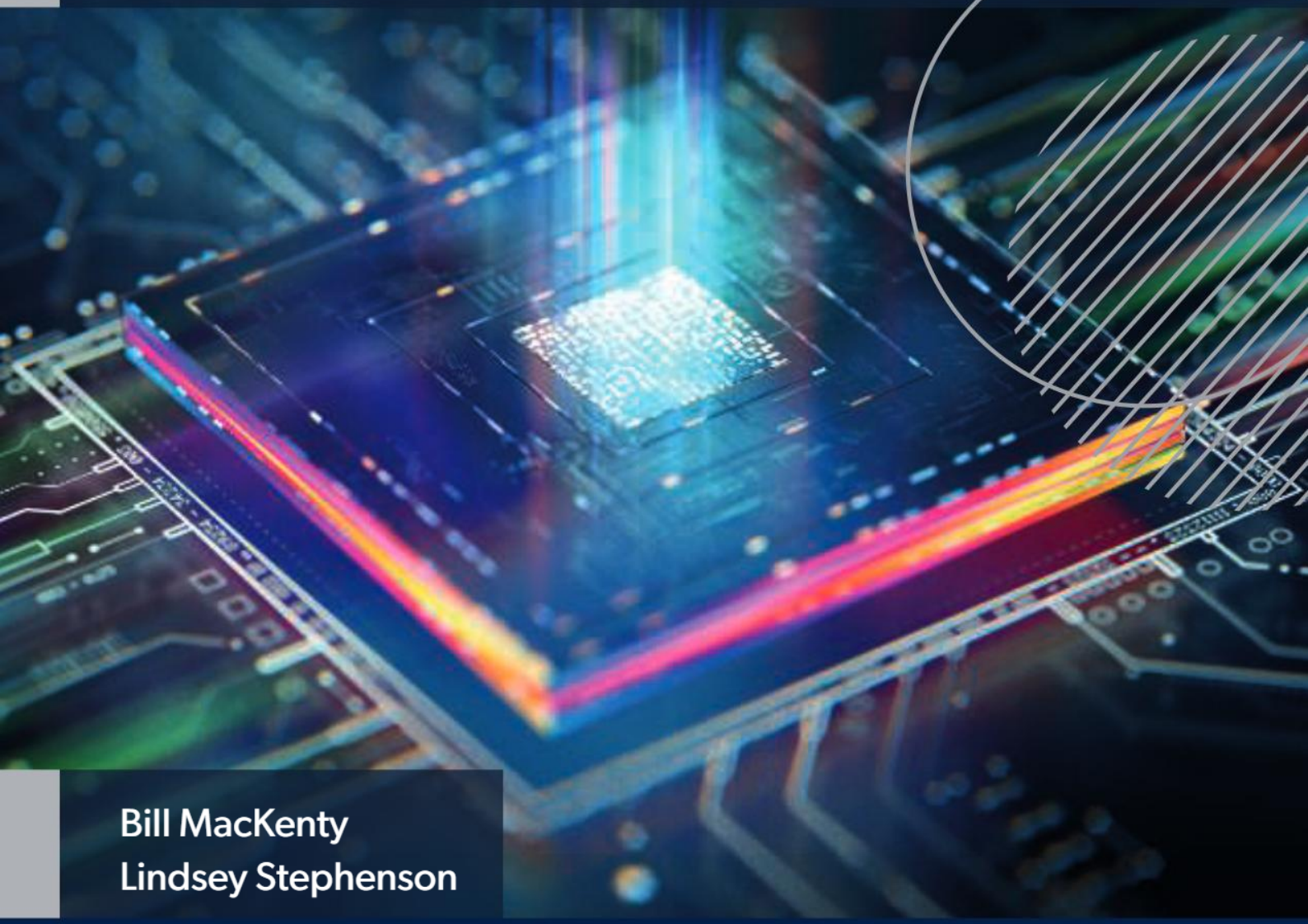
Oxford Resources for IB
Diploma Programme

2025 EDITION

COMPUTER SCIENCE

COURSE COMPANION



Bill MacKenty
Lindsey Stephenson

OXFORD

Take learning online with Kerboodle

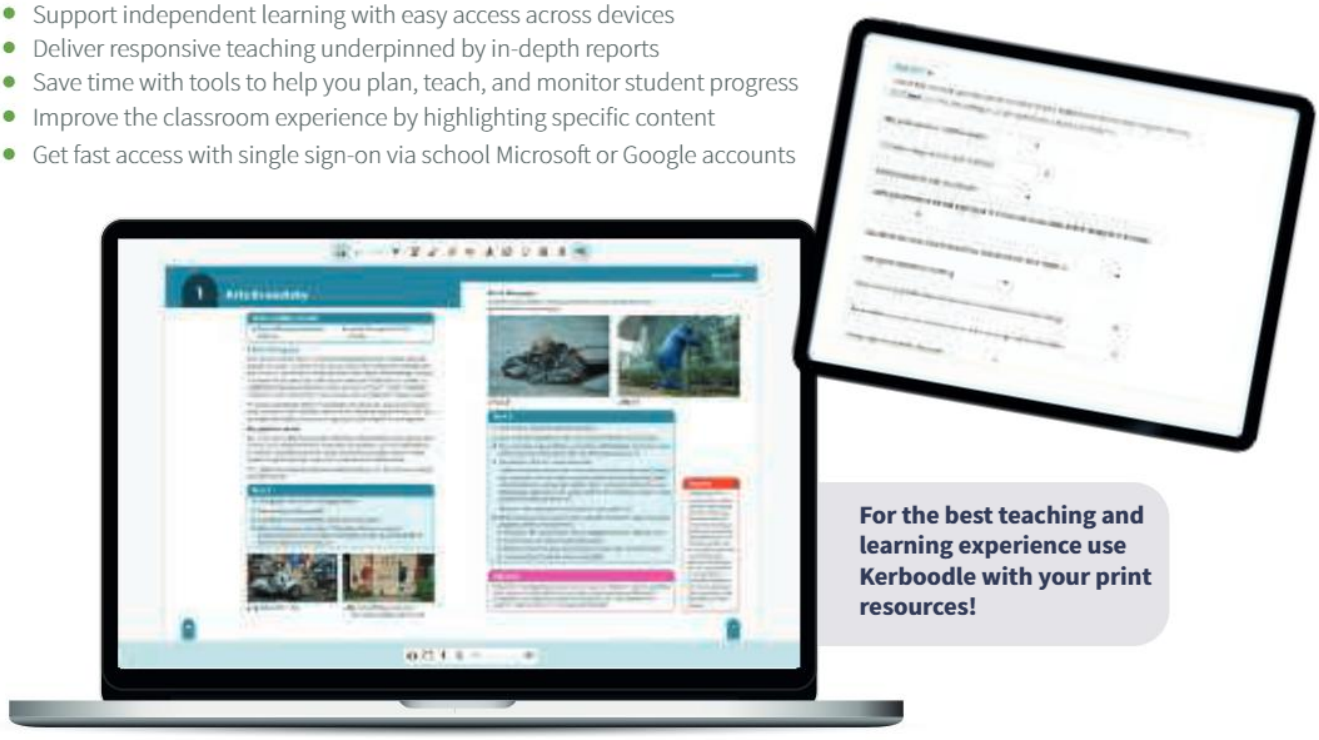
What is Kerboodle?

Kerboodle is a digital learning platform that works alongside your print textbooks to create a supportive learning environment. Available for UK and international curricula, Kerboodle helps you save time and reinforces student learning with a range of supportive resources.

Find out more and sign up for a free trial!

Use Kerboodle to:

- Enable learning anywhere with online and offline access to digital books
- Enhance student engagement with activities and auto-marked quizzes
- Boost performance and exam confidence with assessment materials
- Support independent learning with easy access across devices
- Deliver responsive teaching underpinned by in-depth reports
- Save time with tools to help you plan, teach, and monitor student progress
- Improve the classroom experience by highlighting specific content
- Get fast access with single sign-on via school Microsoft or Google accounts



For the best teaching and learning experience use Kerboodle with your print resources!

For more information, visit:
www.oxfordsecondary.com/kerboodle

Need help?
Contact your local Educational Consultant: www.oxfordsecondary.com/contact-us

Oxford Resources for IB
Diploma Programme

2025 EDITION

COMPUTER SCIENCE

COURSE COMPANION

Bill MacKenty
Lindsey Stephenson
James Abela

OXFORD
UNIVERSITY PRESS

OXFORD
UNIVERSITY PRESS

Great Clarendon Street, Oxford, OX2 6DP, United Kingdom

Oxford University Press is a department of the University of Oxford. It furthers the University's objective of excellence in research, scholarship, and education by publishing worldwide. Oxford is a registered trade mark of Oxford University Press in the UK and in certain other countries.

© Oxford University Press 2025

The moral rights of the authors have been asserted

First published in 2025

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, transmitted, used for text and data mining, or used for training artificial intelligence, in any form or by any means, without the prior permission in writing of Oxford University Press, or as expressly permitted by law, by licence or under terms agreed with the appropriate reprographics rights organization. Enquiries concerning reproduction outside the scope of the above should be sent to the Rights Department, Oxford University Press, at the address above.

You must not circulate this work in any other form and you must impose this same condition on any acquirer

British Library Cataloguing in Publication Data
Data available

978-1-382-06392-0

978-1-382-06393-7 (ebook)

10 9 8 7 6 5 4 3 2 1

The manufacturing process conforms to the environmental regulations of the country of origin.

Printed in Great Britain by Bell and Bain Ltd., Glasgow

The manufacturer's authorised representative in the EU for product safety is Oxford University Press España S.A. of El Parque Empresarial San Fernando de Henares, Avenida de Castilla, 2 – 28830 Madrid (www.oup.es/en or product.safety@oup.com). OUP España S.A. also acts as importer into Spain of products made by the manufacturer.

Acknowledgements

The "In cooperation with IB" logo signifies the content in this textbook has been reviewed by the IB to ensure it fully aligns with current IB curriculum and offers high-quality guidance and support for IB teaching and learning.

The Publisher wishes to thank the International Baccalaureate Organization for permission to reproduce their intellectual property.

The authors have the following acknowledgements and thanks:

Bill MacKenty would like to dedicate this book to his wife Dagmara and daughter Jana. He would also like to thank his students and supportive administrators.

Lindsey Stephenson would like to thank Mamica, Dad and Jamie for all the support, the original Design crew for advice, and her friends and family.

The publisher and authors would like to thank the following for permission to use photographs and other copyright material:

Cover: da-kuk / E+ / Getty Images. **Photos: p2:** Wang Zhibo / Shutterstock; **p3(t):** Gorodenkoff / Shutterstock; **p3(b):** Iaroslav Neliubov / Shutterstock; **p5:** Andrew Derr / Shutterstock; **p6:** Skrypnykov Dmytro / Shutterstock; **p7(l):** © Stefans02; **p7(r):** MZinchenko / Shutterstock; **p8:** S. Singha / Shutterstock; **p11:**BLKstudio / Shutterstock; **p22:** Macrovector / Shutterstock; **p25(t):** effjott.art / Shutterstock; **p25(b):** patruflo / Shutterstock; **p26(t):** Maxx-Studio / Shutterstock; **p26(m):** Poravute Siriphiroon / Shutterstock; **p26(b):** Proxima Studio / Shutterstock; **p27(tl):** Science Photo Library / Alamy Stock Photo; **p27(tm):** Dimedrol68 / Shutterstock; **p27(tr):** Andrei Shumskiy / Shutterstock; p27(mt):nikkytok / Shutterstock; **p27(mb):** Pixel-Shot / Shutterstock; **p27(b):** Derek Brumby / Shutterstock; **p28:** leo_photo / Shutterstock; **p32(l):** Ar_TH / Shutterstock; **p32(r):** Gorodenkoff / Shutterstock; **p34:** iWissawa / Shutterstock; **p38:** Jason Winter / Shutterstock; **p49(t):** Fouad A. Saad / Shutterstock; **p49(b):** VectorArtFactory / Shutterstock; **p50:** pingebat / Shutterstock; **p51(t):** Dan74 / Shutterstock; **p51(m):** Hsyn20 / Shutterstock; **p51(b):** Andrew Rybalko / Shutterstock; **p52:** Sasin Paraksa / Shutterstock; **p67:** posztos / Shutterstock; **p68:** Iconic Bestiary / Shutterstock; **p74:** Frogella / Shutterstock; **p75(t):** MyPro / Shutterstock; **p75(b):** Rob Bouwman / Shutterstock; **p78:** izzuanroslan / Shutterstock; **p80:** catinsyrup / Shutterstock; **p81(t):** releon8211 / Shutterstock; **p81(b):** eamesBot / Shutterstock; **p84:** Skyline Graphics / Shutterstock; **p86(t):** libor.pal / Shutterstock; **p86(mt):** Meaw_stocker / Shutterstock; **p86(mb):** Yuthtana artkla / Shutterstock; **p86(b):** Reproduced with permission of Meggitt SA ; **p87(t):** Elkins Eye Visuals / Shutterstock; **p87(m):** Cultura Creative / Shutterstock; **p87(b):**Black Salmon / Shutterstock; **p88(t):** Sombat Muycheen / Shutterstock; **p88(b):** New Africa / Shutterstock; **p91:** Dabarti CGI / Shutterstock; **p93:** ViDI Studio / Shutterstock; **p100:** EschCollection / Stone / Getty Images; **p108:** Zdeněk Malý / Alamy Stock Photo; **p109(t):** VTECH Vector and Footage / Shutterstock; **p109(b):** Oasishifi / Shutterstock; **p110(t):**PromKaz / Shutterstock; **p110(b):** Skrypnykov Dmytro / Shutterstock; **p112:** JirawatSrimai / Shutterstock; **p113:** Cisco; **p116:** BestForBest / Shutterstock; **p117(t):** elenabsl / Shutterstock; **p117(b):** kozhedub_nc / Shutterstock; **p124:** Aji Asmara / Shutterstock; **p125:** Aji Asmara / Shutterstock; **p134:**MyPro / Shutterstock; **p135:** The 360 Degree / Wikimedia Commons (CC BY-SA 4.0); **p141:** Ninetechno / Shutterstock; **p147(t):** zentilia / Shutterstock; **p147(m):** vinap / Shutterstock; **p147(b):** Alfmaler / Shutterstock; **p149:** Pabkov / Shutterstock; **p151:** vs_vadim / Shutterstock; **p166:** Eugene Mymrin / Moment / Getty Images; **p172:** World Data Center for Climate; **p193:** GGDesigns / Shutterstock; **p223:** Cultura Creative / Shutterstock; **p226(t):** Just Jus / Shutterstock; **p226(b):**Prostock-studio / Shutterstock; **p227:** Dmitry Kalinovsky / Shutterstock; **p229:** didesign021 / Shutterstock; **p236:** Andriy Onufriyenko / Moment / Getty Images; **p237:** VectorMine / Shutterstock; **p239:** mentalmind / Shutterstock; **p240(t):**sfam_photo / Shutterstock; **p240(b):** Visual Generation / Shutterstock; **p242:** MONOPOLY919 / Shutterstock; **p243:**Doggygraph / Shutterstock; **p244:** catris photos / Shutterstock; **p256:** Photobond / Shutterstock; **p260:** Odua Images / Shutterstock; **p262:** Sunward Art / Shutterstock; **p279:** Sidartha Carvalho / Shutterstock; **p280:** Dream01 / Shutterstock; **p284:**Andrey_Popov / Shutterstock; **p287:** chuckchee / Shutterstock; **p288:** fokke baarssen / Shutterstock; **p292:** Qpt / Shutterstock; **p301:** Rebeca R.S / Shutterstock; **p302:** Rebeca R.S / Shutterstock; **p311:** Zapp2Photo / Shutterstock; **p312(t):**Prostock-studio / Shutterstock; **p312(b):** a-image / Shutterstock; **p316:** Cravetiger / Moment / Getty Images; **p318(t):** fizkes / Shutterstock; **p318(b):** ADragan / Shutterstock; **p319(t):** SurfsUp / Shutterstock; **p319(b):** Tero Vesalainen / Shutterstock; **p320:** VectorMine / Shutterstock; **p323:** PeopleImages.com - Yuri A / Shutterstock; **p326:** Wiktoria Matynia / Shutterstock; **p336:** Andriy Onufriyenko / Moment / Getty Images; **p337:** Wenzel Design / Shutterstock; **p338:** Suvit Topaiboon / Shutterstock; **p359:** Pixel-Shot / Shutterstock; **p362:** Lindsay Edwards / Oxford University Press; **p363:** Mjak / Shutterstock; **p396:** Mark Medcalf / Shutterstock; **p416:** andresr / E+ / Getty Images; **p417:** zynpdoodle / Shutterstock; **p418(t):** Eric Isselee / Shutterstock; **p418(b):** Andrew Ong / Shutterstock; **p422(t):**mpohodzhay / Shutterstock; **p422(m):** Nicescene / Shutterstock; **p422(b):** PeopleImages.com - Yuri A / Shutterstock; **p426:** Mikael Damkier / Shutterstock; **p431:** Jason / Shutterstock; **p435(t):**Yandong Yang / Shutterstock; **p435(b):** Karramba Production / Shutterstock; **p437:** Dragon Images / Shutterstock; **p473:**worldswildlifewonders / Shutterstock; **p480:** chaythawin / Shutterstock; **p492:** l i g h t p o e t / Shutterstock; **p523:**PHOTOCREO Michal Bednarek/ Shutterstock; **p528:** Studio Romantic / Shutterstock; **p535:** Rawpixel / Shutterstock; **p548:** Atiwan Janprom / Shutterstock.

Microsoft screenshots © Microsoft 2024. Used with permission from Microsoft.

PyCharm IDE and JetBrains DataGrip screenshots © 2000-2024 JetBrains s.r.o.

Eclipse IDE screenshots © Eclipse Foundation AISBL. All Rights Reserved.

Artwork by Q2A Media, Nathan Jarvis, Green Tree Designing Studio Pvt., and Oxford University Press.

Every effort has been made to contact copyright holders of material reproduced in this book. Any omissions will be rectified in subsequent printings if notice is given to the publisher.

Links to third party websites are provided by Oxford in good faith and for information only. Oxford disclaims any responsibility for the materials contained in any third party website referenced in this work.

Contents

A Concepts of computer science

A1 Computer fundamentals **2**
A1.1 Computer hardware and operation 3
A1.2 Data representation and computer logic 38
A1.3 Operating systems and control systems 66
A1.4 Translation (HL only) 90
A1 End-of-topic questions **98**

A2 Networks **100**
A2.1 Network fundamentals 101
A2.2 Network architecture 124
A2.3 Data transmissions 144
A2.4 Network security 154
A2 End-of-topic questions **165**

A3 Databases **166**
A3.1 Database fundamentals 167
A3.2 Database design 173
A3.3 Database programming 191
A3.4 Alternative databases and data warehouses (HL only) 215
A3 End-of-topic questions **235**

A4 Machine learning **236**
A4.1 Machine learning fundamentals 237
A4.2 Data preprocessing (HL only) 251
A4.3 Machine learning approaches (HL only) 264
A4.4 Ethical considerations 307
A4 End-of-topic questions **314**

B Computational thinking and problem-solving

B1 Computational thinking **316**
B1.1 Approaches to computational thinking 317
B1 End-of-topic questions **335**

B2 Programming **336**
B2.1 Programming fundamentals 337
B2.2 Data structures 350
B2.3 Programming constructs 362
B2.4 Programming algorithms 376
B2.5 File processing 407
B2 End-of-topic questions **413**

B3 Object-oriented programming (OOP) **416**
B3.1 Fundamentals of OOP for a single class 417
B3.2 Fundamentals of OOP for multiple classes (HL only) 473
B3 End-of-topic questions **542**

B4 Abstract data types (ADTs) (HL only) **548**
B4.1 Fundamentals of ADTs 549
B4 End-of-topic questions **581**

Internal assessment (IA): The computational solution **582**

External assessment: Paper 1 and Paper 2 **600**

Index **611**

Answers: www.oxfordsecondary.com/ib-compsci-support

Introduction

The Diploma Programme (DP) computer science course is designed for students in the 16 to 19 age group. The curriculum seeks to examine key concepts in computer science: computer fundamentals, networks, databases and machine learning, and then apply practical skills to support the computational thinking process to solve problems. As with all the components of the DP, this course fosters the IB learner profile attributes (see page vi) in the members of the school community.

Syllabus structure

Topics are organized into two separate themes:

- Theme A: Concepts of computer science
- Theme B: Computational thinking and problem-solving

These themes represent the connection between abstract ideas of how computing system operate (Theme A) and their application using the practical skills of computer science to solve problems through the process of computational thinking (Theme B).

Theme A: Concepts of computer science	Theme B: Computational thinking and problem-solving
A1 Computer fundamentals A1.1 Computer hardware and operation A1.2 Data representation and computer logic A1.3 Operating systems and control systems A1.4 Translation (HL only)	**B1 Computational thinking** B1.1 Approaches to computational thinking
A2 Networks A2.1 Network fundamentals A2.2 Network architecture A2.3 Data transmissions A2.4 Network security	**B2 Programming** B2.1 Programming fundamentals B2.2 Data structures B2.3 Programming constructs B2.4 Programming algorithms B2.5 File processing
A3 Databases A3.1 Database fundamentals A3.2 Database design A3.3 Database programming A3.4 Alternative databases and data warehouses (HL only)	**B3 Object-oriented programming (OOP)** B3.1 Fundamentals of OOP for a single class B3.2 Fundamentals of OOP for multiple classes (HL Only)
A4 Machine learning A4.1 Machine learning fundamentals A4.2 Data preprocessing (HL only) A4.3 Machine learning approaches (HL only) A4.4 Ethical considerations	**B4 Abstract data types (ADTs) (HL only)** B4.1 Fundamentals of ADTs

Coding within the computer science course

You will not be surprised to learn that there is quite a lot of coding in the computer science course. However, this course is accessible, especially at standard level, if you are willing to put in the work. Everyone learning to code experiences moments of failure followed by moments of joy when things finally click. Everyone learns coding at their own pace but, most importantly, everyone can learn to code. There are lots of resources to help you, including this book!

Getting started

This book covers a lot of theory, with no assumption of prior knowledge. You may find it useful to begin by reading the programming topic. There are practice questions to test your learning. The book also includes lots of worked examples and activities to help you get started with your coding.

Many people also find it useful to follow along with online videos when coding. There are many websites to help you if you get stuck, and the coding community can be very supportive.

Practice makes perfect

As with any other skill you might learn throughout your lifetime—speaking another language, driving, painting, playing piano—practice makes perfect! It is true that the more you practise a skill, the easier it becomes, and coding is no exception.

Computational thinking is the ability to understand and identify solutions to problems. When you first start on your coding journey, the vocabulary and coding syntax can be a lot to remember. Writing down what you want to do—your solution—step-by-step, and then translating this into code can help you develop both your computational thinking and your coding skills.

When learning to code, you will probably have to reference example code repeatedly, checking all spaces, full stops and quotation marks. The more you practise this, the more you will remember commands without needing to reference code. As you become more confident, the code you are referencing will become more complex and you will be able to develop more intricate programs. Don't worry if you find yourself referencing previous code even after a few months—most coders do!

Resilience

Part of the fun of coding is problem-solving. It is very rare that code works straight away. This can be frustrating when you are learning, but it is fun to practise resilience and try another way until you solve the problem. The best coders learn from their mistakes, so the more mistakes you make, the more you learn! The best thing you can do when learning to code is just to try and then try again.

ATL Self-management skills

When starting to learn to code, it is often useful to copy and paste the coding fragments into a document and then add notes to help you remember what they do. The first time you encounter an error, take a screenshot of the error and write the solution with the screenshot. The next time you encounter the error, you will know how to solve it.

Create a new document or folder to store all your notes on coding and problem-solving. Maintain this document throughout the course, adding notes as your coding skills develop.

Looking at past IB examination questions will help you to understand what code you need to memorize for the exam. However, keep in mind that not all past paper questions will be relevant for this course.

Find some past papers for this course. As you progress through the course, work through the questions and annotate your answers. Reference the relevant section of your notes for easier revision, and add tips to help you remember things you find challenging.

Where to get help

When learning to code, the following resources may be useful.

- Your teacher: Teachers are usually a good place to start when learning a new skill.
- Textbooks: There are many books dedicated to learning to code.
- Online communities: These are a good place to get specific help when a section of code does not work.
- Friends: Working together to develop code can be helpful. Talking through your ideas and working together to solve problems helps you to learn.

Learning to code can be challenging but it can also be rewarding. Wherever you are on your coding journey, practising coding and talking about coding will help you to learn.

Course book definition

The IB Diploma Programme course books are resource materials designed to support students throughout their two-year Diploma Programme course of study in a particular subject. They will help students gain an understanding of what is expected from the study of an IB Diploma Programme subject while presenting content in a way that illustrates the purpose and aims of the IB. They reflect the philosophy and approach of the IB and encourage a deep understanding of each subject by making connections to wider issues and providing opportunities for critical thinking.

The books mirror the IB philosophy of viewing the curriculum in terms of a whole-course approach: the use of a wide range of resources, international mindedness, the IB learner profile and the IB Diploma Programme core requirements, theory of knowledge, the extended essay, and creativity, activity and service (CAS).

Each book can be used in conjunction with other materials and, indeed, students of the IB are required and encouraged to draw conclusions from a variety of resources. Suggestions for additional and further reading are given in each book and suggestions for how to extend research are provided.

In addition, the course companions provide advice and guidance on the specific course assessment requirements and on academic honesty protocol. They are distinctive and authoritative without being prescriptive.

IB mission statement

The International Baccalaureate aims to develop inquiring, knowledgeable and caring young people who help to create a better and more peaceful world through intercultural understanding and respect.

To this end, the organization works with schools, governments and international organizations to develop challenging programmes of international education and rigorous assessment.

These programmes encourage students across the world to become active, compassionate and lifelong learners who understand that other people, with their differences, can also be right.

The IB learner profile

The aim of all IB programmes is to develop internationally minded people who work to create a better and more peaceful world. The aim of the programme is to develop this person through ten learner attributes, as described below.

Inquirers: They develop their natural curiosity. They acquire the skills necessary to conduct inquiry and research and show independence in learning. They actively enjoy learning, and this love of learning will be sustained throughout their lives.

Knowledgeable: They explore concepts, ideas and issues that have local and global significance. In so doing, they acquire in-depth knowledge and develop understanding across a broad and balanced range of disciplines.

Thinkers: They exercise initiative in applying thinking skills critically and creatively to recognize and approach complex problems and to make reasoned, ethical decisions.

Communicators: They understand and express ideas and information confidently and creatively in more than one language and in a variety of modes of communication. They work effectively and willingly in collaboration with others.

Principled: They act with integrity and honesty, and with a strong sense of fairness, justice and respect for the dignity of the individual, groups and communities. They take responsibility for their own actions and the consequences that accompany them.

Open-minded: They understand and appreciate their own cultures and personal histories and are open to the perspectives, values and traditions of other individuals and communities. They are accustomed to seeking and evaluating a range of points of view and are willing to grow from the experience.

Caring: They show empathy, compassion and respect towards the needs and feelings of others. They have a personal commitment to service and to act to make a positive difference to the lives of others and to the environment.

Risk-takers: They approach unfamiliar situations and uncertainty with courage and forethought and have

the independence of spirit to explore new roles, ideas and strategies. They are brave and articulate in defending their beliefs.

Balanced: They understand the importance of intellectual, physical and emotional balance to achieve personal well-being for themselves and others.

Reflective: They give thoughtful consideration to their own learning and experience. They are able to assess and understand their strengths and limitations in order to support their learning and personal development.

A note on academic integrity

It is of vital importance to acknowledge and appropriately credit the owners of information when that information is used in your work. After all, owners of ideas (intellectual property) have property rights.

To have an authentic piece of work, it must be based on your individual and original ideas with the work of others fully acknowledged. Therefore, all assignments, written or oral, completed for assessment must use your own language and expression. Where sources are used or referred to, whether in the form of direct quotation or paraphrase, such sources must be appropriately acknowledged.

How do I acknowledge the work of others?

The way that you acknowledge that you have used the ideas of other people is through the use of footnotes and bibliographies.

Footnotes (placed at the bottom of a page) or endnotes (placed at the end of a document) are to be provided when you quote or paraphrase from another document or closely summarize the information provided in another document. You do not need to provide a footnote for information that is part of a "body of knowledge". That is, definitions do not need to be footnoted as they are part of the assumed knowledge.

Bibliographies should include a formal list of the resources that you used in your work.

"Formal" means that you should use one of the several accepted forms of presentation. This usually involves separating the resources that you use into different categories (for example, books, magazines, newspaper articles, internet-based resources and works of art) and providing full information so that a reader or viewer of your work can find the same information. A bibliography is compulsory in the extended essay.

What constitutes malpractice?

Malpractice is behaviour that results in, or may result in, you or any student gaining an unfair advantage in one or more assessment component. Malpractice includes plagiarism and collusion.

Plagiarism is defined as the representation of the ideas or work of another person as your own. The following are some of the ways to avoid plagiarism:

- using the words and ideas of another person to support your arguments must be acknowledged
- passages that are quoted verbatim must be enclosed within quotation marks and acknowledged
- email messages, and any other electronic media, must be treated in the same way as books and journals
- the sources of all photographs, maps, illustrations, computer programs, data, graphs, audio-visual and similar material must be acknowledged if they are not your own work
- when referring to works of art, whether music, film dance, theatre arts or visual arts and where the creative use of a part of a work takes place, the original artist must be acknowledged.

Collusion is defined as supporting malpractice by another student. This includes:

- allowing your work to be copied or submitted for assessment by another student
- duplicating work for different assessment components and/or diploma requirements.

Other forms of malpractice include any action that gives you an unfair advantage or affects the results of another student. Examples include taking unauthorized material into an examination room, misconduct during an examination and falsifying a CAS record.

How to use this book

The aim of this book is to develop conceptual understanding, aid in skills development and provide opportunities to cement knowledge and understanding through practice.

Feature boxes and sections throughout the book are designed to support these aims, by signposting content relating to particular ideas and concepts, as well as opportunities for practice. This is an overview of these features:

Developing conceptual understanding

Guiding questions

Each topic begins with a guiding question to get you thinking. When you start studying a topic, you might not be able to answer these questions confidently or fully, but by studying that topic, you will be able to answer them with increasing depth. Hence, you should consider these as you work through the topic and come back to them when you revise your understanding.



Linking questions

Linking questions within each topic highlight the connections between content discussed there and other parts of the course.

Theory of knowledge (TOK)

This is an important part of the IB Diploma Programme. It focuses on critical thinking and understanding how we arrive at our knowledge of the world. The TOK features in this book pose questions for you that highlight these issues.

AHL

Parts of the book have a coloured bar on the edge of the page or next to a question. This indicates that the material is for students studying the DP computer science course at higher level. AHL means "additional higher level".

Link features in the margin will direct you to other parts of the book where a concept is explored further or in a different context. They may also direct you to prior knowledge or a skill you will need, or give a different way to think about something.

Developing skills

Approaches to learning

These approaches to learning (ATL) features prompt you to develop strategies to support the ATL skills of communication, self-management, research, thinking and social skills.

This problem in practice (TPIP)

This feature explores examples of computer science applied to solving real-world problems.

Internal and external assessment

These sections of the book provide guidance on how to apply the computational thinking process to your computational solution, which is the internal assessment for computer science, and how to prepare for your Paper 1 and Paper 2 examinations.

Practical skills

The key practical skills in computer science are algorithmic thinking and programming. These boxes highlight opportunities to develop and apply your practical skills as part of the computational thinking process.

Key term

These introduce the definitions of important terminology used in computer science.

Practising

Worked examples

These are step-by-step examples of how to answer questions. You should review these examples carefully, preferably after attempting the question yourself.

Practice questions

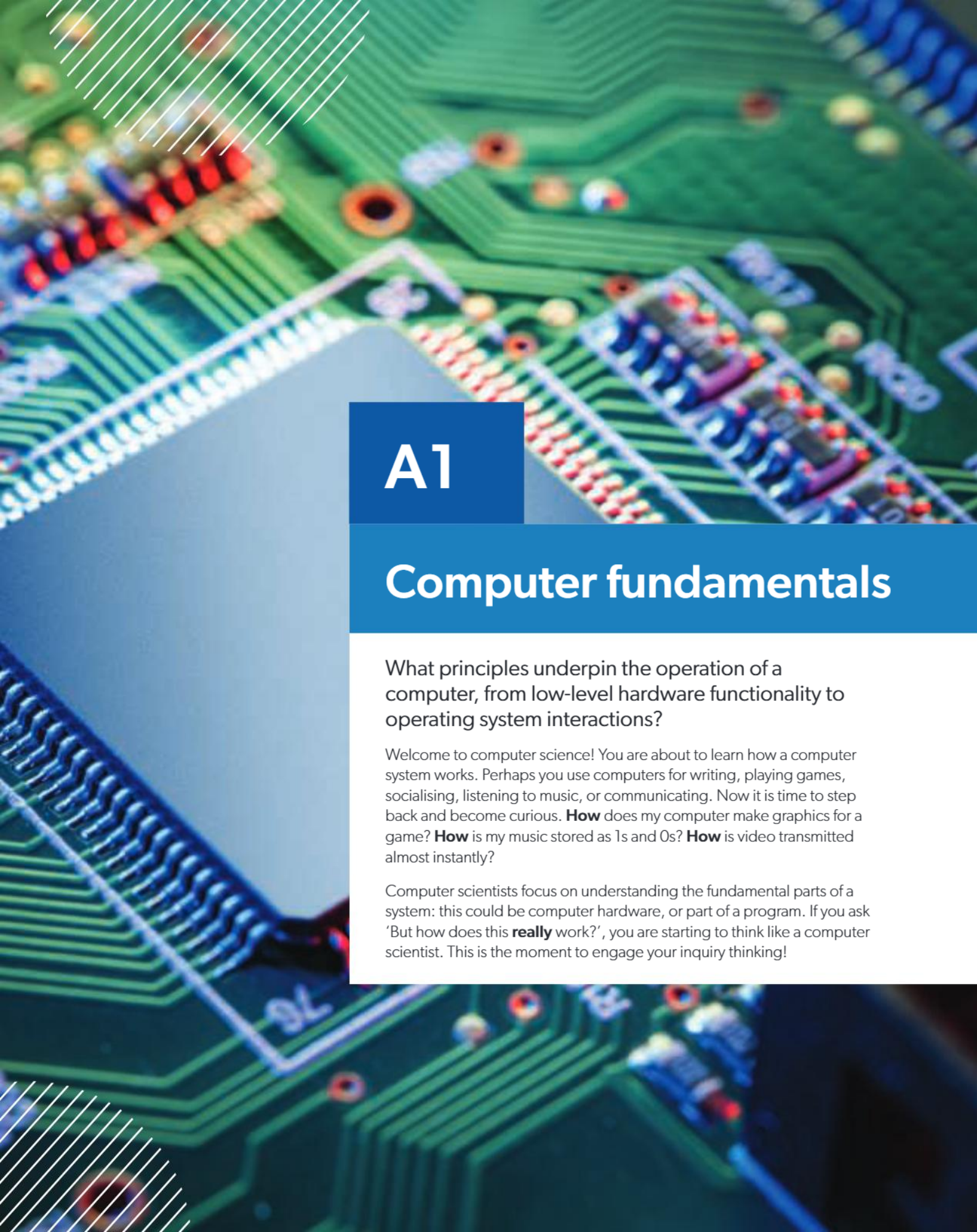
These are designed to give you further practice at using your knowledge and to allow you to check your own understanding and progress.

Activity

These give you an opportunity to apply your knowledge and skills, often in a practical way.

End-of-topic questions

Use these questions at the end of each topic to draw together concepts from that topic and to practise answering exam-style questions.



A1

Computer fundamentals

What principles underpin the operation of a computer, from low-level hardware functionality to operating system interactions?

Welcome to computer science! You are about to learn how a computer system works. Perhaps you use computers for writing, playing games, socialising, listening to music, or communicating. Now it is time to step back and become curious. **How** does my computer make graphics for a game? **How** is my music stored as 1s and 0s? **How** is video transmitted almost instantly?

Computer scientists focus on understanding the fundamental parts of a system: this could be computer hardware, or part of a program. If you ask 'But how does this **really** work?', you are starting to think like a computer scientist. This is the moment to engage your inquiry thinking!

A1.1 Computer hardware and operation

Syllabus understandings

A1.1.1 Describe the functions and interactions of the main CPU components

A1.1.2 Describe the role of a GPU

AHL **A1.1.3** Explain the differences between the CPU and the GPU

A1.1.4 Explain the purposes of different types of primary memory

A1.1.5 Describe the fetch, decode and execute cycle

AHL **A1.1.6** Describe the process of pipelining in multi-core architectures

A1.1.7 Describe internal and external types of secondary memory storage

A1.1.8 Describe the concept of compression

A1.1.9 Describe different types of services in cloud computing



▲ **Figure 1** What is inside a CPU?

A1.1.1 Describe the functions and interactions of the main CPU components

In this section, you will learn about the function and interaction of computer hardware, focusing specifically on the central processing unit (CPU), which is the engine of a computing system.

You will explore **components** of the CPU, such as the arithmetic logic unit (ALU) and control unit (CU). You will discuss the roles and interactions of registers, including the instruction register (IR), program counter (PC), memory address register (MAR), memory data register (MDR), and accumulator (AC). Additionally, you will examine the various buses—control, data, and address—that facilitate communication within the CPU and across the system. This section also covers different types of processors, from single-core to multi-core processors, as well as specialized co-processors.

Key term

Components Distinct functional units that perform specific operations essential for processing instructions.

The CPU

The CPU is the primary computational engine of the computer, responsible for executing instructions. It plays a central role in coordinating data movement within the system, working in conjunction with other components. Understanding its components and their interactions is fundamental to understanding computer operations.

The CPU contains components, registers and buses.



▲ **Figure 2** A CPU

You will learn in more detail how these parts work together in section A1.1.5 The fetch, decode and execute cycle.

TOK

To what extent does understanding the fundamentals of a system help a person use it?

Consider this question in the context of both a car engine and a computer. Does your answer change, depending on the context? Now consider the same question in the context of something less tangible, like human rights or your physical health.

Does understanding how a system works change the way a person can use it?

Components

A component refers to a distinct functional unit or part within the CPU that has a specific role in the processor's operation. The CPU—being the computational engine of the computer—contains key components which work together to execute instructions.

Key components inside of the CPU include the following.

Arithmetic logic unit (ALU) Performs arithmetic and logical operations. It is where the actual computation happens, such as addition, subtraction, and logical operations like AND, OR, NOT, and so on.

Control unit (CU) Responsible for orchestrating the fetch–decode–execute cycle. Its primary functions include decoding and interpreting instructions fetched from memory and generating control signals to activate the appropriate hardware units within the CPU. This involves parsing the instruction's opcode (operation code), which determines the specific action such as reading data, writing data, performing calculations or testing logic (ADD, SUB, AND, OR, and so on).

Registers

A register is a small-capacity, very fast storage location available within the CPU, used to store data temporarily during the execution of programs. It is capable of holding instructions, storage addresses or data.

Table 1 Key registers inside a CPU

Register	Description
Instruction register (IR)	Holds the current instruction being executed. It acts as a temporary holding area for the instruction before it is decoded and executed.
Program counter (PC)	A register that stores the address of the next instruction to be executed. It is incremented automatically after each instruction is executed, pointing to the next instruction in the program's memory location.
Memory address register (MAR)	Stores the address in memory where the next piece of data or instruction is to be found or stored. The MAR interfaces directly with the system's memory address bus.
Memory data register (MDR)	Holds the data or instruction that is being transferred to or from primary memory.
Accumulator (AC)	A special-purpose register used to store intermediate results of operations. It is often used for arithmetic and logical operations. It can also be used as a temporary storage location for other data. While commonly used with arithmetic/logic operations, accumulators can also function as temporary storage for various types of data.

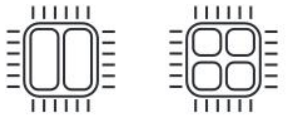
Buses

A bus is a shared communication pathway which transfers data between components within a computer. Key buses connecting a CPU with other components include the following.

Control bus Carries control signals from the Control Unit (CU) to other components, managing actions and timing. These signals can be unidirectional or bidirectional. Examples of control signals are read/write commands, interrupt signals, timing, and acknowledgment signals.

Data bus A pathway for transferring actual data between the CPU, memory, and other components. Its width (e.g., 32-bit, 64-bit) influences transfer speed. The data bus is bidirectional, allowing data flow in both directions.

Address bus A pathway which carries memory addresses from the CPU to specify where data should be read or written.



▲ **Figure 3** Multi-core CPUs. A multi-core CPU can have more than 128 cores

Types of CPU processors

Single-core processors possess one processing unit (core) integrated into a single circuit. This core is the fundamental unit that reads and executes instructions from processes. With a singular processing path, it handles one instruction at a time, following a sequential execution model. This **architecture** was standard in early CPUs, where task completion relied on the linear processing of instructions. Its primary limitation is in executing **parallel processing** demands. As computational tasks become more complex and multitasking becomes essential, single-core processors face limitations in performance, leading to potential bottlenecks in processing efficiency.

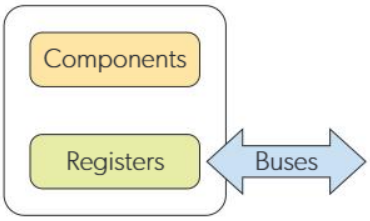
Key terms

Parallel processing A computing technique in which multiple processors or cores within a single machine, or across multiple machines, simultaneously execute different parts of a task or multiple tasks to improve the overall speed and efficiency of computation.

Architecture The design and organization of a computer system's hardware and software components. This includes the structure and functionality to perform computational tasks.



▲ **Figure 4** GPUs are common examples of co-processors



▲ **Figure 5** Components and registers are on the CPU chip. Buses connect the inside of the CPU to outside components such as memory

Multi-core processors consist of two or more independent cores, each capable of processing instructions simultaneously. These cores are integrated onto a single integrated circuit die (chip) or multiple dies in the same package. This architecture enables the processor to handle multiple instructions at once, significantly improving performance over single-core designs, especially for multitasking and parallel processing tasks. Each core can execute a different thread (sequence of instructions) concurrently, enhancing computational speed and efficiency. Multi-core processors are better suited to modern computing needs, including advanced multitasking, complex computations, and high-demand applications. They offer improved performance and efficiency by distributing workloads across multiple processing units.

Co-processors are specialized processors designed to supplement the main CPU, offloading specific tasks to optimize performance. They can be integrated into the CPU or exist as separate entities. By taking on specific tasks, such as graphics **rendering**, mathematical calculations, or data encryption, co-processors free the main CPU to focus on general processing tasks. This division of labour enhances the overall system performance and efficiency. Common examples include graphics processing units (GPUs) for rendering images and videos and data signal processors (DSPs) for handling signal processing tasks.

Key term

Rendering The process of generating an image from a model by executing a series of computational operations that transform 3D scene data into a 2D image.

ATL Self-management skills

The correct use of subject-specific terminology is essential to your ability to communicate your knowledge and understanding of key concepts in computer science. Correct terminology enables the efficient transfer of highly technical information between computer scientists.

In this section, you learned about the main computer components: what they do and how they work. Make a glossary that has an entry for each term that includes this information, such as in the example below.

Term (Abbreviation)	Definition	Linked terms
Arithmetic logic unit (ALU)	Core component of the CPU, where the actual computation happens.	Central processing unit (CPU), control unit (CU)

Include page references so you can find where the term was used in this book — this can be useful for revision.

Continue to add terms to your glossary throughout the course.

A1.1.2 Describe the role of a GPU



▲ **Figure 6** A modern GPU

A graphics processing unit (GPU) is a specialized electronic circuit containing numerous processing cores. For example, the Nvidia GeForce RTX 4080 has 9728 cores.

A GPU is designed to rapidly manipulate and alter memory, accelerating the creation of images for output to a display device. Unlike central processing units (CPUs), which handle a broad range of computations, GPUs possess a highly parallel structure, ideal for complex graphical calculations. GPUs can be integrated (part of a CPU) or discrete (on a separate card). GPUs communicate with software using APIs such as DirectX and OpenGL. As well as processing graphics, GPUs are increasingly used for machine learning and other computationally intensive workloads.

GPU architecture

GPUs have a distinct architecture which sets them apart from conventional CPUs and allows them to process large blocks of data concurrently, leading to more efficient processing for certain types of tasks.

Table 2 Features and characteristics of GPUs

GPU feature	Characteristic	Example
Parallel processing capabilities	Unlike CPUs, which have a few cores optimized for sequential, serial processing, GPUs have thousands of smaller cores designed for parallel processing.	In image processing, a task such as applying a filter to an image can be divided into smaller tasks where the filter is applied to different parts of the image simultaneously. A GPU, with its thousands of cores, can process multiple pixels at the same time, significantly reducing the time required to apply the filter to the entire image.
High throughput	GPUs are optimized for high throughput, meaning they can process a large amount of data simultaneously. This is particularly beneficial in graphics rendering and complex calculations.	In graphics rendering, such as in video games or 3D simulations, a GPU's high throughput allows it to process and display complex scenes in real-time. It can calculate the colour, position and texture of thousands of pixels concurrently, enabling detailed graphics.
Memory	GPUs are equipped with high-speed memory (VRAM), which handles the large textures and data sets required in high-resolution video rendering and complex scientific calculations.	In the context of high-resolution video rendering, the GPU relies on its VRAM to store and manage the textures and data needed for rendering scenes. The high-speed memory allows for the rapid manipulation of this data, enabling the rendering of high-resolution video in real-time without buffering or significant delays.

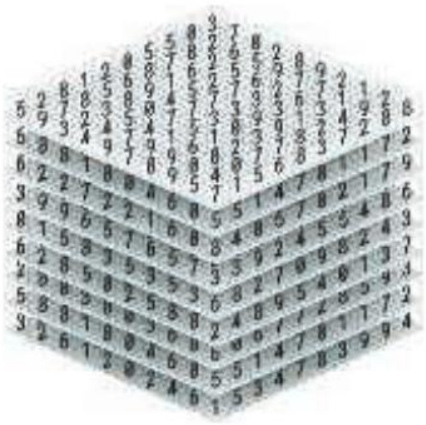
Real-world applications that require graphics rendering and machine learning

GPUs are indispensable for rendering complex graphics in video games. They enable the rendering of high-resolution textures, realistic lighting effects and smooth frame rates, enhancing the gaming experience, providing higher frame rates, and off-loading rendering work from the CPU.



▲ **Figure 7** A screenshot of an alien world from the game No Man's Sky

Consider Figure 7. Notice the lighting, shading, and colours in the sky and grass. Think of the individual leaves on the tree. Each of those colours, pixels, and shapes needs to be calculated and rendered. It is a lot of computational work, especially if the player is moving around (because the lighting and perspective of the shapes change) and the grass and clouds are moving in the breeze.



▲ **Figure 8** GPUs have thousands of cores capable of handling multiple operations simultaneously. This is beneficial for multidimensional data sets where operations often need to be performed across various dimensions in parallel

You will learn more about machine learning in topic A4 Machine learning.



▲ **Figure 9** Researchers can use models and simulations to test drugs or understand disease. Human DNA has 3.1 billion base pairs—that is a lot of possible combinations to model and simulate!



▲ **Figure 10** GPUs can facilitate very rapid changes to lighting, colours and more. In this image, **every single pixel** needs to be assigned a colour and brightness

GPUs are increasingly used in AI and machine learning. Their ability to perform parallel processing allows for faster processing of large data sets which is essential in training neural networks. For example, neural networks, which are at the heart of many AI applications, require the processing of large amounts of data during their training phase. These training processes involve extensive matrix multiplications and other operations which can be parallelized effectively on a GPU.

Scientific computing and large simulations

GPUs are used in various scientific fields for large simulations and data analysis. Their parallel processing capabilities allow for quicker computations in areas such as physics simulations, climate modelling and bioinformatics. For example, in bioinformatics, GPUs play an important role in the processing and analysis of genetic information. One specific application is in genome sequencing, where GPUs are used to align sequences and identify genetic variations quickly. This process involves comparing a massive number of DNA sequences (millions of sequences) against reference genomes to identify mutations and variations, a task that is highly parallelizable.

Graphics design and video editing

In graphics design, especially in the creation of 3D models and environments, GPUs enable designers to visualize their work in real-time. For example, when using software such as Blender or Autodesk Maya, GPUs are utilized to render complex scenes, including lighting effects, shadows and textures, in real-time.

A1.1.3 Explain the differences between the CPU and the GPU

Design philosophy, usage scenarios

Design philosophy

The design philosophy of CPU architecture emphasizes **flexibility and generalizability**, enabling CPUs to efficiently process a wide variety of instructions and data types. In addition, CPUs are designed for **low latency**, meaning they prioritize getting things done quickly, even if it is just one task at a time. Flexibility, generalization, and low latency translate to design choices where CPUs typically have a smaller number of cores compared to GPUs, but each core is more powerful with features such as larger caches and complex logic units. This allows CPUs to handle a wider variety of instructions efficiently.

Another design choice is a focus on branch prediction. CPUs excel at predicting which instruction will be needed next and fetching it in advance. This minimizes wasted time and keeps the core running smoothly.

Finally, instruction versatility is an important design choice. CPUs are built to understand and execute a large set of instructions, making them ideal for running general-purpose software such as web browsers, office applications, and even video games (though not for the intensive graphics processing needed in some games).

AHL

The GPU is built for **high throughput**. It is optimized for tasks that can be decomposed into smaller, independent pieces. GPUs have a large number of cores, each less powerful than a CPU core but designed for simpler tasks. This allows GPUs to process a large amount of data simultaneously. GPUs are optimized for single instruction, multiple data (SIMD) operations, where the same instruction is applied to many data elements at once. GPUs are designed to move data efficiently between cores and memory, prioritizing high bandwidth over complex logic components in each core.

Usage scenarios

Usage scenarios for CPUs include running operating systems and managing system resources, executing general-purpose software tasks, decoding and handling user input (mouse clicks, key presses), and multitasking between different applications.

Usage scenarios in GPUs include processing graphics and rendering images and videos for gaming and video editing, accelerating scientific simulations and machine learning algorithms, encoding and decoding video streams, and cryptocurrency mining.

Core architecture, processing power, memory access, power efficiency

Core architecture

An element in the core architecture of a CPU is the **instruction set architecture (ISA)**, which defines the fundamental operations a CPU can perform. Each instruction in an ISA specifies a particular operation involving arithmetic operations, data movement, logical operations, control flow changes, or system interactions. Unique to a CPU are specific types of instructions such as system management instructions and complex branching instructions.

GPUs also have an ISA. Each instruction in a GPU's ISA is designed towards handling extensive arithmetic operations and data movement, and there is less emphasis on complex logical operations and control flow changes compared to CPUs. This is because GPUs are optimized for throughput over task versatility. Unique to a GPU's ISA are specific types of instructions optimized for graphics rendering and parallel data processing tasks, such as the following.

SIMD instructions: Allow a single operation to be applied simultaneously to a large set of data, which is ideal for the parallel nature of graphics processing and certain types of computational tasks in scientific computing and deep learning.

Texture mapping and manipulation instructions: Essential for graphics processing, these instructions handle tasks like pixel interpolation and texture fetching, which are important for rendering images and videos.

Processing power

Processing power refers to the ability of the device to perform computational tasks. It is a measure of how much work a CPU or GPU can perform in a given amount of time, which directly impacts the performance of software applications running on these processors. Different factors can influence the processing power of a CPU or GPU; for example, the number of cores, **clock speed**, thermal management, and power delivery to the processor.

TOK

CPUs and GPUs are specialized to solve specific types of problems. Working together, they can leverage their specializations to solve much more complex problems than they could individually.

To what extent do the differences between GPUs and CPUs influence our understanding and approach to computational efficiency and problem-solving in modern computing?

In what ways can other systems be combined to solve more complex problems? Are our most stubborn problems the result of not combining the right systems?

Key term

Instruction set architecture (ISA) The element of the CPU that specifies the commands that a processor can understand and execute, such as arithmetic operations, data handling, and control instructions.

Key terms

Processing power The ability of the CPU to execute instructions, often quantified in terms of clock speed.

Clock speed The frequency at which a CPU executes instructions. It is typically measured in gigahertz (GHz).

AHL

TOK

GPUs were devised to render graphics, off-loading that work from the CPU. In 2024, GPUs are indispensable for machine learning and are perfect for highly repetitive tasks.

To what extent do CPU design choices influence our understanding of their role and efficiency in processing diverse instructions? These choices may include prioritizing flexibility, generalizability or low latency. How does the CPU's ability to work with diverse instructions compare to the specialized tasks handled by GPUs?

The nature of the choices made during design can dictate how a system is used in the future. Should systems be highly specialized? Do any design choices for current systems limit our ability to solve highly complex problems?

CPUs are designed with fewer, more powerful cores than GPUs. They feature higher clock speeds and advanced technologies such as branch prediction and out-of-order execution, which optimize sequential task processing. Multithreading capabilities and a high instructions per cycle (IPC) rate enable CPUs to efficiently manage multiple tasks and complex computational instructions.

GPUs possess a large parallel architecture with hundreds to thousands of cores, enabling efficient handling of large-scale parallel processing tasks. High memory bandwidth and specialized cores, such as tensor cores, enhance their ability to process large blocks of data quickly and effectively. The SIMD capabilities allow GPUs to perform the same operation on multiple data points at once, maximizing throughput for suitable tasks. While individual GPU cores may operate at a lower clock speed and with simpler instructions compared to CPU cores, the sheer number of these cores allows for a tremendous amount of parallel processing power. Remember, the "simple instructions" in GPU cores are designed for parallel execution, making them specialized rather than inherently less powerful.

Memory access

Memory access in the context of computing hardware such as CPUs and GPUs refers to how these processors retrieve and manipulate data stored in computer memory. Each type of processor handles memory access differently based on its architectural design, which impacts its overall performance.

CPUs utilize a memory hierarchy to manage data access efficiently (for more on this, refer to A1.1.4). This hierarchy typically includes several levels of caches (L1, L2, and sometimes L3). This hierarchy is optimized to minimize memory latency—the delay from issuing a memory request to receiving the data. CPUs often operate in multi-core environments, necessitating mechanisms such as cache coherence protocols. These protocols ensure that multiple CPU cores have a consistent view of the data in the memory, preventing data conflicts and ensuring data integrity across the cores.

Modern GPUs often use a unified memory architecture, which allows them to access a large, shared pool of memory which both the GPU and CPU can address. GPUs are designed with high bandwidth memory. These memory types are optimized for the high-throughput requirements of GPU tasks, enabling fast data transfer rates that support the processing capabilities of hundreds to thousands of parallel cores. Unlike CPUs, which are optimized for low-latency access, GPUs prioritize memory throughput.

To summarize memory access, CPUs utilize low-latency memory because they need to rapidly switch between tasks, retrieve data from memory, and execute operations based on that data with minimal delay. GPUs utilize high memory throughput because they handle large volumes of data and need to feed hundreds to thousands of parallel cores simultaneously.

Power efficiency

CPUs and GPUs use electrical power to perform computational tasks. Power efficiency is a significant aspect of processor design and operation, especially in environments where energy consumption impacts cost, thermal management, and system longevity.

For CPUs, power efficiency is often defined by how much computing work can be performed per watt. This ratio measures the computational output relative to power consumption, providing a benchmark to compare the efficiency of different CPU models. Higher performance per watt indicates a more power-efficient CPU.

Modern CPUs incorporate advanced power management technologies that adjust the power usage based on the workload. Techniques such as dynamic voltage and frequency scaling (DVFS) allow CPUs to reduce power consumption when full processing power is not needed. Another aspect of power efficiency is thermal design power (TDP). TDP is the maximum amount of heat generated by a CPU that the cooling system in a computer is designed to dissipate under normal conditions. Efficient CPUs manage to deliver more performance while staying within a lower TDP envelope.

GPUs, particularly those used in high-performance computing and gaming, also prioritize power efficiency, given their potential for high power consumption. Since GPUs handle many tasks simultaneously, their power efficiency often benefits from their ability to spread workload across many cores, reducing the power per task when compared with serial processing. Like CPUs, many GPUs incorporate features that help reduce power usage when full graphical power is not required, such as lowering clock speeds or powering down idle cores. GPUs are generally more power-efficient at parallel processing tasks than CPUs.



▲ **Figure 11** Managing heat is directly related to performance

CPUs and GPUs working together: Task division, data sharing, and coordinating execution

CPUs and GPUs must collaborate effectively to optimize computing tasks. Understanding how they work together is important.

CPUs are designed for general-purpose processing, and GPUs are designed for parallel processing capability. General-purpose processing is executing a variety of instructions with complex logic and decision-making. Parallel processing is performing the same operation simultaneously on multiple pieces of data. You can think of this like the roles in a professional kitchen: the head chef (CPU) ensures everything is in order. The specialized cooks (GPU) handle the high-volume tasks.

Task division

When CPUs and GPUs work together, tasks are typically divided based on their nature and requirements. Sequential and control-intensive tasks remain the domain of the CPU, which manages the system, performs logic and control operations, and processes tasks that require frequent decision-making. Some examples of tasks executed by a CPU are OS management, network communication, and input/output handling.

Parallelizable data-intensive tasks are offloaded to the GPU, where hundreds or thousands of smaller, independent tasks can be executed simultaneously. This includes operations such as matrix multiplications in machine learning algorithms, pixel processing in graphics rendering, and data analysis in scientific computations.

Data sharing

For CPUs and GPUs to work together effectively, they must share data. Initially, data is stored in primary memory, accessible by the CPU. For the GPU to process this data, it must be transferred to the GPU's memory through the peripheral component interconnect express (PCIe) bus, which can be a bottleneck. Some architectures offer unified memory, allowing both the CPU and GPU to access the same physical memory space, simplifying data sharing and minimizing transfer overheads.

AHL

Coordinating execution

Coordinating the execution between CPUs and GPUs involves using programming languages such as CUDA (for Nvidia GPUs) and OpenCL. These languages provide the necessary tools to manage how tasks are divided between CPUs and GPUs, including memory management and task synchronization. This often involves synchronization primitives like barriers or events. Modern systems can dynamically allocate tasks to CPUs and GPUs based on the current workload and the nature of the tasks, optimizing for performance and energy efficiency.

A **barrier** is a synchronization mechanism used to ensure that multiple threads or processes reach a certain point in execution before any are allowed to proceed. Think of it as a checkpoint in a race that all runners (threads) must reach before the race can continue to the next segment. In parallel programming, barriers are used to implement a point of synchronization where threads pause their execution until all participating threads have reached the barrier point. Once the last thread arrives at the barrier, all threads are released to proceed with their subsequent operations.

An **event** is a synchronization primitive that allows threads to wait for certain conditions to be met before continuing their execution. Unlike barriers, which synchronize a group of threads at a predefined point, events are more flexible and can be used to signal one or more waiting threads that a specific condition has occurred, such as the completion of a task or the availability of required data.

A1.1.4 Explain the purposes of different types of primary memory

Registers, cache (L1, L2, L3), random-access memory (RAM), and read-only memory (ROM)

Key terms

Volatile memory Requires power to maintain the stored data. When the power is turned off, the data is lost. An example of volatile memory is RAM (random-access memory).

Non-volatile memory Retains stored information even when the power is turned off. Examples include ROM (read-only memory), SSDs (solid state drives), and HDDs (hard disk drives).

Primary memory serves as the central workspace for the CPU, facilitating the storage and quick access to data and instructions **which are in active use**.

Registers

The fastest and smallest type of memory, built directly into the CPU. They store data, instructions and addresses the CPU is actively executing. This memory is **volatile**. The fundamental unit of data handled by a CPU's architecture is the "word size", which describes the size of a register. In general, registers hold 32 or 64 bits of data.

Cache (L1, L2, L3)

High-speed memory residing on or close to the CPU. Caches bridge the speed gap between registers and RAM, holding frequently used data and instructions for quick retrieval. This memory is volatile.

- L1 cache typically ranges from 32 KB to 256 KB per core, with data and instruction caches separate in some architectures.
- L2 cache typically ranges from 256 KB to 16 MB per core or shared across multiple cores.
- L3 cache typically ranges from 2 MB to 32 MB shared across all cores in a CPU.

Main memory (RAM)

The primary workspace of the computer. RAM temporarily stores the currently running operating system, processes, and active data and instructions. This memory is volatile. RAM capacity is typically measured in gigabytes (GB). In 2024, 16 GB of RAM would be adequate for multitasking, light gaming, and content creation, while 32+ GB would be ideal for power users, heavy gaming, video editing, and professional applications. 32-bit operating systems generally have a limit of around 4 GB of RAM, while 64-bit systems can address a much larger amount of RAM. The authors are quite certain these memory baselines will increase significantly in the future.

Read-only memory (ROM)

A non-volatile memory that stores essential instructions and data for the computer to start up (for example, the BIOS or firmware). Data in ROM is typically not modifiable during normal computer operation, although it is modifiable via special processes. ROM's role is primarily for firmware storage and it is not directly involved in the day-to-day memory access hierarchy involving registers, cache and RAM. It is better considered as a separate entity focused on system boot-up and low-level startup operations.

The interaction of the CPU with different types of memory to optimize performance

Computing systems are designed to be as efficient as possible. The CPU interacts with different types of memory in a hierarchical manner to optimize performance. This interaction is guided by the principles of minimizing latency and maximizing throughput for data and instruction access. **Latency** is the time it takes for data to move from its source to its destination. Latency is usually described as low or high. Throughput is the amount of data that can be processed or transmitted in a given amount of time.

The hierarchical memory system, from registers to RAM, serves an important role in this optimization process. The CPU interacts with these different memory types as follows.

Registers

Direct interaction: The CPU has internal registers, which are the fastest type of memory available. These registers are used to store immediate data which the CPU needs for current operations, such as operands for arithmetic operations, address pointers, and the results of operations.

TOK

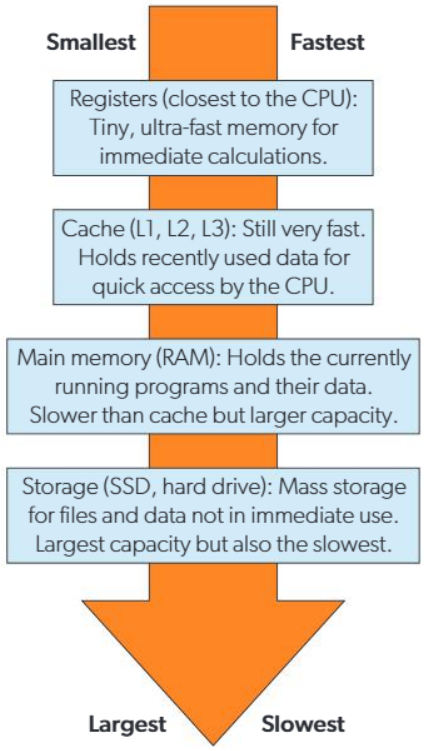
One of the guiding principles in hierarchical memory design is to keep frequently used, related data as close to the CPU as possible: this enables faster processing of data and instructions.

How does the hierarchical structure of memory components influence our understanding of processing speed and data accessibility in computer systems?

Grouping similar data involves making assumptions about *how* data is similar. How might assumptions about relatedness of data be helpful or problematic?

Key term

Latency The time delay between the transmission of a data packet and its reception. It is typically measured in milliseconds. Latency represents the total time taken for the data to travel from the source to the destination. It is synonymous with the term "lag".



▲ **Figure 12** The hierarchical memory system. The closer the memory is to the CPU, the faster and smaller in capacity it is

Optimization: By utilizing registers for the most immediately necessary data and instructions, the CPU minimizes the need to access slower types of memory, significantly speeding up processing times.

Cache (L1, L2, L3)

Hierarchical use: Cache memory serves as a high-speed intermediary between the CPU and the slower main memory (RAM). It is divided into levels (L1, L2, L3) based on proximity to the CPU, with L1 being the smallest and fastest, and L3 being larger and slightly slower but still faster than RAM.

Data and instruction prefetching: Modern CPUs use sophisticated algorithms to predict which data and instructions will be needed soon, and pre-emptively load them into the cache. This anticipatory action reduces the time the CPU spends waiting for data, thus optimizing performance.

Spatial and temporal locality: Caches exploit the principles of spatial (data near recently accessed data is likely to be accessed soon) and temporal (recently accessed data is likely to be accessed again soon) locality to keep relevant data close at hand, further optimizing performance.

Main memory (RAM)

Central data repository: RAM holds the operating system, applications and data that are currently in use. It provides a much larger space for data storage compared to caches and registers.

Interaction through memory controller: The CPU interacts with RAM via the memory controller, a chipset that manages data transactions between the CPU and RAM. This controller plays a critical role in managing access times and optimizing the flow of data between the CPU and RAM.

Virtual memory: The system can use a portion of the hard drive (or SSD) as virtual memory, extending the available memory space. The CPU manages the swapping of data between RAM and virtual memory, though this process is significantly slower than accessing RAM directly.

Read-only memory (ROM)

Boot-up process: While not directly involved in the CPU's data processing tasks, ROM contains the firmware or BIOS necessary for the initial booting of the computer and basic hardware initialization. The CPU accesses this read-only data at startup to load the operating system from secondary storage (for example, HDD or SSD) into RAM.

The relevance of terms "cache miss" and "cache hit"

The CPU first checks the registers for the data it needs. If the data is not found in the registers, the CPU checks the L1 cache, then the L2 cache, and then L3 cache, and finally the main memory (RAM). Sometimes, the CPU finds the data it needs (a hit) and sometimes the data is not found (a miss).

A cache hit occurs when the CPU finds the data it needs in a cache. This is the fastest case, as the CPU can access the data almost instantaneously. A cache miss occurs when the data is not found in the cache. In this case, the CPU needs to fetch the data from main memory, which is slower than accessing the cache.

To minimize cache misses, the CPU uses a variety of techniques, including the following.

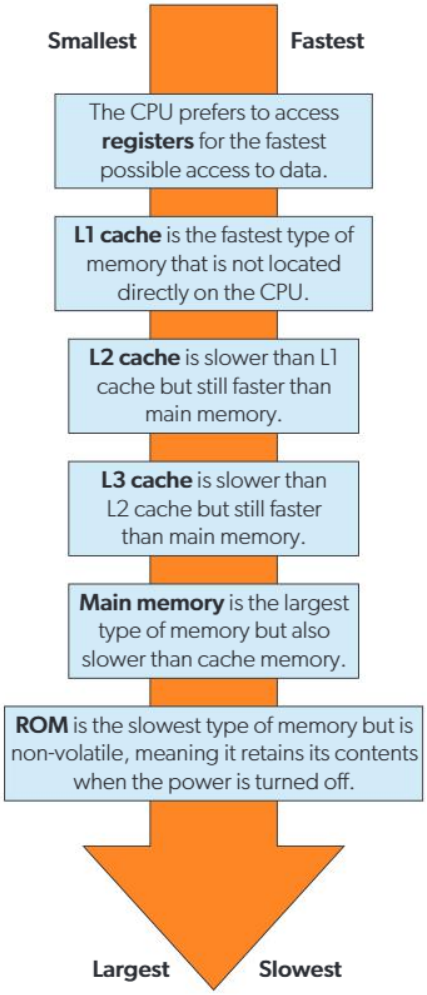
Prefetching: The CPU can predict which data it will need in the future and fetch it into the cache before it is actually needed.

Memory allocation: The operating system can allocate data to appropriate memory locations based on usage patterns to maximize cache hits.

Cache replacement policies: The operating system can choose which data to evict from the cache when it is full to optimize cache hit rates.

By optimizing the interaction between the CPU and different types of memory, the CPU can significantly improve the performance of computer systems.

Figure 13 shows some additional points to remember about the interaction of the CPU with different types of memory.

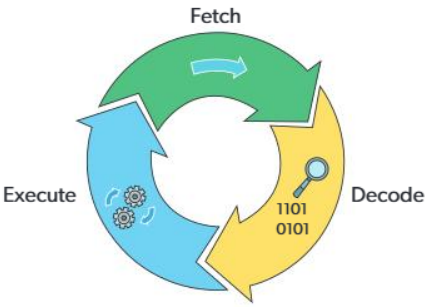


▲ **Figure 13** The interaction of the CPU with different types of memory

Table 3 Just how fast is memory?

Unit of time	Definition	Examples
1 nanosecond (ns)	one billionth of a second	The time to execute one machine cycle by an Intel Pentium 4 1 GHz microprocessor. Light travels 12 inches (30 cm) in 1 ns.
1 microsecond (μs)	one millionth of a second	The time to execute one machine cycle by an Intel 80186 microprocessor.
1 millisecond (ms)	one thousandth of a second	The length of the flash strobe on a camera. It takes around 50–80 ms to blink an eye.
1 second (s)	1 second	The average time to say 'one Mississippi' aloud.

Type of memory	Access time	Size	Latency
Register	0.2–0.5 nanoseconds	Smallest	Lowest
Level 1 (L1) cache	3–7 nanoseconds	Small	Lower
Level 2 (L2) cache	5–10 nanoseconds	Larger than L1 cache	Moderate
Level 3 (L3) cache	10–40 nanoseconds	Larger than L2 cache	Moderate to high
Main memory (RAM)	50–70 nanoseconds	Largest	Highest



▲ **Figure 14** The fetch–decode–execute cycle

A1.1.5 Describe the fetch, decode and execute cycle

The fetch–decode–execute cycle is the fundamental cycle of instruction execution in a computer. This cycle is also known as the instruction cycle. When you run a program, it is executed by your CPU. The fetch–decode–execute cycle describes how instructions are executed.

It consists of three main steps.

1. **Fetch:** The CPU fetches an instruction from memory.
2. **Decode:** The CPU decodes the instruction, which means it interprets the instruction into a set of low-level operations that the CPU can execute.
3. **Execute:** The CPU executes the instruction, which means it carries out the low-level operations that were specified in the decoded instruction.

The fetch–decode–execute cycle is repeated for each instruction in a program. The speed of the CPU is largely determined by how quickly and efficiently it can execute this cycle. Executing a single machine language instruction involves a sequence of operations which are performed by the CPU. These operations can be broadly categorized into three main phases: fetch, decode and execute.

Representing primary memory

Whenever you think about primary memory (RAM), drawing a table like Table 4 can be helpful. This is a simplified representation where the memory address is on the left, and the corresponding data or instructions are on the right. This is especially useful when working with the fetch–decode–execute cycle because it helps you understand the relationship between addresses and their contents in RAM.

Table 4 Example reference table showing memory address and corresponding data or instructions

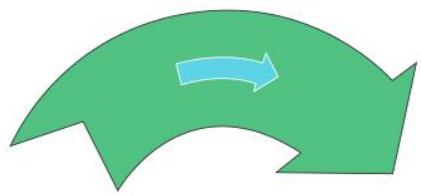
Address	Data/Instructions
01	`42`
02	`69`
03	`LOAD 01`
04	`ADD 01, 02`
05	`STORE 04, 03`

In the Table 4 example:

- address 01 and 02 store data values
- address 03 contains a LOAD instruction to load data from address 01
- address 04 has an ADD instruction to add the contents of addresses 01 and 02
- address 05 includes a STORE instruction to store the result from address 04 into address 03.

This example is quite basic, and real primary memory is much more complex, with all values stored in binary. However, using this representation can help you conceptualize how the fetch–decode–execute cycle operates in practice.

Fetch phase



▲ **Figure 15** Fetch

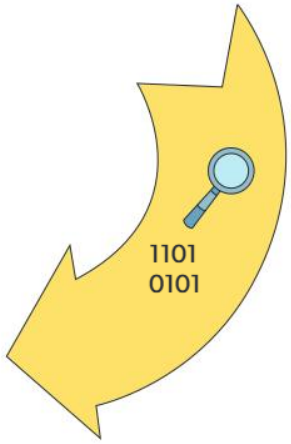
In the fetch phase (Figure 15), the CPU retrieves a machine language instruction from main memory (typically RAM). This involves the CPU sending a request to the memory to fetch the instruction. The address of the instruction to be fetched is stored in the memory address register (MAR), and the fetched instruction is then transferred to a special register called the instruction register (IR) in the CPU.

Decode phase

In the decode phase (Figure 16), the CPU interprets the machine language instruction fetched during the previous phase. This phase is managed primarily by the CPU's control unit, which decodes the instruction by analysing these components:

- **opcode (operation code):** dictates the type of operation
- **operands:** the data to be operated on
- **addressing modes:** determine how to locate the operands.

Additionally, the decode phase involves checking the validity of the instruction and ensuring that the operands are within an acceptable range. This phase also calculates the effective address of the operands, which is the actual memory address where the operands are located.



▲ **Figure 16** Decode

Execute phase

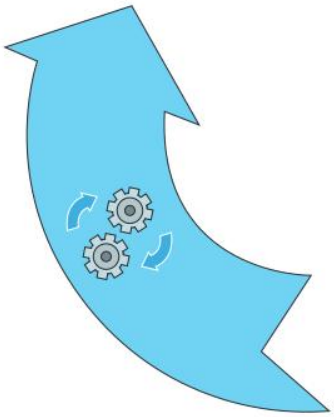
During the execute phase, the CPU performs the operation specified by the instruction decoded in the previous phase. The specific operation carried out is dictated by the opcode. Common operations include the following.

Arithmetic operations: Addition, subtraction, multiplication, and division, typically performed by the ALU.

Logical operations: Logical functions like AND, OR, and NOT, also managed by the ALU.

Memory access operations: Load (reading data from memory into a register) and store (writing data from a register to memory) operations, which involve interaction with the system's memory.

Control operations: Conditional and unconditional jumps which alter the flow of execution based on specific conditions.



▲ **Figure 17** Execute

TOK

Recognising patterns is an important part of being a computer scientist. The fetch–decode–execute cycle is a pattern that is still used today. Modern computers can cycle through this pattern more than a billion times a second. What patterns could be developed today that may be helpful in the future?

Outside of computing, are there patterns of doing work that are better than other patterns?

Outside of computing, how do you identify if some working patterns are better than other patterns? Consider this in the context of completing homework for school. You might work for 30 minutes, then take a 10 minute break and repeat this cycle until you have finished the work. Is that the best pattern? How do you know?

How does knowledge of a cycle of instruction shape our perception of computational efficiency and program execution?

ATL Research skills

Opcodes (operation codes)

Opcodes are part of the instruction in a machine language program which specifies the operation to be performed. Here are some examples of opcodes in assembly language, which is directly related to machine code opcodes.

- `MOV` Move data from one location to another.
- `ADD` Add two operands.
- `SUB` Subtract one operand from another.
- `MUL` Multiply two operands.

Each opcode corresponds to a specific machine language instruction. The binary codes for these operations can differ between each architecture, such as x86, ARM or MIPS.

Research the following opcodes to find out what operations they specify: `DIV`, `AND`, `OR`, `JMP`.

Worked example 1

The instruction `ADD R1, R2, R3` adds the values in registers R2 and R3 and stores the result in register R1. Describe the fetch, decode and execute phases of an ADD instruction cycle in a CPU.

Solution

Fetch phase

1. Memory address: The program counter (PC) points to the memory address where the instruction is stored.
2. Fetch: The CPU fetches the instruction **`ADD R1, R2, R3`** from this address.
3. IR: The fetched instruction is placed into the instruction register (IR).

Decode phase

1. Control unit: The control unit reads the instruction in the IR.
2. Decoding: The control unit decodes the instruction and understands it needs to perform an addition.
3. Operands: It identifies R2 and R3 as the source registers containing the operands.
4. Destination: It identifies R1 as the destination register where the result will be stored.

Execute phase

1. ALU operation: The ALU receives the contents of R2 and R3.
2. Addition: The ALU performs the addition Value in R2 + Value in R3.
3. Store Result: The result of the addition is stored back into R1.

Worked example 2

The instruction `CMP R1, R2` compares the values in registers R1 and R2 and sets the CPU status flags (for example, zero flag, carry flag) based on the comparison result. Describe the fetch, decode and execute phases of a CMP instruction cycle in a CPU.

Solution

Fetch phase

1. Memory address: The PC holds the memory address of the next instruction to be executed, which is where the **`CMP R1, R2`** instruction is stored.
2. Fetch: The CPU fetches the **`CMP R1, R2`** instruction from the memory location specified by the PC.
3. IR: The fetched instruction **`CMP R1, R2`** is placed into the IR.

Decode phase

1. Control unit: The control unit reads the instruction in the IR to understand what needs to be done.
2. Decoding: The control unit decodes the instruction and determines that it is a compare operation. It understands that it needs to compare the values in R1 and R2.
3. Operands: The CU identifies R1 and R2 as the source registers containing the operands for the comparison.
4. No Destination: Unlike arithmetic operations, **`CMP`** does not have a destination register for storing results, as it only affects the status flags.

Execute phase

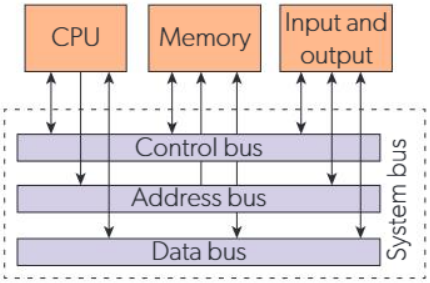
1. ALU operation: The ALU receives the values stored in registers R1 and R2.
2. Comparison: The ALU compares the values in R1 and R2. This operation involves subtracting the value in R2 from the value in R1 to determine the result.
3. Set flags: Based on the comparison result, the ALU updates the status flags:
   - Zero flag: Set if R1 is equal to R2.
   - Negative flag: Set if R1 is less than R2.
   - Carry flag: Set if there is a borrow in subtraction (often used in unsigned comparisons).
   - Overflow flag: Set if there is an arithmetic overflow (less relevant for a basic compare).
4. PC update: The PC is incremented to point to the next instruction in sequence, preparing the CPU for the next fetch phase.

Worked example 3

Outline how a CPU processes a command for calculating an average score.

Solution

1. Initial setup
    - Scores: Stored in consecutive memory addresses.
    - Number of scores: Stored in a special memory location.
    - Sum of scores: Accumulated in Register R1.
    - Current score: Loaded into Register R2.
    - Counter: Number of scores processed, kept in Register R3.
    - Average: To be calculated and stored in Register R4.
2. Fetch-decode-execute cycle 1: Initialize registers
    - Fetch: `LOAD R1, #0` (Initialize R1 to zero for sum)
    - Decode: Recognize `LOAD` operation. Operand is immediate value `#0`.
    - Execute: Set R1 to 0.
    - Fetch: `LOAD R3, #0` (Initialize R3 as counter).
    - Decode: Decode to a load immediate value into R3.
    - Execute: Set R3 to 0.
3. Load number of scores
    - Fetch: `LOAD R5, address_of_num_scores` (Load number of scores into R5).
    - Decode: Decode to load from memory address into R5.
    - Execute: Access memory and put the number of scores into R5.
4. Sum scores loop
    - Loop start: Check if R3 (counter) is less than R5 (total scores).
    - Fetch: `LOAD R2, (address_of_scores + R3)` (Load current score to R2).
    - Decode: Decode load instruction, calculate address as offset by R3.
    - Execute: Access memory, load score into R2.
    - Fetch: `ADD R1, R1, R2` (Add current score to sum).
    - Decode: Recognize add operation, operands R1 and R2.
    - Execute: Compute sum, store back in R1.
    - Fetch: `ADD R3, R3, #1` (Increment counter).
    - Decode: Decode addition, immediate increment.
    - Execute: Increment R3 by 1.
    - Conditional loop: Check if R3 is still less than R5.
    - Execute: If R3 < R5, jump back to loop start.
5. Calculate average
    - Fetch: `DIV R4, R1, R5` (Divide sum by number of scores to get average).
    - Decode: Decode divide operation.
    - Execute: Perform division, store result in R4.



▲ **Figure 18** Interaction between buses and registers. Arrows indicate data flow

The interaction between memory and registers via the three buses (address, data, and control)

Data, instructions, and control signals need to be transferred between a CPU and off-chip components in an organized, predictable and efficient manner.

This movement between memory and registers is facilitated through a set of three **buses.**

A bus is a communication system which transfers data between various components within a computer. The term is used to refer to both the physical connection—wires and printed circuit tracks—as well as the protocols (rules and signalling standards) used to manage the communication.

The three buses are the address bus, data bus, and control bus. These buses form the communication channels between the CPU, main memory, and other components within a computer system. Sometimes these three buses are referred to collectively as the **system bus**.

Bits and bytes are units of data. You will learn more about them in subtopic A1.2.

Address bus

The address bus carries memory addresses from the CPU to the memory controller. The memory address specifies the location of the data or instruction that the CPU wants to access. The address bus is typically a parallel bus, meaning it consists of multiple wires, one for each bit of the memory address. The number of wires in the address bus determines the maximum amount of memory that the CPU can address.

Data bus

The data bus carries data between the CPU and memory. It can be used to transfer both instructions and data. The data bus is typically a parallel bus, meaning it consists of multiple wires, one for each byte of data. The width of the data bus determines the maximum amount of data that can be transferred in a single operation.

Control bus

The control bus carries various control signals necessary for managing and coordinating the interactions between the CPU and other system components such as the memory. These signals include—but are not limited to—read, write, and acknowledge. The read signal prompts the memory controller to retrieve data from a specified address and send it to the CPU. The write signal initiates a data storage operation at a specific memory address. The acknowledge signal confirms the completion of a data transfer. Additional signals on the control bus may include interrupt requests, which allow peripherals to request CPU attention, and clock signals, which help synchronize data transfers and ensure that operations occur at the correct times and in the right sequence.

The three buses work together to enable the CPU to read data from memory, write data to memory, and access instructions from memory. The address bus specifies which memory location to access, the data bus transfers the data between the CPU and memory, and the control bus provides the signals that control the data transfer.

Examples of the interaction between memory and registers

Worked example 4

Outline how a CPU processes a command to read the value stored at memory location 0x1A3B.

Solution

1	Address bus	The CPU sends the memory address 0x1A3B to the memory controller via the address bus. This bus carries the memory address specifying where the data the CPU needs is located.
2	Control bus	The CPU sends a read signal over the control bus to the memory controller. This signal instructs the memory controller to retrieve data from the address specified.
3	Data bus	Once the memory controller receives the read signal, it fetches the data from memory location 0x1A3B and sends it back to the CPU using the data bus. This bus carries the actual data from memory to the CPU.
4	Control bus	An acknowledge signal is sent back to the CPU via the control bus once the data has been successfully transferred to indicate that the read operation is complete.

Worked example 5

Outline how a CPU processes a command to write the value 0x7E to memory location 0x4F2.

Solution

1	Address bus	The CPU uses the address bus to send the target memory address 0x4F2 to the memory controller, indicating where the data should be stored.
2	Data bus	The value 0x7E is sent to the memory controller via the data bus. This bus carries the data that the CPU intends to store in memory.
3	Control bus	The CPU sends a write signal to the memory controller over the control bus. This signal commands the memory controller to store the data received on the data bus at the address provided by the address bus.
4	Control bus, continued	After the data has been written to the specified memory location, the memory controller sends an acknowledge signal back to the CPU via the control bus, confirming that the write operation has been successfully completed.

A1.1.6 Describe the process of pipelining in multi-core architectures

Pipelining

Pipelining allows a processor to execute multiple instructions at the same time. This can significantly improve the overall throughput of the system.

The following laundry analogy is not perfect, but it helps to explain pipelining.

Non-pipelined laundry

1. Wash: Put a load of clothes in the washing machine and wait for the cycle to finish.
2. Dry: Transfer the wet clothes to the dryer and wait for them to dry.
3. Fold: Take the dry clothes out and fold them.

Key term

Pipelining A technique for improving the performance of computer processing by dividing the execution of a process into multiple parts and allowing those parts to operate simultaneously.

AHL



▲ **Figure 19** The laundry process

In this scenario, each task must be fully completed before starting the next. If each task takes 30 minutes, completing a single load of laundry would take 1.5 hours.

Pipelined laundry

1. Wash (Load A): Put the first load of clothes in the washing machine.
2. Dry (Load A): When Load A finishes washing, transfer it to the dryer.
3. Wash (Load B): While Load A is drying, start a second load of laundry in the washing machine.
4. Fold (Load A): When Load A finishes drying, fold the clothes.
5. Dry (Load B): When Load B finishes washing, transfer it to the dryer.
6. Wash (Load C): While Load B is drying, start a third load of laundry in the washing machine.
7. Fold (Load B): When Load B finishes drying, fold the clothes.

Notice that, in the pipelined example, tasks 2 and 3 can happen at the same time (apart from moving the clothes from one task to the next). You do not need to wait for the clothes to finish drying before you start the wash cycle. Similarly, tasks 4, 5 and 6 can all happen at the same time.

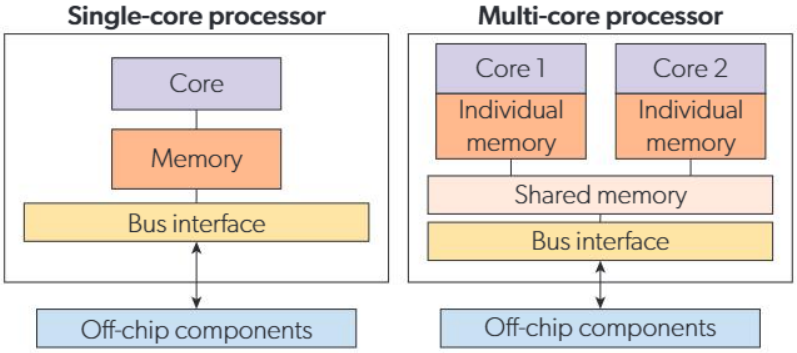
By overlapping tasks, you increase efficiency. After the initial setup time, you can complete a load of laundry every 30 minutes instead of every 1.5 hours.

Multi-core processors

Key term

Multi-core architecture A computing architecture where a single physical processor contains multiple integrated cores. A processor can have a single core or multiple cores.

Each core in a **multi-core** processor includes all the fundamental components you would find in a single-core processor, such as an ALU, control unit, registers, and often their own Level 1 (L1) and sometimes Level 2 (L2) cache. While each core operates independently, they typically share some higher-level caches (like L3 cache) and primary memory (RAM). This is different from single-core processors, which do not need mechanisms for inter-core communication.



▲ **Figure 20** Comparison of single-core and multi-core processor architecture

This integration facilitates faster communication between the cores compared with separate processors or chips, leading to improved performance for applications designed to take advantage of parallel processing.

How pipelining improves the overall system performance in multi-core architectures

Pipelining enables multi-core computing. Each core typically has its own pipeline, allowing it to fetch, decode, execute, and write back instructions concurrently. This means that while one instruction is being executed, another can be decoded, and yet another can be fetched, thus improving overall processing efficiency and speed.

In the fetch phase, each core independently fetches instructions from memory. When cores share L3 cache or RAM, the fetch mechanism must manage cache coherency. Cache coherency ensures CPUs have data consistency and data correctness when multiple caches store copies of the same data from main memory.

In the decode phase, after fetching, each core decodes the instruction into its opcode and operands. This decoding happens in parallel in each core, allowing diverse instructions to be processed simultaneously across the cores.

In the execute phase, each core executes its decoded instructions. This could involve ALU operations (such as arithmetic and logic), accessing registers, or interacting with memory. Execution units in each core work independently, though they might need to synchronize access to shared resources.

Finally, in the writeback phase the results from the execution phase are written back to the memory or registers. This phase is also managed independently by each core, though write operations to shared memory may require coordination to maintain data integrity and coherence.

Overview of how cores in multi-core processors work independently and in parallel

Each core in a multi-core processor can operate independently of the others. This means that each core has its own set of resources, including registers, the ALU, and sometimes its own L1 cache, allowing it to execute a separate thread or process. The operating system (OS) allocates different tasks or threads to individual cores based on scheduling algorithms and the current load on the system.

In parallel operation, the cores work together on a single task to improve performance. This is often achieved through parallel processing, where a task is divided into smaller subtasks which can be executed simultaneously by different cores. For example, in a quad-core processor a task can be divided into four parts, with each core working on one part. This division and simultaneous execution of tasks can significantly reduce the time required to complete complex computations or process large amounts of data.

TOK

What happens when several different people try to talk to you at the same time? Imagine you are watching a movie, chatting with a friend and reading a message. How easy is it to pay attention to everything that is happening? What is the cost of doing many things at the same time? Is faster always better?

What are the implications of simultaneous instruction execution in computing systems?

AHL

For cores to work in parallel effectively, they must coordinate and communicate with each other. This is often facilitated by shared resources, such as L2 or L3 cache, which allow cores to exchange data and synchronize their operations. The effectiveness of this communication and the ability to minimize contention for shared resources are important factors in the overall performance of multi-core processors in parallel tasks.

Worked example 6

Imagine a task which involves processing a list of numbers to calculate their squares and then storing the results. Work through the steps of a problem to clarify how pipelining functions.

Solution

1. Divide the task into the four stages: fetch, decode, execute and writeback.
2. Break the problem into smaller tasks, and put these into the four stages from part 1.
   i. Fetch: Retrieve numbers from an array.
   ii. Decode: Determine the operation to be performed (in this case, squaring the number).
   iii. Execute: Perform the calculation (square the number).
   iv. Writeback: Store the result back in a result array.

Here is how the pipelining could be visualized over time (T1, T2, T3, and so on represent time units).

Explanation

Look at T1 to T4. In the first time unit (T1), Core 1 fetches the first number. In the next time unit (T2), while Core 1 decodes the squaring of the first number, Core 2 fetches the second number. In the next time unit (T3), Core 1 executes the squaring of the first number, Core 2 decodes the second number and Core 3 fetches the third number, and so on. At T4, each core is working on a different stage of the process for different data elements.

The tasks are overlapping. Notice how each core picks up a new task as soon as it completes its part of the previous task. For instance, as soon as Core 1 finishes executing the squaring of the first number at T3, it immediately moves on to the writeback stage at T4 and then fetches the next number at T5.

Each core processes different data elements in parallel, significantly speeding up the overall process compared to a single-core processor performing each task sequentially.

Time	Core 1	Core 2	Core 3	Core 4
T1	Fetch 1	----------	----------	----------
T2	Decode 1	Fetch 2	----------	----------
T3	Execute 1	Decode 2	Fetch 3	----------
T4	Writeback 1	Execute 2	Decode 3	Fetch 4
T5	Fetch 5	Writeback 2	Execute 3	Decode 4
T6	Decode 5	Fetch 6	Writeback 3	Execute 4
T7	Execute 5	Decode 6	Fetch 7	Writeback 4
T8	Writeback 5	Execute 6	Decode 7	Fetch 8
T9	Fetch 9	Writeback 6	Execute 7	Decode 8

AHL

A1.1.7 Describe internal and external types of secondary memory storage

Unlike primary memory (RAM), which is volatile and temporary, **secondary memory storage** is non-volatile and retains data even when the computer is powered off, enabling data persistence. Examples of secondary memory storage include hard disk drives (HDDs), solid-state drives (SSDs), optical discs, and flash drives. Secondary memory can be broadly classified into two categories: internal and external storage. You might also encounter secondary memory storage referred to as external memory, auxiliary storage, or mass storage.

Internal hard drives (SSD, HDD) and embedded multimedia cards (eMMCs)

Internal hard disk drives (HDDs)

HDDs are the most common type of internal storage, and they are typically used in desktop computers, laptops and servers. HDDs offer high storage capacities and are very reliable, but they can be comparatively slow to access data (average sustained data transfer rate for a hard disk is between 100 MB/s to 200 MB/s).

Hard disk drives were named "hard" to distinguish their rigid, non-removable platters coated with magnetic material, where data is stored, from the flexible and removable media used in earlier floppy disks. The term "hard" reflects the physical durability and fixed nature of these storage devices, in contrast to the easily bendable early floppy disks.

An HDD consists of a spindle that rotates at high speed and a read/write head that moves across the surface of the platters. The read/write head can read and write data to the platters by magnetizing tiny areas on the surface.

Advantages of HDDs include large storage capacities, high reliability, and relative affordability. Disadvantages of HDDs include slow data access speeds, and they can also be noisy. HDDs are used in desktop computers, laptops, servers and external hard drives.

Internal solid state drives (SSDs)

SSDs serve the same purpose as traditional hard disk drives (HDDs) but are faster and consume less power. Unlike HDDs, which use mechanical parts and magnetic platters to read and write data, SSDs have no moving parts, leading to quicker access times and lower latency. The average data transfer rate for an SSD can significantly exceed that of HDDs, in some cases reaching 2,500 MB/s to 7,000 MB/s.

Solid state technology's non-volatile nature allows it to retain data without power, similar to traditional hard drives but without the physical constraints of rotating disks.

The core storage mechanism in an SSD is NAND flash memory, characterized by its ability to retain data without power. Data is stored in floating-gate transistors (memory cells) arranged within a grid-like structure on the flash chip. To store data, the floating gate in each cell is electrically charged or discharged. This trapped charge represents a binary value (0 or 1). NAND flash is organized into blocks, which are further subdivided into pages. Data is written to flash memory in pages and erased in blocks.



▲ **Figure 21** A 5.25 inch floppy disk

Key term

Secondary memory storage A type of data storage which is not directly accessible by the CPU and is used for storing data on a long-term basis.

Key term

Hard disk drive (HDD) A type of storage device that stores data on magnetic disk platters.



▲ **Figure 22** A hard disk drive

Key term

Solid state drive (SSD) A type of storage device that uses integrated circuit assemblies to store data persistently, typically using flash memory.



▲ **Figure 23** Two types of SSD



▲ **Figure 24** A 16 MB eMMC card

Key term

Embedded multimedia card (eMMC) A type of solid-state storage device that is typically used in portable devices such as smartphones, tablets and digital cameras.



▲ **Figure 25** An external hard drive

Key terms

External hard disk drives (HDDs) and solid-state drives (SSDs) External storage devices that can be connected to a computer via a cable (for example, a USB or Thunderbolt cable).

Optical drives Devices that can read and write data to optical discs, such as CDs, DVDs and Blu-ray discs.

To read data, the charge level within the floating gate is sensed, determining the stored binary value. A specialized SSD controller manages operations within the SSD, including data transfer and error correction. Due to the physical limitations of flash cells (they can only endure a finite number of write and erase cycles), SSDs employ wear-levelling algorithms. These techniques ensure that write and erase operations are distributed evenly across all blocks in the memory, prolonging its lifespan. NAND flash comes in several varieties (SLC, MLC, TLC, QLC), with each storing a different number of bits per cell. These types offer variations in cost, speed and endurance.

Advantages of SSDs include rapid access times, higher data transfer rates, reduced power consumption, and increased durability due to the lack of mechanical parts. However, SSDs can be more expensive per gigabyte than HDDs and may have limited write cycles, although technology improvements are continually mitigating these downsides.

Embedded multimedia cards (eMMCs)

An **eMMC** is a type of flash storage found in many low to mid-range devices. It integrates the multimedia card interface, flash memory, and flash memory controller on a small package on the device's motherboard. While eMMC is solid-state and offers decent performance, it is generally slower than most SSDs, especially in terms of sequential read/write speeds and input/output operations per second (IOPS). eMMC storage usually offers lower capacities than SSDs (typically ranging from 16 GB to 256 GB) and is less expensive. Its lower cost makes it an attractive option for manufacturers of budget-friendly devices.

eMMCs utilize NAND flash memory, described above. Advantages of eMMCs include fast data access speeds, highly reliable, relatively shockproof. Disadvantages of eMMCs include greater expense than HDDs, and comparatively lower storage capacities than HDDs and SSDs.

External hard drives, optical drives, flash drives, memory cards, and network attached storage

External hard drives (SSD, HDD)

External hard drives come in a variety of sizes and capacities, and they can be used to back up important data, transfer files between computers, or simply store data. The basic functionality of external HDDs and SSDs is the same as for internal HDDs and SSDs.

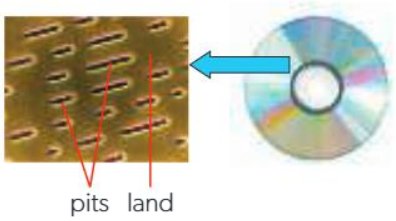
Optical drives

Optical drives were once a common type of external storage, but they have been largely supplanted by HDDs and SSDs. However, optical drives are still useful for archiving large amounts of data and for compatibility with older devices.

An optical disc stores data as tiny indentations known as pits, encoded in a spiral track on the disc's surface. The areas between pits are known as lands.

The process of reading and writing data on an optical disc involves a combination of physical structure, optical technology, and digital encoding. An optical disc is made of a polycarbonate plastic disc, with a reflective metal layer (usually aluminium) on which data is encoded. The top of the disc may have a label, and the entire disc is coated with a protective lacquer.

The data on an optical disc is stored in a single spiral track that starts near the centre and moves outward, containing a sequence of pits and lands. To read data, an optical drive uses a laser diode to emit a beam of light that passes through the polycarbonate layer of the disc and reflects off the metal layer. The presence of pits and lands causes variations in the way light is reflected back to a photodiode sensor in the player.



▲ **Figure 27** Pits and lands on an optical disk



▲ **Figure 26** An external DVD drive

Specifically, when the laser hits a land, it is reflected directly back, but when it hits a pit, the light is scattered or reflected at a different angle, causing a change in the intensity of the light received by the sensor. The photodiode sensor detects these changes in light intensity and converts them back into an electrical signal, which is then processed and interpreted as the original digital data.

The advantages of optical drives include high storage capacities and compatibility with older devices. However, they have slower data access speeds, and discs can be fragile and susceptible to scratches and dust.



▲ **Figure 28** A laser in an optical drive

Flash drives

Flash drives, also known as USB flash drives, are small, portable storage devices that can plug directly into a computer's USB port. They are convenient for transferring small to medium-sized files between computers. Flash drives are also relatively affordable.

Flash drives utilize NAND flash memory, described above.

The advantages of flash drives include portability, fast data transfer speeds and relative affordability, while disadvantages include lower storage capacities than HDDs and SSDs, and they can be susceptible to data loss if not properly handled. Losing a flash drive can present a security risk if confidential data is stored on the flash drive.



▲ **Figure 29** A typical flash drive

Memory cards

Memory cards are small, removable storage devices that are typically used in portable devices, such as smartphones, tablets and digital cameras. They come in a variety of sizes and capacities. Memory cards are very convenient for transferring files between devices and for backing up data.

Memory cards utilize NAND flash memory, described above. Advantages of memory cards include portability, compact design, and varied storage capacities. However, memory cards can be lost or damaged easily, and may not be compatible with all devices.



▲ **Figure 30** A collection of memory cards



▲ **Figure 31** Network attached storage (NAS)

Network attached storage

Network attached storage (NAS) devices provide centralized storage for multiple devices on a network. NAS devices are typically connected to a router, and they can be accessed by any device on the network. NAS devices are ideal for sharing files, backing up data and storing media files. NAS typically uses hard disk drives (HDDs), solid state drives (SSDs), or a combination of both in hybrid setups.

NAS can be configured with RAID (redundant array of independent disks) to protect data against drive failures and ensure data availability. NAS can be easily expanded by adding additional hard drives to accommodate growing data needs.

Advantages of NAS include centralized storage, shared access for multiple devices, scalable storage, and a high degree of control for system administrators. The disadvantages of NAS devices are they are more expensive than individual external drives, they require configuration, and they require a network connection.

A1.1.8 Describe the concept of compression

You compress data to reduce its size, enable more efficient storage, and enable faster transmission over networks. Compression can significantly decrease the amount of disk space needed for files and the bandwidth required for transferring them, facilitating more efficient use of resources.

In general, the more storage you need, the more expensive it is. Network speed can be metered, meaning you pay for what you use. The basic point here is that compression can save money.

Additionally, compression is vital in managing large data sets, optimizing web content delivery, and enhancing the performance of applications by minimizing load times and storage requirements.

TOK

Do you remember what you learned in all your subjects at school last week? What about last term? And last year? If you need to revise this information, how do you find it? Compare the persistence and relative accessibility of this data. Things you learned more recently may be easier to remember without prompting. For topics you studied some time ago, you may need to reread your class notes. Making revision notes while you are learning a topic can help you remember it later.

How does the distinction between secondary memory storage and primary memory challenge our understanding of data persistence and accessibility?

Lossless and lossy compression methods

Lossless and lossy compression are two methods of reducing the size of digital data. While both aim to decrease file size, they differ in their approach and the impact they have on the original data.

Lossless compression

Lossless compression identifies redundant patterns and eliminates them without losing any essential information. This ensures that the decompressed file is an exact replica of the original file, preserving its integrity and quality. However, the compression ratio achieved through lossless methods is generally lower compared with lossy compression.

Worked example 7

A text file contains the following sequence of characters:

`AAAAAAABBBBBCCCCCCCCCCDDDDDDDDDD`

Use **run length encoding (RLE)** to compress this sequence.

Solution

To use RLE to compress this sequence, count the number of consecutive occurrences of each character.

There are 7 As, 5 Bs, 9 Cs, and 10 Ds. So the code becomes:

`7A5B9C10D`

In the worked example, the compressed version 7A5B9C10D requires significantly fewer characters to represent the original sequence. When decompressing, the process is reversed, and the original sequence is perfectly reconstructed. This illustrates how lossless compression removes redundancy without losing any original data, ensuring an exact replica of the original file upon decompression.

Lossy compression

Lossy compression employs a more aggressive strategy to achieve higher compression ratios. It deliberately discards some data, typically minor details that are less noticeable to the human eye or ear. This allows for significantly smaller file sizes, making it suitable for applications such as audio, video and images. However, the discarded information can compromise the quality of the reconstructed file.

For example, consider an image that is a simple 4×4 pixel grid, where each pixel is either black or white. This image could be part of a larger picture, with a segment looking like this:

Black	White	Black	White
White	Black	White	Black
Black	White	Black	White
White	Black	White	Black

For a lossy compression, you might reduce the resolution by averaging the colours of every 2×2 block, a process known as downsampling. In this simplistic case, the result of averaging black and white is grey, so the 4×4 grid might be compressed to a 2×2 grid like this:

Grey	Grey
Grey	Grey

Here, the specific details of which pixels were black and which pixels were white are lost—only the average colour is retained. This demonstrates how lossy compression achieves higher compression by sacrificing some details—in this case, the exact colour of each pixel—while trying to preserve the overall impression of the image.

Differences between lossless and lossy compression

Lossless compression prioritizes data integrity and ensures that the original file is preserved, while lossy compression prioritizes smaller file sizes and sacrifices some data quality for the sake of efficiency. The choice between lossless and lossy compression depends on the specific application and the trade-off between file size and data integrity.

TOK

There is a difference between freshly made food and frozen food; between a home cooked meal and fast food. Compression changes an original data into something smaller, and in some cases (with lossy compression) we lose something from the original.

How does the practice of data compression influence our understanding of resource efficiency and cost management in digital storage and transmission? What broader implications does this have for the accessibility and performance of technology?

Table 5 Key differences between lossless and lossy compression

	Lossless	Lossy
Data integrity	Preserves the original data, ensuring that the decompressed file is identical to the original file.	Introduces variations in the reconstructed file due to the deliberate discarding of certain data deemed less critical for the intended use case.
Compression ratio	Achieves lower compression ratios compared to lossy compression. This is because it aims to preserve all original data, reducing the redundancy potential.	Achieves higher compression ratios by discarding parts of the data that are considered less important, significantly reducing file size at the cost of some loss in quality.
Applications	Preferred for applications where data integrity is critical, such as backups, archival storage, and editing original documents.	Used in audio, video and images, where minor loss of quality may be acceptable for smaller file sizes.
Perceptual redundancy	Does not use perceptual redundancy.	Lossy compression algorithms often exploit perceptual redundancy—the fact that the human senses may not distinguish between certain subtle differences in data. This allows them to discard information that is less noticeable without significantly impacting the overall quality.
Reversible vs irreversible	Reversible, allowing for the perfect reconstruction of the original data from the compressed file.	Irreversible, meaning once data has been discarded during the compression process it cannot be restored, leading to a permanent loss of certain information.

Key term

Run-length encoding (RLE) A basic form of lossless compression. The number of consecutive occurrences of each character are counted and the code changed to a mix of numbers and letters. For example, XXXXXYYY becomes 5X3Y.

Run-length encoding, transform coding

Run-length encoding (RLE) and transform coding are two techniques employed in data compression, each with its own approach and applications.

Run-length encoding is a simple and efficient lossless compression technique that is well suited for data with repetitive patterns. It works by identifying and replacing consecutive occurrences of the same value with a single code that represents the length of the run. For instance, the sequence AAAAA would be encoded as 5A.

RLE is particularly effective for compressing data such as text files, which often contain long runs of blank spaces or punctuation marks. It is also used in image compression algorithms such as fax compression, where it efficiently represents large areas of uniform colour. (If you want to make your computer science teacher feel old, ask them what a fax machine is!)

Transform coding is a more complex and powerful compression technique based on mathematical transformations. It works by applying a transformation to the data, such as the discrete cosine transform (DCT), which rearranges the data into a representation that highlights its statistical structure.

By transforming the data into a more manageable form, transform coding can identify and exploit redundancy in the data more effectively. This allows it to achieve higher compression ratios compared with RLE, particularly for data with complex patterns.

Transform coding is widely used in image compression algorithms such as JPEG, where it transforms the image into a frequency domain representation before applying quantization to reduce the number of bits required to represent each component. It is also used in audio compression algorithms such as MP3, where it transforms the audio signal into a frequency domain representation before applying various coding techniques to achieve efficient compression.

Applications of RLE and transform coding

RLE is commonly used for compressing text files, images with repetitive patterns (for example, fax images), and binary data with long runs of consecutive zeros.

Transform coding is primarily used for compressing images and audio signals, where its ability to exploit statistical redundancy and achieve high compression ratios is highly beneficial.

Table 6 Types of compression and their applications

Type	Example	Application
Lossless	• GIF (graphics interchange format) While GIF can support lossless compression, it is not always the case. GIF has a limited colour palette (256 colours) so is better for simple graphics. • TIFF (tagged image file format) TIFF is commonly used in professional photography and desktop publishing, as it supports lossless compression and various colour depths. • PDF (portable document format) Images or objects within a PDF might be compressed using lossy techniques, even while the overall structure supports lossless). • ZIP • PNG (portable network graphics)	Backups, documents, archives
Lossy	• JPEG (Joint Photographic Experts Group) • MP3 (MPEG-1 Audio Layer III)	Images, audio, images
Run-length encoding (RLE)	• Fax compression • Text compression	Fax images, text files
Transform coding	• JPEG • MP3 • MPEG (Moving Picture Experts Group)	Images, audio, videos

The choice between RLE and transform coding depends on the specific type of data and the desired compression ratio. For data with simple repetitive patterns, RLE offers a straightforward and efficient approach. For data with complex patterns and higher compression requirements, transform coding provides a more powerful and versatile solution.

A1.1.9 Describe different types of services in cloud computing

The term "cloud" refers to cloud computing, a paradigm which allows organizations to access and utilize computing resources (such as servers, storage, databases) over the internet to offer flexible resources. The cloud is not a physical entity but a network of remote servers in data centres around the globe. Organizations can save money by only paying for the computational resources they use, rather than paying for an expensive server which they might not use at full capacity all the time.

The three main cloud service models are software as a service (SaaS), platform as a service (PaaS), and infrastructure as a service (IaaS). Each model provides a different level of control and flexibility over IT infrastructure and software, influencing resource availability and management. The choice between these models depends on the specific needs and capabilities of the organization.



▲ **Figure 32** How people imagine the cloud



▲ **Figure 33** What the cloud really looks like

Software as a service (SaaS)

SaaS is a cloud computing model where **software applications** are hosted and managed by a third-party provider and accessed by organizations through a web browser or mobile app. SaaS eliminates the need for organizations to install and maintain software on their own devices, making it convenient and cost-effective.

Key characteristics of SaaS

All-inclusive: Organizations have access to the entire application suite, including updates and maintenance, without any upfront costs or ongoing maintenance.

Scalability: SaaS applications can be easily scaled up or down to meet changing organizational needs, ensuring optimal performance and resource utilization.

Accessibility: SaaS applications are accessible from anywhere with an internet connection, providing organizations with flexibility and mobility.

Security: SaaS providers implement robust security measures to protect sensitive organizational data and ensure the integrity of the applications.

Examples of SaaS applications

Salesforce: This is a classic example of a SaaS application. It provides a core CRM (customer relationship management) platform delivered entirely over the web, with subscriptions managed by Salesforce.

SurveyMonkey: SurveyMonkey is an online survey development cloud-based company. It provides a tool that allows users to create and distribute surveys, collect responses, and analyse data, all through a web interface. There is no need to install any software—it is a perfect example of a pure SaaS application.

Canva: Canva is a graphic design platform that allows users to create social media graphics, presentations, posters, documents, and other visual content. It is accessible entirely through a web browser, offering design tools and templates without the need for any desktop software. It epitomizes the pure SaaS model.

Hybrid SaaS examples

Gmail/Google Apps: While core components of Google Apps (Gmail, Docs and Sheets) are SaaS, Google also offers downloadable components.

Zoom, Microsoft Teams and Skype: These all have web-based platforms, which could be considered SaaS. However, they also offer installable desktop applications. It is more accurate to say they offer a SaaS component alongside traditional software installation.

Microsoft Office 365: This blends SaaS and traditional software models. Core components such as Word Online or Excel Online are pure SaaS. However, Office 365 subscriptions include downloadable desktop versions of the full software suite.

Moodle: An open-source learning management system. Most organizations self-host their solution, installing and managing it on their own web servers or through a hosting provider. This traditional model does not fit the strict definition of SaaS, where the software is hosted by the service provider and accessed over the internet without the need for local installation. However, there is a SaaS option called MoodleCloud. This offers Moodle's capabilities as a hosted online service, allowing users to use Moodle's features without needing to install, host or maintain the software themselves.

Platform as a service (PaaS)

PaaS provides a cloud-based environment for developers to build, deploy, and manage their applications. It offers a pre-built infrastructure including operating systems, databases, middleware, networking, and developer tools. This frees developers from managing the underlying infrastructure and allows them to focus on coding.

Key characteristics of PaaS

DevOps-friendly: PaaS supports rapid development and deployment cycles. DevOps is a set of practices which combines software engineering (Dev) and information technology (Ops).

Infrastructure abstraction: PaaS simplifies application management by handling OS updates, database administration, and networking configuration.

Reduced operational costs: PaaS eliminates the need to purchase and maintain hardware and software.

Examples of PaaS platforms

- Google App Engine
- Heroku
- Microsoft Azure App Service
- Amazon Elastic Beanstalk
- Red Hat OpenShift

Imagine a student named Alex, who is passionate about creating a personalized fitness application called FitTrack. Alex has great ideas for features such as workout tracking, nutritional advice, and social challenges, but has limited experience and resources for setting up and managing servers, databases, or dealing with complex infrastructure issues.

Alex decides to use a PaaS solution. With PaaS, Alex does not need to worry about installing operating systems, configuring databases, or managing network settings. Instead, Alex can focus entirely on writing the code for FitTrack using preferred programming languages and tools.



▲ **Figure 34** An athlete using a fitness app

The PaaS environment provides all the backend services Alex needs. It automatically handles scaling the application to accommodate more users as FitTrack grows in popularity. It also takes care of security updates and maintenance tasks without Alex having to dive into the details.

By using PaaS, Alex can bring FitTrack from idea to live application much faster and with fewer hurdles, enabling a focus on innovation and user experience rather than infrastructure management. This example demonstrates how PaaS empowers developers with limited infrastructure knowledge or resources to build and scale applications effectively.

Infrastructure as a service (IaaS)

IaaS provides virtualized computing resources, such as servers, storage and networking, over the internet. It gives businesses the flexibility to provision and manage their own IT infrastructure without the need to invest in physical hardware and software.

Virtualization refers to the process of creating a virtual version of something, such as operating systems, servers, storage devices or network resources, rather than a physical version. This technology involves using software to simulate the functionality of hardware to create a virtual computing system, enabling multiple virtual machines (VMs) to run on a single physical machine's hardware resources. Each VM operates independently and can run its own operating systems and applications as if it were a separate physical device. Virtualization allows for more efficient utilization of hardware, greater flexibility, scalability, and isolation between virtual machines for improved security and ease of management.

Key characteristics of IaaS

Customization: IaaS allows businesses to tailor their IT infrastructure to their specific needs and requirements.

Scalability: Resources can be easily provisioned and scaled up or down as needed, ensuring optimal performance and cost efficiency.

Control: Businesses have full control over the configuration and management of their IaaS infrastructure.

Examples of IaaS providers

- Amazon Web Services
- Microsoft Azure
- Linode
- Google Cloud Platform
- IBM Cloud
- Oracle Cloud Infrastructure

This problem in practice

SaaS provides a fully managed solution where the service provider manages all aspects of the software, including the underlying infrastructure, application code, and updates. Organizations simply access the software through a web browser or mobile app. It is ideal for services such as customer relationship management (CRM) systems, email platforms, project management tools, and collaborative workspaces.

With SaaS, organizations have no need to install, configure or maintain the software, reducing need for technical support. It eliminates the need for upfront software purchases or ongoing maintenance costs, making it a cost-effective option for organizations of all sizes. SaaS providers can quickly scale up or down the number of users and resources to meet changing demand, ensuring optimal performance and cost efficiency.

PaaS provides a pre-built development environment where developers can build, deploy and manage their applications without worrying about the underlying infrastructure. It is suited for developing web applications, mobile apps, APIs, and microservices.

PaaS provides a pre-integrated set of tools, frameworks and middleware that are managed by the service provider, simplifying the development process. Developers do not need to invest in their own infrastructure, saving them time and money.

IaaS provides virtualized computing resources, such as servers, storage and networking, that businesses can provision and manage on-demand. It is suited for high-performance computing, big data analytics, seasonal or bursting workloads, and web hosting (allowing organizations to host websites and applications entirely on IaaS infrastructure).

With IaaS, organizations control the configuration and management of their IaaS infrastructure, optimizing performance and tailoring the environment to their specific needs.

They can scale resources up or down as needed, ensuring optimal performance and cost efficiency, and tailor their IT infrastructure to their unique requirements, including specific hardware specifications and software configurations.

TOK

When you think of your own computer, you may think your data, your applications and your memory are all stored on the machine on your desk. However, as the cloud has grown and evolved, this is not always true. Organizations asked a simple question: why do they need to pay for computing resources that they are not using? With subscription options, as with utilities such as water or electricity, they can pay only for the computing resources they use.

How does the paradigm of cloud computing challenge traditional notions of resource allocation and cost efficiency in organizations? What implications does this have for our understanding of technology infrastructure and its scalability?

Choosing the right cloud service model

The choice between SaaS, PaaS and IaaS depends on the specific needs and capabilities of the organization. Here is a summary of the key considerations.

Table 7 SaaS, PaaS and IaaS comparison

Feature	SaaS	PaaS	IaaS
Control	Lowest	Medium	Highest
Flexibility	Lowest	Medium	Highest
Resource management	Service provider manages everything	Service provider manages most resources	Customer manages most resources
Cost	Most cost-effective	Medium cost	Most expensive
Suitability	Basic applications, general user access	Custom applications, rapid development	Custom applications, specialized hardware requirements

In general, SaaS is a good choice for organizations that need simple, easy-to-use applications and are willing to pay for the convenience. PaaS is suitable for organizations that want more control over their applications but still value rapid development and reduced maintenance costs. IaaS is the best choice for organizations that have complex IT requirements, need full control over their infrastructure, and are willing to manage their own resources.

Cloud computing allows organizations to provide the optimal balance of control, flexibility and cost-effectiveness.

Practice questions

1. Describe two applications of GPUs other than graphics rendering. [3 marks]
2. Describe how GPUs contribute to enhancing video game experiences. [3 marks]

AHL

3. Discuss how the design philosophy of CPUs and GPUs reflect their roles in computing systems. [5 marks]
4. Describe the significance of energy efficiency in the design of CPUs and GPUs. [3 marks]
5. Evaluate the effectiveness of the collaborative work between CPUs and GPUs in a computing system. [6 marks]
6. Describe the roles and characteristics of registers and cache memory within a computer system. [4 marks]
7. a. Outline the role of ROM in a computer system. [2 marks]
   b. Compare ROM with volatile memory types, referring to data preservation and access during power cycles. [3 marks]
8. Describe the primary functions of the data bus during the execute phase. [3 marks]
9. Describe how the control bus facilitates the interaction between the CPU and memory during a read operation from memory. [3 marks]
10. Evaluate the impact of bus architecture (address, data and control) on the performance of CPU operations. [4 marks]

AHL

11. Describe the basic principle of pipelining as used in multi-core architectures. [4 marks]
12. Describe how pipelining can improve the throughput of a multi-core processor. [3 marks]
13. Describe the potential challenges of implementing pipelining in multi-core architectures. [3 marks]
14. Evaluate the effectiveness of pipelining in a scenario where a multi-core processor is used for a highly parallel task. [6 marks]
15. Explain how data is stored and accessed on an optical disc. [6 marks]
16. Describe the advantages of using NAS in a multi-device environment. [3 marks]
17. Describe the differences between internal HDDs and eMMCs in terms of performance and use. [3 marks]
18. Evaluate the use of external SSDs for data backup compared to using flash drives. [5 marks]
19. Evaluate the impact of the physical limitations of NAND flash memory on the performance and longevity of storage devices like SSDs and eMMCs. [5 marks]

A1.2 Data representation and computer logic

Syllabus understandings

A1.2.1 Describe the principal methods of representing data

A1.2.2 Explain how binary is used to store data

A1.2.3 Describe the purpose and use of logic gates

A1.2.4 Construct and analyse truth tables

A1.2.5 Construct logic diagrams



▲ **Figure 35** How are images, sound and video represented in binary?

In this subtopic, you will learn about principal methods of representing data. Representing data in a computing system involves encoding information in a structured format, such as binary, that can be processed and stored by digital devices.

This subtopic will also cover binary, hexadecimal, and character encodings such as ASCII and UTF-8, each tailored to specific applications, from data storage in binary to text and numerical representations. You will explore how binary serves as the cornerstone for data storage in computers. At its core, binary operates on a base-2 numeral system, representing the simplest form of data encoding. This system represents every data type, from basic numerals to rich multimedia, as sequences of 0s and 1s. These binary states often correlate with physical conditions such as on or off (electrical charge presence), north or south (magnetic orientation for storage), or low or high (voltage levels).

Next, you will learn about constructing and analysing truth tables. Truth tables are used to describe the functionality of logic gates and circuits, showing every possible input combination and the resulting output. They are important for understanding and predicting the behaviour of digital systems.

Finally, you will learn about constructing logic diagrams. Logic diagrams visually represent the relationships of logic gates within a circuit. They are essential for designing, analysing and understanding how digital systems operate, facilitating the conceptualization and implementation of digital logic solutions.

A1.2.1 Describe the principal methods of representing data

The representation of integers in binary

Binary is a base-2 number system, using only two digits, 0 and 1, to represent all integers. Binary numbers may be shown by a subscript 2, for example, 1011101_2. The subscript helps to clarify which number system is being used. 11_{10}, 11_2, and 11_{16} are all different numbers. The first number is base-10, the second is base-2 and the third is base-16.

In binary, each digit in a binary number is called a bit (short for binary digit). Each bit position has a corresponding power of 2, starting from 2^0 at the rightmost bit and increasing to the left. The first bit represents 2^0, the next bit represents 2^1, then 2^2, 2^3, 2^4, 2^5, 2^6, 2^7, 2^8, and so on. Bits are almost always organized into groups of 8 known as a byte. A group of 4 bits is known as a nibble.

Table 8 Key terms for data representation

Term	Definition
Decimal/Denary	A base-10 numbering system that uses ten digits, 0 through 9. It is the standard system for denoting integer and non-integer numbers.
Binary	A base-2 numbering system that uses two digits, 0 and 1. Each digit is called a bit.
Hexadecimal	A base-16 numbering system that uses sixteen digits, numbers 0 to 9 and letters A to F representing numbers 10 to 16. Each digit (number or letter) is called a nibble because it represents four bits.
Bit	The smallest unit of data in computing, representing a single binary digit (0 or 1).
Byte	A unit of digital information that consists of 8 bits. It is the standard chunk size for most computer architectures and represents a single character of data.
Nibble	A unit of digital information that consists of 4 bits, or half of a byte. It can be represented as a single hexadecimal digit.
Place value	The value of a digit in a number, determined by its position within the number. For example, in the decimal number 345, the place value of 4 is 40 because it is in the tens place.
+	The addition operator
–	The subtraction operator
/	The division operator
*	The multiplication operator
%	The modulo operator (modulo refers to an operation that finds the remainder after division of one number by another)
**	The exponent operator

It can be helpful to read binary numbers from right to left, and to set out the number in a place value table.

Binary is the most compact representation for computers, as it directly aligns with their hardware, which uses transistors operating in two states.

Subtopic A1.1 discusses how data is represented in binary.

Convert from binary to decimal

1. Identify the place values. Each bit in a binary number represents a power of 2, starting from 2^0 at the rightmost bit and increasing to the left. This table is useful when working with binary.

Bit								
Place value	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
Decimal value	128	64	32	16	8	4	2	1

TOK

Everything in a computing system is eventually reduced to binary. Binary was initially used for early computing systems because it was easy to implement, is reliable and expresses logic well.

How does the use of the binary system in computing shape our understanding of numerical representation and data processing? What implications does this have for the broader field of digital technology and its applications?

2. Multiply each bit by its place value. For each 1 in the binary number, multiply it by the corresponding power of 2.
3. Add the products: Add up the results of all the multiplications to get the decimal value. For example, converting 1011 in binary to decimal looks like the following.

Bit					**1**	**0**	**1**	**1**
Place value	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
Decimal value					8	0	2	1

$1 * 2^3 + 0 * 2^2 + 1 * 2^1 + 1 * 2^0 = 8 + 0 + 2 + 1 = 11$

Note that, in computing, the asterisk (*) is used instead of the multiplication symbol (×).

Worked example 8

1. State the decimal number represented by each of these binary numbers.
   a. 1011 b. 1001
2. Write 11001_2 as a decimal number.

Solution

1. a. To change a number from binary to decimal, work out the decimal value of each digit of the binary number.

   Draw a place value table. Write the bits first, then the equivalent binary place values.

Bits	1	0	1	1
Place value	2^3	2^2	2^1	2^0

   Finally, add the working to convert each bit to a decimal number.

Working	$1 * 2^3 = 8$	$0 * 2^2 = 0$	$1 * 2^1 = 2$	$1 * 2^0 = 1$
Decimal value	8	0	2	1

   Find the sum of these individual decimal values: $8 + 0 + 2 + 1 = 11$.

   So, 1011 in binary is the same as 11 in decimal.

   b. Draw a place value table. Write in the bits and fill in the rest of the table.

Bit	1	0	0	1
Place value	2^3	2^2	2^1	2^0
Working	$1 * 2^3 = 8$	$0 * 2^2 = 0$	$0 * 2^1 = 0$	$1 * 2^0 = 1$
Decimal value	8	0	0	1

   Find the sum of the decimal values: $8 + 0 + 0 + 1 = 9$.

   So, binary 1001 is decimal 9.

2. Draw a place value table. Fill in the working and equivalent decimal values.

Bit	1	1	0	0	1
Place value	2^4	2^3	2^2	2^1	2^0
Working	$1 * 2^4 = 16$	$1 * 2^3 = 8$	$0 * 2^2 = 0$	$0 * 2^1 = 0$	$1 * 2^0 = 1$
Decimal value	16	8	0	0	1

   Find the sum of the decimal values: $16 + 8 + 0 + 0 + 1 = 25$.

   So, 11001_2 is decimal 25.

Worked example 9

State the decimal number represented by the binary number 10011010.

Solution

Write the number in a place value table. Complete the table.

Bit	1	0	0	1	1	0	1	0
Place value	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
Working	$1 * 2^7$	$0 * 2^6$	$0 * 2^5$	$1 * 2^4$	$1 * 2^3$	$0 * 2^2$	$1 * 2^1$	$0 * 2^0$
Decimal value	128	0	0	16	8	0	2	0

Find the sum of the decimal values: 0 + 2 + 0 + 8 + 16 + 0 + 0 + 128 = 154.

So binary 10011010 is decimal 154.

Activity

1. Write these decimal numbers as binary numbers.
   a. 12
   b. 28
   c. 71
2. Find all the unique 4-bit (4-digit) binary numbers.
   How many are there? What are their decimal equivalents?

Convert from decimal to binary

1. Divide the decimal number by 2. Repeat as many times as necessary to get to the end of the number. Keep track of the remainders (0 or 1). Note that, in computing, the solidus symbol (/) is used instead of the division symbol (÷).
2. Write the remainders in reverse order: these form the digits of the binary number.

Binary is usually represented in 8 bits, known as a byte. When a base-10 (decimal) number is stored in an 8-bit register, it must first be converted to base-2 (binary). When converting decimal to binary for this purpose, ensure you use all 8 places in the register. Remember to work from right to left, from smallest value to largest. Fill any places that do not have a digit with a zero. For example, 4_{10} (decimal 4) is 100 in binary. However, 4_{10} in an 8-bit register would look like this: 00000100.

Worked example 10

1. Convert each decimal number to binary.
   a. 13 b. 45
2. State the binary number that represents decimal 97.
3. How do these decimal numbers appear in an 8-bit register?
   a. 207 b. 34

Solution

1. a. Divide the decimal number by 2, keeping track of the remainders.

13 / 2 = 6 R (remainder) **1** — Write the remainder. The binary number starts with **1**

Divide your answer to the first division (6) by 2.

6 / 2 = 3 R **0** — Write the second remainder **in front of** the first.

The binary number is now **0 1**

Divide your answer to the second division (3) by 2.

3 / 2 = 1 R **1** — Write the third remainder in front of the second.

The binary number is now **1 0 1**

Divide your answer to the third division (1) by 2.

1 / 2 = 0 R **1** — Write the fourth remainder in front of the third.

The binary number is now: **1 1 0 1**

When the answer to the division is 0, the remainder is the final digit in your binary number.

So, decimal 13 is 1101 in binary.

b. Divide 45 by two until you get an answer of zero. Write down the remainder at each step.

45 / 2 = 22 R **1**	The first (smallest) binary digit is **1**.	**1**
22 / 2 = 11 R **0**	The second binary digit is **0**. Write this in front of 1 from the previous step.	**0** 1
11 / 2 = 5 R **1**	The third binary digit is **1**.	**1** 0 1
5/ 2 = 2 R **1**	The fourth binary digit is **1**.	**1** 1 0 1
2 / 2 = 1 R **0**	The fifth digit is **0**.	**0** 1 1 0 1
1 / 2 = 0 R **1**	The sixth (highest) digit is **1**.	**1** 0 1 1 0 1

The answer to the division is zero, so this is the final digit.

So, decimal 45 is 101101 in binary.

2. Divide 97 by two until you get an answer of zero. Write down the remainder at each step.

97 / 2 = 48 R 1	The first (smallest) binary digit is 1.	**1**
48 / 2 = 24 R 0	The second binary digit is 0.	**01**
24 / 2 = 12 R 0	The third binary digit is 0.	**001**
12 / 2 = 6 R 0	The fourth binary digit is 0.	**0001**
6 / 2 = 3 R 0	The fifth binary digit is 0.	**00001**
3 / 2 = 1 R 1	The sixth binary digit is 1.	**100001**
1 / 2 = 0 R 1	The seventh (and last) binary digit is 1.	**1100001**

Binary number 1100001 is decimal 97.

3. a. Convert the number to binary in the same way, by repeatedly dividing by 2.

207 / 2 = 103 R 1	Write the binary number, working right to left.	**1**
103 / 2 = 51 R 1	Write each remainder in front of the previous one.	**11**
51 / 2 = 25 R 1		**111**
25 / 2 = 12 R 1		**1111**
12 / 2 = 6 R 0		**01111**
6 / 2 = 3 R 0		**001111**
3 / 2 = 1 R 1		**1001111**
1 / 2 = 0 R 1	The 8-bit number is complete.	**11001111**

So, 207_{10} appears as 11001111 in an 8-bit register.

b. Convert the number to binary in the same way, by repeatedly dividing by 2.

34 / 2 = 17 R 0	Write the remainders, from right to left.	**0**
17 / 2 = 8 R 1		**10**
8 / 2 = 4 R 0		**010**
4 / 2 = 2 R 0		**0010**
2 / 0 = 1 R 0		**00010**
1 / 2 = 0 R 1		**100010**

The binary conversion is complete, but there are fewer than 8 bits. Fill in the empty spaces at the front (the large end) of the number with zeros.

So, 34_{10} appears as 00100010 in an 8-bit register.

Activity

1. State the decimal of:
   a. 11000111
   b. 11111111
   c. 10101111
2. State the 8-bit binary of:
   a. 79
   b. 203
   c. 241

The representation of integers in hexadecimal

Hexadecimal is a base-16 number system. Hexadecimal uses 16 digits—0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F—to represent integers. A subscript 16 may be used to show numbers are written in hexadecimal, for example, $C47B9_{16}$.

Hexadecimal is more human-readable than binary, especially for longer numbers, as it uses fewer digits to represent the same value. Both binary and hexadecimal are efficient for computer processing and storage. Each hexadecimal digit, called a nibble, directly corresponds to 4 bits in binary, making conversion between them straightforward.

As with binary, using a table can be a helpful for remembering which place holds which value.

Place value	16^7	16^6	16^5	16^4	16^3	16^2	16^1	16^0

It can be helpful to read hexadecimal numbers from right to left.

Convert from hexadecimal to decimal

1. Identify the place values. Each digit (bit) in a hexadecimal number represents a power of 16, starting from 16^0 at the rightmost digit and increasing to the left. This table is useful when working with hexadecimal.

Nibble									
Place value	16^8	16^7	16^6	16^5	16^4	16^3	16^2	16^1	16^0

2. Convert letters to decimal equivalents. If the hexadecimal number contains letters (A–F), convert them to their corresponding decimal values (A = 10, B = 11, C = 12, D = 13, E = 14, F = 15).
3. Multiply each digit by its place value. For each digit in the hexadecimal number, multiply it by the corresponding power of 16.
4. Add the products. Add up the products of all the multiplications to get the decimal value.

Table 9 Hexadecimal numbers and their equivalents in binary and decimal

Hexadecimal	Binary	Decimal
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
A	1010	10
B	1011	11
C	1100	12
D	1101	13
E	1110	14
F	1111	15

Worked example 11

1. Write each hexadecimal number as a decimal number.

   a. 2A b. 1B4 c. 111

2. Change FFF_{16} to a decimal number.

Solution

1. a. As you did when converting binary numbers, draw a place value table with four columns. Put the hexadecimal bits (digits) in the first row, working right to left. If there are fewer than 4 bits, fill in the spaces with zeros. Add the equivalent hexadecimal place values underneath.

Nibble	0	0	2	A
Place value	16^3	16^2	16^1	16^0

Add the working to convert each hexadecimal bit to a decimal number.

Working	$0 * 16^3 =$ $0 * 4096 = 0$	$0 * 16^2 =$ $0 * 256 = 0$	$2 * 16^1 =$ $2 * 16 = 16$	$A * 16^0 =$ $10 * 1 = 10$
Decimal value	0	0	32	10

Now, find the sum of the individual decimal values: 0 + 0 + 32 + 10 = 42.

So, hexadecimal 2A is decimal 42.

b. Draw and complete a place value table.

Nibble	0	1	B	4
Place value	16^3	16^2	16^1	16^0
Working	$0 * 16^3 =$ $0 * 4096 = 0$	$1 * 16^2 =$ $1 * 256 = 1$	$A * 16^1 =$ $11 * 16 = 160$	$4 * 16^0 =$ $4 * 1 = 4$
Decimal value	0	256	176	4

Find the sum of the decimal values: 0 + 256 + 176 + 4 = 436.

So, hexadecimal 1A4 is decimal 436.

c. Draw and complete a place value table.

Nibble	0	1	1	1
Place value	16^3	16^2	16^1	16^0
Working	$0 * 16^3 =$ $0 * 4096 = 0$	$1 * 16^2 =$ $1 * 256 = 256$	$1 * 16^1 =$ $1 * 16 = 16$	$1 * 16^0 =$ $1 * 1 = 1$
Decimal value	0	256	16	1

Add the decimal values: 0 + 256 + 16 + 1 = 273.

So, hexadecimal 111 is decimal 273.

2. The subscript 16 shows that this is a hexadecimal number. Draw and complete a place value table.

Nibble	0	F	F	F
Place value	16^3	16^2	16^1	16^0
Working	$0 * 16^3 =$ $0 * 4096 = 0$	$F * 16^2 =$ $15 * 256 = 3840$	$F * 16^1 =$ $15 * 16 = 240$	$F * 16^0 =$ $15 * 1 = 15$
Decimal value	0	3840	240	15

Add the decimal values: 0 + 3840 + 240 + 15 = 4095.

So, FFF_{16} is 4095_{10}.

Activity

1. State the base-10 of:
   a. C7
   b. FF
   c. AF
2. State the hexadecimal of:
   a. 420
   b. 900
   c. 256

Convert from decimal (integers) to hexadecimal

1. Divide the decimal number by 16. Repeat as many times as necessary to get to the end of the number. Keep track of the remainders (0 to 15).
2. Convert remainders to hexadecimal digits. Remainders from 0 to 9 stay the same. Convert remainders greater than 9 to their hexadecimal equivalents (A, B, C, D, E, F).
3. Write the remainders, in their hexadecimal form, in reverse order. This forms the hexadecimal number.

Worked example 12

1. Change each decimal to a hexadecimal number.
   a. 26 b. 398
2. Write 2040_{10} as a hexadecimal number.

Solution

1. a. Divide 26 by 16 until you get an answer of 0. Write down the remainders.

26 / 16 = 1 R 10	Convert the remainder to hexadecimal: 10 becomes **A**.
	The hexadecimal number starts with **A**
1 / 16 = 0 R 1	Look at the remainder: decimal 1 is also 1 in hexadecimal.
	Write the second remainder in front of the first.
	The hexadecimal number is now **1 A**

So, decimal 26 is written as 1A in hexadecimal.

b. Divide 398 by 16 until you get an answer of 0. Write down the remainders.

398 / 16 = 24 R 14	Convert the remainder to hexadecimal: 14 becomes **E**.
	The hexadecimal number starts with **E**
24 / 16 = 1 R 8	8 is the same in decimal and hexadecimal.
	Write the second remainder in front of the first.
	The hexadecimal number is now **8 E**
1 / 16 = 0 R 1	1 is the same in decimal and hexadecimal.
	Write the third remainder in front of the second.
	The hexadecimal number is now. **1 8 E**

Decimal 398 is written as 18E in hexadecimal.

2. Divide 2040 by 16 until you get an answer of 0. Write down the remainders.

2040 / 16 = 127 R 8	Write 8 and divide 127 by 16. **8**
127 / 16 = 7 R 15	Write 15 in hexadecimal, in front of 8. **F 8**
7 / 16 = 0 R 7	Write 7 in front of F. **7 F 8**

So, 2040_{10} is the same as $7F8_{16}$.

Convert integers from binary to hexadecimal

1. Group binary digits into nibbles. Divide the binary number into groups of 4 bits, starting from the rightmost bit. Each group of 4 bits represents a single hexadecimal digit.
2. Convert each nibble to its hexadecimal equivalent. Use Table 9 to match each 4-bit binary group to its corresponding hexadecimal digit (0–9, A–F).
3. Combine hexadecimal digits: Read the hexadecimal digits in order from left to right to get the final hexadecimal representation.

Worked example 13

1. Convert each binary number to hexadecimal.
   a. 11010110 b. 11101011
2. What is the hexadecimal equivalent of 1001110_2?

Solution

1. a. Break the binary number into nibbles (groups of 4 digits), working from right to left.
Convert each nibble into its hexadecimal equivalent. Use a conversion table to help you.

1	1	0	1	0	1	1	0
D				6			

So, binary 11010110 is D6 in hexadecimal.

b. Break the number into nibbles, working from right to left.

Write the hexadecimal equivalent for each nibble.

1	1	1	0	1	0	0	1
E				9			

So, binary 11101001 is E9 in hexadecimal.

2. Break the number into nibbles, working from right to left. There are only seven digits in this number, so add an extra 0 in front to complete the second nibble. Write the hexadecimal equivalent for each nibble.

0	1	0	0	1	0	1	1
4				B			

So, 1001110_2 is the same as $4B_{16}$.

Convert integers from hexadecimal to binary

1. Convert each hexadecimal nibble to its 4-bit binary equivalent. Use a conversion table to find the 4-bit binary representation for each hexadecimal digit in the number.
2. Combine binary representations. Join the binary representations of all digits in order to get the final binary string.

Worked example 14

1. Convert each hexadecimal number to binary.
   a. $D56_{16}$ b. $1A4_{16}$
2. State the binary equivalent of $CF0_{16}$.

Solution

1. a. Convert each hexadecimal nibble into its binary equivalent.
Use a conversion table if you wish.

Hexadecimal	D				5				6			
Binary bit	1	1	0	1	0	1	0	1	0	1	1	0

So, the binary representation of $D56_{16}$ is 110101010110.

b. Convert each hexadecimal nibble into 4-bit binary. If you don't use a table, keep each nibble separate.

Hexadecimal: 1 A 4

4-bit binary: 0001 1010 0100

Write the binary representation without spaces.

So, the binary representation of $1A4_{16}$ is 000110100100.

2. Convert each hexadecimal nibble into 4-bit binary.

Hexadecimal: C F 0

4-bit binary: 1100 1111 0000

Binary representation: 110011110000

So, the binary representation of $CF0_{16}$ is 110011110000.

Activity

1. State the hexadecimal of:
   a. 11110010 b. 00010001 c. 11011110.
2. State the 8-bit binary of:
   a. $E5_{16}$ b. $4C_{16}$ c. 99_{16}

A1.2.2 Explain how binary is used to store data

How data—integers, strings, characters, images, audio and video—are stored in binary form

Integers

An integer is a type of numerical data that represents a whole number without any fractional or decimal component.

Integers are represented in computers as fixed-width binary numbers, allowing for the representation of both positive and negative whole numbers. A fixed-width binary representation refers to the practice of allocating a constant number of bits to represent each data item within a computer system. This approach ensures every piece of data, regardless of its actual value, occupies the same amount of space in memory.

The primary encoding mechanism for integers include the following.

- The **unsigned integer** representation, which utilizes the entirety of the bit space to denote the magnitude of a number, with the maximum value determined by $2^n - 1$ where n is the number of bits. Unsigned integers only represent non-negative values.
- **Two's complement representation** for signed integers: This method allows for the representation of negative numbers by designating the most significant bit as the sign bit (0 for positive, 1 for negative) and inversely flipping the bits of the absolute value before adding one, to represent negative values efficiently.

A **4-bit unsigned integer** uses all four bits to represent positive numbers and zero. Here, the maximum value is 1111 in binary, which is 15 in decimal. For example, the binary number 1010 as an unsigned 4-bit integer represents the decimal value 10.

A **4-bit two's complement integer** uses the most significant bit (leftmost bit) as the sign bit. If the sign bit is 0, the number is positive or zero. If the sign bit is 1, the number is negative. They can represent values from –8 to 7.

For example, the binary number 1000 as a two's complement 4-bit integer represents the decimal value –8. The first 1 indicates that this is a negative number, and 000 is the binary form of 8 in a two's complement system, which is calculated by inverting all bits of the absolute minimum value (0111 for 7) and adding one.

Strings and characters

Characters and strings are encoded using character encoding standards that map characters to binary values.

ASCII (American Standard Code for Information Interchange) employs a 7-bit binary code to represent 128 unique characters, enabling the encoding of English letters, digits, and punctuation.

Unicode provides a comprehensive system to represent characters from all writing systems through variable-width encodings (such as UTF-8, UTF-16).

TOK

The decomposition of music, images and video almost always involves a reduction of the original work. Viewing original video or an image can be so computationally expensive that it is impractical.

To what extent does the reduction of diverse data types—such as integers, strings, characters, images, audio and video—into binary form inhibit our broader understanding of data and its inherent complexities in digital technology?

Key term

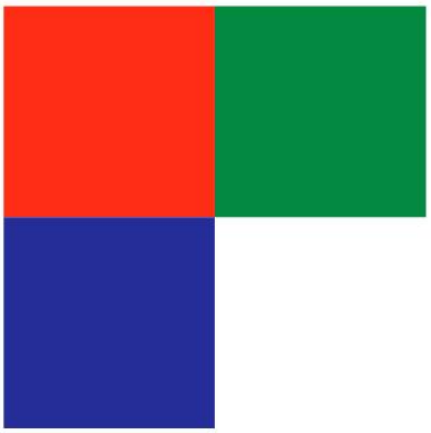
Pixels The tiny dots that make up images. Each pixel carries colour information. The resolution of an image—the detail an image holds—is directly related to the number of pixels it contains.

UTF-8 uses one to four bytes for each character, ensuring global text representation and compatibility.

For example, the character 'A' is represented as 01000001 in ASCII and the same in UTF-8, whereas a character like 'あ' (a Japanese hiragana character) cannot be represented in ASCII and requires more than a single byte in UTF-8, so is represented as 11100011 10000001 10000010.

Images

One of the most common methods of representing colour in digital images is through the RGB colour model, where colours are represented as a combination of red, green, and blue light. Each colour channel is typically represented by one byte (8 bits), allowing for 256 intensity levels (0–255). Thus, a single **pixel's** colour in a true-colour image can be represented by a 24-bit number, made up of 8 bits for each colour channel.



▲ **Figure 36** A 2 × 2 pixel image

In binary form, each pixel's colour is encoded as a sequence of bits. For a 24-bit image, each pixel would be represented by a binary sequence divided into three parts, each corresponding to one of the RGB components. For example, a bright red pixel might be represented as 11111111 00000000 00000000, where the red channel is at its maximum intensity, and the green and blue channels are off.

For a simple example, consider storing a very small image of 2 × 2 pixels where each pixel is encoded in 24-bit RGB, as shown in Figure 36:

Top-left pixel is red: 11111111 00000000 00000000

Top-right pixel is green: 00000000 11111111 00000000

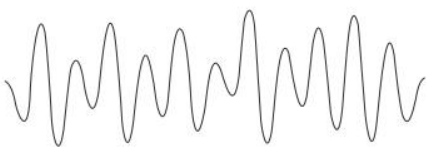
Bottom-left pixel is blue: 00000000 00000000 11111111

Bottom-right pixel is white (all colours at full intensity): 11111111 11111111 11111111

These binary sequences directly represent the colour of each pixel. In a BMP (bitmap) file format, they are stored directly as such, possibly preceded by a file header specifying the image's dimensions and colour depth. In contrast, formats like JPEG first process and compress this information before storing it in binary form.

Key term

Hertz (Hz) The unit of frequency, equivalent to one event per second. A sampling rate of 48 kHz means 48,000 samples per second.



▲ **Figure 37** An analogue sound wave. To sample this, you measure the amplitude 44 thousand times a second (if you are using 44.1 kHz sampling)

Audio

To store audio in binary form, analogue sound waves must be converted into a digital representation that can be processed, stored, and played back by electronic devices. This digital representation involves several key steps and concepts, including sampling, quantization, and possibly compression, depending on the chosen audio format and quality.

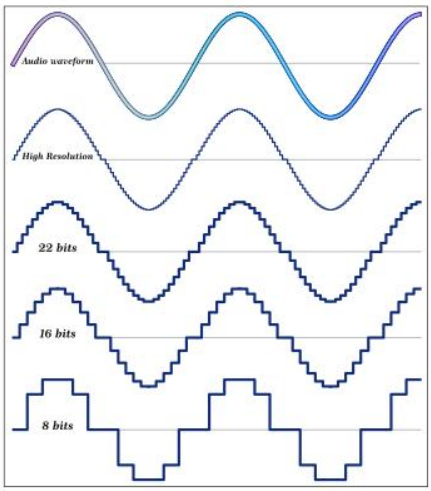
Sampling is the process of measuring the amplitude (volume) of a sound wave at regular intervals to create a series of discrete data points. The rate at which these measurements are taken is known as the sampling rate, measured in **hertz (Hz)**, which indicates the number of samples taken per second. Common sampling rates include 44.1 kHz (used in CDs), 48 kHz (common in professional audio), and 96 kHz (high-resolution audio). A higher sampling rate captures more detail of the sound wave but requires more data.

Quantization is the process of mapping the amplitude of each sampled point to a nearest value within a finite set of possible values. This step effectively converts the continuous amplitude of the sound wave into a digital format. The bit depth

determines the number of possible values for the amplitude of each sample, directly influencing the precision of the sound's digital representation. Common bit depths include 16-bit (65,536 possible values) and 24-bit (16,777,216 values), with professional audio using higher values. Higher bit depth allows for a more precise representation of the sound wave's amplitude, resulting in higher fidelity audio.

Each quantized sample is then encoded as a binary number, with the bit depth determining the length of this number. For example, in a 16-bit audio file, each sample is represented by a 16-bit binary number. In stereo audio, two separate channels (left and right) are recorded and stored, effectively doubling the amount of data compared to mono (single-channel) audio. Multi-channel audio, such as 5.1 surround sound, involves more channels and more data.

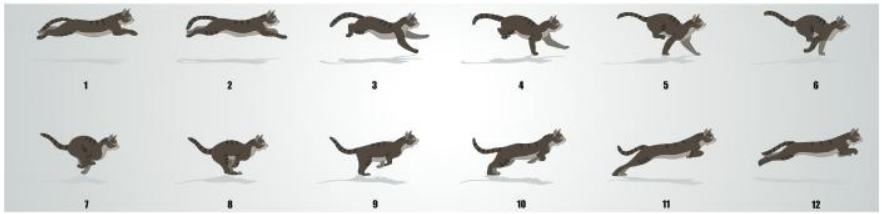
For example, consider a simple sine wave captured at a sampling rate of 44.1 kHz and a bit depth of 16 bits. At each sample point, the amplitude of the wave is measured and mapped to the nearest value within the range provided by the 16-bit depth. If the wave's amplitude at the first sample point corresponds to half of the maximum possible amplitude for positive values, it would be quantized and represented as the binary number 0111111111111111 (in a signed binary format, where the first bit indicates the sign).



▲ **Figure 38** A sampled sound wave at different bits

Video

Video is a sequence of still images (frames) presented at a rate that gives the impression of continuous motion. Each frame operates as a bitmap image, where the pixels are encoded in binary. For instance, within a 24-bit colour depth framework, every pixel is encoded using three bytes (24 bits), which delineate the intensity levels of the red, green and blue components.



▲ **Figure 39** 12 frames of a cat jumping

In addition to visual data, video files comprise an audio track, which undergoes a process of sampling, quantization and binary encoding, as with standalone audio files. The binary representations of audio and video are combined (multiplexed) to ensure that the auditory and visual elements of the video are synchronized during playback. This multiplexing process interleaves the audio and video data in a manner that maintains their temporal alignment, enabling simultaneous and coherent audio-visual playback.

The encoding of video into a binary format involves representing each frame's visual information as a sequence of bits. This binary representation facilitates the storage, processing and transmission of video data by digital systems. For the actual display of the video, various video coding formats, such as H.264 (AVC) or H.265 (HEVC), are utilized. These formats provide the necessary instructions for decoding the binary data back into a sequence of images that, when played at the appropriate speed, recreate the original video content.

Consider a 10-second video clip recorded at 30 frames per second (fps) and a resolution of 1920 by 1080 pixels (Full HD), with 24-bit colour depth. Each frame consists of 1920 × 1080 pixels = 2,073,600 pixels.

Each pixel is represented by 24 bits (or 3 bytes), so one frame is:
2,073,600 pixels × 3 bytes = 6,220,800 bytes (about 6.2 MB).

For a 10-second clip at 30 fps, the total size is:
6.2 MB/frame × 30 frames/second × 10 seconds = 1,860 MB (about 1.86 GB).

The fundamentals of binary encoding and the impact on data storage and retrieval

Binary encoding



▲ **Figure 40** Binary

The primary purpose of a binary encoding scheme is to convert data from its original form (which could be text, numbers, audio, video, and so on) into a binary form that a computer can process, store, or transmit.

At its core, binary encoding involves representing data using bits. A single bit can represent two states, often conceptualized as off/on, false/true, low/high, or 0/1. Eight bits form a byte, which is the basic unit of data storage. Larger data units—such as kilobytes (KB), megabytes (MB), gigabytes (GB), and so on—are multiples of bytes and are used to quantify digital data storage.

Table 10 Data units and their equivalents in bytes and bits

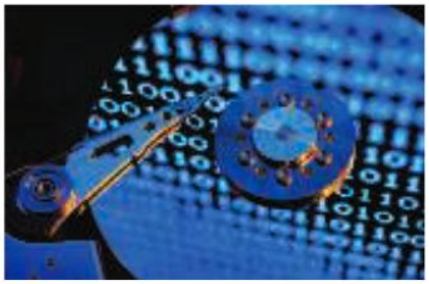
Unit	Equivalent in bytes	Equivalent in bits
1 bit	$\frac{1}{8}$ (0.125)	1
1 nibble	$\frac{1}{2}$ (0.5)	4
1 byte	1	8
1 KB (kilobyte)	1,024	8,192
1 MB (megabyte)	$1{,}024^2$ or 1,048,576	$1{,}024^2 * 8$ or 8,388,608
1 GB (gigabyte)	$1{,}024^3$ or 1,073,741,824	$1{,}024^3 * 8$ or 8,589,934,592
1 TB (terabyte)	$1{,}024^4$ or 1,099,511,627,776	$1{,}024^4 * 8$ or 8,796,093,022,208
1 PB (petabyte)	$1{,}024^5$ or 1,125,899,906,842,624	$1{,}024^5 * 8$ or 9,007,199,254,740,992

(After petabytes, there are exabytes, zettabytes, yottabytes, brontobytes and geopbytes).

Different types of data require different binary encoding schemes. For instance, text is commonly encoded using ASCII or Unicode, where each character is assigned a unique binary value. For images, encoding schemes like JPEG or PNG translate pixel colour and intensity into binary. Audio and video data are encoded into binary using formats that consider temporal changes and compression needs.

Data storage

Binary encoding allows for the efficient storage of data, with specific encoding schemes optimized for types of data to minimize space without sacrificing quality (for example, compression algorithms). The binary system scales well with technological advancements in storage media, from magnetic tapes to solid-state drives. Despite exponential growth in storage capacity, the basic binary nature of these storage technologies has remained constant. Finally, standardized binary encoding formats ensure that data can be stored, retrieved, and understood across different systems and platforms, facilitating interoperability and data exchange.



▲ **Figure 41** Hard drive disk

Data retrieval

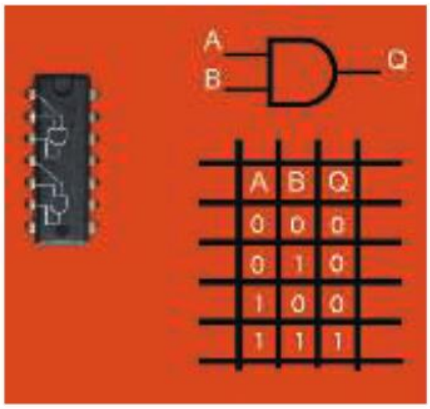
Binary encoding, coupled with the binary architecture of computer processors, enables rapid data retrieval and processing. Computers are inherently designed to work with binary data, which allows them to quickly perform operations on encoded data.

The efficiency of searching and analysing stored data is significantly influenced by how it is encoded. Indexing techniques and algorithms are optimized for binary data, enabling quick searches through large data sets and complex analyses, such as pattern recognition in machine learning models.

Finally, binary encoding schemes often incorporate mechanisms for error detection and correction, which are important for reliable data retrieval. For example, parity bits help identify errors that might occur during data storage or transmission.

A1.2.3 Describe the purpose and use of logic gates

Logic gates are electronic circuits that operate on one or more binary inputs to produce a binary output, based on a specific logical function. Each type of logic gate implements a Boolean operation such as AND, OR, NOT, NAND, NOR, XOR and XNOR, corresponding to the fundamental operations in Boolean algebra. The output of each gate reflects the result of its logical operation. Logic gates can be constructed using transistors—semiconductor devices that act as electronic switches. The arrangement of these transistors determines the type of logic gate and its corresponding logical operation.



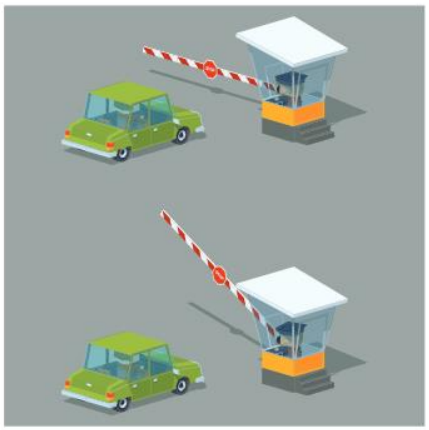
▲ **Figure 42** An AND logic gate

The word "gate" in "logic gate" metaphorically signifies its function as a control mechanism for the flow of information in digital circuits, similar to how a physical gate controls the passage of entities through an opening. In the context of digital electronics, a logic gate performs logical operations on one or more binary inputs to produce a single binary output.

When working with logic gates, the number one (1) represents True, or on, while the number zero (0) represents False, or off.

The purpose and use of logic gates

Logic gates perform simple logical operations, such as AND, OR and NOT. The outputs of these gates depend on the input values and the type of gate, enabling basic data processing functions.



▲ **Figure 43** A gate controlling access

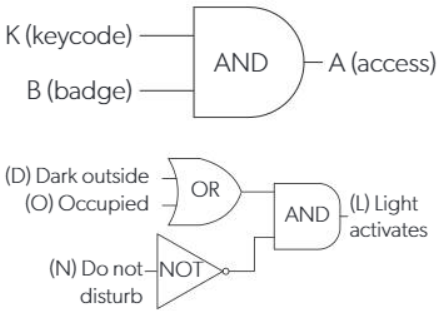
For example, a fire alarm system may have multiple sensors for smoke, heat and carbon monoxide monitoring connected to an OR gate. If any sensor triggers (input is 1), the output of the OR gate activates the alarm system. If there is smoke OR heat OR carbon monoxide, then an alarm is sounded.

By combining different types of logic gates, more complex circuits can be created. Logic gates are physical manifestations of Boolean algebra. Boolean algebra is a branch of mathematics that deals with variables and operators, using truth values (true or false) to perform logical operations.

Logic gates are used:

- in digital displays to control the representation of numbers, characters and symbols
- in control systems, such as elevators or automated doors, helping to process input signals (like buttons being pressed) to produce appropriate responses (like opening a door)
- in timing circuits, such as clocks and timers, determining the timing of various operations
- in communication systems, helping with the processing and transmission of digital signals
- in safety systems, such as alarm systems, where they process inputs from various sensors, to trigger alarms or other safety responses.





▲ **Figure 44** Logic gates for a simple security system as shown in the photograph (top), and for a custom control system (bottom)

The function and applications of logic gates in computer systems

Logic gates enable conditional logic in computer operations, allowing for decision-making processes based on binary conditions. This is foundational for implementing if-else and switch-case statements in programming languages.

Consider a simple security system that requires two conditions to be met before access is granted: a correct keycode and a security badge scanned.

This can be modelled using AND logic, where both conditions must be true (1) for access to be granted.

Consider a custom light control system where a light should be turned on either if it is dark outside or if the room is currently occupied. There is an additional condition that if a "do not disturb" mode is active, the light should remain off regardless of the other conditions.

This scenario can be modelled using OR logic for the first set of conditions and AND logic along with NOT to consider the "do not disturb" condition.

The role of logic gates in binary computing

Data processing and decision making

Logic gates are used to implement conditional statements in hardware, allowing a computer to execute specific instructions based on whether certain conditions are true or false. For example, an AND gate outputs a true signal only when all of its inputs are true, which is like executing an "if all conditions are true" statement in programming. By combining different logic gates, complex decision-making processes can be built, enabling the execution of complex algorithms and control flows within digital circuits.

Arithmetic operations

In the ALU of a computer's CPU, logic gates are arranged into circuits capable of performing arithmetic operations on binary numbers. For instance, adder circuits, constructed from a combination of AND, OR and XOR gates, enable the execution of addition, which can be extended to subtraction, multiplication and division through algorithmic approaches.

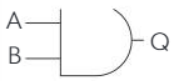
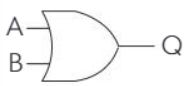
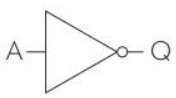
Memory storage

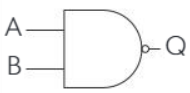
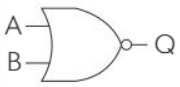
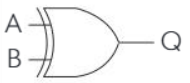
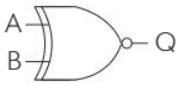
Logic gates are integral to the design and function of memory devices. Flip-flops, which are circuits made from logic gates, can store a bit of data by maintaining a stable state until explicitly changed. This principle is scaled up to create registers, RAM, and other storage devices, allowing binary data to be stored and retrieved as needed.

Boolean operators: AND, OR, NOT, NAND, NOR, XOR, XNOR

Truth tables provide a clear and systematic way to visualize the behaviour of a logical expression for all possible inputs. The table will have a column for each input and each output, wherever they occur in the system. The ultimate output is labelled Q.

Table 11 Boolean operators

Gate	Symbol	Function	What the gate asks	Truth table
AND		Outputs 1 (true) only if all its inputs are 1.	Are both inputs on?	Input: A, B; Output: Q 0 0 0 0 1 0 1 0 0 1 1 1
OR		Outputs 1 if at least one of its inputs is 1.	Is either input on?	Input: A, B; Output: Q 0 0 0 0 1 1 1 0 1 1 1 1
NOT		Outputs the inverse of its input: outputs 1 if the input is 0, and vice versa. This gate is also known as an inverter.	Is A off?	Input: A; Output: Q 0 1 1 0

Gate	Symbol	Function	What the gate asks	Truth table
NAND		Outputs 1 unless all its inputs are 1 (the inverse of the AND gate).	Is either input off?	Input: A, B; Output: Q 0 0 1 0 1 1 1 0 1 1 1 0
NOR		Outputs 1 only if all its inputs are 0 (the inverse of the OR gate).	Are both inputs off?	Input: A, B; Output: Q 0 0 1 0 1 0 1 0 0 1 1 0
XOR		Outputs 1 if the inputs are different: if one is 1 and the other is 0.	Are the inputs different?	Input: A, B; Output: Q 0 0 0 0 1 1 1 0 1 1 1 0
XNOR		Outputs 1 if the inputs are the same: both 0 or both 1 (the inverse of the XOR gate).	Are the inputs the same?	Input: A, B; Output: Q 0 0 1 0 1 0 1 0 0 1 1 1

TOK

Truth tables are powerful tools for summarizing logical expressions. Some systems are so complex they cannot be summarized in a truth table.

How does the use of truth tables to determine outputs from inputs based on a problem description affect our understanding of logical reasoning and problem-solving? What limitations might this impose on our comprehension of complex systems?

A1.2.4 Construct and analyse truth tables

Truth tables to determine outputs from inputs for a problem description

A truth table is a diagram of the outputs from all possible combinations of input. Truth tables are used to predict the output of simple logic circuits.

In the previous section, each gate had a logic diagram. Sometimes, you might encounter problems that only have a description of a problem. You must read the description carefully and decide which logic gates would be appropriate, then construct a suitable truth table. The underlying process is the same for constructing the truth table, but you must first understand the problem correctly.

Worked example 15

A security system activates an alarm if both a motion sensor and a door sensor are activated simultaneously. Draw a suitable truth table for this system.

Solution

First, consider the input(s).

There are two inputs: a motion sensor and a door sensor. Label them A (motion sensor) and B (door sensor).

Second, consider the output(s).

In this case, there is one output: an alarm is activated. Label this Q (activate the alarm).

The alarm only sounds if the door AND motion sensor are activated at the same time. So, the required logic gate is an AND gate.

Now construct the truth table.

There are two inputs, A and B, and one output, Q. The total number of possible outputs is 2^n. In this case $n = 2$, so $2^2 = 4$. The system uses an AND logic gate. This outputs 1 (True) only if all its inputs are 1. Draw and complete the logic table.

A	B	Q
0	0	0
0	1	0
1	0	0
1	1	1

Rewrite the output in the context of the question. Q is only 1 (the alarm is activated) if both A (the motion sensor) and B (the door alarm) are True. This makes sense and fits the problem description perfectly.

Worked example 16

In an automatic lighting system, a light turns on if at least one of two sensors detects darkness. The lighting system also has a manual override switch that, when activated, ensures the light stays off regardless of the input from the sensors.

Draw and complete a truth table for this system.

Solution

First, think about the input(s). There are two light sensors: label them A and B. There is also a manual override switch: label this C.

Next, think about the output(s). In this case, the light is the output. Label it Q.

To decide which logic gate(s) you need, read the question carefully: "A light turns on if at least one of two sensors detects darkness". So, the light turns on if sensor A OR sensor B OR both sensors detect darkness, so the first gate is an OR gate.

The override switch is an additional logic gate: when activated, this means the light always stays off, irrespective of the inputs. So, the second gate is a NOT gate.

The logical statement is (A OR B) AND (NOT C).

Construct the truth table. You have three inputs, and you know the total number of possible outputs is $2^n = 2^3$ (since $n = 3$) = 8. There are eight outputs, so give your table eight rows.

A	B	A OR B	NOT C	Q
0	0	0	0	0
0	0	0	1	0
0	1	1	0	1
0	1	1	1	0
1	0	1	0	1
1	0	1	1	0
1	1	1	0	1
1	1	1	1	0

Check if your answer makes sense. In row 7 of the table, notice that **both** sensors (A and B) have detected darkness. Notice also that C (the override switch) is not on. And Q, the light, is True, which means it is on.

In row 8 of the table, notice that both A and B have detected darkness, but because C (the override switch) is True, the light is not on (Q is False).

Worked example 17

An automatic watering system for plants only activates when it is daytime and either the soil moisture is low or the temperature is high.

Draw and complete a truth table for the system.

Solution

First, think about the inputs. There are three: daytime (A), soil moisture (B) and temperature (C).

Second, think about the output: the watering system activates (Q).

Next, think about the logic gates you need: A AND (B OR C).

Now construct the truth table. There are three inputs. The total number of possible outputs is $2^n = 2^3$ (since $n = 3$) = 8 outputs. Your truth table needs eight rows.

A	B	C	(B OR C)	Q
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	1	0
1	0	0	0	0
1	0	1	1	1
1	1	0	1	1
1	1	1	1	1

Finally, check your work. In row 1, even though it is daytime, neither the soil moisture nor the temperature are true. Therefore, the water system is not activated. However, in row 6, it is daytime and the temperature is high (1), so the water system activates.

Key term

Boolean expression An algebraic expression formed using Boolean variables and logical operators such as AND, OR , NOT, NOR, and so on.

Truth tables and their relationship to a Boolean expression with inputs and outputs

So far, you have built truth tables from logic diagrams and problem descriptions. You can use **Boolean expressions** to build truth tables without the need for diagrams or problem descriptions. Boolean expressions are clearly stated, with little to no need to interpret them. Here are three Boolean expressions:

1. A NOT (B OR C)
2. A AND (B XOR C)
3. (A NOR B) AND C

You will generally encounter truth table problems with only three inputs.

Worked example 18

Construct a truth table for this logic statement: (A AND B) OR NOT C.

Solution

Identify the inputs. In this case there are three: A, B and C.

Construct the truth table. You have three inputs, and you know the total number of possible outputs is 2^n. Since $n = 3$, the number of outputs is $2^3 = 8$. Add eight rows to your truth table.

A	B	C	A AND B	NOT C	Q
0	0	0	0	1	1
0	0	1	0	0	0
0	1	0	0	1	1
0	1	1	0	0	0
1	0	0	0	1	1
1	0	1	0	0	0
1	1	0	1	1	1
1	1	1	1	0	1

Worked example 19

Construct a truth table based on the following logic statement: (A XOR B) AND NOT C.

Solution

Identify the inputs; in this case there are three: A, B and C.

Construct the truth table. There are three inputs, and you know the total number of possible outputs is 2^n. Since $n = 3$, there are $2^3 = 8$ outputs.

A	B	C	A XOR B	NOT C	Q
0	0	0	0	1	0
0	0	1	0	0	0
0	1	0	1	1	1
0	1	1	1	0	0
1	0	0	1	1	1
1	0	1	1	0	0
1	1	0	0	1	0
1	1	1	0	0	0

Activity

Construct a truth table for each of these logic statements:

1. (A NOR B) OR C
2. (A AND B) NOR C
3. A XOR (B OR C)

Truth tables derived from logic diagrams to aid the simplification of logical expressions

Truth tables derived from logic diagrams help simplify logical expressions by providing a clear and complete overview of the output for all input combinations.

Given a slightly more complex logic circuit, how should you approach solving it? Remember, with a truth table you are concerned with predicting all possible input combinations and their corresponding outputs.

Creating a truth table from a logic diagram

1. List all input combinations. With two inputs (A and B) there are four combinations (00, 01, 10, 11), so add four rows to your truth table. This will help you to not forget a possible input.

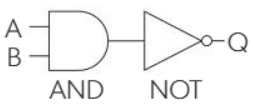
   For n given inputs, there are a total of 2^n possible input combinations. So, with two inputs there are $2^2 = 4$ input combinations. For three inputs, there are $2^3 = 8$ input combinations.

   Make sure that you arrange your inputs logically. Notice that the inputs count up in binary, starting at 0 in the first row. (The decimal equivalent is shown here for reference only—do not include this in your table.)

2. Apply the first operation (AND). Calculate the AND operation result for each input combination.

3. Apply the second operation (NOT) to the result of the AND operation. This gives the final output of the circuit. Do you recognize this simple logic circuit? Hint: it has an AND and a NOT gate.

   This is the same output as a NAND gate.



▲ **Figure 45** A more complex logic diagram

A	B
0	0
0	1
1	0
1	1

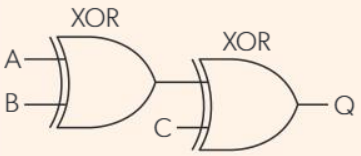
Decimal
0
1
2
3

A	B	AND output
0	0	0
0	1	0
1	0	0
1	1	1

A	B	AND output	NOT output
0	0	0	1
0	1	0	1
1	0	0	1
1	1	1	0

Worked example 20

Construct a truth table for this diagram.



Solution

Determine the total possible number of input permutations. With three inputs (A, B and C), there are eight possible combinations ($2^n = 2^3 = 8$). List all the input combinations in a logical order by counting up in binary: columns A, B and C will show numbers 0 to 7 in binary. (Not 1 to 8: you must start counting at 0.)

A	B	C
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1

Calculate the first XOR operation result for each input combination of A and B.

A	B	C	A XOR B
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

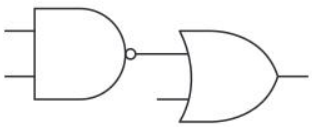
Calculate the second XOR operation by comparing A XOR B and C.

A	B	C	A XOR B	Q
0	0	0	0	0
0	0	1	0	1
0	1	0	1	1
0	1	1	1	0
1	0	0	1	1
1	0	1	1	0
1	1	0	0	0
1	1	1	0	1

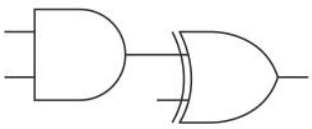
Activity

Construct truth tables for each of these diagrams

1



2



Simplifying output expressions with algebra and Karnaugh maps

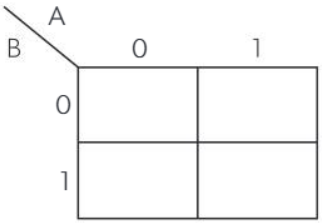
Karnaugh maps (K-maps) are a method for simplifying Boolean algebra expressions by minimising the number of terms needed to express a logic function. A K-map can also be thought of as a special type of truth table that makes finding patterns easier.

Constructing a K-map

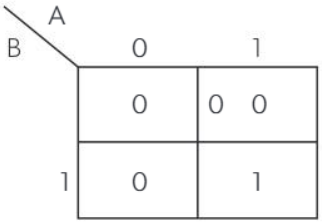
AND		
A	B	Q
0	0	0
0	1	0
1	0	0
1	1	1

1. First, construct a truth table. K-maps are generally used with no more than four inputs, and the IB typically uses three inputs for truth tables. This example uses the truth table shown here.
2. Consider the number of inputs. In this case, there are two inputs.
3. Now construct a K-map.

   Draw a grid: the number of columns = the number of possible inputs for A. The number of rows is the number of possible inputs for B. In this case, both A and B can be 0 or 1, so draw a 2 × 2 grid. From the top left corner of the grid, draw a diagonal line away from the grid to separate the two inputs. Above the line, add a label, A, and column headings showing all the possible inputs for A. Under the line, add a label, B, and row headings showing all the possible inputs for B. Put the smallest values nearest the diagonal line.



4. Populate the table. Looking at the truth table from step 1, ask yourself: "if A is 0 and B is 0, what is the output?" The answer is 0. This value goes in the top left cell of the K-map, where the A = 0 column and the B = 0 rows intersect. Repeat the question, using the correct A and B input values for each cell. The K-map is complete.



Worked example 21

Create a K-map for this truth table.

A	B	Q
0	0	0
0	1	1
1	0	1
1	1	0

Solution

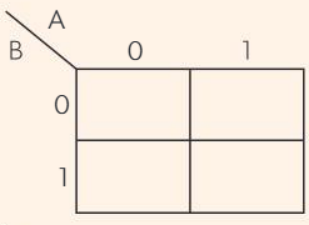
First, construct the truth table. In this case, the truth table has already been provided in the question.

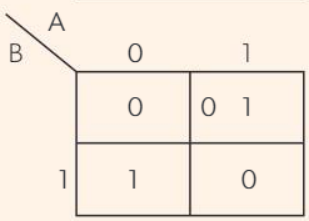
Next, consider the number of inputs. In this case, there are two inputs.

Now construct a K-map. Note that you have column headings with all the possible input combinations for A at the top, and row labels with all the possible input combinations for B on the left.

Populate the table. Looking at the truth table, you can ask yourself: "if A is 0 and B is 0, what is the output?" The answer is 0. If A is 1 and B is 0, the output is 1.

Your K-map is complete.





Worked example 22

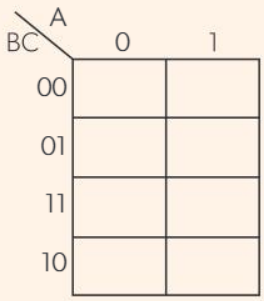
Create a K-map for this truth table.

A	B	C	Q
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

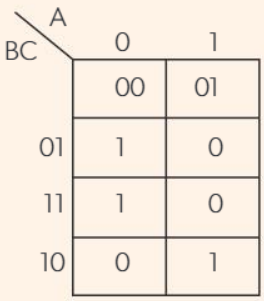
Solution

First, consider the number of inputs. In this case, there are three inputs: A, B and C.

Construct a K-map. At the top, write column headings with all the possible input combinations for A. On the left, write row labels with all the possible input combinations for B and C.



Populate the table. Looking at the truth table, ask: "if A is 0 and BC is 00, what is the output?" The answer is zero. If A is 1 and BC is 00, the output is 1. Repeat until the K-map is complete.



Simplifying K-maps

To simplify a K-map, such as the one in the solution to Worked example 22, you first group the 1s together. There are some important rules to remember as you create groups of 1s.

1. A group can only contain one, two, four or eight 1s.
2. You should make as large a group as possible—do not make a group of two if you can make a group of four.
3. Each group should be shaped as a horizontal or vertical rectangle or square.
4. The groups may overlap if necessary, and you can wrap around the edges of the K-map to form a group.
5. Every 1 on the map must be included in one group.

A Gray code, or reflected binary code, is a binary numbering system where two successive values differ by one bit. This is useful in K-maps to minimize errors in switching circuits by reducing the possibility of error during the change of state from one code to another. In the context of K-maps, the Gray code ensures that adjacent cells differ by only one variable, simplifying the process of grouping and identifying commonalities among terms, which helps in minimizing Boolean expressions.

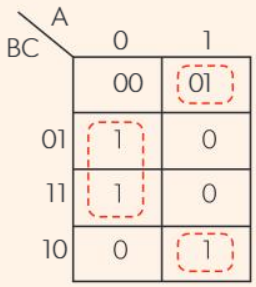
Note that row labels in K-maps are not in sequential binary order; they differ by only one bit (e.g., 00 and 01, 01 and 11).

Worked example 23

Simplify the K-map from Worked example 22.

Solution

First, group the 1s together.



C not A + A + B not C

Next, determine the variables for each group.

Each group corresponds to a term in the simplified Boolean expression.

Look at the rows and columns that your group spans. For each variable do the following.

If a variable **does not** change within a group (i.e., all 1s in the group are in rows or columns marked by the same variable state), that variable **is part** of the term. In the example, notice how C does not change in the column on the left. You can also notice that A does not change. Looking at the two groups in the right column, notice that A does not change and C does change.

If a variable **changes** state within a group (i.e., includes both its true and complemented forms) it **is not included** in the term for that group. In the example, you can observe that B does change in the group on the left.

Now, write each term.

Include a variable in the term if it **is the same** across the entire group (e.g., all rows or columns the group spans are labelled with the variable in the true or complement form).

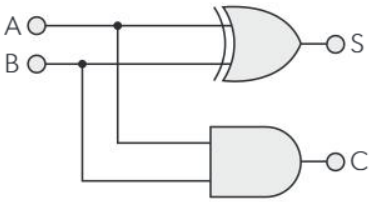
Omit the variable if the group spans areas where the variable changes from true to complement or vice versa.

Combine the terms. Once you have a term for each group, combine them with OR operations. This results in the simplified Boolean expression for the original function defined by the K-map.

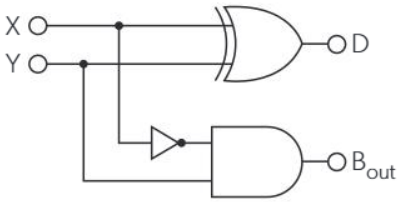
A1.2.5 Construct logic diagrams

Using logic diagrams to demonstrate how logic gates are connected and interact in a circuit

Logic diagrams are composed of connected logic gates, inputs, and outputs. Logic diagrams are normally read left to right. When reading and interpreting a logic diagram, remember that its main purpose is to demonstrate the flow of logic. These diagrams can be directly transposed to electronic circuits, serving as a blueprint to build a physical circuit.



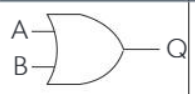
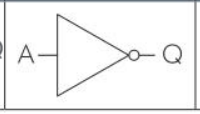
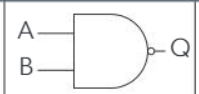
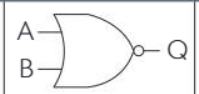
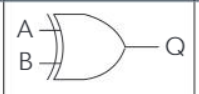
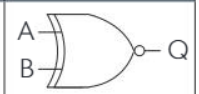
▲ **Figure 46** A half adder. The half adder adds two single binary digits, A and B. It has two outputs, sum (S) and carry (C)



▲ **Figure 47** A half subtractor. The inputs X and Y are the numbers to be subtracted (specifically, X – Y) while D is the difference and B is the borrow signal (if the circuit needs to borrow)

Use standard gate symbols for AND, OR, NOT, NAND, NOR, XOR and XNOR gates

Table 12 Logic gate symbols

AND	OR	NOT	NAND	NOR	XOR	XNOR
						

You should use standard gate symbols when constructing logic diagrams. It is helpful to label your logic gate to avoid any confusion. Make sure your inputs and outputs are clearly labelled.

Worked example 24

A simple security system only activates an alarm when both a motion sensor and a door sensor detect a breach simultaneously. Create a logic diagram to represent this situation.

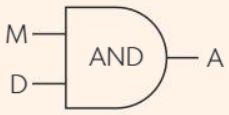
Solution

First, think about and label the inputs: M for motion sensor, D for door sensor.

Next, think about and label the outputs: A for alarm.

Consider what logic gate is needed: in this case, it is a simple AND gate.

Your diagram might look like this:



Worked example 25

An automatic watering system for plants that should only activate under two conditions: it is daytime and either the soil moisture is low or the temperature is high.

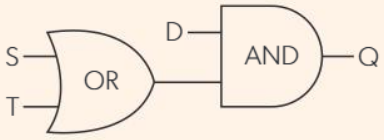
Create a logic diagram based on this problem.

Solution

First, think about and label the inputs. There are three in this problem: daytime (D), soil moisture (S) and temperature (T).

Next, think about and label the output, the watering system activates (Q).

Thinking about the logic gates, create A AND (B OR C).



Process inputs diagrammatically to produce outputs

When systems or circuits are represented in a diagram, inputs are represented by lines or arrows coming into the diagram from the left side. These may be labelled with letters (such as A, B or C) or descriptive names (such as motion sensor or proximity sensor).

In Worked example 25, the inputs are S for soil moisture, T for temperature, and D for daylight. The output, Q, tells us whether the watering system activates. Note that the output goes towards the right side of the diagram.

Remember to read logic diagrams from left to right unless clearly stated otherwise. You can assume variables on the left side are inputs.

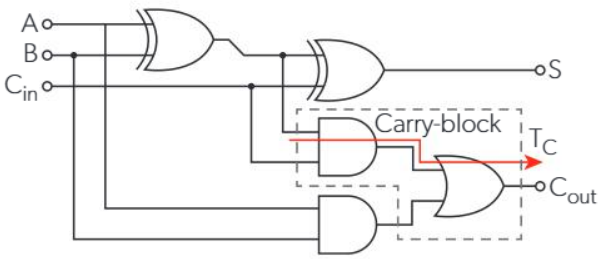
Activity

Look back at the logic diagrams you have drawn or been given in previous sections of this unit.

Identify all the inputs and the outputs. If you spot any mistakes, correct them!

Combine gates to perform more complex logical operations

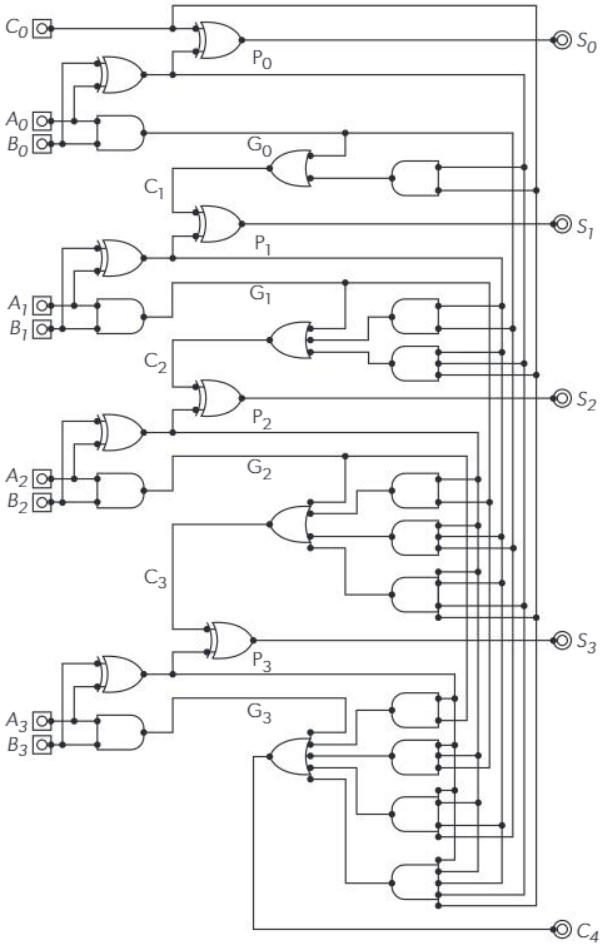
Combinations of gates can lead to more sophisticated logical operations.



▲ **Figure 48** A full adder is used to add two binary digits along with a carry from a previous addition

Use Boolean algebra rules to simplify complex logic diagrams and expressions

Boolean algebra simplifies complex logic diagrams and expressions, which allows expressions to be standardized using logical operators like AND, OR and NOT. This standardization helps in analysing and simplifying the expressions systematically.



▲ **Figure 49** A fast adder is a type of electronics adder. Note that you will not be asked to decode a diagram as complex as this one

Table 13 Mathematical notation

Operation	Description	Mathematical notation	Programming notation	Digital circuit symbol
AND	Logical conjunction	$A \wedge B$	A && B	AND gate
OR	Logical disjunction	$A \vee B$	A || B	OR gate
NOT	Logical negation	$\overline{B}$	!B	NOT gate
NAND	NOT AND	$\overline{A \wedge B}$	!(A && B)	NAND gate
NOR	NOT OR	$\overline{A \vee B}$	!(A || B)	NOR gate
XOR	Exclusive OR	$A \oplus B$	A ^ B	XOR gate
XNOR	Exclusive NOR (Equivalence)	$(A \oplus B)^-$	!(A ^ B)	XNOR gate

TOK

Logic diagrams serve as powerful tools in the visualization and design of electronic circuits and microprocessor layouts, bridging the gap between abstract logical processes and tangible physical systems. Their use shapes the way we approach the construction, interpretation and troubleshooting of complex hardware architectures.

To what extent does this influence our understanding of logical processes and the construction of physical systems? What are the advantages and disadvantages of using logic diagrams in this context?

Boolean algebra rules

A Boolean algebra law is a principle or rule that defines the behaviour and relationships of Boolean variables and operations. These laws are used to manipulate and simplify Boolean expressions, ensuring consistency and predictability in logical reasoning and digital circuit design. There are several mathematical laws within Boolean algebra. These are three common laws.

1. **The commutative law** applies to AND and OR operations. It means that the order of variables does not affect the outcome. For example, for variables A and B, (A AND B) gives the same result as (B AND A). (A OR B) = (B OR A).
2. **The idempotent law** simplifies expressions with repeated variables. It states that when a variable is combined with itself, the result is the variable itself. For example, (B AND B) = B. (1 AND 1) = 1.
3. **The involution law** simplifies double negation. It states that the negation of the negation of a variable returns the original variable. In other words, if you negate a variable twice, you get the original variable back. For example, NOT (NOT A) = A. NOT (NOT 1) = 1.

Worked example 26

Use truth tables to demonstrate the equivalence or difference of these logical expressions.

A AND B B AND A

How does this demonstrate the commutative law?

Solution

Construct a truth table for the logical expression A AND B.

A	B	A AND B
0	0	0
0	1	0
1	0	0
1	1	1

Now construct a truth table for the logical expression B AND A.

A	B	B AND A
0	0	0
0	1	0
1	0	0
1	1	1

The truth tables show that the results of (A AND B) and (B AND A) are identical for all possible values of A and B.

The commutative law states that the order of the variables does not affect the outcome for AND and OR operations. The truth tables demonstrate the commutative law, showing that A AND B is equivalent to B AND A.

Worked example 27

This expression is used in a digital circuit: $(A \wedge A) \vee \overline{\overline{B}}$

Rewrite the logic expression as simply as possible.

Solution

$(A \wedge A) \vee \overline{\overline{B}}$

$\overline{\overline{B}}$ means NOT NOT B.

So, the expression can be written as:
(A AND A) OR NOT NOT B.

The idempotent law simplifies expressions with repeated values. Using the idempotent law, $A \wedge A$ simplifies to A.

The involution law simplifies double negation. Using the involution law, $\overline{\overline{B}}$ simplifies to B.

So, the simplified expression is $A \vee B$, which can be written as A OR B. This uses fewer logic gates and is easier to implement.

Worked example 28

Use the variables A = 1 and B = 0 to show that the commutative law applies to logical AND and OR operations.

Solution

The commutative law in Boolean algebra states that the order in which the operations are performed does not affect the outcome for the logical AND and logical OR operations.

Consider the commutative law for AND.

$A \wedge B = B \wedge A$

So, if A = 1 and B= 0, your equations would be:

1 AND 0 = 0

0 AND 1 = 0

Both expressions evaluate to False (0), showing that the order does not affect the result.

Now consider the commutative law for OR.

$A \vee B = B \vee A$

So, if A = 1 and B = 0, your equation would be:

1 OR 0 = 1

0 OR 1 = 1

Both expressions evaluate to True (1), demonstrating that the outcome remains consistent regardless of the order.

So, the commutative law applies to both AND and OR operations.

Practice questions

20. Define the term "binary number system". [2 marks]
21. Describe how integers are represented in the hexadecimal number system. [3 marks]
22. State the hexadecimal equivalent of decimal 93. Show your working. [2 marks]
23. Describe the process of converting a binary number to a hexadecimal number. [3 marks]
24. Explain the process of converting analogue audio signals into binary. [3 marks]
25. Describe how logic gates are used in timing circuits such as clocks and timers. [4 marks]
26. Describe the role of NAND and NOR gates in the implementation of safety systems. [3 marks]

A1.3 Operating systems and control systems

Syllabus understandings

A1.3.1 Describe the role of operating systems

A1.3.2 Describe the functions of an operating system

A1.3.3 Compare different approaches to scheduling

A1.3.4 Evaluate the use of polling and interrupt handling

AHL

A1.3.5 Explain the role of the operating system in managing multitasking and resource allocation

AHL

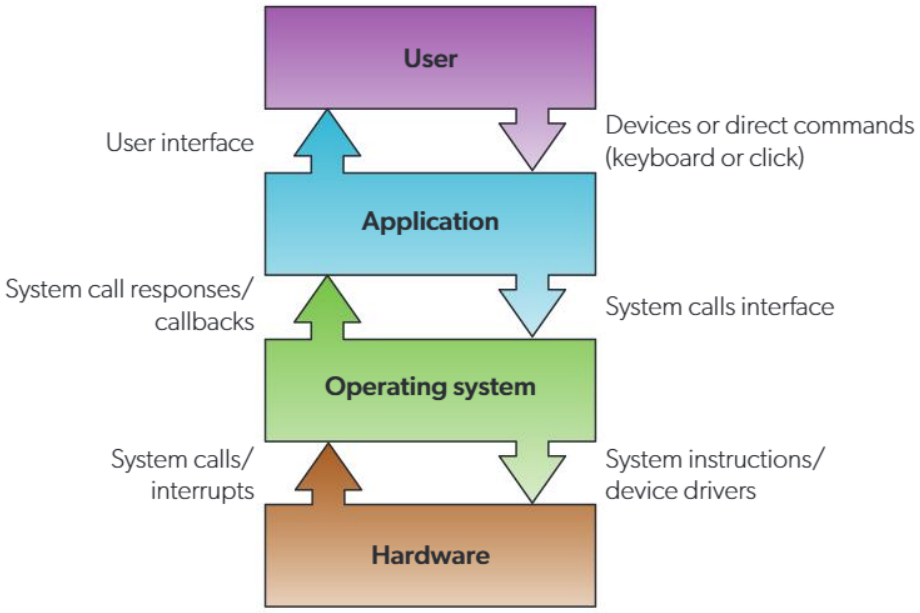
A1.3.6 Describe the use of the control system components

AHL

A1.3.7 Explain the use of control systems in a range of real-world applications

A1.3.1 Describe the role of operating systems

An operating system (OS) serves as an interface connecting the user and application to the hardware of a computer system. The role of an operating system is to manage system resources—such as the CPU, memory, disk, network and peripherals—and to provide services to the user and other applications, such as file management, process management, security, and user interface.



▲ **Figure 50** A simplified diagram showing how an operating system serves as an intermediary between applications and hardware

TOK

To what extent does the operating system's role in hiding the complexity of hardware management influence our understanding of computing systems? What are the implications of this for users and application developers?

Some examples of operating systems include Linux, Windows, Orbis (based on BSD, for the PlayStation), macOS, Android, BSD, Unix, and iOS. Each operating system has its own features, advantages, and disadvantages, depending on the design philosophy, intended use, and user preference. Some operating systems are highly specialized and only work on very specific types of hardware.

Remember: there is not a "best" operating system. There is only to what extent an operating system is the best fit for a given purpose.

Operating systems abstract hardware complexities to manage system resources

Operating systems manage system resources to ensure fairness, security and efficiency with competing processes, multiple users, and many different types of hardware.

Table 14 System resources that an operating system manages

System resource	How an OS manages the resource
CPU	Schedules and allocates CPU time among various processes and threads, implementing scheduling algorithms to manage process execution, prioritize tasks and handle interrupt requests.
Memory	Manages physical and virtual memory, including allocation and deallocation of memory to processes, swapping between RAM and disk (paging), and protection of memory spaces to prevent processes from interfering with each other.
Storage	Manages storage through file systems. It handles file creation, deletion, reading, writing, permissions and organization, as well as disk space allocation and file system integrity.
I/O (input/output)	Manages data flow to and from I/O devices such as keyboards, mice and network interfaces. Manages interrupts from I/O devices; for example, when mouse movement is detected.
Network stack	Manages the network stack to facilitate data transmission over network interfaces, handling protocols, network connections, and bandwidth allocation
User interface (UI)	Provides the frameworks and tools needed to build graphical and command-line interfaces, translating user actions into system calls. While not technically a resource, the UI is critical for interacting with a computer.
Security and access controls	Enforces security policies, managing user authentication, authorization, encryption, and audit logs. Among other areas of security, it controls access to files, applications, and system settings.
Process and task management	Manages the lifecycle of processes and threads, including creation, execution, suspension and termination, along with inter-process communication (IPC) and synchronization mechanisms.
Power management	Manages power resources to optimize battery life, including controlling power usage by hardware components and implementing power-saving modes.
External devices and peripherals	Manages external devices connected via USB, Bluetooth, or other interfaces, ensuring compatibility and proper operation through device drivers. This is different from I/O.

It is worth considering the **chaos** that would occur if individual applications or users were allowed to allocate memory, schedule the CPU, and write to storage without considering all the other running processes in a system. It would be chaotic, insecure and inefficient.

Consider an orchestra. What would happen if all the musicians decided to play any note they wanted, whenever they wanted, for as long as they wanted? In a computer, an operating system performs a similar role to that of a conductor in an orchestra. It manages all the resources while providing a secure and coherent method or interface to utilize those resources.



▲ **Figure 51** An orchestra needs a conductor

Abstraction is one of the fundamental concepts of computational thinking, which you will study in topic B1.



▲ **Figure 52** You do not want to worry about allocating threads when playing video games

Abstraction can be defined as hiding intricate details beneath a simpler, more manageable layer (or interface). An application does not need to directly manage memory—the operating system handles that, ensuring the application has the resources it needs (or fails gracefully if it does not).

When you are playing a video game, are you worried about allocating threads, managing memory, and ensuring that your peripherals are generating interrupts? Of course not! You just want to have fun playing a game.

Another example of abstraction is driving a car. You do not need a deep understanding of internal combustion engines to drive. The pedals, steering wheel and transmission control provide an interface between the driver and the underlying mechanical processes.

When you simplify any complex system into an easy-to-use interface, you are abstracting the complex system.

A1.3.2 Describe the functions of an operating system

The previous section outlined the resources managed by an operating system. This section looks specifically at what and how an operating system manages those resources.

What is a process?

A process is an instance of a program being executed.

A process is the execution of a program: When you double-click an application icon or run a command, the instructions within that program are loaded into memory, and the operating system creates a process to manage its execution. A program is a passive set of instructions. A process is the active, dynamic execution of those instructions. You may have multiple processes running from the same program.

Key term

Sandboxing Isolating a process or application, restricting its access to system resources beyond what it strictly needs. This can involve memory protection and also includes limiting file system access, controlling network communication, and so on.

A process is a unit of resource ownership: Each process is given its own sandboxed area of resources by the operating system. This includes the following.

- Memory: A private section of memory to store the program's code, data, and the stack (which keeps track of function calls and local variables).
- CPU time: The process receives slices of time from the CPU to run its instructions, with the operating system scheduling when each process gets to run.
- Open files/network connections: A process can open files, interact with the network, and use other system resources.

The OS keeps track of which resources belong to which process.

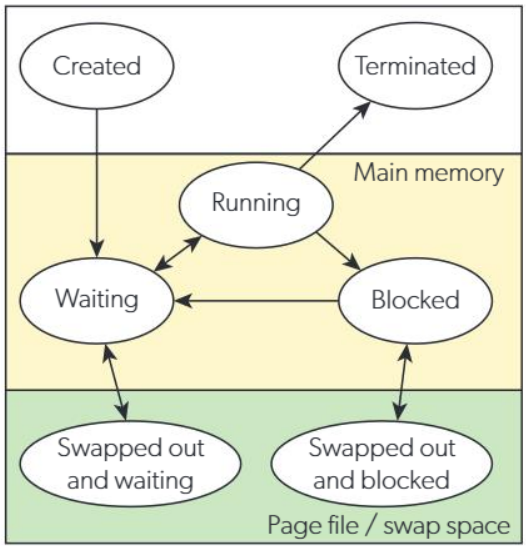
A process is an entity managed by the OS: The operating system maintains information about each process, including the following.

- Process state: Whether it is running, waiting for resources, blocked on input or output, or terminated.

- Process ID (PID): A unique identifier assigned to the process.
- Registers: The temporary storage locations within the CPU that hold the data the process is currently working on.
- Priority: How important the process is relative to others, influencing scheduling decisions.
- Resource usage: Statistics on memory, CPU time, files, and other resources used by the process.

A process can have different process states: The process states are as follows.

- New: The process is being created and loaded into memory.
- Ready: The process is ready to run and is waiting for its turn on the CPU.
- Running: The process is currently executing instructions on the CPU.
- Blocked (or Waiting): The process cannot continue until an external event occurs (for example, waiting for a file to be read from disk, waiting for network data, or waiting for user input).
- Terminated: The process has finished its execution and is being cleaned up by the operating system.



▲ **Figure 53** Processor states

By tracking the state of a process, the OS can do the following.

- Schedule processes: Prioritize ready processes for CPU time.
- Manage resources: Release resources held by blocked processes, potentially allocating them to other processes.
- Error handling: Identify and handle processes stuck in unusual states.

A process has threads: A thread (or thread of execution) is the smallest sequence of programmed instructions that can be independently managed by a scheduler.

Within a process, you can have multiple threads of execution that share the same memory space but can potentially execute in parallel on multi-core systems.

For example, a word processor could:

- have one thread to handle rendering the document on the screen
- have another thread to handle spell checking in the background
- have a separate thread to manage user input.

Operating systems maintain system integrity by implementing the following policies and practices

System integrity refers to the state of a computer system when the system performs its intended functions correctly and reliably, without unauthorized or unintended alterations. It implies that the system's hardware, software and data maintain their expected and trustworthy state. System integrity includes concepts such as correctness, completeness, data accuracy, protection from manipulation, and resilience. You could never rely or trust a system if it did not have integrity.

Background operations, or background processes, are computer processes that execute without direct user interaction or a visible user interface. Background operations perform tasks which support the applications you might be using or the overall functioning of the system.

Process isolation enforces boundaries between running processes. This means each process operates in its own dedicated virtual memory space, preventing accidental or malicious interference with other processes or the operating system itself.

Memory protection means that processes have separate memory spaces, preventing one process from interfering with another or with the OS.

User mode (or user space) and kernel mode separates operations which require direct hardware access (kernel mode) from those that do not (user mode). You do not want individual applications deciding where they should read or write into memory.

Resource management is primarily concerned with allocating resources fairly and efficiently among running processes, ensuring system responsiveness and preventing conflicts.

CPU scheduling assigns processes to ensure that critical system operations receive the CPU time they need without unnecessarily hindering user applications. You will learn more about scheduling in the next section. The basic idea is that, in a system with 400 or 500 different processes, each gets a fair share of resources including priority processes and interrupts.

Access control and permissions assign individual and group permissions to every single file and directory in the computer.

User and group permissions implement user and group accounts with specific permissions, controlling access to system resources and limiting what background operations can do based on their required access level. For example, a guest would not have read and write permissions in your private document directory.

File system security manages access rights to files and directories, ensuring that only authorized processes and users can read, modify, or execute specific files. In the Linux operating system every file and directory has permissions for read, write and execute permissions for owner, group, and other.

Most operating systems also support access control lists (ACLs), which can provide more flexible control to system resources.

Security features generally can be contextualized within the CIA triad, keeping resources confidential, with integrity, and available.

Encryption protects data integrity and confidentiality by encrypting sensitive information stored on the system. For example, certain versions of Windows offer BitLocker, which encrypts a disk, macOS offers FileVault, and Linux offers dm-crypt/LUKS as options for encryption.

Firewalls monitor network traffic at the **packet** level to detect, block or allow packets into the system.

Key term

Scheduling The process of arranging, controlling, and optimizing work and workloads to achieve specific goals.

Key term

Packet A formatted unit of data carried by a network. It is a small segment of a larger message that is divided and transmitted over a network, typically in Internet Protocol (IP)-based communications.

Antivirus software monitors files for malicious behaviour (changing an important system file, for example) and signatures (does anything in this file match a known virus pattern?).

Firewalls and antivirus software can benefit from integration with the operating system. This integration allows them to efficiently access system resources, monitor system activities at a low level, and provide better protection against threats. However, the ultimate effectiveness of these security measures depends on factors like the breadth of threat definitions, the frequency of updates, the sophistication of detection algorithms, and the overall security architecture, rather than solely on whether they are provided by the OS.

You will find more about the function of firewalls in topic A2 Networks.

Update management keeps an operating system secure and performant in an evolving threat landscape.

Patch management is the process of deploying patches to software systems to ensure they are up-to-date and secure. Patches are a set of code changes designed to address problems or update an existing computer program or its supporting data. Patches fall into different categories, including bug fixes, security fixes, feature enhancements, and compatibility updates. Patches often include mechanisms to verify the integrity and authenticity of updates before installation.

Rollback features allow a return to the last known-good state of the system. These features enable the system to revert to a previous state if an update or change causes issues, preserving system integrity. In Windows OS, this is known as "System Restore". On macOS, the "Time Machine" feature serves a similar purpose.

Data integrity checks can use mechanisms such as checksums, hashes, or error-correcting codes to detect and potentially correct data corruption caused by transmission errors or storage issues.

Checksums are digits representing the sum of the correct digits in a piece of stored or transmitted digital data, against which later comparisons can be made to detect errors in the data.

A hash is a fixed-size string or number generated from a data set using a hashing function that converts varied length inputs into a compressed numerical value, primarily used for data integrity checks and quick data retrieval.
Checksums and hashes verify the integrity of system files and data by comparing them against known good checksums or hashes, detecting corruption or unauthorized changes.

Hashing and hash functions are covered in topic B4 Abstract data types (HL only).

File system consistency checks help ensure the integrity and consistency of file systems, which can become corrupted due to improper shutdowns, hardware failures, or uncaught exceptions. Tools like fsck (file system check) are employed, typically during system boot or designated maintenance windows to verify the coherence and validity of the file system.

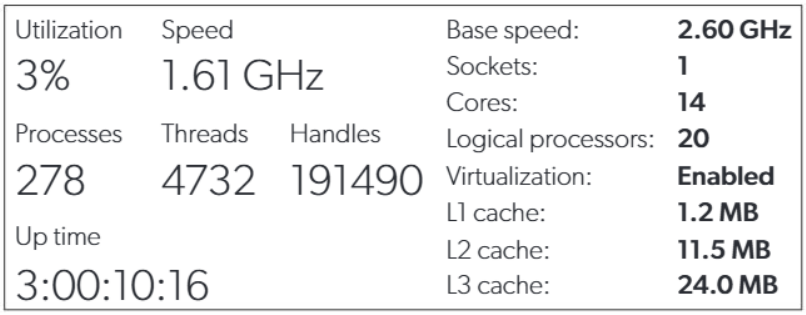
Monitoring and **logging** are both tools to understand current system performance and issues (monitoring) and past system performance and issues (logging).

Logging collects detailed activity of system operations, security events, and errors, providing a way to audit system behaviour, detect anomalies, and troubleshoot issues. Almost every action instantiated by a user or process is logged.

```
Apr 1 17:29:32 production-web sshd[2917441]: Failed password for
invalid user teste from Apr 1 17:29:33 production-web sshd[2917441]:
Received disconnect from 167.99.159.235 port
Apr 1 17:29:33 production-web sshd[2917441]: Disconnected from invalid
user teste 167.99.
Apr 1 17:29:38 production-web sshd[2924950]: pam_unix(sshd:auth):
authentication failure; user=root ruser= rhost=43.156.203.90
Apr 1 17:29:38 production-web sudo: pam_unix(sudo:session): session
closed for user root Apr 1 17:29:39 production-web sshd[2924950]:
Failed password for root from 43.156.203.90
Apr 1 17:29:40 production-web sshd[2923570]: pam_unix(sshd:auth):
authentication failure;ruser= rhost=180.101.88.237 user=root
Apr 1 17:29:40 production-web sshd[2924950]: Received disconnect from
43.156.203.90 port
Apr 1 17:29:40 production-web sshd[2924950]: Disconnected from
authenticating user root 4 th] Apr 1 17:29:42 production-web
sshd[2923570]: Failed password for root from 180.101.88.237
Apr 1 17:29:47 production-web sshd[2923570]: Failed password for root
from 180.101.88.237
Apr 1 17:29:50 production-web sshd[2923570]: Failed password for root
from 180.101.88.237
Apr 1 17:29:52 production-web sshd[2923570]: Received disconnect from
180.101.88.237 port Apr 1 17:29:52 production-web sshd[2923570]:
Disconnected from authenticating user root 1 uth]
Apr 1 17:29:52 production-web sshd[2923570]: PAM 2 more authentication
failures; logname= host=180.101.88.237 user=root
```

▲ **Figure 54** An example of a system log from the Linux OS. This specific log shows failed logins to a web server in a brute-force attack. Note the time stamp, server, process (SSHD), process ID and message: these are common in most logging systems. Learning to read, interpret, and act on log entries is an important skill in information technology and can be useful in computer science

Performance monitoring enables real-time tracking of various metrics such as CPU usage, memory consumption, network bandwidth, and application response times.



▲ **Figure 55** A snapshot of a task manager, where CPU usage is being tracked in real time. At 3% utilization, this CPU is not working very hard

TOK

Understanding the underlying details of an operating system helps you to gain a clearer picture of why the operating system is so vital to the overall health and usability of computing systems. But this is not essential unless you are writing instructions for that system.

To what extent does understanding the specific functions and resource management roles of an operating system shape our perception of its importance in maintaining system integrity and facilitating user interaction with computing systems?

Memory management

Memory management involves allocating and deallocating memory to processes which need it, and ensuring that each process has separate and protected memory address space. Memory management also implements virtual memory, which allows the processes to use more memory than the physical memory available, by swapping some pages of memory to the disk when needed.

Memory pages are fixed-size, contiguous blocks of virtual memory that form the basic unit of allocation and transfer between primary storage (RAM) and secondary storage (disk) in a virtual memory system. The operating system, in conjunction

with the memory management unit (MMU), maintains page tables to track the mapping of these pages between virtual and physical address spaces. If you discuss memory **without** using virtual memory, you use the term "page frames".

Virtual memory separates the linear memory addresses used by processes from the underlying physical memory layout. The OS, with hardware support from the MMU, maps virtual addresses to physical addresses dynamically. This allows for processes to have large, contiguous address spaces independent of physical RAM limitations, with the OS transparently paging data between RAM and disk as needed.

File systems

A file system is a method used by an operating system to control how data is stored and retrieved. It provides a block-level organization for files on physical storage devices such as hard disks, SSDs, or USB flash drives. The file system defines the rules for naming files, the structure of directories (or folders), and how files and directories are displayed to the user. It also manages the allocation of disk space to files and directories and keeps track of which areas of the disk are available for use. Examples of file systems include NTFS (used by Windows), ext4 (used by Linux), and APFS (used by macOS).

Device management

Device management enables interaction between the operating system and hardware via device drivers. These drivers **convert** commands from the operating system into specific signals (instructions) tailored for each hardware component, enabling data transfer (communication) across a variety of devices. Conversely, hardware devices can initiate communication with the operating system by sending signals or interrupts, which the operating system processes and responds to accordingly.

Device management is responsible for handling interrupts and errors emanating from hardware devices, offering a uniform interface for applications to access hardware resources. For example, user actions such as keyboard strokes or mouse movements generate interrupts, alerting the operating system to new user input. Similarly, a network interface card receiving a data packet triggers an interrupt, prompting the operating system to handle the incoming data.

Scheduling

Scheduling is primarily concerned with determining the order and timing of process execution.

Imagine a constantly changing pool of 500 different processes, all needing processor time. How should these processes be executed? First-come, first-served? Should the execution time depend on the importance of the process? The size of the process? Deciding how to allocate CPU time (instruction cycles) falls to scheduling. Scheduling considers priority, fairness, efficiency, and system load. There is more about scheduling in the next section.

Security

Operating systems provide firewalls, encryption, access control, user authentication, security patches, sandboxing, logging, and data integrity.

Accounting

Accounting is the tracking and recording of resource usage, such as CPU time, memory usage, disk space, and network bandwidth. In the early days of computing, accounting was used to charge users or departments using multi-user mainframes based on how much of a resource they used. Interestingly, accounting of resources has returned with cloud-based computing, where organizations pay for what they use.

The operating system gathers data on how long each process runs, the amount of memory it utilizes, its disk input/output operations, and other resource consumption metrics. Based on historical usage patterns, the OS can make more informed scheduling decisions to optimize resource distribution and ensure fair allocation.

Accounting is implemented in system logs. The operating system maintains detailed logs of activity. Each entry in a log includes timestamps, process IDs, user IDs, specific resources requested (CPU time, memory allocation, network bandwidth), and the duration for which these resources were granted. In some systems, the logs might also record details on denied resource requests, failed process executions, or errors encountered during resource allocation.

Windows users can view logs by looking for the "Event Viewer" application, macOS users can open the "Console" application, and Linux users can look through the /var/log directory.

Reading log files on your computer is an effective, non-destructive method to help you understand the different aspects of the system your operating system manages.



▲ **Figure 56** A confusing GUI—it is difficult to understand because it has been poorly designed

Graphical user interface

When most users think of an operating system, they think of the graphical user interface or GUI (pronounced "gooey"). Windows OS looks different from macOS, which looks different from some Linux OS GUIs. As a computer scientist, you know an OS serves as an intermediary between the user, application and the hardware of a computer system. The GUI is just one part (albeit an important part) for the users of the OS.

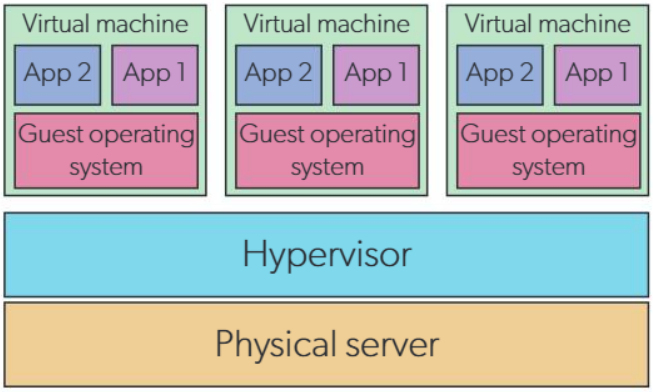
A GUI facilitates interacting with a computer system using visual elements, such as icons, menus, windows and buttons. The graphical user interface makes the computer system easier to use, and allows the user to perform tasks with a mouse, keyboard, touchscreen, or other input devices.

GUIs are constantly being updated and improved to make them easier and more intuitive to use. A well-established field within computing is **human–computer interaction**, which guides thinking about computing interfaces.

Virtualization

Virtualization involves creating virtual instances of physical hardware. Virtualization enables a single physical machine to host multiple virtual machines (VMs), with each VM operating as though it has its own dedicated hardware. Think of virtualization like dividing up a big house into separate apartments. Each apartment gets its own rooms and resources, even though they all share the same physical building.

The technical key to virtualization is the hypervisor, a specialized software layer which enables the interaction between virtual machines and the physical hardware. The hypervisor divides the underlying computer into smaller, virtualized chunks. Each VM is given a portion of these virtualized resources. The hypervisor ensures that VMs stay within their boundaries and don't interfere with each other. The hypervisor tricks each virtual machine into thinking it actually has direct access to the physical hardware, even though it is a virtualized environment.



▲ **Figure 57** A hypervisor model

There are two main benefits of virtualization. The first benefit is efficient use of computing resources—multiple VMs can share a single system's CPU, memory and storage, increasing hardware utilization and reducing infrastructure costs. The second is isolation, where VMs are isolated from each other. A failure or security compromise in one VM does not affect others.

Networking

Operating systems facilitate networking via a software stack designed to manage network communications. Networks operate by segmenting data into packets and transmitting those packets across networks. When an operating system receives packets, it puts them back together so applications can use them.

The network stack, or network protocol stack, comprises networking protocols and layers that collectively enable network communication. This stack is commonly associated with the TCP/IP model, detailed in A2. Key points to remember include:

- operating systems implement the network stack
- the process of ensuring reliable, robust, and secure digital communication is complex, requiring the cooperation of multiple software layers.

Operating systems support and manage network hardware such as network interface cards (NICs) via device drivers. These drivers abstract the hardware details, providing a standardized interface to the network stack for data transmission and reception, regardless of the hardware used.

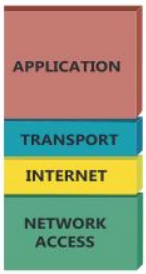
Network configurations on operating systems include setting IP addresses for network interfaces, either statically by the user or dynamically through dynamic host configuration protocol (DHCP). This step is vital for device identification on a network and their subsequent communication.

To safeguard network communications, operating systems deploy security measures such as firewalls to manage packet flow, encryption (for example, SSL/TLS for web traffic) for secure data transmission, and authentication protocols for identity verification.

Operating systems also come equipped with a variety of network utilities and command-line tools for troubleshooting, configuration and monitoring purposes. These tools include:

- **`ping:`** to test host reachability
- **`ifconfig/ipconfig:`** for network interface configuration
- **`netstat:`** to view network connections
- **`traceroute/tracert:`** to trace packet routes to a destination.

You will learn about networking in more detail in topic A2 Networks.



▲ **Figure 58** The TCP/IP model



▲ **Figure 59** Network configuration

TOK

Optimization and efficient use of resources in computer systems are central to decision-making when creating computing systems—they drive the development of systems.

To what extent do different scheduling approaches influence our understanding of optimizing system performance and managing process execution? Why might a computer scientist or software engineer need to know about scheduling, and at what depth would they need to know it?

A1.3.3 Compare different approaches to scheduling

Manage the execution of processes by allocating CPU time to optimize system performance

Computer operating systems manage the execution of processes and threads by scheduling CPU time, memory, and other computing resources.

The reason operating systems consider different scheduling algorithms is because they want to optimize the performance of the computer system. Process importance and requirements can change dynamically, and operating systems often adjust scheduling priorities in real-time.

Imagine hundreds of processes every second (some high priority, some low priority, some very complex, some very simple), all entering and leaving the processing queue at incredible speed. Ineffective scheduling can cause latency and poor performance. Operating systems measure the efficiency of process scheduling through throughput, turnaround time, waiting time, and CPU utilization.

First-come, first-served, round robin, multilevel queue scheduling and priority scheduling

Table 15 Approaches to scheduling

Scheduling approach	Definition
First-come, first-served (FCFS)	Processes are assigned CPU time in the order they arrive in the ready queue, with no pre-emption. The primary metric is arrival time. The CPU serves one process completely before moving on to the next in the queue. It is non-pre-emptive, meaning once a process starts execution, it runs to completion.
Round robin	A pre-emptive scheduling algorithm designed to ensure all processes receive an equal share of the CPU time in a cyclic order. It is characterized by a fixed time slice or quantum, after which the currently running process is swapped out of the CPU if it has not finished execution, allowing the next process in the queue to execute. If a process does not finish during its quantum, it is placed at the back of the ready queue. This continues in a circular queue fashion, hence the name.
Multilevel queue scheduling	Divides the ready queue into several separate queues, each with its own scheduling algorithm and priority level. Processes are permanently assigned to a queue based on their priority or other characteristics, such as process type (system versus user, foreground versus background). Scheduling between the queues can be done based on fixed priority (where each queue has a fixed priority over others) or using round robin (cycling through queues in order).
Priority scheduling	Assigns a priority to each process, and the CPU is allocated to the process with the highest priority. In pre-emptive priority scheduling, if a new process arrives with a higher priority than the currently running process, the current process is suspended, and the CPU is allocated to the new process. In non-pre-emptive priority scheduling, the current process continues until completion before the system checks the queue for the next highest-priority process.

Table 16 Key terms for scheduling

Term	Definition
Slice	The amount of CPU time allocated to a process before it is interrupted or replaced by another process.
Quantum	The fixed amount of time that a process is allowed to run in a pre-emptive multitasking environment, such as in the round robin scheduling algorithm.
Pre-emption	The ability of the scheduling system to interrupt the currently running process to assign CPU time to another process of higher priority.
Fairness	Fairness in scheduling algorithms refers to the equitable allocation of CPU time to processes, ensuring that all processes have a fair chance of executing.
Starvation	Starvation occurs in a scheduling context when a process is perpetually denied necessary resources (for example, CPU time) to make progress due to the continuous interference of other processes.
Efficiency	Efficiency refers to the optimal utilization of system resources, particularly the CPU, to maximize throughput and minimize turnaround time, waiting time and response time.

Table 17 Comparing the approaches to scheduling

	Scheduling type			
Comparator	**FCFS**	**Round robin**	**Multilevel**	**Priority**
Fairness and starvation	Fair in the sense that processes are served in the order they arrive. However, it can lead to the convoy effect, where short processes get stuck behind long ones, leading to poor utilization and potential starvation for processes arriving during a long process's execution.	Designed to be fair by giving each process an equal time slice. However, the choice of time quantum is critical; too short leads to excessive context switching, and too long makes it resemble FCFS.	Inherently prioritizes certain types of processes over others, which can lead to starvation of processes in lower-priority queues.	Risks starvation for low-priority processes unless mechanisms like ageing (incrementally increasing the priority of waiting processes) are implemented.
Efficiency and context switching	The least efficient in terms of CPU utilization and response time, especially with a mix of long and short processes.	Improves CPU utilization and average response time but incurs overhead from frequent context switching.	Can be efficient by allowing different types of processes to be handled appropriately according to their needs but adds complexity in managing multiple queues.	Can be very efficient if priorities are assigned correctly, ensuring that critical processes receive CPU time when needed. However, without pre-emption, high-priority tasks could still wait unnecessarily.

Table 18 Use cases for different scheduling algorithms

Algorithm	Use case
FCFS	Environments with predictable and uniform process lengths, where simplicity and predictability are valued over throughput.
Round robin	Time-sharing and multitasking systems, where it is essential to give each user or task a fair share of the CPU while ensuring responsive system performance.
Multilevel queue	Complex systems with processes of varying types and priorities, such as operating systems that need to balance system and user tasks.
Priority	Real-time systems, where completing high-priority tasks in a timely manner is critical, and the system designer can effectively assign priorities based on the tasks' real-time requirements.

A1.3.4 Evaluate the use of polling and interrupt handling

Interrupt handling and polling are two methods of managing and handling communication between the CPU and devices or programs.



▲ **Figure 60** An interrupt

An **interrupt** is a signal to the processor from a device attached to the computer or from a program within the computer that causes the processor to stop and figure out what to do next. In essence, it interrupts the current processor activity to indicate that it needs immediate attention. When an interrupt is received, the processor saves its state (or context) and starts executing the interrupt service routine (ISR) to deal with the interrupt. After the ISR is finished, the processor resumes its previous state and continues its prior task. This mechanism allows the CPU to respond promptly to important events, improving the system's efficiency and responsiveness.

Polling is a technique in which the processor repeatedly checks the status of a device at regular intervals to see if it needs attention. This is done by executing a sequence of instructions that inquire if the device has data to transfer or if it can accept new data. Polling is essentially a loop that continuously checks the status of all devices one by one. This method is straightforward but can be inefficient because the processor is busy checking devices instead of performing other tasks. Polling is used when device activity is infrequent or when immediate action on events is not required.

ATL Communication skills

Discuss with a partner or group the implications of interrupts and polling for managing real-world scenarios such as user inputs and network communications.

Event frequency, CPU processing overheads, power source (battery or mains), event predictability, controlled latency, security concerns

When evaluating polling and interrupts there are different factors to consider, detailed in Table 19.

Table 19 Factors to consider for evaluating polling and interrupts

	Definition	Interrupt	Polling
Event frequency	Refers to how often an event (such as a device needing service or data ready for processing) occurs.	In a high-frequency environment, interrupts are often preferred because they allow the CPU to be notified immediately when an event occurs, minimising the waiting time and improving responsiveness.	For low-frequency events, polling might be adequate and simpler to implement, as the CPU can afford to check periodically without significant performance degradation.
CPU processing overhead	Involves the extra work the CPU must do beyond its primary tasks, such as managing interrupts or polling devices.	Interrupts can cause overhead due to the need for saving and restoring the CPU state and executing the interrupt service routines.	Polling introduces overhead by consuming CPU cycles just to check if a device needs attention, potentially reducing the efficiency of the CPU for other tasks.
Power source (battery or mains)	Refers to the source of power, battery or mains.	Devices running on batteries, such as mobile devices, benefit from interrupts because they can allow the CPU to enter a low-power state (sleep mode) when not processing interrupts.	Polling, due to its continuous checking nature, can drain the battery faster. Power efficiency is less of a concern for devices plugged in to mains, allowing more flexibility in choosing between interrupts and polling.
Event predictability	Refers to how predictable the timing of events is.	In systems where events occur at known, regular intervals, polling can be an effective method because the CPU can check for events at the optimal times.	In systems with unpredictable events, interrupts are superior because they ensure that the CPU can respond to events whenever they occur without wasting cycles on checking.
Controlled latency	Refers to the ability to have predictable and manageable delays between an event's occurrence and the system's response to it.	Interrupts can provide lower and more controlled latency because the CPU can respond as soon as an event occurs.	Polling might introduce variable latency, depending on when an event happens in relation to the polling cycle.
Security concerns	Refers to potential vulnerabilities these mechanisms can introduce.	Interrupts can be exploited to cause denial-of-service attacks if a device or program overwhelms the CPU by flooding it with interrupts.	Polling, while generally less susceptible to such attacks, might overlook security checks if the implementation assumes regular checks are sufficient for detecting anomalies.

Real-world scenarios including keyboard and mouse inputs, network communications, disk input/output operations, embedded systems, real-time systems

Keyboard and mouse inputs

Interrupts are typically used for keyboard and mouse inputs to ensure immediate responsiveness. When a key is pressed or the mouse is moved, an interrupt signals the CPU to process the input right away, providing a seamless experience to the user. Polling these devices would introduce noticeable lag and decrease user satisfaction.

Network communications

Network communications use interrupts where incoming packets can arrive unpredictably. An interrupt-driven approach allows the network interface card (NIC) to alert the CPU of incoming data, ensuring timely processing and reducing latency. Polling could lead to delays in data processing and increased latency, which is undesirable, especially in high-speed networking environments.

Disk input/output operations

Interrupts are preferred for disk I/O operations to notify the CPU when a read or write operation is complete. This allows the system to efficiently manage data transfers without constantly checking the status of disk operations, freeing the CPU to perform other tasks. Polling for disk I/O completion could negatively impact system performance due to the relatively slow nature of disk operations compared to CPU speeds.

Embedded systems

Embedded systems, such as sensors in IoT devices, often rely on interrupts for efficient power management. These devices usually operate on battery power and need to conserve energy by staying in a low-power state until an event (e.g., a sensor detecting a change) triggers an interrupt. Polling would require the device to be awake more frequently, consuming more power and reducing battery life.

Real-time systems

Real-time systems, such as those used in medical devices, automotive controllers, or industrial machinery, require interrupts to meet strict timing constraints. In these systems, controlled latency is critical. The system must respond to inputs or changes in conditions within a guaranteed time frame to ensure safety and effectiveness. Polling could not provide the timely response needed for these critical applications.



▲ **Figure 61** When you move your mouse, a series of steps occur

Moving your mouse

When you move your mouse, a series of steps occur from the physical movement to the cursor moving on the screen.

Physical movement: You move the mouse. The motion is detected by the mouse's sensors.

Signal generation: The mouse's sensor converts the physical movement into electronic signals, encoding the direction and distance of the movement.

Data packet formation: The electronic signals are formatted into data packets by the mouse's internal processor. These packets include information about the movement and any button clicks.

Transmission to the computer: The data packets are sent to the computer. This happens over a wired connection (USB, PS/2) or wirelessly (Bluetooth, RF).

Interrupt signal: Upon receiving the data, the computer's USB or Bluetooth interface generates an interrupt signal to the CPU, indicating that there is input data to be processed.

Interrupt service routine (ISR) activation: The CPU pauses its current tasks, saving its state, and executes the ISR designated for handling mouse inputs. This routine is part of the operating system's device drivers.

Data interpretation: Within the ISR, the data packets from the mouse are read and interpreted to understand the movement and button states.

Cursor movement calculation: The operating system calculates the new position of the cursor based on the interpreted mouse movement data and the current cursor position. Factors like pointer speed settings are taken into account.

Screen update: The operating system sends instructions to the graphics processing unit (GPU) to update the cursor's position on the screen.

Cursor redraw: The GPU redraws the cursor at its new location on the display, making the movement visible to you.

Resume previous CPU tasks: Once the ISR has completed its execution, the CPU restores its previous state and resumes its interrupted tasks.

This process happens **so rapidly** that the cursor movement on the screen appears seamless and instantaneous to the user. The frequency with which the mouse reports its position to the computer is measured in Hz (times per second). A common reporting rate for modern mice can range from 125 Hz (125 times a second, an 8 ms delay) to over 1000 Hz (1000 times a second, a 1 ms delay).



▲ **Figure 62** A gaming mouse

Gaming mice significantly reduce latency. As they are engineered with higher polling rates, these mice communicate their position to the computer more frequently, reducing the delay between a click and its corresponding action on-screen.

A1.3.5 Explain the role of the operating system in managing multitasking and resource allocation

The challenges of multitasking and resource allocation, including task scheduling, resource contention, and deadlock

Multitasking

Multitasking in computing refers to the ability of an operating system (OS) to manage and execute multiple processes.

Multitasking focuses on the **process level**. Modern operating systems also handle multithreading, where multiple threads within a process can share resources and execute concurrently. These are two related, but different concepts.



▲ **Figure 63** Multitasking

Resource allocation

Resource allocation is the process of efficiently distributing computer resources among various tasks, processes, or users. It is the operating system which manages multitasking and efficient resource allocation. It is a complex job, highly dynamic, and critical to producing efficient performance in a computer system.

Multitasking mechanisms

The OS tracks each running program as a **process** with its own set of resources (memory, CPU time, open files). It maintains process tables with information such as process state (running, waiting, blocked), priority, and resource usage. This is called process management.

AHL

AHL

Based on different ever-changing factors, the OS decides what type of scheduling to use to execute a process queue.

When the OS switches between processes, it saves the state of the current process (register values, memory pointers) in its process table and loads the state of the next process to be scheduled. This process happens extremely fast, often in microseconds, creating the illusion of simultaneous execution. This mechanism of switching is called context switching.

Resource allocation mechanisms

The OS allocates and manages various system resources to ensure efficient functioning of multiple processes.

Memory management: Paging is when the OS divides physical memory into fixed-size blocks called pages. It maps these pages to the virtual address space of each process. This gives the illusion of a larger, continuous memory space and makes memory usage more efficient.

Segmentation is when the OS may organize memory into variable-sized segments for each process (for example, code segment, data segment). Segmentation provides logical structure and enhanced memory protection.

Refer to section A1.3.2 for more information on virtual memory.

Virtual memory is a combination of paging and segmentation that allows the OS to extend available memory using secondary storage (HDD or SSD), giving processes access to seemingly more RAM than is physically installed.

Device management: The OS uses device drivers to communicate with specific hardware components. This ensures proper access, prevents conflicts among processes, and provides an organized way to use peripherals like printers and network cards.

File systems: The OS provides an abstraction layer on top of the raw storage hardware (like hard disk drives or solid-state drives). This layer is the file system and is responsible for defining how data is stored, named, and organized into files and directories (folders). Common file systems include NTFS (Windows), APFS (macOS), and ext4 (Linux). File systems work with the underlying storage devices at the block level. A block is a fixed-size chunk of storage, typically several kilobytes in size. When you read or write a file, the file system translates those operations into interactions with specific blocks on the disk.

Resource monitoring and limits: The OS tracks how processes use resources such as the CPU, memory, and disk input/output (I/O). It can set quotas or limits to prevent a single process from using disproportionate resources, ensuring overall system stability.

Key terms

Fairness The equitable distribution of CPU time and resources among all processes or threads in a system.

Starvation (also known as indefinite blocking) When a process is perpetually denied the resources it needs to progress, due to the continual allocation of resources to other processes.

Task scheduling

Task scheduling ensures that all running tasks are allocated time on the CPU efficiently, balancing system responsiveness, resource utilization, and fairness amongst tasks.

Fairness is deciding which process gets the CPU at any given time and involves balancing fairness and efficiency. High-priority tasks should not starve, while low-priority ones should not monopolize resources.

Different scheduling algorithms, such as priority scheduling or round robin scheduling, each have strengths and weaknesses, making choosing the optimal strategy a complex decision based on current system load.

Pre-emption is suspending a running process for a higher-priority one, which can lead to context switching overhead, impacting performance.

In extreme cases, low-priority tasks might never get CPU time if higher-priority ones constantly demand it. **Starvation** prevention strategies are essential to guarantee basic responsiveness even for less critical tasks.

Resource contention

Resource contention occurs when multiple programs or processes attempt to access the same resource simultaneously, but the resource cannot accommodate multiple concurrent accesses.

Processes can enter a **deadlock** state where each process needs a resource held by another, leading to a system standstill. Deadlock detection and avoidance algorithms are necessary to prevent these scenarios and keep the system operational.

Similarly, **livelock** occurs when processes can get stuck in a cycle of constantly requesting and releasing resources, never making progress.

Thrashing occurs when pages are continuously swapping in and out of physical memory due to insufficient memory. Memory management techniques like demand paging and virtual memory allocation need to be optimized to prevent thrashing.

Deadlock

Deadlock is a specific condition in concurrent computing where two or more processes are each waiting for another to release a resource, or more than two processes are waiting for resources in a circular chain, so that none of the processes can proceed.

Operating systems must be effective in **identifying deadlock** situations. The OS can employ algorithms to monitor resource dependencies and detect potential deadlocks before they occur.

Once a deadlock is detected, the OS must take action to break the cycle (**release the deadlock**) and free up resources. This might involve terminating processes, pre-empting tasks, or reallocating resources.

Proactive measures such as deadlock avoidance algorithms can prevent deadlocks from happening in the first place. This is known as **prevention**. These algorithms analyse resource requests and deny certain requests to avoid creating cyclic dependencies.

A1.3.6 Describe the use of the control system components

A control system is a device or set of devices which manage, command, direct or regulate the behaviour of other devices or systems using control loops. Some examples of controls include automatic elevator control, automatic washing machine, and traffic signal control system.

ATL Thinking skills

Consider the challenges in balancing processes for optimal computer system performance discussed in this section. Suggest how these challenges can best be addressed.

TOK

Whenever a control system has the potential to cause harm, exceptional care must be taken to ensure the design and construction of the system is safe.

How can understanding the components and mechanisms of control systems—such as input, process, output and feedback—contribute to avoiding harm in real-world scenarios such as traffic signal control?

In this section you are going to explore the input–process–output (IPO) model, feedback, and the components of a control system. The section ends with real-world examples of control systems.

The input, process, output and feedback mechanism (open-loop, closed-loop)

The IPO model describes how control systems function to achieve a desired result. Feedback sends data about the output back into the system to adjust or control the ongoing processing.

Table 20 Key terms for the IPO model and feedback

	Definition	In practice
Input 	The data or signals that are fed into a system. In a control system, this could be user input, sensor data, or information received from another system. Inputs can have dynamic ranges and types of inputs (analogue, digital) which control systems accept. The quality and accuracy of input data is important in determining the system's effectiveness.	**Setting a desired value or goal:** For example, inputting a target temperature, speed or position. **Receiving sensor data:** For example, a sensor detecting the current temperature, speed or position.
Process 	The manipulation or transformation of input data. In control systems, this usually happens in the CPU or a microcontroller, where the input data is processed according to a set of predefined instructions or algorithms. In modern control systems can include complex algorithms for real-time processing.	**Comparing desired value to sensor readings:** The controller determines the difference (error) between the input and the output. **Running a control algorithm:** The controller calculates the necessary adjustments to minimize the error. **Generating a control signal:** The controller determines the signal needed to drive actuators.
Output 	The actual response or behaviour of the system being controlled. This is the measurable result of the process component's control action. Examples include the actual temperature of a room, the speed of a motor, or the position of a robotic arm.	**Actuator actions:** For example, motors turning on and off, valves opening and closing, heaters activating. Any physical changes made to the system by actuators. **Change in the system's state:** The actual temperature, speed, position, and so on, being influenced by the actuators.
Feedback 	The mechanism by which the system's output is monitored, evaluated, and then fed back into the system to influence future behaviour. This enables the system to adapt and correct itself in real time. An open-loop control system executes a predefined set of operations without using feedback to modify its actions based on the output. A closed-loop control system continuously monitors its output and adjusts its inputs based on that feedback to maintain the desired state of the system.	**Monitoring output:** Sensors capture the system's current state (such as temperature, speed, position) as output data. This data is critical for evaluating the system's performance against desired outcomes. **Evaluation of system performance:** The feedback mechanism assesses how closely the output matches the target or desired state. This involves comparing the output data to the setpoints or goals established during the input stage. **Adjustment signals:** Based on this evaluation, feedback is used to generate adjustment signals. These signals are instructions sent back to the controller to correct any deviation from the target state.

AHL

Worked example 29

Consider a cruise control system in a car. Identify what parts of the system represent the input, process, output and feedback.

Solution

Input: Driver sets the desired speed with the accelerator and cruise control switch. Sensors measure the car's current speed.

Process: The cruise control system compares the desired speed to the actual speed. It calculates how much to adjust the throttle and sends control signals to the engine's throttle actuator.

Output: The throttle actuator adjusts the fuel flow. The car's speed changes.

Feedback: As the car is constantly adjusting the throttle based on the speed, this is a closed-loop feedback control system.

Activity

In this group activity, you will examine the control system of a dishwasher to understand the components and mechanisms of control systems.

Materials needed

- Internet access for research
- Notebook or digital document for recording observations
- Poster board or presentation software for group work

Instructions

1. Introduction to dishwasher control systems

   In small groups, review the basic components and functions of control systems.

   - Input: Data or signals received by the system.
   - Process: Operations performed on the input to produce the output.
   - Output: The final result or action taken by the system.
   - Feedback: Information about the output used to adjust the input/process.
   - Controller: Manages the operations of the dishwasher.
   - Sensors: Detect changes in the environment.
   - Actuators: Carry out actions.
   - Transducer: Converts one form of energy into another.
   - Control algorithm: Rules or calculations determining how the system reacts to inputs.

2. Research and diagram creation
   - In your group, conduct research on how dishwasher control systems work.
   - Identify and describe the input, process, output and feedback mechanisms.
   - Detail the roles of controllers, sensors, actuators, transducers and control algorithms in a dishwasher.
   - Create a detailed diagram of the dishwasher's control system, labelling all components and showing the flow of information from input to output, including feedback loops.
3. Group presentation
   - In your group, prepare a presentation to share findings with the class.
   - Ensure the presentation covers the key components and mechanisms of the dishwasher's control system.
   - Include your diagram in the presentation.
   - Explain how each component works together to manage and regulate the dishwasher's behaviour.
   - Highlight the differences between open-loop and closed-loop systems within the dishwasher context.
4. Class discussion
   - Engage in a class discussion to compare and contrast the dishwasher control system with other control systems.
   - Discuss the importance of feedback in the dishwasher and how it improves system performance and reliability.

AHL

Controller, sensors, actuators, transducer, and control algorithm

Table 21 Key components and how they are used

Component	Definition	Example and use
Controller	A device or software which processes input from sensors. Based on a control algorithm, the controller then sends signals to actuators.	Example: Programmable logic controller (PLC) Use: In a manufacturing assembly line, a PLC can coordinate the sequence of machinery operations, ensuring that parts are assembled correctly. It processes input data from various sensors to control actuators like motors and pneumatic arms, running the assembly process. (See Figure 64.)
Sensors	Devices which detect and measure physical quantities (e.g., temperature, light, pressure) and convert them into signals the controller can process.	Example: Thermocouple Use: In an industrial furnace, a thermocouple measures the temperature inside the furnace. This data is sent to the controller, which maintains the furnace temperature within a specified range by adjusting the fuel supply. (See Figure 65.)
Actuators	Mechanical or electronic devices that affect changes in a system or environment based on signals from the controller, such as motors or valves	Example: Solenoid valve Use: In an automated irrigation system, solenoid valves control the flow of water to different zones of a field. Based on moisture sensor data, the controller opens or closes these valves to ensure optimal soil moisture levels for crops. (See Figure 66.)
Transducer	Converts one form of energy or signal into another, allowing for seamless communication between various control elements. Transducers play a dual role across sensors and actuators, enabling the conversion of physical quantities into electrical signals and vice versa.	Example: Piezoelectric pressure sensor Use: In automotive tyre pressure monitoring systems, piezoelectric pressure sensors act as transducers by converting the tyre's pressure into electrical signals. These signals are then processed by the vehicle's onboard computer to alert the driver if tyre pressure is outside the recommended range. (See Figure 67.)
Control algorithm	A mathematical formula or logic implemented by the controller to determine the output based on the input data, desired outcome, and current state of the system.	Example: PID control Use: In a home heating system, a PID (proportional–integral–derivative) control algorithm could be used to maintain the room temperature at a comfortable level. The algorithm adjusts the heating output based on the difference between the actual and desired temperatures (error), the error over time (integral), and the rate of change of the error (derivative), ensuring minimal fluctuation and efficient energy use.



▲ **Figure 64** A programmable logic controller (PLC)



▲ **Figure 65** A thermocouple



▲ **Figure 66** A solenoid valve



▲ **Figure 67** A pressure sensor

AHL

A1.3.7 Explain the use of control systems in a range of real-world applications

Complex control systems with sensors, algorithms and actuators

Autonomous vehicles

Autonomous vehicles are complex control systems that rely on a sophisticated interplay of sensors, algorithms and actuators. While they follow the overarching IPO model, their underlying technologies are highly specialized. Sensors like LIDAR (for 3D mapping and object detection), radar (for object detection in bad weather), cameras (for visual recognition tasks), ultrasonic sensors (for close-range proximity), and IMUs (for the vehicle's own motion tracking) provide essential data about the environment.

Control algorithms implement path-planning algorithms, calculate routes, and utilize localization techniques to precisely determine the vehicle's position. In addition, control algorithms enable obstacle avoidance algorithms to predict collisions, and AI-powered decision-making systems to interpret traffic situations. Finally, actuators such as electromechanical steering systems, electronic throttle control, and integrated anti-lock braking systems (ABS) physically carry out the navigational decisions made by the control system.

Home thermostats

Home thermostats and large-scale building management systems (BMS) both regulate temperature using the same core components: sensors, controllers and actuators. Home thermostats typically rely on simple sensors such as bimetallic strips or digital thermistors. These sensors provide temperature readings to the controller, which may use basic on/off logic or more sophisticated PID algorithms for smoother control. The controller then activates actuators such as relays or switches to turn the furnace or air conditioner on or off.

Building management systems take this principle to a larger scale. A network of temperature sensors provides detailed data to a centralized controller, which incorporates factors such as occupancy, weather, and energy costs. Advanced BMS systems might use predictive modelling and machine learning to optimize both comfort and energy efficiency. These systems control complex HVAC (heating, ventilation and air conditioning) equipment, including chillers, boilers and fans. The systems may even integrate with lighting and blinds for comprehensive energy management.

Automatic elevator control systems

Automatic elevator control systems aim to optimize elevator operations, ensuring passengers experience minimal wait and travel times. These systems use sensors to monitor the current position of elevators and where passengers have requested stops (floor positions). A central controller then processes this information, employing algorithms to determine the most efficient way to assign elevators to call requests. This intelligent direction of traffic flow reduces waiting and journey times. Actuators control the motors that physically move the elevator cars between floors.

Automatic washing machines

Automatic washing machines streamline the laundry process by integrating sensors, intelligent controllers and actuators.



▲ **Figure 68** Fully autonomous vehicles will use proximity sensors and share sensor data on a mesh network

TOK

Sophisticated control systems are used in autonomous vehicles, using multiple sensors and highly complex processing systems to make driving decisions and maintain safety standards.

How do control systems enhance the functionality and safety of autonomous vehicles? What assumptions about safety, humans and computers are embedded into the design of these systems?



▲ **Figure 69** Home thermostats regulate temperature using sensors, controllers and actuators



▲ **Figure 70** Elevator control systems reduce waiting and journey times

They often utilize weight sensors or water level sensors to assess the size of the laundry load. A central controller, perhaps using fuzzy logic algorithms, then determines optimal cycle settings. These settings might include wash duration, water temperature, agitation levels and spin speeds, matching the washing process to the needs of the specific load. Actuators control critical elements: water inlet valves regulate water flow, drain valves allow for water removal, and powerful motors drive the washing drum during agitation and spin cycles. Some advanced washing machines might even include sensors to monitor the turbidity (dirtiness) of the water, adjusting the cycle accordingly for greater efficiency.



▲ **Figure 71** Traffic control systems might use real-time traffic data to optimize timing patterns

Traffic signal control systems

Traffic signal control systems manage traffic flow, aiming to reduce congestion and improve safety at intersections. These systems employ a range of sensors to gather real-time traffic data. Inductive loop detectors, embedded in the road surface, sense the presence of vehicles through electromagnetic fields. Cameras, coupled with computer vision, identify and count vehicles, sometimes even differentiating between vehicle types. Pressure-sensitive pads detect pedestrians at crosswalks, allowing for safe crossing. A controller processes this sensor data. Older systems use pre-timed schedules, whereas actuated controllers offer dynamic signal timing based on traffic conditions. Advanced adaptive traffic control systems (ATCS) may even use AI algorithms to optimize traffic flow across entire networks of intersections. Ultimately, the system directs actuators—the traffic lights themselves (red, amber, green) and pedestrian signals—to implement the optimized timing patterns.

Irrigation control systems

Irrigation control systems bring precision and automation to agricultural watering, promoting healthy crop growth while minimizing water waste. Soil moisture sensors include tensiometers, measuring soil water tension, or capacitance-based sensors that detect moisture levels directly. A central controller, often incorporating programmed crop-specific watering schedules, processes the sensor data. Advanced systems might integrate weather data (forecasts, rain sensors) to further optimize irrigation cycles. The controller then directs actuators, typically solenoid-operated valves, to open and close irrigation lines, ensuring water reaches the fields precisely when needed. Some sophisticated systems may even employ variable-rate irrigation, tailoring water delivery to specific zones within a field, based on localized sensor readings.



▲ **Figure 72** Home security systems rely on a network of sensors

Home security systems

Home security systems protect against intrusion, fire, and other hazards through a network of sensors, a central controller, and various actuators. Sensors are the eyes and ears of the system, including motion detectors (often using passive infrared technology), door and window contacts that sense when they are opened, and glass break detectors (acoustic sensors with a shock sensor—the shock sensor detects vibrations directly on the glass). Additionally, smoke and heat detectors and carbon monoxide sensors monitor for environmental hazards. A control panel processes the data from these sensors, using its programmed logic to determine if a real threat exists. If so, it activates actuators such as loud sirens to deter intruders, smart locks for added security, and sends notifications to homeowners or a monitoring service.

Automatic doors

Automatic doors offer convenience and conserve energy by automating entry and exit points. They rely on motion or optical sensors to detect people approaching.

Motion sensors often use microwave or infrared (IR) technology. Microwave sensors send out pulses and analyse the reflected signal for changes, while IR sensors detect differences in heat patterns. Optical sensors, like those used in sliding doors, create a grid of infrared beams—breaking these beams indicates the presence of a person. A controller processes this sensor data and may incorporate logic to prevent false triggers (for example, only opening when someone approaches from a certain direction). The controller then signals actuators to open and close the door smoothly—these are typically electric motors coupled with gear systems or hydraulic mechanisms. Some systems include safety sensors, such as light curtains, as an additional precaution to prevent the door from closing on an individual or object.

Practice questions

27. Describe how operating systems handle storage and file systems. [3 marks]
28. Describe how operating systems ensure system integrity and manage background processes. [3 marks]
29. Describe the importance of networking capabilities in operating systems, including the management of network hardware and protocols. [4 marks]
30. a. Describe the round robin scheduling algorithm. [3 marks]
    b. Discuss its potential drawbacks. [3 marks]
31. Explain how multilevel queue scheduling can be used to manage processes of varying types and priorities. [4 marks]
32. Compare and contrast the efficiency and fairness of round robin and priority scheduling algorithms. [6 marks]
33. Explain the polling method and discuss its potential disadvantages in terms of CPU efficiency. [6 marks]
34. Compare and contrast interrupts and polling in terms of power consumption and suitability for battery-operated devices. [6 marks]
35. Evaluate how security concerns differ between interrupt handling and polling. [6 marks]
36. Explain the role of controlled latency in choosing between interrupt handling and polling for real-time systems. [6 marks]
37. Explain the mechanisms used by operating systems for resource allocation, including memory and device management. [6 marks]
38. Discuss the challenges associated with task scheduling, including fairness, pre-emption, and starvation. [6 marks]
39. Discuss the issue of resource contention in multitasking environments and the strategies used to mitigate its effects. [6 marks]
40. a. Outline the importance of deadlock management in operating systems. [2 marks]
    b. Describe how operating systems can prevent and resolve deadlocks. [4 marks]
41. State the four main components in control systems. [4 marks]
42. Outline the role of feedback in a control system. [2 marks]
43. Explain the functions of sensors and actuators in a control system, including how they interact with other components. [6 marks]
44. State two real-world applications of control systems. [2 marks]
45. Outline how control systems are used in automatic washing machines. [2 marks]
46. Explain the role of sensors, controllers and actuators in the functioning of autonomous vehicles. [6 marks]
47. Explain how traffic signal control systems use sensors and controllers to manage traffic flow at intersections. [6 marks]
48. Evaluate the use of control systems in home security, focusing on the types of sensors employed and their functions. [6 marks]

AHL

A1.4 Translation

Syllabus understandings

AHL **A1.4.1** Evaluate the translation processes of interpreters and compilers

A1.4.1 Evaluate the translation processes of interpreters and compilers

Translation is the process of changing source code into machine-executable instructions.

Table 22 Key vocabulary for translation

Term	Definition
Machine instructions, machine code or machine language *(these terms are synonymous)*	The most basic commands that a computer's processor directly interprets and executes. Machine instructions are encoded in binary. Each pattern of bits represents a specific operation for the processor. The set of available machine instructions and their binary representations vary depending on the processor's architecture (for example, x86, ARM, MIPS).
Assembly language	A (slightly) more readable representation of machine instructions. Assembly language uses short text codes (mnemonics) for opcodes, like `MOV` for move, `ADD` for add, and so on. Assembly might allow you to label memory locations and use those labels instead of raw numerical addresses.
Compiler	A compiler translates code into machine code prior to the code being executed.
Interpreter	An interpreter reads code line by line and translates each instruction into machine code as the program is running. Interpreters perform the translation on-the-fly, executing the code line by line without producing an intermediary binary executable.
Translation	In programming languages, translation is the process of changing human understandable computer instructions (a program) into instructions a machine can execute (machine instructions). There are two common methods for translation: interpreted and compiled.
Programming language	A formal system (lexicon) of instructions and syntax which computers can execute.
Binary executable	A computer file that contains machine code instructions in a format that a particular operating system (such as Windows, macOS or Linux) can directly load into memory and execute.
Bytecode	A form of instruction set designed for efficient execution by a software interpreter.
Parser	A software component which analyses a sequence of input data (often text) according to specific grammar rules. Parsers decompose input into a hierarchical structure (often a parse tree or abstract syntax tree) that reveals the relationships between different elements.
Semantics	The precise meaning associated with specific programming constructs and the expected behaviour of code. It answers the question "What does this instruction do?"
Standard library	A collection of pre-written modules, classes, and functions that provide building blocks for software development within a specific programming language. It is conventionally distributed and maintained as part of the language's implementation.

Translation changes higher-level languages into machine code or instructions.

Table 23 Simplified example of translation

Higher level language	Assembly language	Machine instructions
`score = 42`	`LDR R1, =0x80001000;` Load the memory address 0x80001000 into register R1.	`1110 0100 1000 0001 0010 0000 0000 0000`
	`MOV R2, #42;` Load the value 42 into register R2.	`011 0000 0100 0010`
	`STR R2, [R1];` Store the contents of R2 into the memory location pointed to by R1.	`0101 0000 0001 0010`

The actual machine code and assembly is dependent on the processor's architecture. The example in Table 23 is in the ARM v7 RISC Architecture.

The mechanics and use cases of each translation approach

Interpreted programming languages

An interpreted language is translated into machine language line by line, as the program is run. It reads, interprets, and executes each line of code in sequence. Interpreters stop at the first error they encounter and report it, making them useful for debugging. The program will not continue until the error is resolved. Since interpretation happens in real-time, interpreted programs generally run slower compared to compiled programs because each line of code is translated during execution. They often use less memory as they do not generate an output file containing machine code. Instead, they read the source code directly. Interpreted languages are typically more platform-independent—the interpreter acts as a layer between the source code and the machine code of the underlying operating system.

Interpreted programming languages include Python, PHP, Ruby and JavaScript.

This problem in practice

Python is widely used in the sciences, where researchers can quickly develop and iterate programs to analyse and visualize data, transform complex data sets, and implement sophisticated machine learning. As an interpreted language, Python enables rapid development. The time to write, execute and test code is very short. In comparison to interpreted languages, compiled languages take longer to execute (but tend to run faster).

Katie Bouman led a team that created a new algorithm to produce the first image of a black hole. Python played a significant role in this scientific achievement. Researchers used various Python lvibraries to process the massive amounts of data collected by the telescope network and to perform complex computational tasks necessary to produce the image.



▲ **Figure 73** A radio telescope

AHL

Table 24 The steps of the translation process

Translation step	Description	Example in Python
Initiate execution	The user executes the program: they might click "run" in an IDE, double-click an icon, or run the program from a command line.	`print("Hello World!")`
Lexical analysis	The interpreter decomposes the source code into tokens (keywords, identifiers, operators, and so on).	For `print("Hello World!")`, tokens might include **`print`** as a keyword, **`(`** and **`)`** as operators, and **`"Hello World!"`** as a string literal.
Syntax analysis	The token stream is parsed (by a syntax parser) to verify it adheres to the programming language's grammar rules.	The syntax parser verifies that `print("Hello World!")` follows Python's syntax for a function call: a function name followed by parentheses which enclose arguments.
Semantic analysis	The interpreter checks for semantic errors such as type mismatches and undeclared variables within the parsed structure.	Checks if print is a function that exists and can be called with a string argument, ensuring there are no semantic errors.
Code generation	The parsed code is translated into an intermediate representation (bytecode) specific to the interpreter. In the example given, the byte code is designed to be executed by a virtual machine (Python Virtual Machine).	Converts `print("Hello World!")` into Python bytecode. This bytecode is a lower-level, platform-independent representation of the source code, prepared for execution but not in machine language yet.
Execution	The interpreter executes the bytecode instructions one at a time. This involves fetching, decoding, and executing the specified operation on the data or instruction.	The Python interpreter executes the bytecode for `print("Hello World!")`, leading to the string `"Hello World!"` being displayed on the screen.

The use case for interpreted languages

Interpreted languages have a very fast feedback loop during development.

Interpreted languages do not have a compile–run cycle (write code, compile code, execute code, go back to writing code). You make code changes and very quickly see results. This leads to faster iteration during development than compiled languages. The immediacy of code execution facilitates a more dynamic and exploratory coding process.

Interpreted languages often employ dynamic typing, which allows variables to be assigned to any type of object at runtime, reducing the amount of boilerplate code required for type definitions and enabling faster development.

Scripts written in interpreted languages are generally platform-independent, running on any operating system that has the interpreter installed. This eliminates the need for platform-specific code, making it easier to deploy automation solutions across diverse environments.

Interpreted languages such as Python provide comprehensive standard libraries which have a wide range of functionalities, from file I/O to network communication. Additionally, a large ecosystem of third-party modules further extend this functionality, allowing for the rapid development of scripts with complex capabilities without the need for extensive custom coding.

The ease of use, increased development speed, ease of modification, and platform independence makes interpreted languages an attractive choice for rapid development.

Compiled programming languages

A compiled language is translated from source code to machine language in its entirety before the program is executed, through a process known as compilation. The compiler translates the entire code into a binary executable, which the computer's processor can then run directly. Compiled programs are checked for errors during the compilation process. If errors are found, the compiler will report them, and the executable is not generated until these errors are resolved. This upfront error checking can make debugging less interactive compared to interpreted languages but often results in faster execution times since the program is directly executed in machine language without the need for real-time translation. Compiled languages tend to use more memory during the compilation process but can be more efficient in execution since they generate a directly executable file. This process can make compiled programs less platform-independent, as the executable file is specific to the operating system and hardware for which it was compiled, requiring different compilations for different platforms.

Compiled languages include C, C++, Rust, and Go. Examples of compilers are GCC, CLANG, LLVM, and Microsoft Visual C++ (MSVC).



▲ **Figure 74** Video games and VR experiences are largely programmed in compiled languages to get the best possible performance out of the machine

Table 25 Overview of compilation process

Translation step	Description	Example in Rust
Initiate compilation	The user initiates the compilation process, possibly through a command in a terminal or an action in an integrated development environment.	`println!("Hello, world!");`
Lexical analysis	The compiler breaks down the source code into tokens (keywords, identifiers, operators, and so on).	For `println!("Hello, world!");`, tokens include **`println!`** as a macro, `(` and `)` as delimiters, and **`"Hello, world!"`** as a string literal.
Syntax analysis	The token stream is analysed to ensure it conforms to the language's grammar rules, constructing a syntax tree out of the program.	The syntax parser checks that `println!("Hello, world!");` correctly applies Rust's syntax for macro invocation, ensuring proper structure.
Semantic analysis	The compiler checks for semantic errors such as type mismatches, undeclared variables, and ensures the logic of the program is sound.	Verifies that **`println!`** is a valid macro call and that the supplied string literal matches expected parameter types for printing.
Optimization	The compiler optimizes the code, improving performance without changing its functionality. This step may involve reordering instructions, eliminating redundant code, and optimizing data storage.	No direct example, as this involves transforming the code under the hood to run more efficiently.
Code generation	The optimized code is translated into machine code, generating an executable binary file specific to the target platform.	Translates the Rust program into machine code, creating an executable file named main or main.exe on Windows.
Execution	This is a separate step initiated by the user or the operating system after compilation.	The executable file is run, directly printing **`"Hello, world!"`** to the console without needing to interpret the source code at runtime.

The difference in error detection, translation time, portability, and applicability for different translation processes, including just-in-time compilation and bytecode interpreters

Just-in-time (JIT) compilation

JIT compilation is a hybrid translation process which compiles source code or bytecode into machine code just before execution, rather than compiling the source code into machine code in advance or interpreting it directly at runtime. JIT compilers work by compiling the program's bytecode (an intermediate representation of the code) into machine code dynamically at runtime, as needed. This approach combines the development speed benefits of interpretation with the execution speed benefits of compilation. Code sections that are executed frequently can be compiled once and cached for subsequent executions, greatly improving performance over pure interpretation.

JIT compiled languages include Java, C# (.NET Framework), JavaScript (in certain environments), and Python (PyPy implementation).

Table 26 Comparison of JIT and compiled translations

	JIT compilation	Compiled
Error detection *Identifying and reporting mistakes or inconsistencies in a program's code that prevent it from executing correctly.*	JIT compilers perform error detection during runtime before the compilation of bytecode to machine code. This allows for some runtime errors to be caught early in the execution process but primarily focuses on errors related to bytecode integrity and compatibility with the execution environment.	Compiled languages offer extensive compile-time error checking, including syntax errors, type errors, and some semantic errors. This process ensures that many potential issues are caught before the program is executed.
Translation time *Period required to convert the source code of a program into an executable form.*	Translation occurs at runtime, which means there is an initial performance overhead as bytecode is compiled into machine code. However, this is mitigated over time as frequently executed paths are optimized and cached.	The translation from source code to machine code happens entirely before execution, which means there is no translation time during runtime. However, the development process can be slower due to the need for recompilation after changes to the code.
Portability *The ease with which software can be transferred from one computing environment to another.*	JIT compilation itself is highly dependent on the underlying hardware and operating system, as the generated machine code is platform-specific. However, the bytecode being compiled is typically platform-independent, allowing the same bytecode to be executed on different systems with a compatible JIT compiler.	The machine code generated by compilers is platform-specific, requiring recompilation for each target platform. This limits the portability of compiled executables across different systems without access to the original source code and a suitable compiler.
Applicability *Suitability or relevance of a programming language or technology for addressing specific types of problems or requirements.*	Best suited for applications where performance is critical and the overhead of on-the-fly compilation can be justified by the execution speed of native code. Commonly used in environments that run complex, long-lived applications like server-side applications or integrated development environments.	Ideal for applications that demand high performance and efficiency, such as system software, video games, and applications requiring intensive computation. The upfront cost of compilation is offset by the fast execution speed.

Bytecode interpreters

Bytecode interpreters execute programs written in an intermediate bytecode language rather than directly executing source code or machine code. Bytecode is a lower-level, platform-independent representation of the source code. Bytecode is generated by compiling the high-level source code. The bytecode interpreter then executes this bytecode. This two-step process allows for portability since the same bytecode can run on any platform with a compatible interpreter. Bytecode interpreters balance execution speed and development efficiency, providing faster execution than direct interpretation of source code while maintaining some level of platform independence.

Bytecode interpreted languages include Python (CPython Implementation), Ruby (YARV) and Erlang (BEAM VM).

Table 27 Comparison of bytecode and interpreted translations

	Bytecode interpreters	Interpreted
Error detection *Identifying and reporting mistakes or inconsistencies in a program's code that prevent it from executing correctly.*	Error detection occurs at runtime, as the bytecode interpreter executes the program. While some static errors can be caught during the initial bytecode compilation phase, most error checking happens as the code is interpreted.	Interpreted languages perform error detection at runtime, as the interpreter parses and executes the source code. This allows for immediate feedback during development but can lead to runtime errors that were not caught during development.
Translation time *Period required to convert the source code of a program into an executable form.*	The translation of bytecode to executable actions happens in real-time as the program runs, which can slow down execution. However, since bytecode is a more compact and optimized form than high-level source code, it can be executed more quickly than directly interpreting source code.	There is minimal to no upfront translation time, as source code is executed on-the-fly. This enables rapid development cycles but at the cost of slower execution speed compared to compiled or JIT-compiled code.
Portability *The ease with which software can be transferred from one computing environment to another.*	Bytecode is platform-independent, allowing the same bytecode to run on any system with a compatible interpreter. This greatly enhances the portability of software, making bytecode interpreters particularly useful for cross-platform applications.	High-level source code interpreted languages are highly portable, as the same code can be executed on any platform with a compatible interpreter. This makes interpreted languages ideal for scripts and applications that need to run across different operating systems.
Applicability *Suitability or relevance of a programming language or technology for addressing specific types of problems or requirements.*	Suitable for applications where a balance between performance, development efficiency, and portability is desired. Commonly used in cross-platform environments, mobile applications, and for distributing software that runs in virtual machines (e.g., Java applications).	Best for scripts, rapid prototyping, and applications where development speed and cross-platform compatibility are more important than execution speed. Widely used in web development, educational contexts, and for scripting within larger applications.

AHL

Examples of scenarios where the translation method should be considered

Table 28 Translation in different scenarios

Scenario	Translation method	Example
Rapid development and testing *Aimed at speeding up the software development process so as to create a functional product quickly and efficiently.*	In scenarios where development speed and iteration are important, **interpreted** languages like Python or JavaScript are advantageous. Their syntax is concise, and the absence of a compilation step means changes can be tested immediately. This immediate feedback loop allows for quicker iteration, which is important in early stages of development where modifying code and testing new ideas quickly is more important than raw execution speed.	A tech startup is prototyping an IoT (Internet of Things) application to manage smart devices in real-time. Using an interpreted language, developers can quickly write and test scripts that interact with various devices, adjusting functionality on the fly based on testing feedback without worrying about lengthy compile times.
Performance-critical applications *Speed, efficiency, and stability are paramount for their functionality.*	For applications where performance is critical, **compiled languages** are often the better choice. Languages like C++, Rust, and Go are compiled to machine code, which the CPU can execute directly. This results in faster execution times.	Developing a high-frequency trading platform where transactions need to be executed in microseconds. A compiled language can provide the necessary execution speed and low-level system access to optimize for the fastest possible data processing and response times
Cross-platform development *Compatible with multiple operating systems or platforms.*	The decision between compiled and interpreted languages for cross-platform development depends on the specific requirements and constraints of the project. Languages such as Python and JavaScript are inherently cross-platform, as they run on any system with the appropriate interpreter installed. This makes them suitable for web applications and services where the same codebase needs to run on various server environments and architectures. For desktop and mobile applications, languages like C# (with .NET Core) and Java offer a balance between performance and portability. They compile into intermediate bytecode, which runs on virtual machines designed for different operating systems.	An enterprise software company is developing a customer relationship management (CRM) system that needs to operate on Windows, macOS, and Linux. Using Java, the developers can ensure that the core application logic runs consistently across all platforms, leveraging Java Virtual Machine (JVM) for cross-platform compatibility while still achieving fairly good performance.

AHL

TOK

By diving into the translation process, you can gain insights into the fundamental connections between the code you write and the way it drives system functionality.

To what extent does the process of translating source code into machine-executable instructions influence our understanding of programming languages and their impact on the way systems function?

Practice questions

49. Describe how a high-level instruction is translated into machine code. [4 marks]
50. Outline the role of a parser in the translation process of programming languages. [2 marks]
51. Evaluate the use of bytecode in programming languages translation. [3 marks]

Linking questions

1. What role does multitasking in an operating system play in machine learning (A4)?
2. How might a conditional statement be constructed using Boolean logic gates in a circuit (B2)?
3. What role does task scheduling in an operating system play in managing network traffic and requests (A2.3.3, A2.4)?
4. How does resource allocation in an operating system impact network performance and stability (A2)?
5. What role do GPUs play in non-graphics computational tasks (A4)?
6. To what extent should computer systems not cause harm (TOK)?

End-of-topic questions

Topic review

1. Using your knowledge from this topic, A1, answer the guiding question as fully as possible: What principles underpin the operation of a computer, from low-level hardware functionality to operating system interactions? [6 marks]

Exam-style questions

2. State two components found within the CPU. [2 marks]
3. Describe how buses support the operation of the CPU within a computer system. [3 marks]
4. Distinguish between the instruction register (IR) and the program counter (PC) in terms of their roles within the CPU. [6 marks]
5. Describe the primary difference between GPU and CPU architectures. [3 marks]

AHL

6. Explain how the core architecture of CPUs and GPUs are optimized for their respective tasks. [6 marks]
7. Discuss how the different memory access designs of CPUs and GPUs affect their performance. [4 marks]
8. Explain how the CPU interacts with different types of memory to optimize performance. [6 marks]
9. Compare the performance characteristics of RAM and cache memory in terms of access times and capacity. [4 marks]
10. Define the terms "cache hit" and "cache miss". [2 marks]
11. Describe the fetch phase of the fetch–decode–execute cycle. [3 marks]
12. Describe how the decode phase of the fetch–decode–execute cycle works. [3 marks]

AHL

13. Explain the roles of the fetch, decode, execute, and write-back stages in pipelining within a multi-core processor. [6 marks]
14. Describe the structure and function of an internal solid state drive (SSD). [4 marks]
15. State the binary equivalent of the hexadecimal number 2F. [2 marks]
16. State the decimal equivalent of the binary number 1100101. [2 marks]
17. Explain how integers are represented in binary using both unsigned and two's complement encoding methods. [6 marks]
18. Explain how characters and strings are encoded into binary using ASCII and Unicode. [4 marks]
19. Discuss the advantages and limitations of using binary encoding for video data storage. [4 marks]
20. Evaluate the role of AND, OR, and NOT gates in creating complex decision-making circuits within a computer system. [6 marks]

21. Describe how XOR and XNOR gates function differently from basic AND and OR gates. [4 marks]

22. Describe how the arrangement of transistors in logic gates determines their function. [3 marks]

23. Construct a truth table for the logic expression (A NAND B) NOR C. [4 marks]

24. Deduce a simplification of the Boolean expression using the idempotent law:

    $A \wedge A \wedge B$ [2 marks]

25. Deduce the value of the expression $\overline{\overline{C}}$ using the Involution law. [2 marks]

26. Explain how operating systems manage CPU resources, including process scheduling and interrupt handling. [6 marks]

27. Describe the role of an operating system in managing memory, including allocation, protection, and paging. [4 marks]

28. Describe the role of security management in operating systems, including user authentication and access controls. [3 marks]

29. Evaluate the concept of abstraction in operating systems, particularly how it simplifies interaction with computer hardware for users and applications. [6 marks]

30. Explain how operating systems manage processes, including scheduling and state management. [6 marks]

31. Describe how operating systems handle device management and interrupts. [3 marks]

32. a. Outline the role of virtualization in operating systems. [2 marks]

    b. Describe its benefits for resource management. [2 marks]

33. a. Describe the first-come, first-served (FCFS) scheduling algorithm. [3 marks]

    b. Discuss the effectiveness of the FCFS scheduling algorithm in terms of fairness and efficiency. [3 marks]

34. Discuss the priority scheduling algorithm, including how it manages process priorities and the risks associated with it. [6 marks]

35. a. Describe the basic principles of interrupt handling. [2 marks]

    b. Explain how interrupt handling enhances system responsiveness. [4 marks]

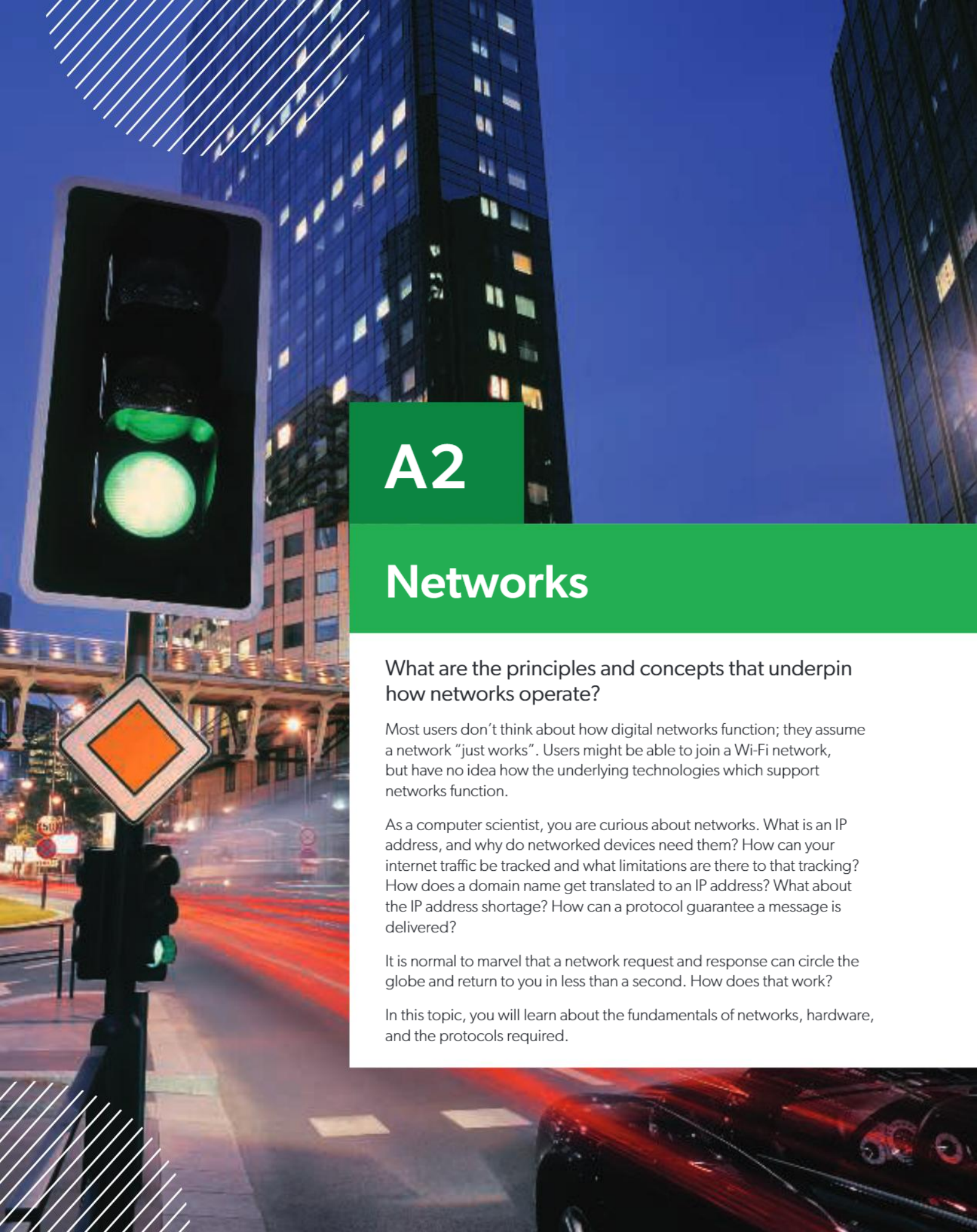
AHL

36. Explain the concept of multitasking in operating systems and how it relates to process and thread management. [6 marks]

37. Explain how a controller functions within a control system using a real-world application. [6 marks]

38. Discuss the importance of control algorithms in managing the operations of control systems, with an example of how they are used in practice. [6 marks]

39. State two differences between compiled and interpreted programming languages. [2 marks]



A2

Networks

What are the principles and concepts that underpin how networks operate?

Most users don't think about how digital networks function; they assume a network "just works". Users might be able to join a Wi-Fi network, but have no idea how the underlying technologies which support networks function.

As a computer scientist, you are curious about networks. What is an IP address, and why do networked devices need them? How can your internet traffic be tracked and what limitations are there to that tracking? How does a domain name get translated to an IP address? What about the IP address shortage? How can a protocol guarantee a message is delivered?

It is normal to marvel that a network request and response can circle the globe and return to you in less than a second. How does that work?

In this topic, you will learn about the fundamentals of networks, hardware, and the protocols required.

A2.1 Network fundamentals

Syllabus understandings

A2.1.1 Describe the purpose and characteristics of networks

A2.1.2 Describe the purpose, benefits and limitations of modern digital infrastructures

A2.1.3 Describe the function of network devices

A2.1.4 Describe the network protocols used for transport and application

AHL **A2.1.5** Describe the function of the TCP/IP model

This subtopic deals with the principles and concepts essential for understanding digital networks. Digital networks are a central part of life and work. Some organizations could not function without digital networks.

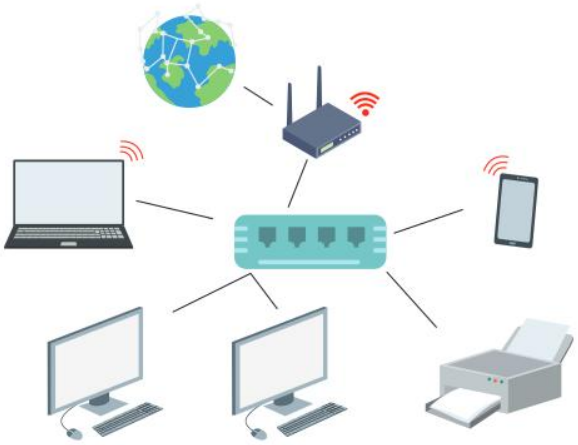
A2.1.1 Describe the purpose and characteristics of networks

There are many different types of networks, each with a specific use case. Networks may share characteristics but have very different purposes.

Local area network (LAN)

A local area network (LAN) connects network devices over a short distance. There is no rigid definition of "short distance", but LANs are designed to operate over distances not exceeding approximately one kilometre. Longer distances introduce latency. LANs are characterized by high data transfer rates and relatively low latency. LANs can be wired or wireless, but are usually a mix of the two. The primary goal of a LAN is to facilitate the sharing of resources such as files, printers, and software applications among multiple users in a local area.

Examples of LANs include a school or college campus, an office building, a hospital, a hotel or a restaurant.



▲ **Figure 1** Local area network (LAN)

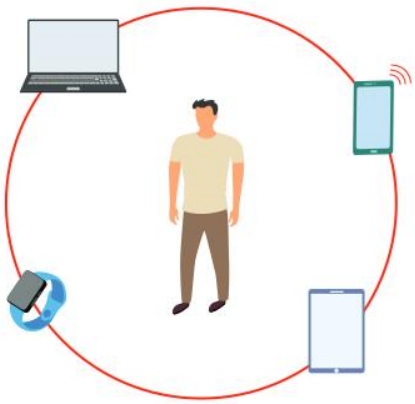
Wide area network (WAN)

A wide area network (WAN) connects network devices across large geographic areas that can span cities, countries, and even continents. WANs are characterized by their ability to maintain data communication over long distances. They usually have lower data transfer rates and higher latency compared to LANs.

The primary purpose of a WAN is to enable businesses, governments and other entities to operate on a wide scale. WANs facilitate the connection of smaller, geographically dispersed networks such as LANs and MANs (metropolitan area networks) into a cohesive system, allowing for centralized data processing, collaborative work, and access to shared resources regardless of location.

Examples of WAN usage include multinational corporations connecting their office networks around the world to share corporate resources and communicate efficiently, and public services providing online access to centralized databases for citizens in different regions.

WANs can be established over leased lines or satellite links, or through public internet connections using virtual private networks (VPNs) to ensure security and privacy. A leased line is a dedicated, fixed-bandwidth communication link between two locations, provided and maintained by a telecommunications company. Unlike typical broadband connections, which are shared among multiple users, a leased line offers exclusive use of the connection for the customer, ensuring consistent speed, low latency, and reliable performance. The infrastructure of WANs might involve multiple transmission technologies and media, ranging from fibre-optic cables to wireless transmissions, to accommodate the vast distances and varied terrains they cover.



▲ **Figure 2** Personal area network (PAN)

Personal area network (PAN)

A personal area network (PAN) is designed for personal use and usually spans no more than 10 metres (about 30 feet). This range is optimal for devices centred around a single person's workspace or within their immediate physical environment. PANs are characterized by their convenience for inter-device communication on a much smaller scale, facilitating direct interaction between personal gadgets such as smartphones, laptops, tablets, wireless headphones and wearable devices.

The primary goal of a PAN is to enable the connection and communication of personal devices for individual use, streamlining the sharing of data and access to personal resources like contacts, internet access, and multimedia files. This network type enhances personal productivity and entertainment by allowing seamless device synchronization, data transfer and internet sharing in a highly localized setting.

Examples of PAN usage include:

- connecting a smartphone to a smartwatch for fitness tracking and to wireless headphones for music streaming
- syncing a laptop with a wireless mouse and keyboard for an uncluttered workspace
- enabling a tablet to access the internet through a smartphone's mobile data connection.

PANs can be established using wireless technologies such as Bluetooth and Wi-Fi Direct, which are specifically designed for short-range communication and require minimal power, making them ideal for personal device connectivity.

Virtual private network (VPN)

A virtual private network (VPN) extends a private network across a public network, allowing users to send and receive data as if their devices were directly connected to the private network. A VPN can function over unlimited distances since it uses the internet to create a secure and encrypted connection between devices and the private network. This encryption ensures that data transmitted over the VPN is protected from unauthorized access.

The primary goal of a VPN is to provide secure remote access to a private network and its resources, such as files, printers, and software applications, from any location with internet access. Remote workers, organizations with global operations, and individuals concerned with privacy and security online all benefit from VPNs. By creating a "tunnel" through the public internet that encrypts data as it travels back and forth, VPNs ensure that sensitive information remains confidential and secure from potential cyber threats.

Examples of VPN usage include:

- employees accessing their company's internal servers and documents securely while working from home
- individuals browsing the internet privately without revealing their **IP address** or location
- connecting to geo-restricted content by appearing to be in a different geographical location.

VPNs can be implemented using software on devices such as laptops, smartphones and tablets, or on routers to secure all traffic from a local network.

Key term

IP address A 32-bit number, typically represented in a dotted decimal format (e.g., 192.168.1.1). Each of the four octets (sections separated by dots) in an IP address is an 8-bit number.

Practice questions

1. State two types of networks that are designed to operate over short distances. [2 marks]
2. Outline the primary goal of a virtual private network (VPN). [2 marks]
3. a. Describe how a wide area network (WAN) functions. [4 marks]
   b. Outline an example of the use of a WAN. [2 marks]
4. a. Describe the characteristics of personal area networks (PANs). [4 marks]
   b. Outline one practical application of a PAN. [2 marks]
5. Describe the advantages of using a LAN in a corporate environment. [4 marks]

A2.1.2 Describe the purpose, benefits and limitations of modern digital infrastructure

The term "infrastructure" refers to the basic physical and virtual components that enable the operation and flow of digital data within a network. Infrastructure is the backbone upon which digital services, applications and communications rely.

You can think of a road system including highways and bridges as an example of infrastructure. All kinds of cars and trucks can drive on the roads and over the bridges. Network infrastructure is like the roads, highways and bridges, and the cars and trucks are like the different network packets moving along the roads.

The internet

The internet is a global network of interconnected devices (routers, servers, computers) which exchange data using standardized protocols. For example, the worldwide web (WWW) uses the internet as infrastructure to manage a global network of hypertext transfer protocol (HTTP)-based resources.

You will learn more about HTTP in section A2.1.4.

Purpose: The primary purpose of the internet is to facilitate reliable, fault-resistant global communication and data exchange.

Benefits: The internet enables global communication and collaboration and provides access to a vast amount of textual, video and audio data. Organizations have used the internet to improve communication with customers and partners, streamline operations and reduce costs and gather valuable customer insights. The internet also enables educational materials and resources to be shared for virtual teaching and learning.

Limitations: Bandwidth can vary greatly depending on geographical distance, network congestion and the quality of infrastructure. The internet must evolve to handle increasing amounts of data, connections and users without degradation in performance or quality of service. Vulnerabilities can pose significant risks to personal, organizational and national security. As critical infrastructure relies on internet-connected systems, security and reliability become even more important.

Cloud computing

Cloud computing is a network of remote servers (as opposed to local servers or personal computers) accessible via the internet and used to store, manage and process data. Cloud computing also offers software as a service, where users only need a web browser to connect and use software applications.

Section A1.1.9 on computer hardware in topic A1 includes a detailed discussion about cloud computing.

For example, schools can use cloud computing to reduce their need for technical support and improve their resources management. Schools can pay a subscription for cloud services rather than buying and managing expensive servers themselves, which they may not need all the time. This saves time and money on often tight budgets.

Purpose: The primary purpose of cloud computing is to provide scalable, on-demand access to computing resources and services.

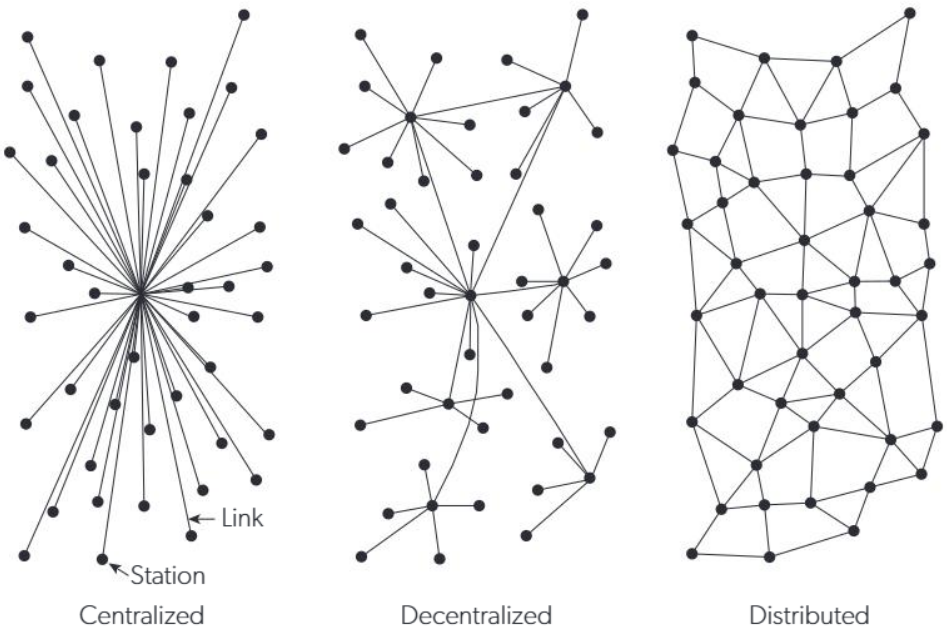
Benefits: Cloud services can be scaled up or down based on demand, providing businesses with flexibility in resource allocation and costs, as organizations pay only for what they use. Cloud computing enables data and applications

to be accessed from anywhere, at any time, enabling collaboration among geographically dispersed teams. Cloud providers can offer data backup and disaster recovery solutions, ensuring data integrity and minimizing downtime.

Limitations: Cloud services require reliable internet access. Limited or unreliable connectivity can hinder access to cloud resources. Storing sensitive data off-site raises concerns about data security, privacy, and regulatory compliance. While cloud providers implement robust security measures, the shared responsibility model requires users to also secure their data.
High data-transfer volumes can incur significant costs. Additionally, bandwidth limitations may affect data-intensive operations, reducing performance. For applications that require real-time processing, the physical distance between the cloud servers and end-users can introduce latency, affecting performance and user experience.

Distributed systems

A distributed system is a network of independent computers that appear to the users of the system as a single coherent system. When discussing a distributed system, computers and devices within the system are referred to as nodes. These nodes communicate and coordinate their actions by passing messages to one another. The more nodes in the system, the more powerful the computational resources.



▲ **Figure 3** The different configuration of nodes in centralized, decentralized, and distributed networks

For example, Apache Hadoop is an open-source framework used for distributed storage and processing of large data sets across clusters of computers. It is designed to scale out from a single server to thousands of machines, each offering local computation and storage. Adding more nodes directly translates to increased processing power, storage capacity, and fault tolerance.

Purpose: The purpose of a distributed system is to coordinate and distribute processes and resources across multiple computers in a network, making them work together to perform tasks more efficiently and reliably than could be possible with a single computer.

Benefits: Distributed systems enable the sharing of resources, providing improved computational speed because tasks are distributed across multiple nodes. This architecture enhances overall system performance and reliability because if one component fails, the system as a whole can continue to operate effectively. Furthermore, distributed systems facilitate flexible and efficient processing by allowing systems to be expanded to accommodate increased loads dynamically.

Limitations: One of the challenges of distributed systems is the complexity of ensuring reliable communication and coordination between components over unreliable networks. Ensuring data consistency and managing concurrency control in an environment where data might be replicated across multiple nodes can be particularly challenging. Security concerns are also amplified in distributed systems, since data is often transmitted over public networks and stored on multiple machines, increasing the attack surface (the number of ways threats can get into the system) for potential threats. The distributed nature of these systems can introduce latency and potentially lower performance for certain operations, compared with centralised systems. Additionally, the design and maintenance of distributed systems require sophisticated algorithms and mechanisms to manage the distribution of tasks, handle failures, and ensure data integrity and security.

Edge computing

Edge computing involves processing data at or near the data source, rather than relying solely on centralized cloud servers. This strategy decreases the physical distance data must travel, leading to faster processing times and lower latency, which is particularly beneficial for applications requiring real-time responses.

For example, think about a smart traffic light system which gathers data about traffic and then changes the timing of the lights to improve the flow of traffic. Where should the processing happen—far away in a remote data centre or next to the traffic lights? Edge computing brings processing near to the data, speeding up the system.

Licence plate recognition systems are another example. Instead of sending raw video feeds to a distant server, the camera sends the processed data (digitized number plates) because the data has been processed near the source.

Purpose: The core aim of edge computing is to bring data processing closer to where the data is generated (the data source). This reduces reliance on a central data-processing warehouse. This decentralization is intended to improve response times and save bandwidth, providing real-time data analysis and processing.

Benefits: Edge computing allows for faster data processing and response times by minimizing the distance data travels. It enhances the efficiency of data transmission for real-time applications such as autonomous vehicles and internet of things (IoT) devices. By processing data locally, edge computing reduces the amount of data that needs to be sent to the cloud, decreasing bandwidth usage and associated costs. It also increases the reliability of data services, as local data processing continues to function even if connectivity to a central server is lost. Furthermore, edge computing can improve security and privacy by

keeping sensitive data on-site, reducing exposure to potential breaches during transmission.

Limitations: The management and maintenance of distributed computing resources can be complex, requiring advanced coordination and automation tools. Security is a double-edged sword: while keeping data local can enhance privacy, it also necessitates robust security measures on numerous devices that might be vulnerable to attack. The initial setup and operational costs can be high, given the need for additional computing resources at multiple locations. Furthermore, there can be inconsistency in computing capabilities across different edge devices, affecting the uniformity of data processing and application performance. Ensuring data consistency across distributed networks also poses a challenge, particularly in environments requiring real-time synchronization across devices.

Mobile networks

Mobile networks are telecommunications networks designed for the connectivity of mobile devices, enabling voice, data, and multimedia communication over significant distances without the need for physical connections. Mobile networks are cellular, covering broad geographical areas and supporting connections for a large number of users with the ability to hand off connections from one cell to another as users move, maintaining seamless connectivity.

Purpose: The purpose of mobile networks is to provide wireless communication services across wide areas, facilitating connectivity for mobile devices anywhere within the coverage area. This enables seamless communication, internet access and data exchange on the go, supporting the increasing mobility of society and the growing demand for constant connectivity.

Benefits: Mobile networks offer the convenience of wireless communication, allowing users to stay connected anywhere there is network coverage. They support a wide range of services, from voice calls and text messaging to high-speed internet access and streaming of multimedia content. The evolution of mobile networks, from 2G to 5G technologies, has significantly enhanced data transmission speeds and reduced latency, improving the user experience for mobile internet services. These networks also enable a vast ecosystem of mobile applications and services, driving innovation in fields such as mobile commerce, telemedicine and remote work.

Limitations: Despite their advantages, mobile networks face several limitations. Coverage gaps or "dead zones" can occur in areas where network signals are obstructed or too far from a cell tower, leading to connectivity issues. The quality of service can vary due to factors such as network congestion, particularly in crowded areas or during peak usage times, impacting data speeds and call quality. Upgrading network infrastructure to newer technologies (for example, from 4G to 5G) requires substantial investment, and the deployment of these advanced networks is often gradual, leading to disparities in service availability. Mobile networks also have inherent security vulnerabilities that can expose users to risks such as eavesdropping, data breaches, and other cyber threats. Additionally, the reliance on finite radio frequency spectrum resources can limit the capacity of mobile networks, necessitating efficient spectrum management and technological innovations to meet growing demand.

Practice questions

6. State the primary purpose of the internet. [1 mark]
7. State one benefit of cloud computing. [1 mark]
8. Describe how distributed systems enhance system performance. [4 marks]
9. a. Describe the use of edge computing in real-time applications. [3 marks]
   b. State one specific benefit and one potential limitation of edge computing. [2 marks]
10. Describe the challenges associated with mobile networks, focusing on service quality and security. [4 marks]

A2.1.3 Describe the function of network devices

There is a wide array of network devices responsible for managing, modulating, routing, converting, sniffing, blocking, shaping and switching packets on a network.

Gateways



▲ **Figure 4** A gateway

Gateways are network devices that act as a bridge between two networks that use disparate protocols. Gateways are used in scenarios where data needs to be translated from one format to another or when different networking technologies need to interact.

Usage scenarios

A gateway can enable communication between an office network and the internet, converting private network addresses to a public address using protocols such as NAT (network address translation). Additionally, gateways might incorporate security functions, filtering, and traffic management to enhance data flow and security across the networks they bridge.

A gateway can also be used to convert data between different network protocols. For example, a gateway might translate email traffic from the simple mail transfer protocol (SMTP) on an enterprise network to another messaging protocol used by an external network, facilitating seamless communication between different email systems.

Within the context of voice over internet protocol (VoIP) communications, a gateway can translate between the digital signals used within an IP network and the analogue signals of traditional phone lines. This enables calls to be made between internet-based VoIP users and traditional telephone users.

In industrial environments, a gateway can bridge the communication gap between a factory's control system and the machines on the production floor, which may use differing communication standards or protocols (for example, translating between Ethernet/IP and a legacy protocol such as Modbus). This helps integrate older machinery into modern control architectures without replacing expensive equipment.

Hardware firewalls

Firewalls are network security devices that monitor and control incoming and outgoing network traffic (packets) based on predetermined security rules. A firewall typically establishes a barrier between a trusted internal network and untrusted external networks, such as the internet, to block malicious traffic and prevent unauthorized access.



▲ **Figure 5** A firewall device

Usage scenarios

A primary use of a hardware firewall is to protect an organization's network by filtering traffic at the packet level. For example, a firewall can block traffic coming from suspicious sources or containing harmful data, preventing attacks on the network.

In scenarios involving network traffic prioritization, firewalls can implement quality of service (QoS) rules to ensure critical applications—such as VoIP calls or video conferencing—receive priority over less critical traffic. By analysing and classifying packets, a firewall can allocate higher bandwidth to these priority packets, reducing latency and improving overall performance for essential services.
For example, your school might prioritize network traffic to educational sites and deprioritize traffic to a gaming site. This ensures valid traffic has enough bandwidth.

In a home environment, a firewall can be integrated into the broadband router to provide basic network protection against external threats. It can prevent potentially harmful communications from entering the home network and can also manage parental controls to restrict access to inappropriate content.

Software and hardware firewalls

Software firewalls are installed as an application on an individual device, controlling incoming and outgoing traffic at the device level based on predefined security rules. They are suitable for providing customized security settings for specific applications and are particularly useful in protecting individual devices from internal threats and unauthorized software actions.

Hardware firewalls are standalone physical units that sit between an entire network and the external world. They manage all the traffic entering and exiting the network through a centralized point, providing broader protection against external threats. These firewalls are designed to handle a higher volume of traffic without impacting the performance of individual network devices.

Software firewalls offer detailed control over individual device activities, while hardware firewalls provide comprehensive protection at the network perimeter for the entire network.

Modems

Modems (modulator–demodulators) are devices that facilitate data transmission over telephone lines or broadband connections by converting digital data from a computer into analogue signals suitable for sending over these lines, and vice versa. They serve as a bridge between the digital data networks and analogue phone systems.



▲ **Figure 6** A modem

Usage scenarios

One primary function of modems is to provide internet connectivity in residential and business environments by interfacing with a service provider's network via standard telephone lines, cable, or satellite connections.

For example, a cable modem converts the digital signals from a router to analogue signals that are suitable for transmission over cable television infrastructure.

In telecommunication setups, modems enable dial-up access to the internet, where they convert the digital data from a computer to an analogue signal that can travel over public switched telephone networks (PSTN). The modem then demodulates the incoming analogue signal back into digital form at the receiving end, allowing for two-way internet communication.

Modems are also instrumental in scenarios involving remote management and telemetry, where they send and receive data to control or monitor remote equipment or systems via telephone lines. This is common in industries such as utilities or services that require remote system diagnostics and updates.

Network interface cards (NICs)

Network interface cards (NICs) are hardware components within a computer or network device which facilitate the interface between the physical network and the device's processing capabilities. NICs perform the critical function of converting electrical signals received from the network into digital data that the computer's processor can understand and vice versa. This conversion is essential for the communication process over computer networks.



▲ **Figure 7** An older style NIC

At its core, a NIC converts raw data signals from the network into usable digital data. For Ethernet NICs, this involves translating the electrical signals transmitted over copper cables into digital data the device's operating system can process. Similarly, for fibre-optic NICs, the NIC converts light signals into digital data. Wireless NICs, or Wi-Fi adapters, function by converting radio frequency signals into digital data.

NICs handle both the transmission and reception of data packets. Each NIC has a transmitter and a receiver, which work simultaneously to send out digital data converted into signals and to receive incoming signals converting them back into digital data.

Integral to its operation, each NIC possesses a unique **media access control (MAC) address**. This address identifies the device on the local network. During data transmission, the MAC address is used to ensure that data packets are delivered to the correct hardware destination on a local network segment.

NICs also incorporate buffers which store data temporarily as it is being sent or received, which helps manage data flow and prevent loss during high traffic periods. Additionally, NICs perform error checking to ensure data integrity, using protocols such as cyclic redundancy check (CRC) to detect errors in the data packets.

Key term

Media access control (MAC) address A unique identifier assigned to a network interface card (NIC) for communications on a physical network. It is a 48-bit hexadecimal number typically written as six pairs of hexadecimal digits (e.g., 00:1A:2B:3C:4D:5E) and is used to ensure that data is sent to the correct device on a local network.



▲ **Figure 8** A router

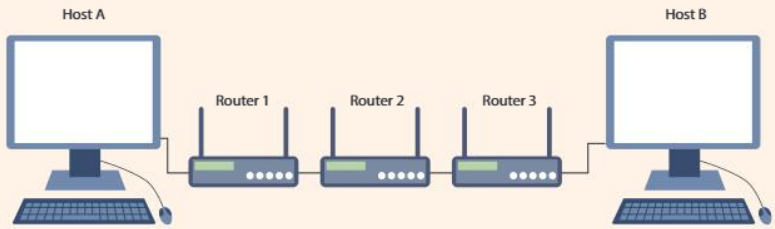
Routers

Routers are network devices which manage the exchange of data between different networks. They direct data packets between networks by determining the optimal paths for data transmission, using routing protocols and routing tables.

At its core, a router examines the destination IP address within each data packet and uses this information, along with a routing table, to determine the best next hop for the packet. Routers maintain and update their routing tables through dynamic or static routing protocols, which help them learn network paths and make intelligent routing decisions.

Routers are responsible for receiving, processing and forwarding data packets to their correct destinations. A packet might pass through many routers to reach its destination. Routers inspect each packet's header to make routing decisions and use protocols such as routing information protocol (RIP) to find the fastest path to the destination.

Worked example 1



Imagine you could follow (or trace) the journey of a packet from its origin to its destination. What might the journey look like? Describe all the processes involved in sending a data packet from one computer to another.

Solution

Imagine a network with the following setup.

- Host A (Source): IP Address 192.168.1.5 (origin)
- Router 1: Interfaces with IP addresses 192.168.1.1 (local network) and 10.10.20.1 (connecting to Router 2)
- Router 2: Interfaces with IP addresses 10.10.20.2 (connecting to Router 1) and 172.16.30.1 (connecting to Router 3)
- Router 3: Interfaces with IP addresses 172.16.30.2 (connecting to Router 2) and 203.0.113.1 (local network)
- Host B (Destination): IP Address 203.0.113.5

A packet's journey may be something like this.

1. Start at Host A
    a. Host A wants to send a packet to Host B.
    b. Host A sends the packet to its default gateway, Router 1, since Host B is not on the same local network.

    At each step (each hop) the packet header is examined and a decision is made how and where to send the packet.
2. Hop 1: from Router 1 to Router 2
    a. Router 1 receives the packet and checks its routing table.
    b. Router 1 determines that the packet destined for 203.0.113.5 should be forwarded to Router 2, via interface 10.10.20.1.
    c. The packet is sent to Router 2.

3. Hop 2: from Router 2 to Router 3
    a. Router 2 receives the packet from Router 1.
    b. Router 2 consults its routing table and finds that the best route to the destination 203.0.113.5 is through Router 3, via interface 172.16.30.1.
    c. The packet is forwarded to Router 3.
4. Hop 3: from Router 3 to Host B
    a. Router 3 receives the packet from Router 2.
    b. Router 3 checks its routing table and identifies that Host B is directly connected to its local network on interface 203.0.113.1.
    c. Router 3 forwards the packet directly to Host B at 203.0.113.5.

Usage scenarios

In a residential setting, routers connect multiple devices to the internet and each other, creating a local network. Home routers often come with built-in wireless access points to support Wi-Fi connectivity.

In business environments, routers connect the enterprise network to the internet or other branch networks over large geographic distances. These routers are more robust and have complex configurations to handle the greater security and traffic management needs of business operations.

High-capacity routers are used by internet service providers (ISPs) and large enterprises to direct internet traffic across the globe. These routers handle massive amounts of data and use complex routing algorithms to maintain internet connectivity and performance.

Switches



▲ **Figure 9** A switch with Ethernet cables

Switches are networking devices that connect devices within a LAN. Switches manage the flow of data within a network by receiving, processing and forwarding data packets (also called frames) to the correct destination device.

At their core, switches receive incoming data packets from one device and use the MAC address information contained within the packet to forward it to the appropriate destination device.

Switches maintain a MAC address table, also known as a forwarding table, which maps each MAC address to the corresponding switch port. This allows the switch to effectively direct traffic by looking up the destination MAC address of each frame and forwarding the frame directly to the port associated with that address.

While switches are efficient at directing traffic, they must handle broadcasts (sent to all devices) and multicasts (sent to a group of devices) differently. For these types of traffic, the switch forwards the frames to multiple ports based on the needs of the network protocol being used.

Usage scenarios

In a small office or home office, a switch can connect multiple devices, such as computers, printers and storage devices, allowing them to share resources and communicate efficiently.

In larger enterprise networks, switches are part of a more complex network infrastructure that includes multiple switches and routers. Here, switches can be configured for advanced functions such as VLAN segmentation and link aggregation to enhance network security, performance and reliability.

In data centre environments, switches play a critical role in managing data traffic between servers and storage systems. High-performance switches in data centres often support additional features such as high-speed throughput rates, low latency, and the handling of large volumes of data traffic.

In schools and universities, switches are used to connect and manage traffic across campus networks, supporting connectivity for thousands of devices used by students, faculty and staff.

Wireless access points (WAPs)

Wireless access points (WAPs) serve as the interface between the wired infrastructure and wireless devices. WAPs convert wired Ethernet data into wireless signals and vice versa, effectively extending the reach of the network to areas where wiring is impractical, impossible or undesirable.

WAPs broadcast wireless signals that wireless devices can detect and connect to. They operate by receiving data packets from the wired network, converting these packets into radio frequencies, and transmitting them wirelessly. The process is reversed for incoming signals from wireless devices, which are converted back into data packets and relayed onto the wired network.



▲ **Figure 10** A WAP

WAPs integrate with the existing network infrastructure using standard network protocols. They can support various security standards such as WPA, WPA2 or WPA3 to ensure encrypted connections.

In larger installations, multiple WAPs can be configured to create a wireless network mesh. This setup allows users to move between different areas (for example, between floors or rooms in a building) without losing connection, as their devices automatically switch to the strongest available signal from the nearest WAP.

WAPs can be managed centrally, often via controllers that configure settings, manage network policies and provide tools for monitoring network usage and performance. This central management helps maintain consistent configurations across multiple WAPs and simplifies the task of updating network settings or firmware.

Usage scenarios

In office environments, WAPs provide employees with the flexibility to connect to the network from anywhere in the building without the need for physical cables, enhancing mobility and collaboration.

Schools and universities use WAPs to cover extensive campus areas, providing students and staff with access to educational resources and the internet from classrooms, libraries and outdoor spaces.

WAPs are widely used in retail environments and public areas such as airports and cafes to offer free Wi-Fi access to customers, improving visitor satisfaction and enabling additional services such as location-based advertising.

In residential settings, WAPs enhance the convenience of wireless connectivity within the home, supporting a wide range of devices including smartphones, tablets and home automation systems.

AHL

How devices map to the layers of the TCP/IP model

Table 1 How devices are mapped to TCP layers

Device	TCP layer	Description of mapping
Gateways	Application layer primarily, but can operate across multiple layers depending on the type of gateway	Facilitate communication between different network protocols, often involving translation of data formats, which can occur at the highest layer (application) of the TCP/IP model. However, some gateways might operate at lower levels if they're translating or interfacing between different types of networks (for example, between a local network and the internet), showcasing their multi-layer functionality.
Hardware firewalls	Primarily internet layer, but can operate on the link layer and transport layer	Primarily inspect IP packets, which places them at the internet layer. However, advanced firewalls (like next-generation firewalls) can inspect data at the transport layer (e.g., TCP/UDP ports) and even up to the application layer, making decisions based on application-specific data. Some also operate at the link layer, controlling access to the physical network.
Modems	Link layer	Operate at the link layer, where they convert digital data from a computer to analogue signals for transmission over various media (such as telephone lines or coaxial cables), and vice versa. This layer is responsible for establishing and maintaining the physical connection between devices.
Network interface cards (NICs)	Link layer	Provide the physical interface between a computer and the network, handling the framing of data for transmission over physical media. They operate at the link layer, directly interfacing with the network cabling and devices.
Routers	Internet layer	Function at the internet layer, where they determine the best path for data packets based on their IP addresses. They are responsible for the routing of packets across multiple networks, making decisions that guide data toward its destination across the interconnected networks of the internet.
Switches	Link layer	Operate at the link layer, providing a central point of connection for devices within a LAN. They manage data frames between devices on the same network, using MAC addresses to forward data to the correct destination.
Wireless access points (WAPs)	Link layer	Function at the link layer, bridging wireless devices to a wired network. They manage the airwaves for Wi-Fi communication and connect those wireless communications to the wired LAN, effectively extending the network to wireless devices.

Practice questions

11. State the primary function of a gateway in a network. [1 mark]
12. Outline the role of hardware firewalls in a network. [2 marks]
13. Describe how modems function within digital communication systems. [4 marks]
14. Outline two advantages of using routers in a network. [4 marks]
15. Describe how switches and wireless access points (WAPs) contribute to the functionality of local area networks (LANs). [4 marks]

A2.1.4 Describe the network protocols used for transport and application

A protocol is a set of rules and standards which defines how data should be structured, transmitted and received across a network. Protocols determine format, timing, sequencing and error checking of data as it is transmitted between devices.

Protocols can be stateful or stateless. A stateful protocol maintains state information about the client–server session across multiple requests and responses. A stateless protocol does not maintain information about sessions. Each communication session is essentially treated as a blank slate.

Transmission control protocol (TCP)

Transmission control protocol (TCP) is a core protocol of the internet. It operates at the transport layer of the TCP model and provides reliable, ordered and error-checked delivery of a stream of bytes between hosts communicating via an IP network. Key features include connection establishment (handshake process), data transfer with acknowledgment, flow control (preventing network congestion) and connection termination. It is used where data integrity and the delivery order of packets are important.

This topic is covered in greater detail in section A2.1.5.

User datagram protocol (UDP)

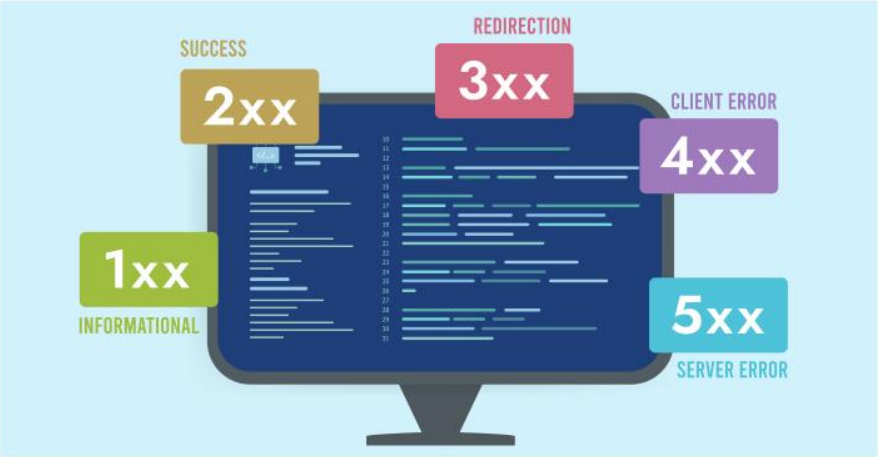
User datagram protocol (UDP) enables a connectionless mode of communication whereby data packets, or datagrams, are sent between devices without establishing or maintaining a stateful connection between the communication endpoints. This approach contrasts with the connection-oriented method employed by TCP, which requires a connection to be established before data can be exchanged.

UDP's connectionless nature makes it faster and less resource-intensive than TCP because it eliminates the overhead associated with setting up and maintaining a connection. This efficiency is particularly valuable in scenarios where occasional data loss is tolerable. Therefore, UDP is widely favoured for real-time applications such as video streaming, online gaming and VoIP. In these use cases, the priority is on reducing latency and maintaining continuous flow of data, even if it means some packets are lost or arrive out of order.

UDP also supports multicasting—the transmission of a packet to multiple destinations in a single send operation—making it suitable for applications such as live broadcasts and group collaborations. Despite its simplicity, UDP's utility in specific contexts—where the overhead of guaranteed delivery is an unnecessary burden—makes it an indispensable part of the network communications landscape.

Hypertext transfer protocol (HTTP)

Hypertext transfer protocol (HTTP) is the foundational protocol for data communication over the worldwide web (WWW). It delineates the exact structure and encoding of data transferred between web servers and browsers, facilitating the transfer of web pages as HTTP messages. The messages sent by a client (usually a web browser) are termed HTTP requests, and the messages sent by the server in response are called HTTP responses. Responses are often preceded by response codes, some of which you may have seen as you have used the worldwide web.



▲ **Figure 11** HTTP responses

Table 2 Some common HTTP response codes

HTTP response code	Description
200 OK	Indicates that the request has succeeded and the server has returned the expected content.
404 Not Found	The server cannot find the requested resource. Often triggered by broken links or incorrect URL input.
500 Internal Server Error	A generic error message indicating that the server encountered an unexpected condition that prevented it from fulfilling the request.
301 Moved Permanently	This response code is used when the requested resource has been permanently moved to a new URL, and any future references should use one of the returned URLs.
403 Forbidden	The server understands the request but refuses to authorize it. This often occurs when access to the requested resource is restricted or denied.

This protocol operates primarily using a request/response model. A client sends an HTTP request to the server, which includes a method (for example, GET to fetch resources, POST to submit data), a header (which conveys metadata such as content type and authentication information), and sometimes a body (which contains data being sent to the server). The server processes this request and returns an HTTP response, which includes a status code (indicating success or failure), response headers (similar to request headers but for response settings), and usually a response body containing the requested resource or error details. This model is fundamental for web interactions, enabling the vast array of web functionalities available.

HTTP is characterized as a stateless protocol, which means that each request made by a client is processed by the server independently, without any inherent knowledge or memory of previous interactions.

This stateless nature requires that any session information needed to be maintained must be included in the HTTP messages themselves or handled by external mechanisms such as cookies.

Cookies are files stored on the client's device which store state information. They are sent with requests to enable stateful communication. Sessions are useful in web applications because they allow the server to maintain user-specific data across multiple HTTP requests (if you are logged in, what you last looked at in an online store, what time zone you are in, and so on).

HTTP state management refers to the methods and techniques used to maintain the state of a user's interaction with a web application across multiple HTTP requests. A common example of state management is the use of cookies, which store data on the client's browser to help maintain continuity between requests.

Hypertext transfer protocol secure (HTTPS)

Hypertext transfer protocol secure (HTTPS) is an extension of the basic HTTP, specifically designed to enhance security for communications over the worldwide web. HTTPS incorporates encryption into the data transmission process to protect the integrity and privacy of the data exchanged between a web server and a browser.

Learn more about encryption in section A2.4.4.



▲ **Figure 12** HTTP and HTTPS protocols

The security in HTTPS is achieved through the transport layer security (TLS) protocol, and formerly through its predecessor, the secure sockets layer (SSL). These protocols encrypt the data before it is sent over the internet and decrypt it upon arrival at its destination. This encryption process ensures that even if the data is intercepted during transmission it remains unreadable and secure from unauthorized access.

The use of HTTPS is indicated in the **scheme** of a URL of a website with the prefix "https://" rather than "http://". This protocol employs a combination of asymmetric and symmetric encryption.

HTTPS significantly enhances the security of web interactions by safeguarding against eavesdropping, tampering and message forgery, thereby providing a trusted foundation for secure communication on the worldwide web.

Key term

Scheme The part of the URL that specifies the protocol used to access the resource. It appears at the beginning of the URL and is followed by "://". The scheme indicates how the resource should be retrieved and can influence the way data is transmitted over the network.

Dynamic host configuration protocol (DHCP)

Dynamic host configuration protocol (DHCP) automatically assigns IP addresses and other essential network configuration parameters to devices when they connect to a network. This automatic configuration is helpful when a device, such as a computer or smartphone, first connects to a network, ensuring it receives the appropriate network configuration settings to communicate effectively with other devices.



▲ **Figure 13** DHCP stands for Dynamic Host Configuration Protocol

The primary function of DHCP is to simplify network administration. Without DHCP, network administrators would need to manually assign IP addresses to each new device joining the network—a time-consuming and error-prone process. DHCP automates this task, ensuring efficient and correct configuration without manual intervention.

When a device connects to a network, it sends a broadcast query requesting the necessary information. In response, the DHCP server assigns an IP address to the device from a predefined range of addresses, known as a scope.

Key term

Subnet mask A subnet mask is a 32-bit number, used to divide an IP address into network and host portions. Subnet masks determine which part of an IP address refers to the network and which part refers to the host. The 1s in the subnet mask indicate the network part, and the 0s indicate the host part. For example, the subnet mask 255.255.255.0 in binary is 11111111.11111111.11111111.00000000.

In addition to the IP address, DHCP also provides other configuration details vital for network communication, such as the following.

- **Subnet mask**, which determines the network segment the device is on.
- Default gateway, the device that forwards traffic to other networks.
- Domain name system (DNS) server addresses, which are used to resolve domain names into IP addresses.

DHCP can allocate IP addresses in two primary manners.

- **Dynamic allocation** is when DHCP assigns an IP address to a device for a limited period, known as a "lease." Once the lease expires, the device must request a new IP address, although often the same IP may be reassigned if still available. This method is particularly useful in environments with frequently changing devices, like guest Wi-Fi networks.
- **Static allocation** is when DHCP assigns a permanent IP address to a device based on its MAC address. This method is used for devices that require a consistent IP address, such as network printers or servers.

By managing IP address distribution and network settings, DHCP helps prevent address conflicts (where two devices on the same network end up with the same IP address) and reduces the overhead associated with network resource management. This capability is integral to maintaining smooth network operations, particularly in large-scale or rapidly changing network environments.

Practice questions

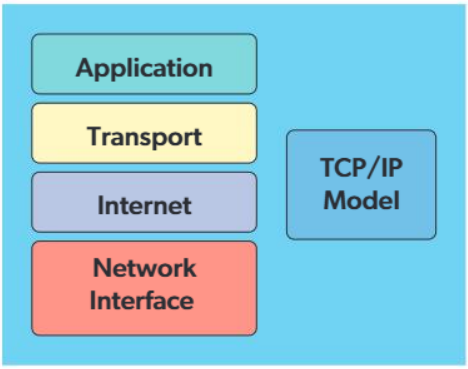
16. Describe the key features of TCP that ensure reliable data transmission. [4 marks]
17. Distinguish between TCP and UDP in terms of connection management and typical use cases. [6 marks]
18. Describe how HTTPS enhances the security of HTTP communication over the worldwide web. [4 marks]
19. Describe the role of the DHCP in network configuration. [4 marks]
20. Describe the benefits and drawbacks of using a stateless protocol like HTTP for web communication. [4 marks]

A2.1.5 Describe the function of the TCP/IP model

Digital networks are complex, interdependent systems with many different devices, protocols, transmission media, and potential problems. Because networks are so complex, conceptual models are used to aid understanding, characterizing, troubleshooting and standardizing network operations. A well-known conceptual model is the TCP/IP model.

Each layer of the model articulates a specific networking activity. In other words, at each layer, something happens to a message before it is passed to the next layer. Normally, network traffic goes through layers sequentially. As a message is received by a computer it travels from the bottom up and, as a message is sent, it travels from the top down.

Application, transport, internet and network interface: The role of each layer and the interaction between them to ensure reliable data transmission over a network



▲ **Figure 14** The different layers of the TCP/IP model

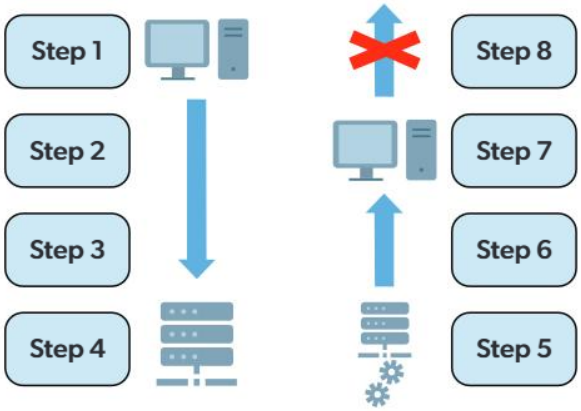
Table 3 Layers and roles of the TCP/IP model

Layer	Role
Application layer	Provides the interface between the applications on a computer and the network. It defines protocols which applications use to exchange data over the network, such as HTTP for web browsing, SMTP for email, DNS, DHCP and simple network management protocol (SNMP) which is used for managing and monitoring network devices.
Transport layer (TCP and UDP)	Facilitates direct communication services for application processes across various hosts. Transmission control protocol (TCP) employs sequencing, checksums, acknowledgments, and retransmissions to guarantee orderly and error-free data delivery to the destination. User datagram protocol (UDP) offers a quicker but less reliable service compared to TCP.
Internet layer	Handles the packetization, addressing, and routing of data across different networks to ensure it reaches its intended destination. The primary protocol in this layer is the internet protocol (IP), which defines IP addresses that uniquely identify each device on the network.
Network interface layer	Concerns itself with the physical and hardware aspects of network communication. It handles how data is physically transmitted over the network media, including aspects such as cable specifications, signal encoding and decoding, data framing, and MAC addressing for Ethernet networks. This layer is where network adapters (such as Ethernet cards and drivers) operate, interfacing directly with the physical network medium (such as copper wires, fibre-optic cables, or wireless signals). Protocols used at this layer include 802.3 (ethernet) and 802.11 (wireless).

Reliability comes from the TCP/IP model's layered approach, where each layer's specific protocols dictate precisely how data should be handled, encapsulated, transmitted, and received. Error-checking, checksums, sequencing and acknowledgments of messages all add to the reliability of the data sent via TCP/IP.

A simplified step-by-step example of how each layer in the TCP/IP model functions

Figure 15 illustrates the journey of data from the user's web browser through the network to the web server and back to the user's browser.



▲ **Figure 15** The different steps of the TCP/IP model

Here is a description of the steps shown in Figure 15.

Step 1. Application layer: User requests a web page

a. The user enters a URL in their web browser and presses enter or clicks.

b. At the **application layer** the web browser (application) constructs an HTTP GET request to retrieve the web page. This request is prepared according to the HTTP protocol operating at the application layer.

c. The request is passed down to the transport layer.

Step 2. Transport layer: Preparing the request for transmission

a. At the **transport layer**, the HTTP request is handed down from the application layer, where it is changed into a segment (called a TCP segment). Each segment is given a TCP header, which includes sequencing information, source, and destination port numbers (e.g., port 80 for HTTP, port 443 for HTTPS), and a checksum for error checking.

b. A TCP connection is established using a three-way handshake process. SYN, SYN-ACK, and ACK packets are exchanged between the client and server to establish a reliable connection.

   i. SYN (synchronization)

      The client begins the process by sending a SYN segment to the destination server. "Hey, I want to talk, do you want to talk?"

   ii. SYN-ACK (synchronization-acknowledgement)

      The destination server receives the SYN segment and responds with a SYN-ACK segment. "Ok, I got your message, yes, I'm ready to hear your message."

   iii. ACK (acknowledgement)

      The client receives the server's SYN-ACK and sends a final ACK segment. "Great. I understand you are ready to talk, and I'm going to start sending data!"

c. After the three-way handshake, the TCP connection is considered established, and data transfer, like our HTTP GET request, can proceed. The segment is passed down to the internet layer. If for some reason the TCP handshake fails, the request will not be passed to the internet layer and the user might encounter a "connection timed out" message.

Step 3. Internet layer: Preparing the segment for routing

a. At the **internet layer**, each TCP segment is encapsulated within an IP packet, which includes the IP address of the source (the user's device) and the IP address of the destination (the web server). In addition, IP packets contain the protocol (the type of data encapsulated, in this case, TCP), a time to live (TTL), which prevents packets from circulating indefinitely on the network, and a header checksum for error detection in the IP header itself.

b. At this layer, the packet is also assigned a next hop (where should this packet be sent to next), which is usually the address of the router on your network.

c. After the packet has been created, it is passed down to the network interface layer.

Step 4. Network interface layer: Final encapsulation and converting to signal

a. The **network interface layer** encapsulates the IP packet (which contains the TCP segment and HTTP request) into an Ethernet frame (assuming you are on an Ethernet network). This frame includes destination MAC address (the MAC address of the next hop device), the MAC address of your own machine, and control information and a checksum for error detection.

b. At this point, the frame is physically transmitted.

   i. The network interface card (NIC) converts the Ethernet frame into electrical signals (or light pulses, for example, depending on the medium) suitable for the physical network connection.

   ii. The signals representing the frame are finally transmitted over the physical medium, be it an Ethernet cable, Wi-Fi signal, or other technology.

c. Please keep in mind the network interface layer is where the protocols and operations become specific to the type of physical network (Ethernet, Wi-Fi, fibre-optic).

Between steps 4 and 5, the next part happens outside of your computer and network: The frame is routed through the internet, potentially hopping through multiple routers, until it reaches the server. Each router uses the frame's destination IP address to determine the next hop towards the destination.

AHL

Key term

Broadcast address The broadcast address is used to communicate with all devices on a network. It is identified by having all host bits set to 1.

Step 5. Receiving the request on the server

a. At the **network interface layer**, the server's NIC receives the electrical (or other) signals from the physical network and converts them into an Ethernet frame. The NIC checks the frame checksum to ensure no errors occurred during transmission. If errors are detected, the frame is discarded. The NIC examines the destination MAC address. If it matches the server's MAC address (or a **broadcast address**), the frame is processed further. The network interface layer strips off the Ethernet header and trailer, leaving the IP packet, and passes the frame to the internet layer.

b. At the **internet layer**, the server's IP layer examines the destination IP address in the packet header. If it matches the server's own IP address, the packet is accepted. The IP header checksum is calculated to verify the header's integrity. Corrupted packets are discarded. Finally, the IP layer removes the IP header, exposing the encapsulated TCP segment. The TCP segment is passed to the transport layer.

c. At the **transport layer**, the TCP layer looks at the destination port number in the TCP segment header. This directs the data to the correct application process listening on that port (port 80 for HTTP, 443 for HTTPS). If the HTTP GET request was split into multiple TCP segments, the TCP layer uses the sequence numbers to reassemble the data in the correct order. The TCP checksum is calculated to ensure no errors were introduced during transmission. Corrupted segments are typically discarded or trigger requests for retransmission. If the TCP segment checksums are valid and the data is successfully reassembled, the intact HTTP GET request is handed up to the application layer.

Step 6. Server responds with the web page

a. At the **application layer**, the web server software generates an HTTP response, containing the requested web page's HTML content.

b. At the **transport layer**, the HTTP response is changed to a TCP segment, encapsulated with TCP headers, and the segments are sent back to the client.

c. At the **internet layer**, the response TCP segments are encapsulated within IP packets.

d. At the **network interface layer**, the packets are converted to frames and changed into signals, and are then transmitted.

Step 7. User receives the web page

a. At the **network interface layer**, the user's network interface card (NIC) receives the electrical (or other) signals from the physical network and converts them into an Ethernet frame. After checking the frame, it is discarded, and the IP packet is sent up to the internet layer.

b. The **internet layer** inspects and then removes the IP header, exposing the encapsulated TCP segment. The TCP segment is passed to the transport layer.

c. At the **transport layer**, if the TCP segment checksums are valid and the data is successfully reassembled the intact HTTP GET request is handed up to the application layer.

d. At the **application layer**, the web browser interprets the HTML content and displays the web page to the user.

Step 8. Connection termination

Finally, after the web page is successfully transmitted, the TCP connection is closed through a connection termination process, where FIN and ACK packets are exchanged to gracefully close the connection.

a. A FIN packet signals that the sender (either the client or the server) has finished sending data and wants to gracefully close the connection in one direction.

b. An ACK packet confirms the receipt of data.

TOK

The process of transmitting a web page from a server to a client involves multiple layers of communication, each handling specific tasks to ensure accurate data transfer. This intricate process underscores the complexity and efficiency of modern networking protocols.

To what extent do we rely on invisible systems in our daily lives, and how does our understanding of these systems (or lack thereof) affect our trust in them?

Few users understand the intricate technical details of networks. To what extent are network engineers and developers ethically responsible for ensuring the security and privacy of data transmitted over the internet?

Practice questions

21. Outline the function of the application layer in the TCP/IP model. [2 marks]
22. Outline how the transport layer ensures reliable data transmission. [2 marks]
23. Describe the role of the internet layer within the TCP/IP model. [3 marks]
24. Describe the function of the network interface layer in the TCP/IP model. [3 marks]
25. Describe the importance of the layered approach of the TCP/IP model in ensuring the reliability of network communications. [5 marks]

A2.2 Network architecture

Syllabus understandings

A2.2.1 Describe the functions and practical applications of network topologies

AHL **A2.2.2** Describe the function of servers

A2.2.3 Compare and contrast networking models

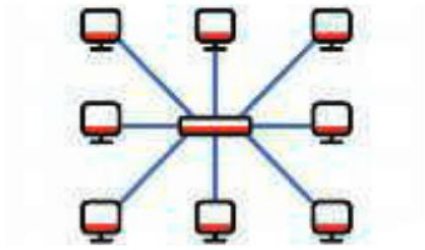
A2.2.4 Explain the concepts and applications of network segmentation

Network architecture refers to the overall design of a network, encompassing its components, layers, protocols, and how they interact.

A2.2.1 Describe the functions and practical applications of network topologies

Network topology is the arrangement of devices (nodes) and connections (links) within a network. Topology can be physical or logical. Physical topology refers to the actual layout of cables, wires, routers, switches and other hardware components. Logical topology refers to how data flows across the network, regardless of the physical arrangement of devices.

Star topology



▲ **Figure 16** A star topology

A star topology is a configuration where each device connects directly to a central device (usually a switch). This creates a star-shape with the central device at the centre.

Reliability of star topologies is high, since each device is independently connected. A single cable or device failure will not take down the entire network. Only the affected device becomes isolated. Of course, if the central switch or hub fails, the whole network fails.

Transmission speed is generally fast. Modern switches and hubs minimize transmission delays. Direct connections between devices and the central node reduce the potential for bottlenecks that might occur in other topologies.

The scalability of star topologies is good, and adding or removing devices is as simple as connecting a new device to an available port on the central switch. However, central switches have a finite number of ports, ultimately limiting how many devices you can add.

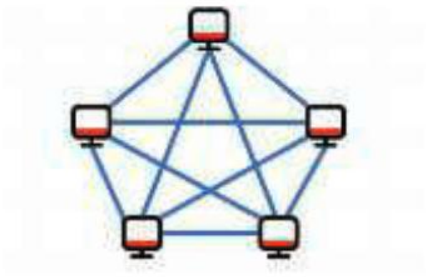
Data collisions are rare because modern switches largely eliminate collisions by managing traffic effectively.

The cost of star topologies can be high because they require more cabling than some alternatives. Each device has its own dedicated line to the centre.

A common example of a star topology is a home network. Most home networks use a star topology with a central router or wireless access point. Business LANs often use a star topology, with computers and other network devices connecting to centrally managed switches.

Mesh topology

Mesh topology is where devices (called nodes) connect directly to as many other nodes as possible. This creates a mesh-like structure. Full mesh networks are when every node is connected to every other node. Partial node networks are when nodes are connected to some but not all other nodes.



▲ **Figure 17** A mesh topology

Reliability of mesh topologies is very high. If one node fails or a connection is broken, data can be automatically rerouted through other nodes. This redundancy provides resilience. Reliability is improved because there is no single point of failure like a central switch.

Transmission speed depends on factors such as the number of hops data takes, the strength of connections between nodes, and network traffic. (A hop refers to data packets being transmitted from one device, such as a router, switch or gateway, to another device.) Mesh networks can be fast, but may also experience some slowdown on heavily used paths.

Scalability of mesh topologies involves adding nodes, powering them on, and placing them within range. Many mesh protocols facilitate automatic configuration, making scalability very easy.

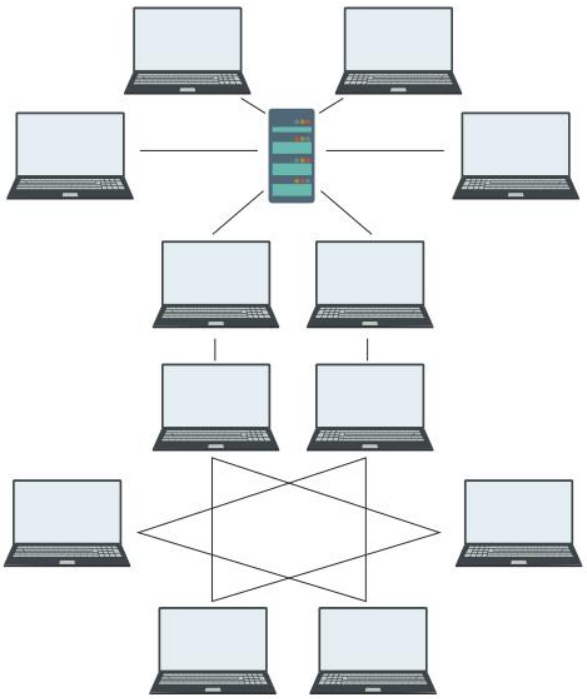
Intelligent routing algorithms in mesh networks help find the most efficient data paths, reducing the chances of multiple signals competing on the same channel. This makes data collisions uncommon.

The cost of a mesh network depends on the type of nodes used. Simple mesh systems can be affordable, while more sophisticated enterprise-grade nodes will be more expensive.

Common examples of mesh topologies include modern Wi-Fi mesh systems (such as Google Nest, Eero) and some communities use mesh networks to provide internet access in areas with limited infrastructure. Mesh networks can provide reliable connectivity for sensors and other devices in manufacturing or warehousing environments.

Hybrid network

Hybrid networks combine elements from different networking topologies (such as star and mesh) to optimize performance, reliability and scalability based on specific needs and conditions. Hybrid networks are designed to leverage the strengths of multiple topology types, avoiding the limitations inherent to any single topology. For example, a hybrid network might use a star topology for its core infrastructure, connecting main servers and routers, while employing a mesh topology for peripheral devices to ensure robustness and redundancy.



▲ **Figure 18** Hybrid network topology

The reliability of hybrid networks is typically very high, benefiting from the combined strengths of the incorporated topologies. The presence of multiple paths for data transmission ensures that the network can maintain connectivity even if certain components fail. This redundancy, drawn from mesh topology aspects, alongside the centralized management from a star or bus topology, ensures a balance between resilience and control.

Transmission speeds in hybrid networks can vary, influenced by the network's specific configuration and the topologies it integrates. The use of direct connections in certain segments (akin to mesh topology) can enhance speed and reduce hops, while centralized elements might streamline data routing and management, improving overall efficiency.

Hybrid networks offer flexible scalability options, allowing expansion in a manner that aligns with the network's evolving needs. Adding new devices or nodes can be facilitated through the easier-to-scale components of the network, such as those based on mesh or star topologies. This enables the network to grow or reconfigure without significant disruptions to existing operations.

The incorporation of intelligent routing algorithms is important in hybrid networks, managing the complexity of data paths across different topologies. These algorithms optimize the routing of data packets, taking into account the diverse characteristics of the network's parts to enhance performance and minimise collisions and congestion.

The cost of implementing and maintaining a hybrid network can vary widely, depending on the complexity of the design and the types of topologies integrated. While the flexibility of hybrid networks allows for cost-effective solutions by combining inexpensive and more costly components judiciously, the overall expense can be higher than that of more straightforward network designs due to increased management and maintenance requirements.

Hybrid networks are particularly useful in complex environments where no single topology meets all the requirements. Examples include large corporate networks that use a combination of star topology for central offices and mesh topology for connecting remote workstations and devices. Similarly, smart cities might employ hybrid networks to integrate various services and infrastructure needs, ensuring robustness, flexibility and coverage.

Practice questions

26. State one advantage of using a star topology in network design. [1 mark]
27. Outline the primary difference between physical and logical network topologies. [2 marks]
28. Describe two features of mesh topology. [4 marks]
29. a. Describe the benefits of hybrid network topology using examples from the text. [4 marks]
    b. Describe a real-world application of hybrid network topology. [2 marks]

A2.2.2 Describe the function of servers

A server is a computer system (either hardware or software) that delivers data, resources or services to other computers (known as clients) over a network. Any computer can be a server, but servers are generally different from desktop or laptop computers you might use. They are designed for very high reliability, fault-tolerance and robust performance.

Domain name system (DNS) server

The function of a DNS server is to translate human-readable domain names (such as www.google.com) into their corresponding numerical IP addresses (such as 172.217.16.14). This translation is required for address resolution and packet routing to work (see A1.2.5). If your local DNS server cannot find the IP address for a domain name you have requested, it employs recursion by querying higher-level DNS servers within the internet's hierarchical structure until it locates the server that can provide the correct IP address.

AHL

A DNS lookup

1. Enter a domain name (such as www.google.com) into your web browser.
2. The request for the domain name's IP address is sent to your local DNS server, which is typically operated by your internet service provider.
3. The local DNS server checks its cache to see if it has a recent record of the IP address for the domain name. If it finds the address in its cache, it returns the IP address to your computer, and the process ends here.
4. Recursive query:
    a. If the IP address is not in the cache, the local DNS server sends a query to a root DNS server. The root server does not know the IP address but directs your local DNS to a top-level domain (TLD) server (for example, for .com domains).
    b. The TLD server, in turn, does not store the IP address but directs to the authoritative DNS server for the domain (which knows the actual IP address).
5. Authoritative DNS server:
    a. The local DNS server then queries the authoritative DNS server for the domain name you requested.
    b. The authoritative server returns the corresponding IP address back to your local DNS server.
    c. The local DNS server caches the IP address for the specified time (defined by the time to live (TTL) in the DNS record) to speed up future requests.
6. The local DNS server sends the IP address back to your computer. Your browser can now use this IP address to connect to the web server hosting the website you requested.
7. Your browser sends a request to the web server at the resolved IP address to load the web page.

DNS servers are spread throughout the world, ensuring that requests can be answered from geographically close locations, reducing latency. DNS servers store resolved domain-name-to-IP address mappings for a period of time. This means frequently requested domain names can be served quickly from the cache, reducing the load on the broader DNS system. For example, once your DNS server knows acm.org can be reached at IP address 104.17.78.30, it does not need to look it up again for a while. Large DNS providers use load balancing techniques to distribute requests across multiple servers, preventing any single server from becoming overloaded.

DNS systems use multiple servers with replicated data. If one server fails, others can still provide resolutions. Top-level root servers are particularly resilient, using multiple, geographically distributed servers to protect against failures.

You learned about DHCP in section A2.1.1.

Dynamic host configuration protocol (DHCP) server

The function of a DHCP server is to automatically assign IP addresses and other network configuration parameters to devices on a network, enabling them to communicate with other IP networks.

When a device connects to the network, it requests networking settings from the DHCP server. The DHCP server selects an available IP address from its pool and assigns it to the device, along with other necessary configuration information such as subnet mask, default gateway, and DNS server addresses. This process is known as obtaining a DHCP lease.

The DORA process

Discovery, offer, request and acknowledgment (DORA) is used by DHCP to assign IP addresses to devices on a network automatically.

- **Discovery:** The client sends a broadcast packet (DHCPDISCOVER) to locate available DHCP servers.
- **Offer:** DHCP servers respond to the client with an offer (DHCPOFFER) that contains an IP address the client can use.
- **Request:** The client responds to an offer by requesting (DHCPREQUEST) the IP address from one of the servers.
- **Acknowledgment:** The server confirms (DHCPACK) the IP address has been assigned to the client for a specific lease time.

DHCP servers are configured to manage a range of IP addresses and distribute them as needed to client devices. This automated management simplifies the process of connecting new devices to the network, ensures that each device has a unique IP address, and reduces the potential for configuration errors.

For example, when a new smartphone connects to a Wi-Fi network, it sends a DHCP discover message. The DHCP server responds with an offer that includes an IP address and other network settings. Once the smartphone accepts the offer, the server sends an acknowledgment, and the smartphone can then communicate on the network using the assigned IP address.

DHCP servers can be set up to assign addresses dynamically (choosing from a pool of available addresses) or statically (assigning a specific address to a device based on its MAC address). This flexibility allows network administrators to ensure that certain devices always receive the same IP address, which can be important for printers, servers, or other network resources that need to have a consistent address.

Large networks often use multiple DHCP servers to ensure reliability and load balancing. If one server fails, another can take over, providing continuous network configuration services to client devices. This redundancy is crucial for maintaining network connectivity and performance.

File server

A file server is a dedicated server used to store and manage data files and directories in a networked environment, allowing multiple users and devices to access and share files. This central repository for storing documents, images, videos, and other data types facilitates collaboration and data sharing among network users while maintaining centralized data management and security.

When a user or a device needs to access a file stored on the file server, they send a request over the network to the server. The file server, upon authenticating the user's access rights, grants or denies access to the requested file or folder. If access is granted, the user can then read, edit, or save changes to the file directly on the server, depending on their permissions. This centralized access control helps in managing and protecting the data more effectively.

File servers are configured with large storage capacities and robust file management software to handle simultaneous access requests from multiple users. This setup ensures that files are available whenever needed and can be efficiently located and retrieved. For example, a file server might host a company's shared documents, making the documents accessible to employees from their computers or mobile devices without the need to physically transfer files using external storage devices.

To enhance performance and reliability, file servers often employ redundancy and backup solutions, such as redundant array of independent disks (RAID) configurations and regular backup schedules. This ensures data integrity and availability even in the event of hardware failures or data corruption.

File servers can be optimized for different scales of operations, from small office environments to large enterprises, by adjusting storage capacities, network configurations and access protocols (such as SMB for Windows environments and NFS for Unix/Linux environments). The choice of file server setup and configuration depends on the organization's size, the volume of data handled, and specific security, performance and accessibility requirements.

Mail server

A mail server is a specialized server that handles the sending, receiving and storing of email for users within a network. It acts as a digital post office, facilitating communication between users by transferring email messages from one account to another over the internet or within an organization's internal network. Mail servers can take the role of mail delivery agent (MDA) or mail transfer agent (MTA) or both.

Mail servers operate based on standardized email protocols such as simple mail transfer protocol (SMTP) for sending emails, post office protocol version 3 (POP3) or internet message access protocol (IMAP) for receiving emails. A mail server ensures that emails reach their intended recipients and that users can retrieve their emails from the server.

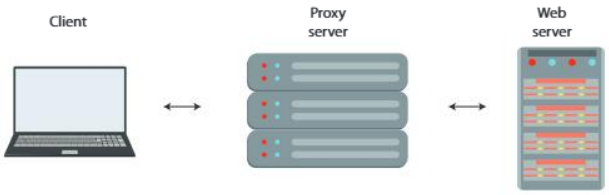
When a user sends an email, the email client on their device connects to the SMTP server. The SMTP component of the mail server processes the outgoing email, determining where to send the message based on the recipient's email address, and then forwards the email to the recipient's mail server. If the recipient is within the same mail server, the email is directly routed to the appropriate mailbox.

For incoming emails, the mail server uses either POP3 or IMAP. POP3 allows email clients to download emails from the server to the client's device and then typically deletes the email from the server. In contrast, IMAP synchronizes the email among multiple devices while keeping the emails stored on the server, allowing users to access their mail from any device with internet access.

Mail servers are equipped with various security measures, such as spam filters and antivirus programs, to protect users from malicious emails and to manage unwanted spam emails effectively. These measures help to ensure that only legitimate emails reach user inboxes. In more recent times, domain keys identified mail (DKIM), sender policy framework (SPF) and domain-based message authentication, reporting and conformance (DMARC) policies are required for email to be accepted without being flagged as spam.

Proxy server

A proxy server acts as an intermediary between a user's device and the internet, providing various functionalities such as enhancing security, improving performance, and ensuring privacy. When a user requests a web page or any online resource, the request is sent to the proxy server first. The proxy server then evaluates the request according to its filtering rules—which may involve content filtering, user authentication, or another policy enforcement—before sending the request to the internet. Upon retrieving the requested resource, the proxy server relays it back to the user, optionally caching the content for faster access in future requests.



▲ **Figure 19** A simplified version of a proxy server

Proxy servers can perform several key roles:

- By masking the user's real IP address, a proxy server enhances privacy and security. It can also prevent direct attacks on an internal network and enforce access controls to sensitive websites or services.
- In an organizational or educational setting, proxy servers are used to block access to specific websites according to content policies, such as sites that are not safe for work or do not comply with institutional guidelines.
- Proxy servers can store copies of frequently accessed web resources. This reduces bandwidth usage, speeds up access for users, and reduces latency, since the information can be served from the proxy server's local cache instead of retrieving it from the original source each time.
- They can monitor and log all web requests, allowing organizations to keep track of internet usage and detect potentially harmful or unauthorized activity.

For example, a company may employ a proxy server to control and monitor its employees' internet access. When an employee accesses the internet, their request goes to the proxy server, which checks the request against the company's internet usage policies. If the request is for a website that is not blocked, the proxy retrieves the content, returns it to the employee's browser, and logs the activity for future review. If the website is blocked, the proxy will deny the request, possibly returning a message explaining the restriction.

Additionally, public proxy servers provide a means for users to conceal their IP addresses and geographical location, offering a degree of anonymity when browsing the internet. This is particularly useful for accessing geo-restricted content or for users in countries with strict internet censorship policies.

Web server

A web server is a software and hardware combination that stores, processes and delivers web pages to clients upon request. When a user enters a URL in their browser or clicks a link, the browser sends a request to the web server hosting the site. The server then processes this request and sends back the requested web page, typically using HTTP protocol, which the browser then renders for the user to view.

Web servers operate using HTTP and its secure version, HTTPS, which define how messages are formatted and transmitted, and what actions web browsers and servers should take in response to various commands.

Common web servers include Apache, Nginx and IIS.

How web servers work

1. Web servers host the files that make up websites, including HTML files, style sheets (CSS), JavaScript files and multimedia content. This content is stored in a manner that makes it easily retrievable and deliverable upon request.
2. When a web server receives an HTTP request from a client (such as a web browser), it interprets the request, locates the requested files, and returns them to the client. If the requested content involves dynamic pages, the server may need to execute server-side scripts (for example, PHP, Python or Node.js) to generate the content before sending it.
3. The server delivers the requested content back to the client over the internet, ensuring that the content is formatted and encoded in a way that the client's browser can interpret and display.
4. Web servers implement security measures, including SSL/TLS encryption for HTTPS, to ensure secure data transmission. They also manage access controls and authentication to protect sensitive content.

A web server example

Imagine a small business, "Bekka's Bakery", has a website to showcase its products and allow customers to place orders online. The website is hosted on a web server. When a customer wants to view the bakery's menu, they enter the website's URL into their browser.

- The browser sends a request to the web server hosting the "Bekka's Bakery" website.
- The web server receives the request and locates the specific HTML file for the menu page.
- If the menu page content is static, the server immediately sends this HTML file back to the browser. If the content is dynamic, the server might first run some server-side scripts to fetch the latest menu items from a database.
- The server then sends the generated HTML content over the internet to the customer's browser, encrypted with HTTPS if supported.
- The customer's browser receives the HTML file and renders the menu page for the customer to view.

ATL Research skills

1. Working individually, research one of these topics.
    a. Analyse the importance of DCHP servers in managing network configurations. Consider how DHCP servers assign IP addresses and manage essential configuration details for network devices. What challenges arise in terms of scalability and security, when managing large networks? How can they be addressed?
    b. Evaluate the role of file servers in organisational settings, focusing on their scalability, reliability, and security measures. Consider how these servers handle large volumes of data, ensure data availability and integrity, and protect against unauthorized access.
    c. Discuss the function and security considerations of proxy servers. How do proxy servers enhance network performance and user security? Think about their role in forwarding client requests to other servers and discuss how they can cache data to improve performance and filter requests to enhance security.

Use your research to make a 5 minute presentation—in person or as a video presentation—to your class or working group.

2. Before you start your research, make a schedule for yourself, including both the research and presentation preparation stages. Break the project into smaller tasks. Remember to build in time to evaluate your research, to make sure you stay focused on the question.
3. Once you have finished your research, decide the key points to include in your presentation. Think about the presentation from the audience's perspective, too: make it succinct and interesting. Consider what questions people may ask you and be prepared to answer them.
4. Present your findings to the class.
5. Reflect on your research and presentation: What did you do well? What have you learned about your research and presentation abilities? What would you do differently if you did it again?

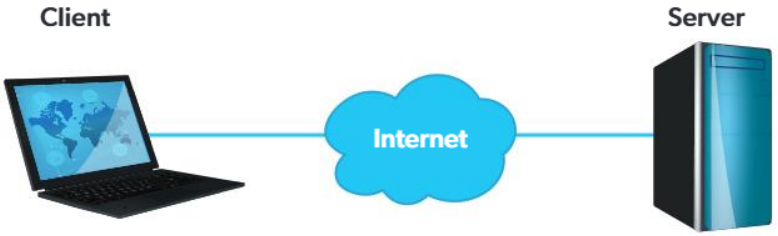
Practice questions

30. State the primary function of a domain name system (DNS) server. [1 mark]
31. Outline the role of a dynamic host configuration protocol (DHCP) server. [2 marks]
32. Describe two functions of a mail server. [4 marks]
33. Describe the benefits and potential drawbacks of using a proxy server in an organizational setting. [4 marks]
34. Describe the role of redundancy in the design of DNS servers. [4 marks]

A2.2.3 Compare and contrast networking models

Client–server model

In the client–server model, a server hosts, delivers, and manages most of the resources and services, which are requested by the client. The client–server model is characterized by request/response, where the client requests and the server responds. This model is centralized, with servers acting as the centralized points of access.



▲ **Figure 20** Client–server communication

Table 4 Advantages and disadvantages of the client–server model

	Advantages	Disadvantages
Control	Central management makes it easier to implement and enforce policies, manage resources, and ensure data consistency.	If adequate resources are not devoted to effective management of the server, it will not operate optimally. Put another way, if you do not take care of the server, it will not take care of you.
Scalability	Although dependent on the server's capacity, it is feasible to scale up by upgrading server resources or adding more servers.	Scaling up often requires significant investment in server capacity and infrastructure.
Security	Centralized security measures can be more straightforward to administer, offering potentially better protection against unauthorised access.	Centralized servers are lucrative targets for cyberattacks, which can compromise data integrity and availability.
Efficiency	Servers can be optimized for specific tasks, improving the efficiency of data processing and resource allocation.	High client demand can overload servers, leading to bottlenecks and reduced performance.
Reliability	With professional maintenance and robust infrastructure, servers can offer high reliability and uptime for clients.	The server is a single point of failure. If the server goes down, the services become unavailable to all clients.

Real-world applications

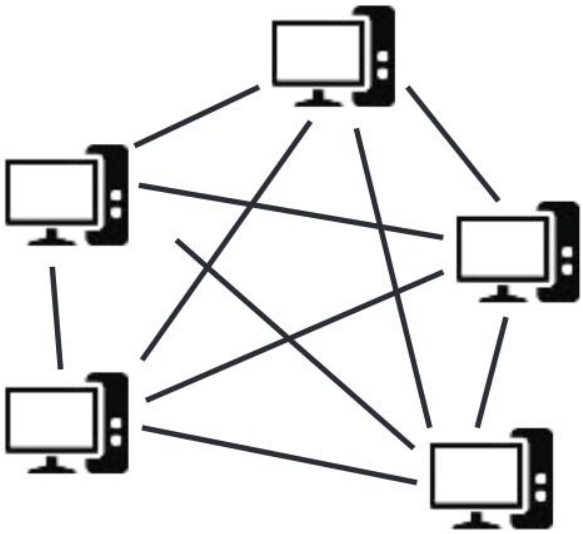
Accessing a website using a browser (client) that requests data from a server hosting the website (server) is a real-world application of the client–server model. The benefits might be centralized management and updating of website content. Enhanced security measures can be centrally applied, and you do not need to worry about multiple versions of software. However, if the server goes down, the website becomes inaccessible. Scalability might require significant resources for traffic management.

Using an email client such as Microsoft Outlook to send and receive messages through a central server such as Microsoft Exchange offers benefits such as centralized control over security protocols and data storage, and easier backup and archiving. However, a single point of failure can compromise access for all users, and there are potentially higher costs for server maintenance and scalability.

A common task is accessing your bank account through a banking app or website where transactions are processed on central servers. This method of access allows for enhanced security protocols, and centralized transaction processing ensures consistency and reliability. Of course, a bank might be a tempting target for cyberattacks, and any downtime would affect all users simultaneously.

Peer-to-peer (P2P) model

The peer-to-peer model is a decentralized network model where each participant (peer) has equal capabilities and responsibilities. Unlike the client–server model, there is no dedicated server. Instead, each peer can act as both a client and a server, sharing resources and services directly with other peers. This model is utilized in file-sharing networks, blockchain technologies, and certain messaging systems.



▲ **Figure 21** A peer-to-peer network is one where each node (peer) is connected to every other node

Table 5 Advantages and disadvantages of the peer-to-peer model

	Advantages	**Disadvantages**
Control	No central authority, enhancing resilience and eliminating single points of failure. Decentralized. Control is distributed among peers.	Without a central authority, problems can be difficult to track down, and usage metrics are difficult to gather.
Scalability	Adding more peers increases the network's resource pool and computational power with minimal cost.	Network performance can vary significantly, depending on the number and capacity of peers.
Security	P2P models are anonymous. This can be seen as a benefit…	…or a drawback, as threats are harder to identify and address. That being said, lack of accountability can make P2P networks susceptible to problematic software (illegal, malware, unethical) being distributed.
Efficiency	Efficient distribution of resources among peers can reduce the load on individual nodes.	Inefficient resource usage can occur if not properly managed, leading to redundancy and excessive overhead.
Reliability	The network can continue operating even if some peers fail, enhancing overall reliability.	Reliability may be compromised by the transient nature of peers, leading to potential data availability issues.

Real-world applications

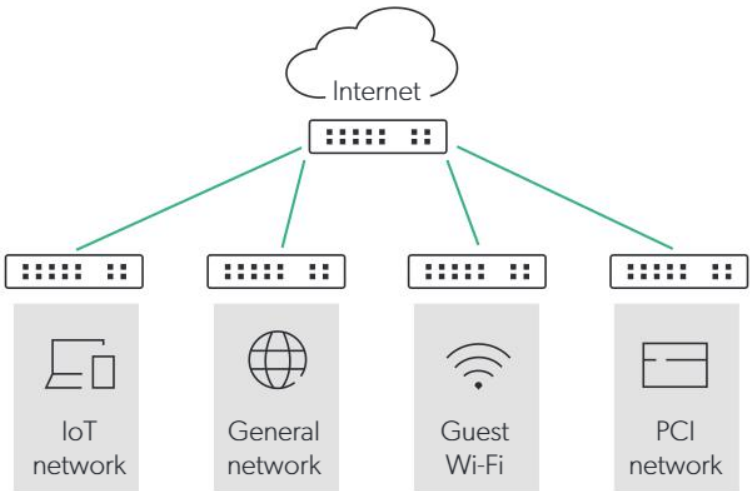
Using BitTorrent is a method to download or upload files where files are shared directly between users' computers without a central server. A major benefit is that it reduces the need for powerful central servers, and it can be faster for sharing large files, as multiple peers can transfer data segments simultaneously. Because it is a P2P model, there is less control over data security and the speed of the network is dependent on the number of peers (availability may vary).

Skype uses a P2P model for establishing direct connections between users for voice calls. Benefits include scalability, as the system can handle more users without needing much infrastructure investment, and potentially better call quality due to direct routing. Drawbacks include reliability, which can be an issue if peers have unstable connections. As with any P2P model, there are challenges in implementing centralized data policies and security measures.

Bitcoin and other cryptocurrencies use blockchain technology, operating on a P2P network where each node holds a copy of the transaction ledger. Benefits include decentralization, which enhances security and reduces the risk of fraud, and there is no single point of failure. Drawbacks include significant energy consumption for consensus mechanisms in some blockchains, and transactions can be slower compared to centralized models.

A2.2.4 Explain the concepts and applications of network segmentation

Network segmentation refers to the division of a computer network into smaller, distinct segments or subnetworks. This approach serves several purposes, primarily enhancing performance and security. By breaking down a larger network into manageable parts, network segmentation allows for more controlled access, reduced congestion, and tailored security policies.



▲ **Figure 22** An illustration of network segmentation

Network segmentation strategically divides a computer network into smaller, distinct parts or subnetworks. Security is improved on segmented networks because only allowed traffic can access network resources within the same segment. Network traffic cannot cross different segments. For example, if you worked in a bank, you would want to separate the wireless network for guests from the network with customer accounts. A device within the guest segment could not cross into the customer account segment.

Segmentation also introduces better performance in networks. For example, if your home network has two teenage users, each of whom have 37 different games, videos and music streams playing at the same time, your network is going to be congested, with poor performance for everyone (including the parents, who might want to watch a video). But if we segment this network and create a "teenage" segment and a "parent" segment, then the heavy teenage traffic will be isolated from the parent traffic. Segmentation confines network traffic to specific logical areas, reducing the load on the network's overall infrastructure.

Dividing a network into segments reduces congestion by distributing traffic more evenly across the network. This distribution helps avoid overloading any single pathway, ensuring a smoother flow of data. Network administrators can identify and upgrade segments experiencing high traffic volumes without overhauling the entire network, leading to cost-effective performance enhancements. For example, if a network was not segmented and 50 students were all streaming high definition video at the same time, they could negatively impact the entire network. However, if those students were on a segmented network, they would not impact other segments of the network (but their segment might be very slow).

Segmentation allows for more manageable network growth. New segments can be added as needed without disrupting existing configurations, thus facilitating scalable and controlled expansion. Administrators can monitor and manage traffic more effectively within segments. This improved oversight enables quicker identification and resolution of issues, optimizing network health and performance.

In general, you should think about segmentation as if you are isolating parts of the same network. Segmentation is a very powerful tool for performance and security, reducing congestion and efficiently managing network resources.

Methods of network segmentation

Network segmentation involves dividing a network into smaller, distinct segments. This division can be achieved through various methods, including subnetting and the use of virtual local area networks (VLANs).

Subnetting

Subnetting is a technique used to divide a larger IP network into smaller, logically segmented networks called subnets. This process involves manipulating the subnet mask, which dictates how the IP address space is divided.

The **host address** is the actual IP address assigned to a device on the network. These addresses must be within the range of the network address and broadcast address, excluding both. As you learned in section A2.1.4, the subnet mask distinguishes between the **network portion** of an IP address and the **host (device) portion**.

Key term

Host address The actual IP address assigned to a device on the network.

Devices with the same network portion (as determined by the subnet mask) are considered to be on the same local network and can communicate directly. Devices with different network portions need a router to facilitate communication.

Review binary numbers and converting between them, decimal and hexadecimal numbers in section A1.2.1.

The number 255 is significant in subnetting because it represents a full set of bits in an 8-bit binary number, where each bit is set to 1. In binary, 255 is represented as 11111111.

To identify the network address, you perform a bitwise AND operation between the IP address and the subnet mask.

The bitwise AND operation compares each bit of the IP address to the corresponding bit of the subnet mask.

- If both bits are 1, the resulting bit is 1.
- If either bit is 0, the resulting bit is 0.

Worked example 2

Apply the subnet mask 255.255.255.0 to the IP address 192.168.1.10.

Solution

Step 1: Convert to binary

IP address: 192.168.1.10

Binary: 11000000.10101000.00000001.00001010

Subnet mask: 255.255.255.0

Binary: 11111111.11111111.11111111.00000000

Step 2: Apply bitwise AND operation

```
IP address:     11000000.10101000.00000001.00001010
Subnet mask:    11111111.11111111.11111111.00000000
-------------------------------------------------
Result:         11000000.10101000.00000001.00000000
```

Step 3: Convert result back to decimal

Result in binary: 11000000.10101000.00000001.00000000

Convert to decimal: 192.168.1.0

Conclusion: the network address is 192.168.1.0.

Explanation

IP address: 192.168.1.10 (11000000.10101000.00000001.00001010)

Subnet mask: 255.255.255.0 (11111111.11111111.11111111.00000000)

Network address: The result of the bitwise AND operation (192.168.1.0) shows which network the IP address belongs to.

Worked example 3

Device A has an IP address 192.168.72.5 and a subnet mask of 255.255.255.0

1. Explain the relevance of the IP address and the subnet mask. What part of the IP address refers to the network? Which part refers to the host?
2. Device B has IP address 192.168.70.5. Device C has IP address 192.168.72.100.
   a. For each device, identify the network portion and host portion of the IP, and state if it is on the same subnet as Device A or not.
   b. Which of these devices can communicate directly with Device A?
   c. Which of these devices needs a router to communicate with Device A?

Solution

A device with an IP address of 192.168.72.100 would be able to directly communicate with Device A, but a device with an IP address of 192.168.70.5 would need to use a router in order to reach device A.

1. The device is identified by its IP address. The subnet mask 255.255.255.0 tells you about the IP address: the first three octets (192.168.72) represent the network portion of the IP address and the last octet (5) represents the host portion.
2. a.
   - Device B: 192.168.70.5
   - Network portion: 192.168.70
   - Host portion: 5

   Device B is not on the same subnet as device A.

   - Device C: 192.168.72.100
   - Network portion: 192.168.72
   - Host portion: 100

   Device C is on the same subnet as device A.

   b. Device C can communicate directly with Device A. Devices on the same subnet can communicate directly without needing a router. Any device with an IP address in the range of 192.168.72.1 to 192.168.72.254 belongs to the same subnet as device A.

   c. Device B needs a router to communicate with Device A. Since they are not on the same network, they cannot communicate directly.

Key term

Access control list (ACL) A set of rules that specifies which users or system processes are granted access to objects, as well as what operations are allowed on given objects.

You will learn more about IP addressing in the next section.

Subnetting improves routing efficiency by reducing the size of routing tables, as routers only need to store routes to subnets rather than individual IP addresses. It allows for better management of IP address space, enabling organizations to allocate IP addresses more efficiently across different departments or geographic locations. Subnets can be used to isolate network segments, limiting broadcast traffic and improving security by controlling access between subnets through routing rules and **access control lists (ACLs)**.

An ACL is a set of rules that specifies which users or system processes are granted access to objects, as well as what operations are allowed on given objects. Each entry in an ACL specifies a subject and an operation.

Objects: These are resources such as files, directories or devices that require access control.

Subjects: These are typically users, groups or system processes that are attempting to access a resource.

Operations: These refer to actions such as read, write, execute or delete that the subject can perform on the object.

Virtual local area networks (VLANs)

VLANs enable the segmentation of a physical network into multiple logical networks. VLANs allow devices to be grouped together, even if they are not connected to the same network switch, and manage network traffic in a way that is independent of their physical location.

VLANs offer the flexibility to group devices according to functional, departmental, or application-specific needs rather than their physical location, enhancing network efficiency and security. By segmenting network traffic into VLANs, sensitive data can be kept separate from other network traffic, reducing the risk of data breaches. Inter-VLAN routing rules can further enhance security by controlling which VLANs can communicate with each other. VLANs limit broadcast domains, reducing unnecessary broadcast traffic on the network. This containment improves network performance by ensuring that broadcast traffic is only sent where it is needed.

Subnetting and VLANs often work together within network segmentation to optimise both the logical and physical aspects of the network.

VLANs allow for logical separation without the need for physical separation of devices. Subnetting can be used in conjunction with VLANs to map logical segments to specific IP address ranges, enhancing the organization and management of network resources.

By combining subnetting and VLANs, networks can achieve a high level of security and performance optimization. VLANs segment the network, while subnetting provides an additional layer of segmentation.

This problem in practice

A small company with 20 employees has a single switch connecting all devices, including computers, printers and servers. The company wants to segregate the network into two distinct sections, one for the accounting department and one for the sales department, to enhance security and reduce unnecessary network traffic between the two departments. The company has one switch and does not want to buy additional hardware.

The accounting department has 10 employees, and the sales department has 10 employees. The switch has 24 ports.

The network administrator assigns ports 1–10 to VLAN 10 (Accounting) and ports 11–20 to VLAN 20 (Sales). The remaining ports can be used for shared resources or further expansion. Accounting department devices are connected to ports 1–10 on the switch. Sales department devices are connected to ports 11–20.



▲ **Figure 23** A 24-port switch

An accounting department computer is plugged into port 3 on the switch, making it part of VLAN 10. A sales department computer is plugged into port 15, making it part of VLAN 20. The result is network segregation, where computers in VLAN 10 (Accounting) can communicate with each other but not with devices in VLAN 20 (Sales), and vice versa. This segregation reduces unnecessary broadcast traffic and enhances security. The network administrator can apply policies—such as security or QoS (quality of service)—differently to each VLAN, according to the needs of each department.

Classless inter-domain routing (CIDR) notation

CIDR notation is a method for specifying IP addresses and their associated routing prefix. It replaces the older system based on classes A, B and C. CIDR notation provides a compact and flexible way to define IP networks and their sizes.

The key points of CDR notation are as follows.

Format: CIDR notation combines an IP address with a suffix indicating the number of bits in the subnet mask. For example, in CIDR notation 192.168.1.0/24, 192.168.1.0 is the IP address, and /24 indicates that the first 24 bits are the network portion.

Subnet Mask: The number after the slash (/) represents the number of bits set to 1 in the subnet mask. For example, /24 corresponds to a subnet mask of 255.255.255.0. In binary, this is 11111111.11111111.11111111.00000000.

Flexibility: CIDR notation allows more efficient allocation of IP addresses by supporting variable-length subnet masking (VLSM). It enables more granular control over network sizes, avoiding wasteful allocation of IP address ranges.

Worked example 4

CIDR example

Imagine you are given the CIDR notation 192.168.100.0/22

Determine the:

1. subnet mask
2. network address
3. range of IP addresses in this subnet
4. broadcast address.

Solution

Step 1: Determine the subnet mask.

The /22 in the CIDR notation means the first 22 bits are set to 1 in the subnet mask.

Binary representation of the subnet mask for /22 is:

11111111.11111111.11111100.00000000

Convert each octet to decimal.

- 11111111 (binary) = 255 (decimal)
- 11111111 (binary) = 255 (decimal)
- 11111100 (binary) = 252 (decimal)
- 00000000 (binary) = 0 (decimal)

Subnet mask: 255.255.252.0

Step 2: Determine the network address.

Convert the IP address and subnet mask to binary and perform a bitwise AND operation.

IP address: 192.168.100.0

11000000.10101000.01100100.00000000

Subnet mask: 255.255.252.0

11111111.11111111.11111100.00000000

Perform bitwise AND calculation.

11000000.10101000.01100100.00000000

11111111.11111111.11111100.00000000

11000000.10101000.01100100.00000000

Result: 192.168.100.0

Step 3: Determine the range of IP addresses.

Calculate the number of hosts per subnet.

For a /22 subnet mask, the number of host bits is 32 – 22 = 10.

Number of hosts = 210 – 2 = 1024 – 2 = 1022 (subtracting 2 for network and broadcast addresses).

The range of IP addresses:

First IP address (Network address + 1): 192.168.100.1

Last IP address (Broadcast address – 1): 192.168.103.254

Step 4: Determine the broadcast address.

The broadcast address has all host bits set to 1:

- Network address: `11000000.10101000.01100100.00000000` (192.168.100.0)
- Host bits: `00000000.00000011.11111111` (0.3.255 in decimal)

  `-----------------------------------`
- Broadcast address: `11000000.10101000.01100111.11111111` (192.168.103.255)

Summary

CIDR notation: 192.168.100.0/22

Subnet mask: 255.255.252.0

Network address: 192.168.100.0

IP address range: 192.168.100.1 to 192.168.103.254

Broadcast address: 192.168.103.255

A2.3 Data transmissions

Syllabus understandings

A2.3.1 Describe different types of IP addressing

A2.3.2 Compare types of media for data transmission

A2.3.3 Explain how packet switching is used to send data across a network

AHL **A2.3.4** Explain how static routing and dynamic routing move data across local area networks

A2.3.1 Describe different types of IP addressing

The internet protocol (IP)

The internet protocol (IP) facilitates the routing and delivery of packets across various networks. Central to the internet protocol is an IP address. An IP address is a unique numerical address assigned to each device connected to a computer network that uses the internet protocol for communication.

The distinction between IPv4 and IPv6 addressing

There are two types of IP addresses, IPv4 and IPv6. IPv4 is a 32-bit addressing scheme, resulting in about 4.3 billion possible unique addresses. Addresses are represented with dotted decimal format, for example, 192.168.1.1. An address segment cannot be lower than 0 nor higher than 255. Many IPv4 addresses are reserved for special purposes (these will be discussed later in this section).

IPv6 is a 128-bit addressing scheme, resulting in approximately 340 undecillion (3.4×10^{38}) possible unique addresses. IPv6 addresses are represented as eight groups of four hexadecimal digits separated by colons (for example, 2001:0db8 :85a3:0000:0000:8a2e:0370:7334). Many IPv6 addresses are also reserved for special purposes.

Table 6 Distinction between IPv4 and IPv6

Feature	IPv4	IPv6
Address size	32-bit address space	128-bit address space
Address representation	Dotted-decimal notation (e.g., 192.168.1.1)	Hexadecimal notation with colons (e.g., 2001:0db8:85a3:0000:0000:8a2e:0370:7334)
Address availability	Approximately 4.3 billion unique addresses	About 340 undecillion unique addresses
Header format (The header is the beginning of a packet and has routing information.)	Relatively complex, with 14 fields	Simplified, with 8 fixed fields
IPsec support (IPsec is a protocol which facilitates encryption, authentication and data integrity)	Optional	Mandatory

The differences between public IP addresses and private IP addresses

Public IP addresses

Public IP addresses are assigned to devices that directly access the internet. These addresses are unique across the entire internet, meaning no two devices can have the same public IP address simultaneously. Managed and allocated by the Internet Assigned Numbers Authority (IANA) and distributed through regional internet registries to ensure uniqueness, public IP addresses are used by servers hosting websites, email servers, and any device that needs to be directly accessible from the internet. Devices with a public IP address can be accessed from anywhere on the internet, assuming the appropriate network and security configurations allow it.

Private IP addresses

Private IP addresses are used within a private network and are not directly exposed to the internet. These addresses allow multiple devices to communicate within the same network. They are defined by RFC 1918, which specifies specific IP address ranges for private use. These ranges are not routed on the internet, making them reusable by anyone in private networks.

Private addresses are commonly used in home networks, office networks and other LAN setups for devices such as computers, printers and smartphones. Devices with a private IP address cannot be accessed directly from the internet without a form of network address translation (NAT), which we will discuss later in this section.

Table 7 Common private IP address ranges

Private address range	Description
From 10.0.0.0 to 10.255.255.255	This range allows for a single large private network with 16,777,216 possible addresses (24-bit block). An example of a private IP address in this range could be 10.1.2.3.
From 172.16.0.0 to 172.31.255.255	This range is suitable for multiple mid-sized private networks, offering 1,048,576 possible addresses across 16 contiguous 20-bit blocks. An example from this range is 172.16.0.1.
From 192.168.0.0 to 192.168.255.255	This range is often used for smaller private networks, such as home Wi-Fi networks. It offers 65,536 possible addresses across 256 contiguous 16-bit blocks. A common example from this range is 192.168.1.1, frequently used as a default gateway address in home routers.

A special note about 127.0.0.1

The IPv4 address 127.0.0.1 is known as the loopback address in IPv4 networking. It is a special address that is used by a computer to direct network traffic back to itself. The entire range 127.0.0.0 to 127.255.255.255 is reserved for loopback purposes, but 127.0.0.1 is the most commonly used address in this range.

IPv6 uses 0000:0000:0000:0000:0000:0000:0000:0001 for this purpose. IPv6 has a convenient shorthand notation for this address, **::1**, where the double colon (::) represents a string of consecutive zeros. This shortened notation is the standard and preferred way of representing the IPv6 loopback address.

The differences between static IP addresses and dynamic IP addresses

Static IP addresses

A static IP address is permanently assigned to a device or server. Once a device is assigned a static IP address, the address does not change. Static addresses must be manually configured and managed, often by a network administrator or by the user, depending on the network's setup. Can you imagine how much work this would be if there were thousands of users in your organization? Static IP addresses are ideal for devices that need to maintain the same IP address, such as web servers, email servers, and other devices that require stable addressing. The advantages of static addressing include ensuring reliable and consistent communication for services that need to be continuously available and accessible.

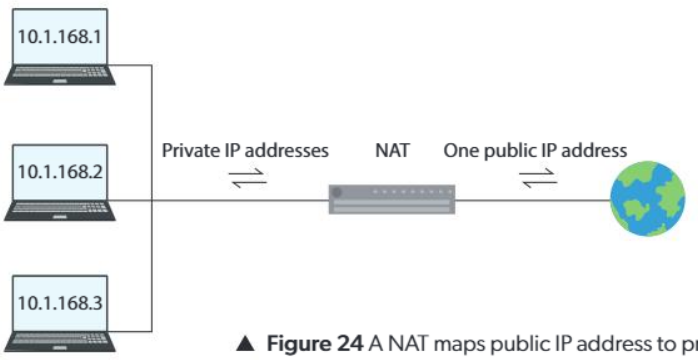
Dynamic IP addresses

Dynamic IP addresses are assigned to devices on a temporary basis. Each time the device connects to the network, it may be assigned a new IP address. Dynamic IP addresses are allocated by a DHCP server, which assigns IP addresses from a pool of available addresses. Dynamic addresses efficiently manage the limited pool of IP addresses by reassigning addresses to different devices as needed.

The role of network address translation (NAT)

There are not enough IPv4 addresses to assign every device a unique address. How would you solve this problem? We have an alternative in IPv6, but IPv6 sometimes requires new network hardware and has been slow to be adopted.

Network address translation (NAT) enables private IP addresses to map to a single public IP address (or a few addresses), drastically reducing the number of public IP addresses required for each device on the internet.



▲ **Figure 24** A NAT maps public IP address to private IP addresses

When a device within the private network initiates a connection to the internet, the NAT device (typically a router) replaces the device's private IP address in the packet header with the router's public IP address before sending it to the internet. It also modifies the source port number so that replies can be accurately directed back to the originating device.

For incoming traffic, the NAT device translates the public IP address and port number back to the corresponding private IP address and port number based on an internal NAT table that tracks active sessions. This ensures that responses from the internet reach the correct device on the private network.

You might have thousands of devices within a private network, but thanks to NAT they would all seem to be coming from a single IP address. This is a powerful tool in minimizing the number of public IP addresses used.

A2.3.2 Compare types of media for data transmission

Wired (fibre-optic and twisted pair) and wireless

Fibre-optic cables use pulses of light to transmit data at very high speeds over considerable distances. These cables contain glass or plastic fibres which transmit light.

Twisted pair cables use pulses of electricity to transmit data. They consist of pairs of insulated copper wires twisted together. Twisted pair cables are divided into two categories: unshielded twisted pair (UTP) and shielded twisted pair (STP). UTP is the most commonly-used cable in computer networking. STP includes additional shielding to further reduce interference, making it suitable for environments with high electromagnetic interference.

Wireless technology uses electromagnetic waves, such as radio frequencies, infrared and microwaves, to send signals. Common wireless technologies include Wi-Fi, Bluetooth, and cellular networks.

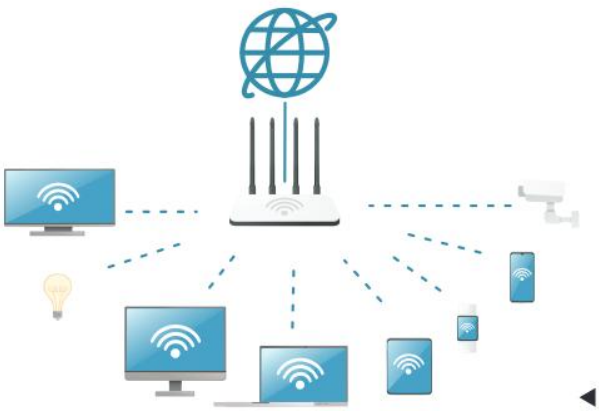
Factors to consider with data transmission include bandwidth, installation complexity, cost, range, interference susceptibility, attenuation, reliability and security.



▲ **Figure 25** Fibre-optic cables



▲ **Figure 26** Twisted pair cables



◀ **Figure 27** Wireless network

Table 8 Compare types of media for data transmission

	Fibre-optic cables	Twisted pair cables	Wireless
Bandwidth	Support extremely high data transmission rates, often reaching speeds of up to tens of gigabits per second.	Provide moderate bandwidth, sufficient for general office and home internet usage, but lower than fibre-optics. Bandwidth varies between categories, with Cat 6 and above supporting higher speeds.	Bandwidth is limited compared to wired options and can be affected by the number of connected devices, distance from the access point, and environmental factors.
Installation complexity	Installation can be complex, requiring specialized tools and skills for connecting and splicing.	Systems are well understood and can be installed and maintained by technicians without the need for highly specialized skills.	Installation involves setting up access points and configuring network settings, which can be simpler and less invasive than laying cables.
Cost	Installation cost of cables and associated equipment can be high compared to other media types.	Relatively inexpensive (especially UTP) and widely available.	Equipment costs can be moderate, but there are no costs for physical cables.
Range	Supports long-distance transmission with minimal signal loss, making it ideal for wide-area network connections.	Not ideal for long-distance transmission without the use of repeaters to boost the signal.	Effective communication range is limited. Although it can be extended with additional hardware, this adds to the cost and complexity.
Interference susceptibility	Immune to electromagnetic interference (EMI), allowing data to be transmitted more reliably in industrial or densely populated urban environment.	Still suffer from interference and signal degradation over long distances or in environments with high EMI.	Susceptible to interference from other wireless devices, physical barriers, and environmental factors, which can degrade performance.
Attenuation (reduction in signal strength as it travels through a medium)	Very low signal attenuation over long distances, maintaining signal integrity.	Higher attenuation, impacting signal strength and requiring the use of signal boosters for longer distances.	Signal strength decreases with distance and through physical barriers, leading to attenuation and potential data transmission issues.
Reliability	Very reliable, with low latency and minimal signal degradation. Not affected by weather conditions.	Generally reliable within their operational limits but can be impacted by EMI.	Reliability can vary, affected by interference, signal strength, and capacity issues in high-traffic environments.
Security	Intercepting data transmitted via fibre-optics is challenging, enhancing the security of the data transmission.	As with fibre-optics, intercepting data transmitted via twisted pair cables is challenging, making them reliable.	More vulnerable to unauthorized access and eavesdropping if not properly secured.

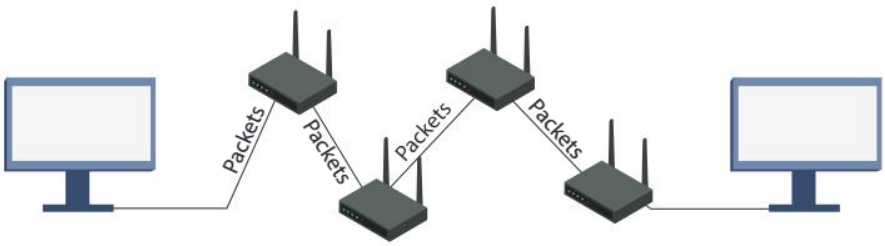
All of these factors can vary widely depending on specific conditions, technology standards, and deployment scenarios. Moreover, actual user experiences may vary significantly from these peak theoretical speeds due to factors like network congestion, distance from the access point or cell tower, physical obstructions, and the specific configurations of the network.

Selecting the appropriate media type for data transmission in a network requires careful consideration of various factors. These factors influence the network's performance, cost and overall effectiveness in meeting the demands of its users.

A2.3.3 Explain how packet switching is used to send data across a network

How exactly are messages passed from your device to another device on another network?

Packet switching is a method used to send data across a network by breaking data into small segments known as **packets**. How those packets are routed is the main focus of this section.



▲ **Figure 28** Which route should a packet take to efficiently arrive at its destination?

Data segmentation

As data is sent to a destination, the data is changed into a packet (or many packets, depending on the size of the data being sent). This process is known as **encapsulation**. Encapsulation is the process of adding or wrapping data with necessary protocol information before it is transmitted over the network.

When data is received by the destination, the data is rebuilt into the original request. This process is known as **decapsulation.** Decapsulation is the process of removing protocol information that was added to the data before it was transmitted over the network.

An example of the process of data segmentation into packets

Imagine you are playing a multiplayer game of Minecraft with a friend online, and you are building a structure together. Here is how packet switching works in this scenario.

1. Action initiation

You decide to place a block of stone in your Minecraft world. This action needs to be communicated to the game server so that your friend, who is also connected to the same server, sees the block appear in the game.



▲ **Figure 29** A character from the popular game Minecraft (online multiplayer mode) walking forward

2. Data segmentation

The action of placing a stone block is converted into data. This data includes information about the type of block (stone), the coordinates where it should be placed, and the player who is placing it. Instead of sending all this data in one big chunk, the game breaks it down into smaller pieces, known as packets.

3. Packet creation (encapsulation)

Each packet contains a portion of the data related to your action. For example, one packet might contain the coordinates of the block, while another packet contains the block type (stone). Additionally, each packet is given a routing header, which includes information like where the packet is coming from (your computer) and where it needs to go (the game server).

4. Adding control information (encapsulation)

The packets also contain control information to ensure they are delivered correctly. This might include sequence numbers so the server knows in what order to reassemble the packets, and error-checking codes to detect if any packets have been damaged during transmission.

5. Independent transmission

These packets are now sent from your computer to the game server. However, instead of travelling together as a single large piece of data, they are sent independently. Each packet can take a different route through the internet depending on the current network conditions.

6. Routing

As the packets travel through the network, routers along the way decide the best path for each packet based on network traffic and availability. For example, one packet might travel through a server in New York, while another might go through a server in Chicago. The routing decision is made dynamically for each packet.

7. Arrival at the server

Eventually, all the packets arrive at the game server. Since they might have taken different routes, they may arrive out of order or at slightly different times.

8. Reassembly at the server (decapsulation)

The game server uses the sequence numbers in the headers to reassemble the packets into the complete action that you originally performed (placing the stone block at specific coordinates). If any packets are missing or damaged, the server can request that those specific packets be resent.

9. Updating the game world

Once the server has reassembled the action, it updates the game world to reflect the stone block being placed. This update is then communicated to your friend's computer in a similar packet-switched manner, so they see the block appear in the game at the same location.

10. Final display

Your friend's game reassembles the packets it receives from the server, and the stone block appears in their game exactly where you placed it, ensuring that both of you see the same world and can continue building together seamlessly.

Bringing it all together

Remember:

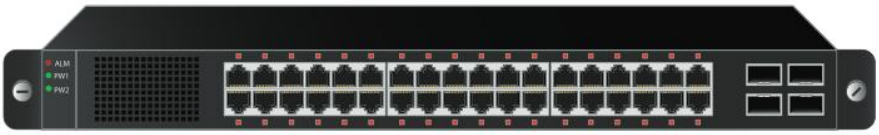
- data is segmented into packets; each packet includes payload (the actual data) and header information, which contains metadata such as the destination address, source address, sequence information and error detection codes
- once all the packets arrive at their destination, they are reassembled in the correct order to reconstruct the original message; this process relies on sequence numbers included in each packet's header.

Packet-switched networks send each packet independently. Packets may travel through different paths to reach the same destination, depending on network congestion, link failures and routing decisions. Packets are routed based on the network conditions and the routing table information available at each node (router). A routing table is a data file in routers that contains rules to determine the best possible path for forwarding a data packet to its destination.

This flexibility allows the network to adapt to changes, such as traffic load, ensuring optimal use of resources and maintaining data flow even if some parts of the network fail. When you draw a picture of the internet, you might draw it as a cloud because you cannot definitively know what path a packet will take.

The role of switches and routers

Switches manage how data packets are moved within a local network segment or a LAN. Switches forward data packets based on MAC addresses. They maintain a MAC address table to efficiently route packets to their correct destination within the LAN. By doing so, switches contribute to reducing network congestion and ensuring that packets reach their intended devices directly.



▲ **Figure 30** A 36-port switch

Routers are responsible for forwarding data packets based on IP addresses. They analyse the destination IP address contained in the packet's header, consult their routing table to determine the packet's next hop, and route the packet towards its destination across multiple networks. Routers enable communication between disparate networks, making them essential for the global interconnectivity of the internet. They assess multiple paths for packet transmission, choosing the most efficient route based on current network conditions, which may include considerations of speed, distance, traffic congestion, or other metrics.

AHL

A2.3.4 Explain how static routing and dynamic routing move data across local area networks

Static routing involves manually configuring routers with specific routes to network destinations. This process requires network administrators to explicitly define paths between devices across a LAN and beyond.

Dynamic routing enables routers to automatically discover and maintain routes to network destinations. Routers communicate using routing protocols to exchange information about network topology changes. This process allows network paths to be adjusted dynamically, based on current network conditions.

Static routing

Static routing involves a network administrator entering routing information directly into the router's configuration. This information includes the destination network address, subnet mask, and the next hop address or exit interface. Each router maintains a routing table that contains the statically configured routes along with any directly connected networks. This table is used to make forwarding decisions for incoming packets. When a packet arrives at a router, the router examines the packet's destination IP address and searches its routing table for a matching route. If a matching static route is found, the router forwards the packet to the specified next hop or exits the interface. If no match is found, the packet is dropped or sent to a default route if configured.

Static routing is ideal for small networks where routes do not change frequently. Routes are predefined, making network behaviour more predictable. Static routes do not require additional processing power or bandwidth to communicate route information, as there is no need for routers to exchange route information dynamically. Finally, by controlling exactly which routes are valid, an administrator can improve the network's security.

Static routing is not without disadvantages. These include the manual effort to maintain routing tables as a network grows. Static routes do not adapt to network changes, such as link failures, which can lead to increased downtime. In larger networks, configuring static routes becomes complex and error-prone.

Dynamic routing

Dynamic routing involves routing protocols (for example, RIP, OSPF, BGP) which define how routers communicate and exchange information. Routing protocols enable routers to discover network destinations and maintain up-to-date routing information. Routers send and receive routing updates to learn about the existence of each other and network destinations and importantly, the time to access neighbouring routers. Routing protocols use algorithms to determine the best path for packet forwarding based on various metrics (for example, hop count, bandwidth, latency). When there is a network change (for example, a link failure), the routing protocol automatically recalculates paths and updates the routing tables accordingly.

One of the main advantages of dynamic routing is routers can quickly adapt to changes in the network, such as link failures or additions by finding new paths. In addition, it is easier to manage in large and growing networks because routes

are updated automatically. Once dynamic routing protocols are configured, the network can adjust to changes without manual intervention. Many dynamic routing protocols are capable of selecting the most efficient route based on various metrics, enhancing network performance.

Among the disadvantages, configuring and maintaining dynamic routing protocols can be more complex than static routing. Dynamic routing requires processing power and bandwidth for routers to exchange route information. The time it takes for all routers to learn about network changes and reach a state of consistent knowledge can affect network performance (this time is called convergence). The exchange of routing information can introduce security risks if not properly secured.

Table 9 Comparison of static and dynamic routing

Factor	Static routing	Dynamic routing
Configuration	Manually configured by specifying exact routes.	Automatically configured through routing protocols.
Maintenance	Requires manual updates for any network change.	Automatically adapts to network changes with minimal manual intervention.
Complexity	Simple to configure for small networks but becomes cumbersome as the network grows.	Initially complex to set up but manages complexity well in large networks through automation.
Resource usage	Minimal: does not require additional processing power or memory for route discovery.	Higher: requires more processing power and memory for route discovery and maintenance.
Convergence	Not applicable, as routes are static and do not automatically adjust to network changes.	Dynamic protocols ensure the network can converge to a new understanding of topology after a change, which can vary in speed.
Scalability	Poor: becomes impractical and challenging to manage as the network size and complexity increase.	Good: scales well with network size and complexity due to automatic route adjustment.
Network size	Best suited for small to medium-sized networks where network changes are infrequent.	Better suited for medium-sized to large networks or networks requiring high availability and adaptability.

Practice questions

35. a. State an example of an IPv4 address. [1 mark]

    b. State an example of a full IPv6 address (not abbreviated or truncated). [1 mark]

36. Outline the difference between public and private IP addresses. [2 marks]

37. Outline the advantages and disadvantages of using static IP addresses versus dynamic IP addresses. [4 marks]

38. Describe the role of Network Address Translation (NAT). [2 marks]

39. List three advantages of using IPv6 instead of IPv4. [3 marks]

A2.4 Network security

Syllabus understandings

A2.4.1 Discuss the effectiveness of firewalls at protecting a network

AHL **A2.4.2** Describe common network vulnerabilities

AHL **A2.4.3** Describe common network countermeasures

AHL **A2.4.4** Describe the process of encryption and digital certificates

A2.4.1 Discuss the effectiveness of firewalls at protecting a network

A firewall is a network security device or software that monitors and filters incoming and outgoing network traffic at the packet level. Firewalls inspect packet headers to filter traffic based on IP addresses and ports. Modern firewalls (also known as next-generation firewalls or layer 7 firewalls) can inspect the content of packets (not just the headers) and filter traffic.

Firewalls inspect each packet of data which attempts to enter or exit a network, making decisions about each packet based on predetermined criteria.

A whitelist is a list of approved entities, such as IP addresses, domain names or applications, which are allowed to communicate with the network. Traffic which matches a whitelist entry is automatically permitted, ensuring that known safe sources or necessary business applications have unimpeded access.

A blacklist contains identifiers such as IP addresses, domain names or applications which are not approved. Blacklists block network traffic deemed harmful or unnecessary for business operations, helping to prevent attacks from known malicious sources.

Firewalls are configured with a set of rules which specify which types of traffic are allowed or blocked based on attributes such as source and destination IP addresses, port numbers, and protocols (TCP/UDP). These rules are prioritized and processed in sequence. The firewall examines each packet against these rules until a match is found, at which point the corresponding action (allow or block) is taken.

TOK

The term firewall originally referred to physical walls within buildings designed to prevent the spread of fire. Over time, it came to be used to refer to network security systems too.

How can using figurative language help in the acquisition of knowledge?

Strengths and limitations of firewalls

Table 10 Strengths of firewalls

Access control	Firewalls control access to network resources. By filtering incoming and outgoing traffic based on IP addresses, domain names, ports and protocols, they ensure that only authorized traffic can access the network, significantly reducing the risk of unauthorized access.
Traffic monitoring and logging	Firewalls monitor all network traffic, allowing for detailed logging and real-time alerts on suspicious activities. This capability is vital for identifying and responding to potential threats promptly.
Versatility and scalability	Modern firewalls are highly versatile, offering capabilities beyond simple packet filtering, such as VPN support, intrusion prevention systems (IPS) and deep packet inspection (DPI). They can be scaled to accommodate the growing network demands of an organization.
Application-level security	Application layer (or next-generation) firewalls can inspect the content of the data packets, providing security measures for specific applications. This allows for more granular control and protection against application-level attacks.

Table 11 Limitations of firewalls

Internal threats	Firewalls are less effective against threats that originate from within the network, such as from a malicious insider. Since firewalls primarily control ingress and egress traffic, their ability to mitigate internal risks is limited.
Sophisticated attacks	Advanced persistent threats (APTs) and some forms of malware can bypass firewall protections through encrypted traffic, zero-day vulnerabilities, or by masquerading as legitimate traffic, making it challenging for firewalls to detect and block such threats.
Configuration and management complexity	Properly configuring and managing firewalls requires a significant level of expertise. Misconfigured firewalls can introduce security vulnerabilities or block legitimate traffic, hindering network operations.
Performance impact	Inspecting and filtering traffic can introduce latency, especially if deep packet inspection or other resource-intensive processes are involved. This can impact the performance of critical applications and services.

Network address translation (NAT) to enhance network security

NAT operates by modifying network address information in the IP header of packets while they are in transit across a traffic routing device. From a security perspective, NAT provides several benefits.

Review section A2.3.1, which discusses NAT in depth.

One of the most notable security features of NAT is IP masquerading, where multiple devices on a private network use one public IP address for all external communications. This hides individual IP addresses of devices on the internal network from the external world, making it more difficult for attackers to directly target internal network devices.

By controlling which internal addresses are translated and allowed to communicate with the external network, NAT serves as a rudimentary form of access control. This can help prevent unauthorized access to certain parts of the network or restrict internet access to specific devices.

A2.4.2 Describe common network vulnerabilities

Table 12 Common network vulnerabilities

Vulnerability	Description and example
Distributed denial of service (DDoS)	Aims to overwhelm a website or online service with excessive traffic from multiple sources, rendering it inaccessible to legitimate users.
	Example: An e-commerce site is overwhelmed with traffic from thousands of infected computers, causing it to crash during a major sales event.
Insecure network protocols	Protocols that lack security measures, making them vulnerable to interception and exploitation.
	Example: Using Telnet or FTP (insecure protocols) for remote server access allows potential eavesdropping on credentials and data.
Malware	Malicious software designed to damage, disrupt, or gain unauthorized access to systems.
	Example: A ransomware attack encrypts files on the victim's computer, demanding payment for decryption keys.
Man-in-the-middle (MitM) attacks	The attacker secretly intercepts and possibly alters the communication between two parties who believe they are directly communicating with each other.
	Example: An attacker intercepts a Wi-Fi connection at a café to capture credit card information sent during an online purchase.
Phishing attacks	Fraudulent attempts to obtain sensitive information such as usernames, passwords and credit card details by disguising oneself as a trustworthy entity in electronic communications.
	Example: An email that appears to be from a bank asking recipients to confirm their account details on a fake website.
SQL injection	An attack that involves inserting malicious SQL statements into an input field for execution, to manipulate or exploit SQL databases.
	Example: Entering a specially crafted SQL command in a website's search box that causes the site to reveal sensitive user information.
Cross-site scripting (XSS)	A vulnerability in web applications that allows attackers to inject malicious scripts into content viewed by other users.
	Example: A comment on a blog that contains a script which, when viewed, steals cookies from other users.
Unpatched software	Software that has not been updated with the latest security patches, making it vulnerable to exploitation.
	Example: An operating system that has not been updated, allowing attackers to exploit known vulnerabilities.
Weak authentication	Security measures that are easy to bypass or crack, often due to simple or predictable passwords, or lack of multifactor authentication.
	Example: A website that only requires a username and a simple password, without requiring any additional form of verification.
Zero-day exploits	Attacks that take advantage of a previously unknown vulnerability in software or hardware, before the developer has released a patch or the public is aware of it.
	Example: An attacker discovers a flaw in a popular browser and develops code to exploit it, spreading malware before the issue is publicly known and patched.

A2.4.3 Describe common network countermeasures

Table 13 Common network countermeasures to mitigate network vulnerabilities

Policy	Description and example
Content security policies (CSPs)	A browser-side mechanism that helps prevent malicious code injection attacks, such as cross-site scripting (XSS). A CSP dictates from which domains and sources the browser can load scripts, stylesheets, images or other resources.
	Example: A website sets a CSP that only allows scripts to load from its own domain and a trusted content delivery network (CDN). This blocks attempts to inject malicious scripts from unknown sources.
Complex password policies	Rules enforcing the use of strong passwords (a mix of upper/lowercase letters, numbers, symbols, minimum length) that are harder to crack. These policies often include password expiration and preventing the reuse of recent passwords.
	Example: A company requires employees to use passwords that are at least 12 characters long and include a mix of uppercase, lowercase, numbers and symbols.
Encrypted protocols	Using protocols like HTTPS (the secure version of HTTP) that encrypt the communication between web browsers and servers. This prevents eavesdropping and man-in-the-middle (MitM) attacks, safeguarding sensitive data in transit.
	Example: You visit an online store with "https://" in the address bar, indicating that your credit card information will be encrypted during transmission.
Secure socket layer (SSL) certificate / transport layer security (TLS) certificate	Digital certificates installed on web servers to enable the use of HTTPS. They validate the server's identity to the client, ensuring you are connecting to the legitimate website.
	Example: There is a padlock icon in your browser's address bar when visiting your bank's website. This signifies the presence of a valid SSL/TLS certificate.
Update software	Regular patching of operating systems, applications, and firmware fixes known vulnerabilities. This reduces the "attack surface" that bad actors can exploit.
	Example: Your operating system prompts you to install a critical security update that patches a recently discovered vulnerability.
Distributed denial of service (DDoS) mitigation tools	Specialized services or software designed to detect and filter out massive amounts of malicious traffic in a DDoS attack. They help ensure that legitimate users can still access websites and services.
	Example: A large news website comes under a DDoS attack. Their mitigation tools detect the surge in traffic, filter out malicious requests and maintain website availability for legitimate users.
Input validation (filtering, whitelisting)	Techniques used in web applications and software to ensure that user-submitted data is in the correct format and does not contain malicious code. Filtering removes unwanted characters, while whitelisting only allows specific, pre-approved characters or patterns.
	Example: You try to create a username on a website with the code `alert('XSS')`. The input validation blocks this attempt, preventing a potential cross-site scripting attack.

Policy	Description and example
Email filtering solutions	Detect and block spam, phishing emails, and malware that frequently comes through email attachments. These solutions use advanced analysis to identify harmful patterns and known malicious senders.
	Example: An email filter blocks a phishing email disguised as an invoice from a familiar vendor because it detects suspicious language and a mismatched sender address.
Intrusion detection systems (IDS)	Network or host-based systems that monitor for suspicious activity or policy violations. They raise alerts when they detect potential attacks, but do not directly block the traffic.
	Example: An IDS on a company network detects unusual port scanning activity likely aimed at finding vulnerable systems.
Intrusion prevention systems (IPS)	Similar to IDS, these systems have the capacity to take proactive action to stop detected attacks, such as blocking malicious traffic.
	Example: An IPS notices four consecutive failed logins from the same IP address and blocks the IP address from access.
Multi-factor authentication (MFA)	A requirement for multiple forms of verification to access systems, reducing the risk of unauthorized access.
	Example: MFA might require a standard password plus a one-time password (OTP) so that users must enter a password along with a time-sensitive one-time code. The OTP may be generated by an app or sent via SMS. This increases the difficulty of compromising an account because it requires more than just a password.
Virtual private network (VPN)	Extends a private network across a public network, creating a secure and encrypted connection between devices and the private network. This encryption ensures that data transmitted over the VPN is protected from unauthorised access. Refer to section A2.1.1 for more detail.

Security testing and employee training

Regular security testing and employee training should not be seen as one-time events. Continuous updates and reinforcement are important in a rapidly shifting threat environment.

Tailored training is essential. Employees in different roles require specific training relevant to their job functions and data access.

Regular security tests, such as penetration tests and vulnerability scans, proactively search for weaknesses within your systems, software and networks. These tests mimic real-world attack methods to uncover exploitable flaws before cybercriminals do. The cybersecurity landscape is constantly changing as new attack techniques and vulnerabilities emerge. Frequent testing ensures you remain updated on your security posture and can adapt defences as needed.

Security testing not only reveals issues but also validates that your countermeasures (firewalls, IDS/IPS, and so on) are correctly configured and working as intended. Many industries, such as finance and healthcare, have regulations mandating regular security testing and audits. These tests provide evidence of your organization's commitment to security.

Employees are often the most vulnerable entry point for cyberattacks. Attackers use social engineering, phishing emails and other tactics to trick employees into giving up credentials or downloading malware. Effective training transforms employees from potential liabilities into your first line of defence.

They learn to recognize phishing attempts, handle sensitive data responsibly, spot unusual activity, and follow security best practices. Regular training reinforces security concepts and procedures, minimizing the risk of mistakes that could lead to security breaches. A security-aware workforce understands threats and actively reports suspicious activity to IT teams. They become partners in maintaining a robust security posture.

Security testing results highlight areas where employees require more training. For example, if a phishing test reveals a high rate of success, training can focus on phishing awareness and email security. Security teams gain valuable data on employee awareness from training quizzes and simulations. This data helps guide the allocation of security resources and future training plans.

Training and testing, when combined, establish a clear understanding of security expectations for employees. This encourages greater accountability and a shared responsibility for data protection.

Wireless security measures

Wireless security goes beyond just securing your Wi-Fi network name and password. While encryption (WPA2 or WPA3) is important, MAC filtering, whitelists and blacklists add another layer of defence for your wireless network.

Every network interface card (NIC) on a device, such as a laptop or phone, has a unique identifier called a MAC address. **MAC filtering** allows you to specify a list of authorized MAC addresses that can connect to your Wi-Fi network. It prevents unauthorized devices from connecting, even if they crack your Wi-Fi password. MAC addresses can be spoofed (faked) by determined attackers. Newer devices may have random MAC addresses for privacy reasons, requiring constant updates to the filter list.

A **whitelist** is a list of approved MAC addresses that are explicitly allowed to connect to your network. Any device not on the whitelist is denied access. A **blacklist**, on the other hand, is a list of specific MAC addresses that are explicitly denied access to your network, even if they know the password. Whitelisting offers a more secure approach as unknown devices are automatically blocked. Blacklists require knowing the MAC addresses of all authorized devices beforehand, which can be cumbersome for a dynamic environment.

MAC filtering, whitelists and blacklists are most effective when combined with strong WPA2/WPA3 encryption. Encryption scrambles the data transferred over your Wi-Fi network, making it unreadable even if someone manages to connect. A skilled attacker with the right tools might still be able to bypass them.

A2.4.4 Describe the process of encryption and digital certificates

Encryption is the process of converting data into a coded form to prevent unauthorized access. It involves using algorithms to transform plaintext into ciphertext, a scrambled version of the original data that can only be read if decrypted. Digital certificates, also known as public key certificates or identity certificates, are electronic documents used to prove the ownership of a public key.

Symmetric and asymmetric cryptography

Symmetric cryptography is a method of encryption where the same key is used for both encryption and decryption of data. This key, often referred to as a secret key, must be shared between the communicating parties in a secure manner. Symmetric encryption algorithms are typically fast and efficient, making them suitable for encrypting large volumes of data. However, the requirement for key exchange over a secure channel can be challenging in terms of key management, especially in systems with a large number of users.

Imagine Alice and Bob want to communicate securely. They decide to use symmetric encryption with a shared secret key. Alice chooses a secret key and securely shares it with Bob in person.

1. Alice writes a message: "Meet me at noon."
2. She encrypts this message using the secret key and the symmetric encryption algorithm, resulting in ciphertext (a scrambled message).
3. Alice sends the ciphertext to Bob over an insecure channel (such as the internet).
4. Bob receives the ciphertext and uses the same secret key and algorithm to decrypt it, recovering Alice's original message: "Meet me at noon."

The same key used for encryption is also used for decryption, and it is important that the key is shared securely between Alice and Bob. If a nefarious hacker had the secret key, then the hacker could also read the message.

Asymmetric cryptography, also known as public-key cryptography, involves a pair of keys for each user: a public key and a private key. The public key is openly distributed and can be used by anyone to encrypt data intended for the owner of the pair. Conversely, the private key is kept secret by the owner and is used to decrypt data encrypted with the corresponding public key. Asymmetric cryptography also enables digital signatures, where a user can sign data with their private key, and others can verify the signature with the user's public key. This method solves the key distribution problem of symmetric cryptography, but it is generally slower due to the computational complexity of the algorithms used.

Imagine that Alice wants to send Bob a secure message, and they decide to use asymmetric encryption. Bob generates a pair of keys: a public key, which he shares with Alice, and a private key, which he keeps secret.

1. Alice writes a message: "Meet me at noon."
2. She encrypts her message using Bob's public key.
3. Alice sends the encrypted message to Bob.
4. Upon receiving it, Bob uses his private key to decrypt the message, reading Alice's original text: "Meet me at noon."

The message is encrypted with Bob's public key, which anyone can use to encrypt messages for Bob. However, only Bob can decrypt the message with his private key, ensuring the message's confidentiality. This method allows secure communication without needing to share a secret key in advance.

Activity

Search online for "Alice and Bob examples". Learn about the history of Alice and Bob, and how they are used in discussions about cryptographic systems and protocols.

Table 14 Key differences between symmetric and asymmetric cryptography

Feature	Symmetric cryptography	Asymmetric cryptography
Key types	Single key (secret key) used for both encryption and decryption.	Two keys: a public key for encryption and a private key for decryption.
Key distribution	Keys must be shared securely between parties, posing a challenge for large systems.	Public keys can be freely distributed, while private keys remain confidential, easing key distribution challenges.
Encryption speed	Generally faster due to simpler algorithms.	Slower because of the complex mathematical operations involved.
Use cases	Suitable for encrypting large volumes of data due to its efficiency.	Often used for secure key exchange, digital signatures, and situations where secure communication is required without prior key exchange.
Security	Depends on the secrecy of the key and secure key exchange methods.	Provides a high level of security, especially for online communications, without the need for secure key exchange.
Computational resources	Less demanding on computational resources, making it more efficient for devices with limited processing power.	More demanding due to the complexity of the algorithms, requiring more processing power for encryption and decryption.

The role of digital certificates

A digital certificate is an electronic document that uses a digital signature to bind a public key with an identity—information such as the name of a person or an organization, their address, and so on. The certificate can be used to verify that a public key belongs to an individual, organization, or device. Digital certificates are issued by trusted entities called certificate authorities (CAs), which verify the certificate holder's identity before issuing the certificate.

Imagine Alice owns a website and wants to secure her site to protect her users' information. To do this, she needs to implement HTTPS on her website, which requires a digital certificate. The underlying mechanism for this secure communication is the TLS protocol, or its predecessor, SSL.

1. Alice generates a key pair: a public key and a private key.
2. She then submits a certificate signing request (CSR), which includes her public key and website information, to a CA.
3. The CA verifies Alice's identity and the authenticity of her website. Upon successful verification, the CA creates a digital certificate for Alice's website, signing it with the CA's private key.
4. Alice installs this digital certificate on her web server.
5. Now, when a user visits Alice's website, their browser checks the digital certificate. Since the certificate is signed by a trusted CA, the browser establishes a secure, encrypted connection using the public key in the certificate. This process is largely invisible to the user but ensures that the communication between the browser and Alice's website is secure.

HTTPS is a very common place to encounter digital certificates in use, but there are other use cases of digital certificates you should be aware of.

SSH is commonly used for secure remote login and other secure network services over an unsecured network. SSH traditionally uses password-based authentication, but it can also be configured to use SSH keys (a form of digital certificates) for authentication. In this setup, a user generates a pair of keys: a private key kept secret and a public key placed on the server. Authentication is performed by proving possession of the corresponding private key.

Digital certificates are also used for code signing, where developers sign their software or code with a digital signature. This signature verifies the software's source and ensures that it has not been tampered with since it was signed. When users download or run the signed software, their system checks the digital signature using the public key in the certificate. If the signature is valid, it indicates that the software is genuine and has not been altered.

VPNs often use digital certificates for client and server authentication. In a typical VPN setup using protocols such as IPsec or SSL/TLS, a server presents a digital certificate to the client and, optionally, the client can also present a certificate to the server. This mutual authentication process ensures that both parties are who they claim to be before establishing the encrypted VPN tunnel. Digital certificates in VPNs enhance security by adding a layer of authentication on top of encryption.

Digital certificates are also used for signing digital documents (PDFs, Word documents, and so on) to verify the signer's identity and ensure the document's integrity has not been compromised since it was signed. This application is essential in legal, financial and official communications where authenticity and non-repudiation are critical.

The use of public and private keys in asymmetric cryptography

In asymmetric cryptography, also known as public-key cryptography, two different but mathematically related keys are used: a public key and a private key.

The public key is used to encrypt data. When someone wants to send a secure message to the key owner, they encrypt the message using the recipient's public key. This ensures that only the recipient can decrypt the message, as only they possess the corresponding private key.

Here is a short summary of how a public key is used to encrypt data.

1. The sender must obtain the recipient's public key. This can be done through a public directory, a digital certificate, or directly from the recipient.
2. The sender writes the message they intend to send. If the message is lengthy or requires additional security measures, it might first be compressed or hashed.
3. The sender uses the recipient's public key and an asymmetric encryption algorithm to encrypt the message. Common algorithms include RSA, ECC (elliptic curve cryptography), and ElGamal, each with its own method for using the public key to encrypt data.

The private key is used to decrypt the data encrypted with the corresponding public key. In the context of secure communications, when a message is received which has been encrypted with the recipient's public key, the recipient uses their private key to decrypt it. This process ensures confidentiality, as only the intended recipient can access the decrypted information.

In digital signing, the private key is used to sign a document or a piece of data. The signer generates a hash of the document and then encrypts this hash with their private key, creating a digital signature. This signature is then attached to the document.

To verify a digital signature, the recipient (or any party wishing to verify the signature) uses the signer's public key to decrypt the attached signature, thus revealing the hash. The recipient then generates a new hash of the received document and compares it to the decrypted hash. If they match, it confirms that the document has not been altered since it was signed and that it was indeed signed by the holder of the corresponding private key, thus verifying the integrity and authenticity of the document.

ATL Research skills

Hash functions are algorithms that take an input (or "message") and return a fixed-size string of bytes, typically a digest that is unique to each unique input. They are designed to be one-way functions, making it computationally infeasible to reverse the process and derive the original input from the hash output. Hash functions are widely used in cryptography for ensuring data integrity, generating unique identifiers, and supporting secure password storage.

Imagine you have a message, "Hello, world!" and you apply a simple hash function to it, such as MD5.

Input: "Hello, world!"

After applying the MD5 hash function, you get the output (digest).

Output: "fc3ff98e8c6a0d3087d515c0473f8677"

This output is a unique representation of the input message. If you slightly alter the input, even just changing "Hello, world!" to "hello, world!" (note the lowercase "h"), the hash output will be dramatically different, demonstrating the sensitivity of hash functions to input changes.

Use web-based resources to find information about hash functions.

- What are the advantages and disadvantages of MD5?
- Research at least one other hash function and compare it with MD5. Explain when you would and would not use each function.

Encryption key management

An encryption key is a string of bits used by encryption algorithms to encrypt data. This key dictates the output of the encryption process, and the same key (in symmetric encryption) or a corresponding key (in asymmetric encryption) is required to reverse the transformation, making the original data readable again. If you have the key, you can open the lock.

Encryption key management refers to the processes and policies for handling, storing and protecting encryption keys.

Proper key management ensures that encryption keys are only accessible to authorized entities. Compromised keys can lead to unauthorized data access, negating the benefits of encryption. By securely managing keys, organizations can reduce the risk of data breaches. Even if data is intercepted, it remains protected if the keys are secure. Many regulatory standards, such as GDPR,

HIPAA and PCI DSS, mandate strict management and protection of encryption keys as part of their data protection requirements. Non-compliance can result in significant fines and legal consequences.

Effective key management ensures that keys are available when needed. Loss of an encryption key can result in the loss of access to critical data, potentially halting operations. Part of key management involves backing up keys securely, ensuring they can be recovered in the event of accidental deletion or loss, without compromising their security. As organizations grow and adopt new technologies, their key management strategies need to scale accordingly. Efficient key management solutions can adapt to increasing volumes of keys and evolving cryptographic standards.

Practice questions

40. Outline the strengths and limitations of using firewalls to protect a network. [4 marks]
41. Describe how network address translation (NAT) enhances network security. [4 marks]

AHL

42. Describe how a specific network countermeasure mitigates a common vulnerability. [3 marks]
43. Distinguish symmetric and asymmetric cryptography, emphasizing their differences and applications. [6 marks]

Linking questions

1. Do networks and databases use the same form of encryption algorithms (A3)?
2. How do cloud computing and distributed systems utilize networking to deliver services (A1.1.9)?
3. How do the concepts of binary and hexadecimal data structures relate to network communications (B2)?
4. Are similar ethical principles needed when transmitting data over a network and using data in machine learning algorithms (TOK)?
5. How can network types or transmissions impact database performance (A3)?
6. What are the similarities and differences between network security and database security (A3)?
7. How do network technologies influence machine learning algorithms (A4)?

End-of-topic questions

Topic review

1. Using your knowledge from this topic, A2, answer the guiding question as fully as possible:

   What are the principles and concepts that underpin how networks operate?

Exam-style questions

2. Describe the purpose and characteristics of a wide area network (WAN). [4 marks]
3. a. Outline two examples of the use of a personal area network (PAN). [4 marks]
   b. Describe the function of network devices in a PAN. [2 marks]
4. a. Describe the purpose and benefits of virtual private networks (VPNs). [3 marks]
   b. State one limitation of VPNs. [1 mark]

AHL

5. Describe the function of each layer within the TCP/IP model. [4 marks]
6. a. Describe the purpose of cloud computing. [3 marks]
   b. Describe the benefits of cloud computing. [3 marks]
   c. Describe the limitations of cloud computing. [3 marks]
7. a. Describe the concept of edge computing. [3 marks]
   b. Describe the benefits of edge computing. [3 marks]
   c. Outline one example of edge computing. [2 marks]
8. Describe the purpose distributed systems. [3 marks]
9. a. Describe the function of modern digital infrastructure. [4 marks]
   b. Outline two examples of networks in real-world use. [4 marks]
10. Describe different types of IP addressing. [3 marks]
11. Distinguish between IPv4 and IPv6. [4 marks]
12. Outline two types of media for data transmission. [4 marks]
13. Explain how packet switching is used to send data across a network, including the processes of encapsulation and decapsulation. [6 marks]
14. Describe the differences between static and dynamic IP addresses. [4 marks]

AHL

15. Describe how static routing and dynamic routing move data across local area networks. [4 marks]
16. Discuss the effectiveness of firewalls at protecting a network. [6 marks]

AHL

17. Outline two common network vulnerabilities. [4 marks]
18. Outline two common network countermeasures. [4 marks]
19. Describe the difference between symmetric and asymmetric cryptography. [4 marks]

AHL

20. Describe the importance of regular security testing. [3 marks]



A3

Databases

What are the principles, structures and operations that form the basis of database systems?

Databases are ordered data repositories. They are used to store data in a logical manner so that the data can then be analysed or used in queries. Data stored in databases are stored in entities (tables) and a computing language called SQL is used to query and manipulate the data. Many databases use relationships to understand the connections between the data being stored. Understanding the connections enables you to discover patterns and trends within the data. The operations that can be performed on the structure of the database help to store data in a highly organized way, leading to informative systems. Database management systems are often used to help those developing and using databases.

A3.1 Database fundamentals

Syllabus understandings

A3.1.1 Explain the features, benefits and limitations of a relational database

A3.1.1 Explain the features, benefits and limitations of a relational database

A database is a systematically stored collection of data. The data is stored in a logical manner, usually in several interconnected tables. You probably interact with databases everyday without even being aware of it. Here are some examples of databases that you may have had interactions with.

Retail management databases: If you use an online shop you are probably going to be interacting with a database as they are used to keep track of the products being sold, the customers buying the products, and the companies supplying the products.

Social media: If you use social media you are interacting with elements of a database. Your profile is part of the records they store about people, your posts are stored in records, and interactions you make across the platform are stored.

Medical records database: Medical records are stored in databases, and any interactions you have with a doctor or specialist are stored within a file, as well as information about the staff in the medical establishment and treatments they offer.

Data might be stored in a single **entity** (known as a flat file). The problem with storing data in a flat file database is that data is often repeated throughout the entity. This leads to data inconsistency because, if there are several similar versions of the same data, it is challenging to know which is the correct data. It also means that deleting data requires looking through masses of data in the flat file, trying to ensure that all traces of the data are removed.

Another problem could be the volume of data being stored. In the flat file model, lots of data is repeated, which uses up valuable storage space very quickly. Therefore, databases are usually split into several entities that have links (known as relationships) between them.

Key term

Entity A table within a database.

This problem in practice

You almost certainly interact with a database on a regular basis. One application of databases is to store information about users in online games.

For example, if you play a massively multiplayer online role-playing game (MMORPG) game, such as *Lost Ark*, a database will be used to track your achievements, your transactions, your in-game chats, your friend list and your inventory. When you save the game, your current status is updated in the database, ready to reload when you rejoin the game.

Owner ID	Owner Name	Address	Patient ID	Name	Type	Vet ID	Name	Address	Date	Treatment	Type	Cost
1029	Alison Bachm an	12 Green Lane 20192	1011	Oskar	Dog	4400	Rachel	345 Ridley St 99554	23/05/25	Worming	Tablet	50
1922	Aria Mathers	458 Rigistr 4993	1012	Seb	Cat	4100	Lucy	29 Entle Street. 3049	23/05/25	Broken Tail	Surgery	400
1029	Alison Bachm an	12 Green Lane 20192	3999	Jaques	Hamster	4400	Rachel	345 Ridley St 99554	23/05/25	Cut of Paw	Medication	50
2032	Theo Naidoo	45 Rue Martignac	2393	Kai	Dog	4400	Rachel	345 Ridley St 99554	23/05/25	Broken Leg	Surgery	450
2032	Theo Naidoo	45 Rue Martignac	2393	Kai	Dog	4400	Rachel	345 Ridley St 99554	17/08/25	Cast Removal	Surgery	200

▲ **Figure 1** An example representation of a flat file database of clients of a veterinary practice

Key term

Anomaly An issue that can occur because of incorrect handling of data in a database.

As you can see from the figure, a flat file database is very inefficient and has several **anomalies**.

Insert anomaly: You cannot add details for a vet without adding information about a patient and an owner, meaning that a vet has to treat a patient before being added to the database.

Deletion anomaly: You cannot delete data without all data in the record being deleted. For example, if you delete all the information about a vet then you lose all the information about the parent and owner.

Update anomaly: If an update is needed (such as a change of address) and you do not change all records, then you will have different data in the database and no longer know which is the correct version.

TOK

One of the biggest databases in the world is that of the Central Intelligence Agency (CIA) in the United States of America. The CIA also maintains the World Factbook. This section of their database is accessible to the public and free to use. The World Factbook stores facts about the history, people and geography of the world, such as:

- Cheomseongdae (which means "star-gazing tower" in Korean) is the oldest surviving astronomical observatory in Asia
- the world's largest mangrove forest is in the Sundarbans National Park in India, a park that is also a UNESCO World Heritage Site.

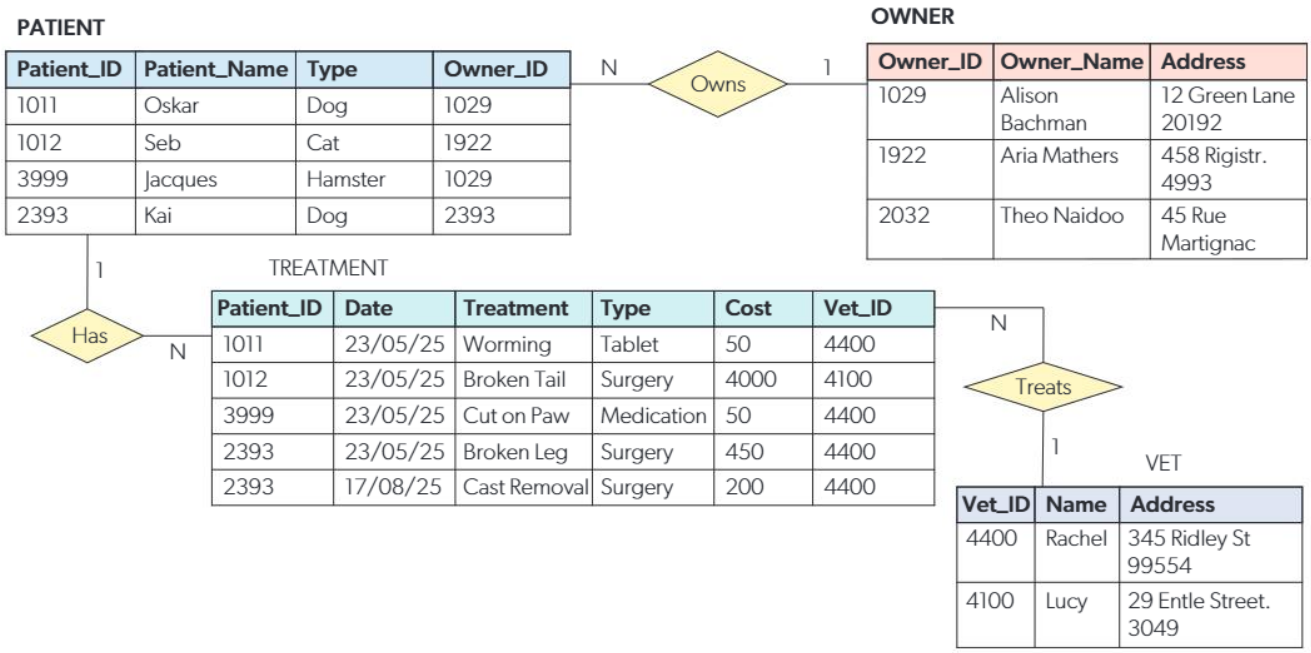
The World Factbook provides information to users about different countries around the world, while the full CIA database is used by the government of the United States to make decisions about political policies.

YouTube also utilizes a massive database. Every day, more than 5 billion videos are watched on YouTube, more than any other video hosting website. Information stored about videos includes the account publishing the video, the length, the style of video, and possibly linked content, enabling users to continuously watch videos they find interesting. YouTube is designed for entertainment, and in some cases to educate.

What responsibilities do the developers of a database have to ensure the data stored within the database is accurate?

Any calculations and queries require accurate data, and incorrect information will create incorrect results. Flat file databases are difficult to scale, as the more data you have, the more likely these problems are to occur. This is why we make use of relational databases.

Figure 2 shows the flat file database that has been **normalized** into a relational database.



▲ **Figure 2** A relational database

Features of a relational database

Entity: The relational database now consists of four entities each containing data. The patient entity stores information about the animal, the owner entity stores the information about the owner of the animal or animals, and vet information is stored in the vet entity. The final entity is the treatment entity, which stores information about the treatment each patient received as well as the date of the treatment.

Primary key: Primary keys are unique identifiers for an instance in an entity. For example, there may be more than one dog called Oskar in the patient entity. However, because of the primary key (the patient ID) we can tell which Oskar we are referring to.

Foreign key: A foreign key creates a relationship between the instances in the entity so we can link information together. For example, you would like to know the owner of each patient so you can update them on progress. To do this, the primary key of the owner from the owner entity is placed in the patient entity. This creates a link. The foreign key in the patient entity allows you to match the owner using the primary key.

Key terms

Normalization A process to organize data in a database to minimize the risk of anomalies.

Primary key A unique value for each row in a database.

Foreign key A primary key value that has been placed in another entity to create a relationship.

Owner_ID (PK)	Name	Address	Phone
393939	Andrea	29 Main St	333 4003
39291	Pia	12 River Lane	393 0102
5893021	Amalia	34 River Lane	392 1019

Owner primary key placed in pet table as a foreign key to create link.

Pet_ID (PK)	Name	Type	Owner_ID (FK)
H849	Paolo	Hamster	393939
C192	Felix	Cat	393939
C059	Bao	Cat	5893021

▲ **Figure 3** Relationship between tables using a primary key as a foreign key

Composite key: Sometimes there are no unique values in a record, so you have to link different values together to make them unique. This can be seen in the treatment entity. The patient ID cannot be used, as a patient may have many treatments. However, if patient ID and date are used they become unique, as a patient would not have more than one treatment in a day.

Concatenated key: Sometimes, when there are no unique values in a record, data from the **attributes** can be placed together to create a unique key. For example, in a table of people it may be a challenge to find a unique key. If you take a substring of a first name, surname, and date of birth you are more likely to get a unique value. This is known as a concatenated key.

Relationships: There must be relationships between the entities otherwise the data exists alone and is not very helpful.

Cardinality: Cardinality is the type of relationship between the entities. There are three you need to be aware of.

- **One-to-one:** An instance in one entity can only be related to one instance in another entity. In the example above, one patient can have one owner.
- **One-to-many:** An instance in one entity can be related to many instances in another entity. In the example above, one vet can perform many treatments.
- **Many-to-many:** Many instances in one entity can be related to many instances in another entity. A real-life example could be that many students are enrolled in many courses.

Modality: Modality is the necessity of the relationship between entities. There are two types of modality you need to be aware of.

- **Optional relationship:** There may be a relationship, but there does not have to be. For example, Actor (mandatory) and TV Show (optional). This relationship is optional because actors do not have to be in TV shows. Some actors are only on stage.
- **Mandatory relationship:** There has to be a relationship between the two entities. For example, Receipt (mandatory) and Items (mandatory). This relationship is mandatory because to have a receipt you have to have bought at least one item.

Key terms

Attribute An attribute represents a characteristic of the data.

Relationship The link between two entities.

Cardinality The type of relationship between two entities.

ATL Research skills

Understanding how the data relates to each other in a database often requires research to specify problems correctly and decompose the problem.

Think about a database you interact with regularly, such as an online shop, a social media site, or a school information system. Identify the data that is stored and investigate what the purpose of the database is. Can you suggest some entities that are being used?

Relationships enable links to be made between the data in a relational database.

Benefits of a relational database

Community support: There are multiple streams of support for users of relational databases.

Concurrency control: Relational databases can be used by concurrent users without error using ACID properties (this will be explained further in this topic).

Data consistency: Due to the lack of data repetition, data is more consistent.

Data integrity: Due to the lack of data repetition, data is more likely to be correct.

Data retrieval: Using SQL commands, data retrieval is easier.

Reduced data duplication: Due to the use of entities and relationship, there is less data duplication.

Reduced redundancy: As data is stored across entities using relationship links, there is less redundant data.

Reliable transaction processing: Relational databases have several checks that occur when transactions take place to ensure that the transactions are reliable.

Scalability: Relational databases are easier to scale than flat file databases.

Security: Relational databases have built-in security to support a multiuser environment.

Limitations of a relational database

Big data scalability: Relational database models are not the best model for storing data that will be mined to make complex decisions.

Design complexity: The different entities and models within the database can be complex and challenging to implement.

Hierarchical data handling: The nature of the database model means trees are stored in a hierarchical manner. This can lead to inefficient traversal of the tree, especially if there are deep hierarchical layers.

Rigid schema: The schemas of the database are rigid data types, and relationships must conform to very specific rules.

Object-relational impedance mismatch: Relational databases are useful for storing data in dedicated databases. However, if the data then needs to be used in an object-oriented space, for example, when dealing with big data, sometimes the models might not be fully compatible.

Unstructured data handling: Unstructured data that does not follow the rigid scheme of the database model can be difficult to place within the model.

Activity

Identify suitable cardinality and modality relationships for:

- teacher and student
- class and teacher
- zoo and animal
- person and clothes
- animals and children.

TOK

The World Data Center for Climate is one of the world's largest databases. It holds hundreds of terabytes of data regarding the world climate. The online database allows users to search the archives and upload data to the website. The search interface is very simple and the way the results are communicated is not user-friendly.

Think about the knowledge that is being communicated by this website (scientific data about the climate for the scientific community). Consider the different users of this database:

- non-specialist users
- climate scientists
- school students researching for a project.

For these three different types of user, consider the user's ability to access information, to understand it as it is presented, and whether the information available matches their needs.



▲ **Figure 4** Search interface for the World Data Center for Climate

What challenges are raised when communicating knowledge in the form of a database?

Practice questions

1. Describe two anomalies that may occur in a flat file database. [4 marks]
2. Explain how primary and foreign keys are used to create a relationship in a database. [3 marks]
3. State an example of a one-to-many relationship. [1 mark]
4. State an example of a mandatory relationship. [1 mark]
5. Identity two benefits of using a relational database. [4 marks]

A3.2 Database design

Syllabus understandings

A3.2.1 Describe database schema

A3.2.2 Construct ERDs

A3.2.3 Outline the different data types used in relational databases

A3.2.4 Construct tables for relational databases

A3.2.5 Explain the difference between normal forms

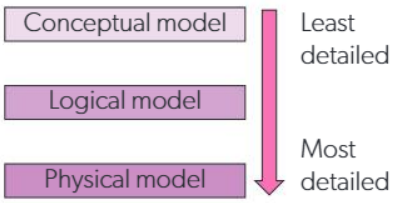
A3.2.6 Construct a database normalized to 3NF for a range of real-world scenarios

A3.2.7 Evaluate the need for denormalizing databases

A3.2.1 Describe database schema

A database **schema** is a blueprint of the database that identifies the different entities (tables), attributes (fields) and relationships (links) within the database, including the restraints on the data. The database schema is a model detailing how the database has been set up, but the schema does not contain any actual data. There are different models within the schema. These models describe the database for different users of the system including users, administrators and developers.

The three schemas that you need to be aware of in this course are the conceptual schema, the logical schema and the physical schema.



▲ **Figure 5** A diagram showing the hierarchical levels of the database schema

The conceptual schema

This model of the database identifies all the entities within the database and the relationship between these entities. This model does not contain specific details such as the attributes within the entities nor information about the restraints on the data, but does identify the types of relationships between the data. This level of the database schema is viewed by both the users and the administrators/developers. **Entity relationship diagrams (ERDs)** are used in the conceptual model to show the relationships.

Examples of relationships

- One player plays for one team.
- One team has many players.
- One team is managed by one manager.

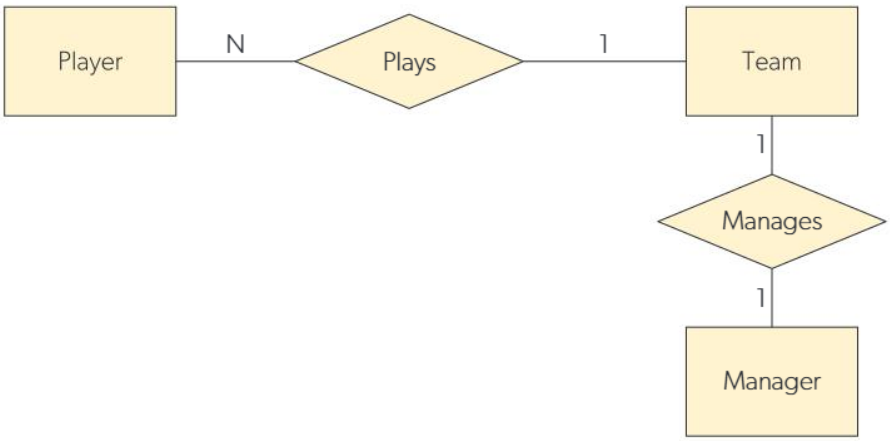
This is shown in Figure 6.

Key terms

Schema A plan showing the overall design of the database.

Entity relationship diagram (ERD) A universally understood diagram showing the database schema.

ERD diagrams are similar to constructing the design of classes in object-oriented programming (OOP), covered in section B3.1.2. Just like in OOP, you need to have a clear model identifying attributes and behaviours that can be understood by others.

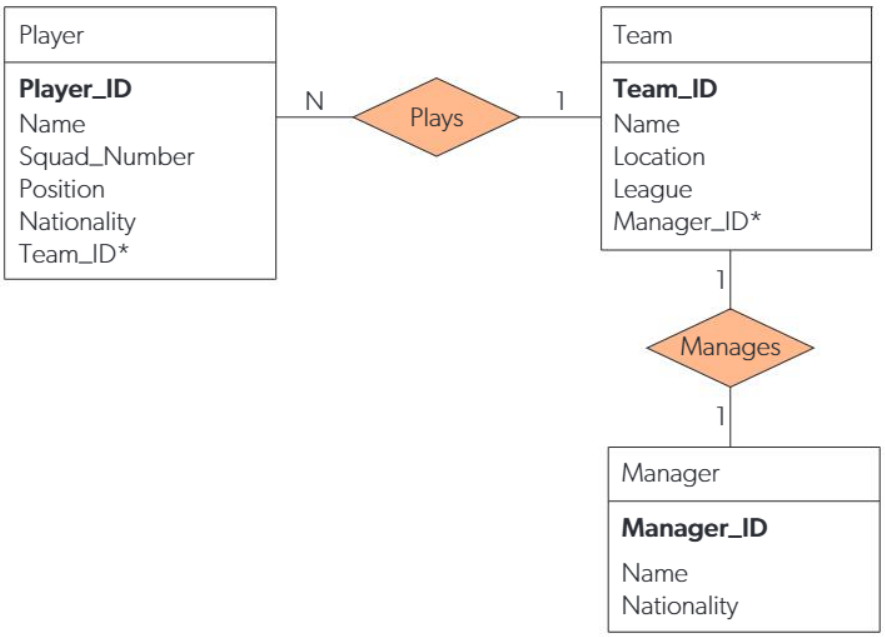


▲ **Figure 6** An example entity relationship diagram at a conceptual schema level

The logical schema

This model of the database identifies the different attributes (the data that will be stored about each item in the entity) within each of the entities. This will include the primary keys and foreign keys which create the relationships between the entities. This level of the schema will be used by both the programmer and the administrator. However, this level will not be used by the user.

An example of the detail required in the logical schema is shown in Figure 7, using the same database example as the example outlined in the conceptual schema. The figure shows the information that will be stored about the Player, the Team, and the Manager. The primary key is bold and the foreign key is denoted with an asterisk (*).

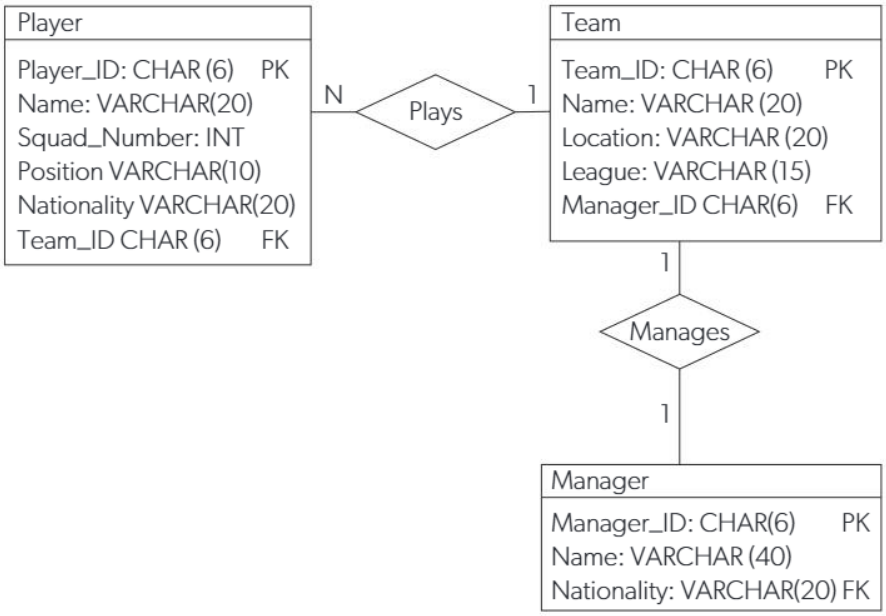


▲ **Figure 7** An example entity relationship diagram at a logical schema level

The physical schema

This model of the databases identifies the different attributes and their data types as well as the primary key and foreign key link that will create the relationships between entities. The physical schema is a description of how the data will be stored in the database, including specific files, indexes and storage structures. This level of schema will be used by the programmer as this is the schema used to initially set up the database.

Continuing with the database examples from the conceptual and logical schema, an example of the detail required in the physical schema is shown in Figure 8. The image shows the entities, the attributes, and their data types. Primary keys are denoted using the initials PK and foreign keys FK.



▲ **Figure 8** An example entity relationship diagram at a physical schema level

A3.2.2 Construct ERDs

Entity relationship diagrams (ERDs) show the entities (tables), attributes (fields), and relationships within a relational database. There are different levels of ERDs within the database schema. This section explains how to construct an ERD.

The different items within the ERD

Entity: An entity is an object that can have data stored about it. Entities are usually named using a noun and are represented using a rectangle. Examples of entities include: Student, Car, Player, Course, Product, Menu item.

Key: An entity needs to have at least one key (the different keys were outlined in A3.1.1).

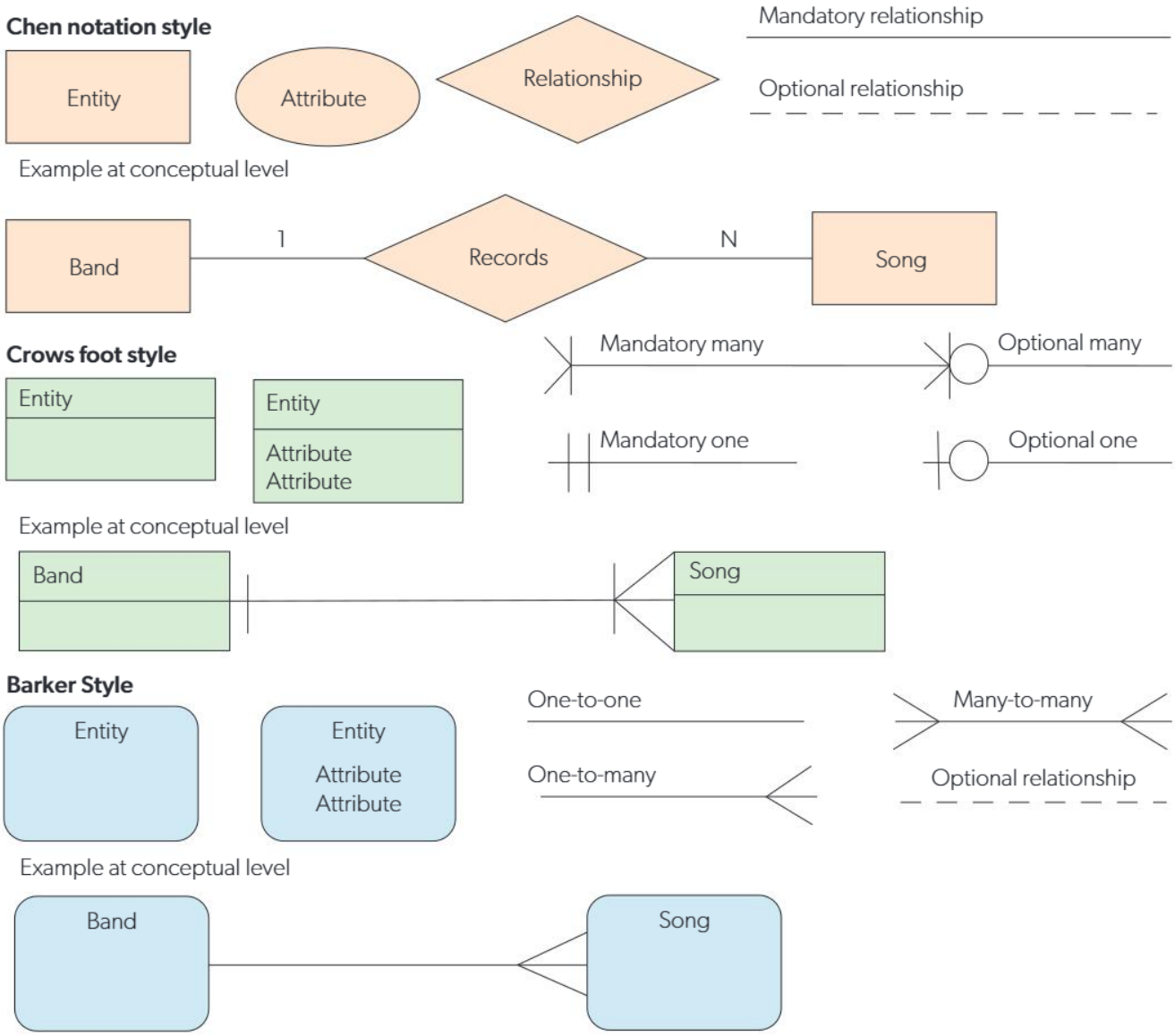
Relationships: The way in which the entities interact with each other is known as the relationship. Relationships can be thought of as the verbs that connect the entities. For example, Student studies Course, Teacher teaches Course, Menu contains Menu items.

Look back at section A1.1.1 for more information about relationships.

In some versions of the entity relationship diagrams, the relationship is identified in a diamond. The type of relationship (known as the cardinality) can be one-to-one, one-to-many or many-to-many. The relationship also needs to show the modality—whether the entity is mandatory or not within the relationship. For example, actor to theatre show is optional as many actors have not acted in theatre productions. But menu to menu item is mandatory since a menu has to have a menu item. Note that relationships are only read **one way**. A menu has menu items, but a menu item does not belong to a menu unless you specify this.

Attributes: The attributes show the data that will be stored about each of the items in the entity. For example, a car may have a make, model, engine size, fuel type and registration number.

There are different representations of ERDs. All show the same data but are represented differently. Three of the recognized conventions are shown in Figure 9.



▲ **Figure 9** Different representations of ERDs

To construct an ERD, follow these steps.

Step 1: Identify the level of ERD required (conceptual, logical or physical). This will determine the detail required.

Step 2: Identify the entities involved and, if required, the attributes and data types.

Step 3: Identify the relationships (using verbs) between the entities.

Step 4: Identify the cardinality of the relationship, one-to-one, one-to-many or many-to-many.

Step 5: Identify the modality of the relationships. Are they optional or mandatory?

ERDs provide a foundation to build a database. By modelling the components of a database in a very clear manner it can be understood by other developers, which improves database maintenance. Working through the model, you can identify entities and relationships. Understanding the interconnectivity allows you to spot mistakes before development begins.

Activity

Construct a logical level ERD diagram for the following system: A doctor treats several patients per day.

You need to identify the entities, attributes, and relationships required.

ATL Communication & Research skills

ERD diagrams communicate the schema of the database clearly so it can be understood by many people. Developing ERD diagrams requires pattern recognition.

Research in pairs the following questions.

- How could ERD diagrams be seen as independent of languages?
- How does the accepted universal ERD diagram enable the database design to be understood by all designers?
- What role do ERD models play in the acquisition of knowledge within computer science?

Share your findings with another pair. What commonalities are there in your answers? What differences do you have?

Work together as a four to create definitive answers you are all happy with.

A3.2.3 Outline the different data types used in relational databases

The data types you have available for use in a database depend on the database system you are using. Look at the documentation provided with your database software. This book will refer to MySQL data types.

There are three main data types in MySQL: string, numeric and date/time. The following examples are not exhaustive, but they outline the main data types available of each kind.

Table 1 String data types

Data type	Example	Description
`CHAR(size)`	P@ssWord!	A string (letters, numbers and special characters) of a fixed length. The length of the fixed string can be 0 to 255.
`VARCHAR(size)`	@Ab41ITCC (A better password)	A string (letters, numbers and special characters) of a variable length. The maximum size can be specified and can be up to 65,535 characters.
`TEXT(size)`	This is example text of longer text.	A string field with a maximum size of 65,535 characters (around 64 KB).
`MEDIUMTEXT`	Used to store backups of books, and so on.	A string field with a maximum size of 16,777,215 characters (around 16 MB).
`LONGTEXT`	Used to store backups of code and applications.	A string field with a maximum size of 4, 294, 967, 295 characters (around 4 GB of text, which is enough to store approximately 8,000 books).
`ENUM(val1, val2, val3…)`	Ms, Mr, Mrs, Mx, Dr, Prof, Other, None	A string field where the value is limited to one of a number of values (for example, a title list on a website).
`SET(val1, val2, val3,….)`	Sport Sport, Art	A string field that can have 0 or more values from a list of values (for example, selecting interests from a list of interests).

Table 2 Numeric data types

Data type	Example	Description
`BIT(size)`	110	A bit value can have a size from 1 to 64.
`BOOL`	0	Zero equals false and non-zero equals true.
`INT(size)`	48575392	A whole number. The size specifies the display digits (up to 255).
`FLOAT(p)`	5.333	A floating point number. The p specifies whether it is treated as a float or double.

Table 3 Date/time data types

Date type	Example	Description
`DATE`	2010-10-29	A date. The format of the date is YYYY-MM-DD.
`DATETIME(fsp*)`	2010-10-29 21:03:45	A date and time. The format of this is YYYY-MM-DD hh:mm:ss.
`TIMESTAMP(fsp*)`		A timestamp will add the current date and time.
`TIME(fsp*)`	13:04:39	A time in the format hh:mm:ss.
`YEAR`	2015	The year stored in YYYY format.

* Fractional seconds precision: the timestamp stores the fraction of the second up to 6 digits of precision. 6-digit precision for seconds, for example, gives the integer value plus 6 decimal digits (for example, 54.283483 seconds). This may seem excessive but, considering the number of databases and the variety of data being stored worldwide, it can be very useful in a number of different scenarios.

- To measure time very accurately. Measuring time in whole seconds would not work, for example, for measuring the fastest laps in a Formula One race, or to decide the winner of the 100 m sprint at the Olympics.
- When many events are happening at the same time. For popular events, such as festivals or music gigs, there can be thousands of people in an online queue and everyone wants to buy tickets at the same time. Using fractional seconds ensures everyone gets a fair place in the queue.
- When working with distributed databases, it is essential that transactions are coordinated. Using fsp can help to effectively manage transactions.

The data type informs MySQL what type of data is held within the data field and, therefore, how it will interact with the data. For example, anything stored as a string data type cannot have calculations performed upon it but numerical data types can. Consistency is important when choosing data types so data can be imported and updated correctly.

The correct data type is essential for developing a database, as it is with programming (discussed in section B2.1.1).

ATL Thinking skills

Developers must understand the data that they will need to store, so they can understand what data types they will need to work with. To do this, they must examine real-world problems.

When developing databases, you are often presented with documents showing how the data is collected. These could be presented as online or offline forms, posts or playlist details.

Using the screenshot of a playlist in Figure 10, identify the:

- attributes that would be stored
- data types of the attributes to be stored.

Throwback Thursday Classics

Song	Band	Length
Master Of New York	Ten Inch Monkeys	3:56
Girls	Rancid Rancid	3:19
Tall People	Joystickhead!	2:50
How Much Do You Love Pies?	No Rest For the Teachers	3:44
Cambridge Circus	T.O.O.T.H.B.R.U.S.H.	4:19
Deaf Tigers	Collapsing at the Disco	2:02

▲ **Figure 10** Screenshot of a playlist

If you do not choose the correct data type, you may get issues with data not being used in the correct way. For example:

- if data is stored as string, you may not be able to use it to perform calculations
- if you want to query using dates, the data needs to be stored as a date
- if you set it as `ENUM`—a predetermined list of values—then if the item is not on the list, users will not be able to store the data they want to.

Activity

Suggest suitable data types for the following attributes.

- Name: the name of a person. e.g. Nora Digby.
- ID Number: an ID number for a person e.g. 38292.
- Vegan: whether a person is vegan or not.
- Height: the height of a person in metres. e.g. 1.92.
- Address: the address of a company e.g. Rotkreuzplatz 89, Zythus, 9332.
- Ticket cost: the cost of a ticket e.g. 34.99.
- Quantity of an item in a warehouse e.g. 392.

TOK

When developing databases, developers make decisions about what data will be stored and what data types will be used to store the data. This is a form of knowledge classification: developers have to determine the acceptable values for each attribute within the database.

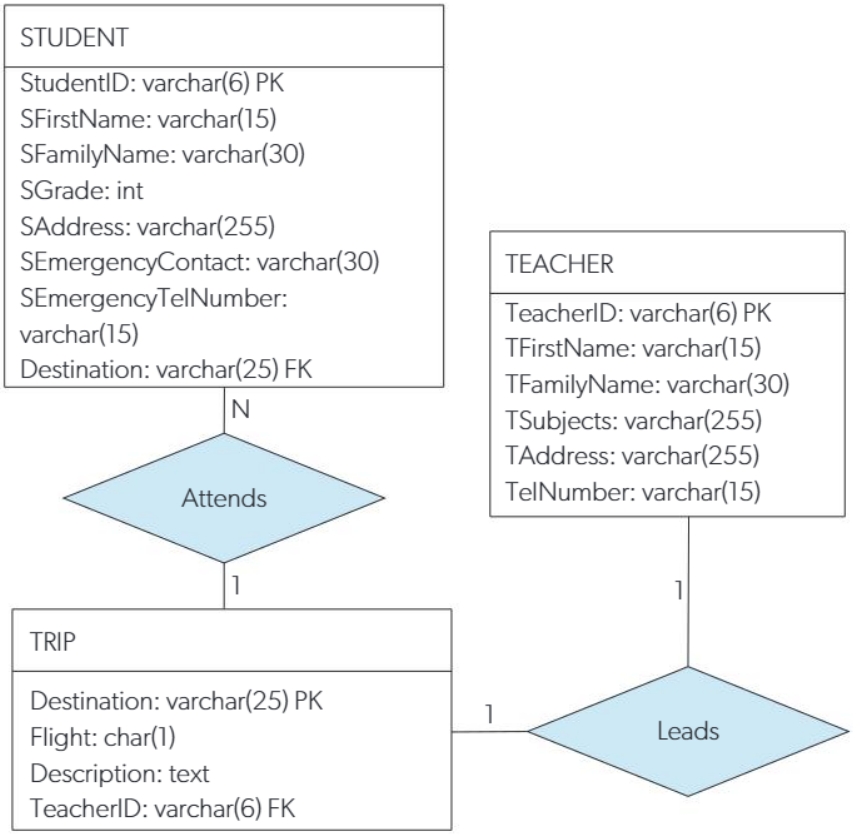
The databases that store scientific calculations are an example of this. If the developer decides to store the data as a float rather than as a double, some accuracy may be lost.

To what extent does the way knowledge is classified using data types affect what we can learn from the data?

A3.2.4 Construct tables for relational databases

Building tables that make use of keys is essential when creating a relational database. It is important you plan the tables correctly to ensure that data integrity is always maintained. One problem of a badly designed database is duplicated data. Duplicated data can lead to problems with deletion since, if your data is not linked together, there may be data left within the database resulting in wrong results from queries (deletion anomaly). If data is updated and the data is not correctly linked, not all associated records may be updated (update anomaly).

The correct linking of data can occur through keys.



▲ **Figure 11** The relationship between entities using keys

Keys link data together, showing clearly that there is a relationship between the two items and that they should be treated as related data.

Using keys, you can avoid deletion anomalies and update anomalies through cascading. Delete cascade makes use of primary keys and foreign keys. When a row with a primary key is deleted, all rows that are referenced with the foreign key are also deleted. This removes all related data, ensuring data is left over in the table.

Update cascade also makes use of primary keys and foreign keys. If a primary key is updated in a table, all subsequent related rows are also updated. This ensures data integrity across the tables.

A3.2.5 Explain the difference between normal forms

Databases that are not designed properly can have the following issues.

Insert anomalies: You cannot add data correctly as the data is dependent on other attributes in the database.

Deletion anomalies: The database may have attributes that rely on non-related attributes. If you delete the non-related attributes, you unintentionally lose data.

Update anomalies: Any update in a badly designed database will be a challenge as you have to look through every section of the database to ensure all items are deleted. This is not a challenge in a small database, but for a database with hundreds thousands, or millions of records, this becomes a significant problem.

Normalization is a developer's tool that ensures a database is correctly designed. Normalization turns an unnormalized, badly organized database into a well-designed database, minimizing the likelihood of anomalies occurring.

There are three stages of normalization you need to be aware of: unnormalized to first normal form, first normal form to second normal form, and second normal form to third normal form. In this course you need to know the differences between the forms and be able to follow the steps to normalize a database to third normal form. One of the main things you need to remember when normalizing is:

The data should rely on the key,
the whole key and nothing but the key.

When creating databases, you are thinking about functional dependencies; that is, what data relies on other data. For example, in Table 4, the primary key for the patient indirectly tells you the patient's name and address, as the primary key is unique to this patient. Normalization makes use of these dependencies to break down the original entity into smaller entities where all data is functionally dependent on the key. Once you have a normalized database, you minimize the chances of the anomalies occurring, and the database is more reliable.

Activity

Identify the anomaly that has occurred in each of the following scenarios.

- Two people have bought the same ticket for an event.
- There are three addresses for the same person.
- There is a pet in the database without an owner.

TOK

Real-world systems are complex. Every interaction in the real world has many nuances. Programmers need to abstract the information into a general model that works for many people. Consider the Amazon database. This contains information about the products people buy and the media (books, music, film, and television shows) people consume using Amazon. The Amazon programmers have to consider:

- what information they need to store about people and what relationships they need to identify in that data to allow them to represent their customers accurately
- what information to store about products that will enable them to make good business decisions.

Identify the challenges that database programmers face when trying to accurately represent real-world data in normalized databases.

To understand normalization, consider the data in Table 4.

Table 4 Data for normalization

Dentist ID	Dentist Name	Dentist Practice	Patient ID	Patient Name	Patient Address	Treatment	Treatment Detail	Date	Price
192	Luis Reorder	Heaton	190	Wesley Kyle	12 Sunny Close	Filling	Filling with technical assistance	22/06/25	150
			138	Ana Torres	87 Lake Lane	Check Up	Check up and cleaning with hygienist	22/06/25	80
193	Henry Stewart	Gosfield	190	Wesley Kyle	12 Sunny Close	Filling	Filling with technical assistance	25/06/25	150
			148	Camille Manon	98 Alma Grove	Crown	Crown including X ray	25/06/25	300
			150	Ole Huber	1 North Wharf	Check Up	Check up and cleaning with hygienist	25/05/25	80
194	Romesh Ranaweera	Spittal Fields	136	Jamie Christoph	3 North Wharf	Filling	Filling with technical assistance	25/05/25	150
			172	Amalie McGovern	19 Maker St	Cleaning	Cleaning with hygienist	25/06/25	50

The first thing you need to ensure is that all values in the table are atomic, meaning they only contain one value in each cell. Table 4 is atomic as all of the values only have one value. Once you have confirmed the atomicity, you place the data into unnormal form (UNF), choosing an initial primary key and identifying the repeating groups. Repeating groups are the new data that needs to be added when a new entry is created. In Table 4, for each new entry, you need to add: Patient ID, Patient Name, Patient Address, Treatment, Treatment Detail, Date and Price.

UNF	Unnormal form
(Dentist_ID	Initial candidate (possible) primary key identified
Dentist_Name	
Dentist_Practice	
Patient_ID	Repeating groups identified using indentation
Patient_Name	
Patient_Address	
Treatment	
Treatment_Detail	
Date	
Price)	

Table 5 Unnormal form to first normal form

UNF	1NF (First normal form)	Actions
(Dentist_ID Dentist_Name Dentist_Practice Patient_ID Patient_Name Patient_Address Treatment Treatment_Detail Date Price)	DENTIST (Dentist_ID Dentist_Name Dentist_Practice) TREATMENT (Patient_ID Patient_Name Patient_Address Treatment Treatment_Detail Date Price Dentist_ID*)	Non repeating data is placed in an entity, entity is named and a primary key given. Repeating data is moved to a new table and a unique key found. The primary key from one table is placed in the second table as a foreign key.

Table 6 First normal form to second normal form

1NF	2NF (Second normal form)	Actions
DENTIST (Dentist_ID Dentist_Name Dentist_Practice) TREATMENT (Patient_ID Patient_Name Patient_Address Treatment Treatment_Detail Date Price Dentist_ID*)	DENTIST(Dentist_ID Dentist_Name Dentist_Practice) TREATMENT(Patient_ID* Treatment Treatment_Detail Date Price Dentist_ID*) PATIENT (Patient_ID Patient_Name Patient_Address)	First entity ignored. Patient data removed from treatment entity as it does not rely on the whole key, just Patient_ID, which is transformed into a foreign key. Patient entity created and primary key identified.

Table 7 Second normal form to third normal form

2NF	3NF (Third normal form)	Actions
DENTIST(Dentist_ID Dentist_Name Dentist_Practice) TREATMENT(Patient_ID* Treatment Treatment_Detail Date Price Dentist_ID*) PATIENT (Patient_ID Patient_Name Patient_Address)	DENTIST(Dentist_ID Dentist_Name Dentist_Practice) APPOINTMENT(Patient_ID* Treatment* Date Dentist_ID*) PATIENT (Patient_ID Patient_Name Patient_Address) TREATMENT(Treatment Treatment_Detail Price)	All entities need to be checked for all attributes relying on the key. Treatment information does not rely on all parts of the key, only on treatment. Treatment becomes a foreign key. Entity renamed. Treatment entity with all treatment details created. All entities now have data that relies only on the key.

ATL Thinking skills

Developing databases into third normal form requires logical thinking. One of the key requirements is abstracting the required information.

Consider the following unnormalized database based on a picture sharing site:

Username

image

image_date

caption

location

location_coordinates

location_country

Develop a flow chart for each of the stage of the normalization process.

The differences between normal forms may seem confusing at first: they are outlined for you here.

First normal form: The repeating data is moved into a new entity. The non-repeating data stays in its own entity and is given a unique identifier (primary key). The primary key from the non-repeating entity is placed in the new entity as a foreign key. In the new entity a unique identifier is found. This will most likely be a composite key. The issue with the first normal form is that the new entity will have attributes that only rely on one part of the key, known as partial key dependencies. This does not align with the normalized database philosophy of "the whole key and nothing but the key".

Second normal form: The partial key dependencies are removed into their own entity and a primary key is identified. The primary key features in the entity it was removed from as both a primary key and a foreign key at this point. The issue with second normal form is that some attributes in the entities do not rely on the key, they are their own information. These are known as transitive or non-key dependencies. These, again, do not align with the normalized database philosophy of "the whole key and nothing but the key".

Third normal form: The transitive or non-key dependencies are moved into their own entity and given a primary key. The primary key becomes a foreign key in the entity it was removed from. Transitive dependencies can be present in any of the entities previously created. Once they are removed, your entities should meet the normalized database philosophy of "the whole key and nothing but the key".

TOK

When normalizing data, developers make assumptions about the data they have been given, deciding what data must be stored and what data can be left out of the final data model. For example, developers may assume that every ticket for a sporting event has a seat associated with it. What are the roles of assumptions when developing a normalized database model?

Activity

Explain the difference between the normalization stages.

A3.2.6 Construct a database normalized to 3NF for a range of real-world scenarios

In programming, encapsulating the data within a class keeps the data safe and accurate. Entities in a database are similar. By keeping all data in one place and minimizing duplication, the data is more secure.

Learn more about encapsulation and information hiding in section B3.1.5.

Two examples of how data normalizing is used in real-life scenarios are an e-commerce database and a library database.

E-commerce database

An e-commerce website called Negozio connects local artists selling handmade gifts to registered customers. Transactions store the item sold, the artist and the seller.

An excerpt of the unnormalized flat file database is shown in Table 8. As you can see in the excerpt, customer data is stored every time the customer makes a purchase. The artist details are stored every time one of their products is sold.

Table 8 Negozio transaction data

Customer ID	Customer Email	Customer Address	Item ID	Item Description	Quantity	Cost	Date	Artist ID	Artist Name	Artist Email
4843	Raya@mail.com	95 East Wharf	5504	Small gold ring	1	49	3/3/25	49455	Maja's Jewellery	MJS@mail.com
			4532	Wooden toy chest	1	120	4/3/25	96854	Sunshine craft	Sunshine@mail.com
			6422	Handmade ceramic bowl	4	25	12/8/25	58584	Sarah's Ceramics	SCC@mail.com
3758	Bligh@mail.com	22 Light Street	6422	Handmade ceramic bowl	6	25	13/8/25	58584	Sarah's Ceramics	SCC@mail.com
			6665	Football print	1	45	4/10/25	1222	Boris Printshop	Boris@mail.com
			9182	Printed apron	2	15	4/10/25	1222	Boris Printshop	Boris@mail.com
4737	Frankie@mail.com	58 Tree Grove	5504	Small gold ring	1	49	3/3/25	49455	Maja's Jewellery	MJS@mail.com
4728	Rory@mail.com	85 Green Street	9383	Ramen bowl	6	20	12/6/25	58584	Sarah's Ceramics	SCC@mail.com
5844	Alysha@mail.com	4 East Street	5504	Small gold ring	1	49	12/6/25	49455	Maja's Jewellery	MJS@mail.com
			4783	Laser cut bag	1	80	12/6/25	1111	Heather's Handicraft	Heather@mail.com

Table 9 shows the normalization process for the negozio.com database. First normal form removes the transaction data to a new table, adding the customer ID as a foreign key. In second normal form, the item and artist data is removed as this only relies on one part of the composite key. In third normal form, the artist information is removed to a new table as the artist information does not depend on the item's primary key.

Table 9 Normalization process for the Negozio database

UNF	1NF
Customer_ID Customer_Email Customer_Address Item_ID Item_Description Quantity Cost Date Artist_ID Artist_Name Artist_Email	CUSTOMER(Customer_ID Customer_Email Customer_Address) TRANSACTION(Item_ID Item_Description Quantity Cost Date Artist_ID Artist_Name Artist_Email Customer_ID*)
2NF	**3NF**
CUSTOMER(Customer_ID Customer_Email Customer_Address) TRANSACTION(Item_ID* Quantity Date Customer_ID*) ITEM(Item_ID Item_Description Cost Artist_ID Artist_Name Artist_Email)	CUSTOMER(Customer_ID Customer_Email Customer_Address) TRANSACTION(Item_ID* Quantity Date Customer_ID*) ITEM(Item_ID Item_Description Cost Artist_ID*) ARTIST (Artist_ID Artist_Name Artist_Email)

Library database

A library database is used to manage a group of libraries to record which books have been borrowed and who has borrowed them. The database also stores details of any outstanding fines per person. For this group of libraries, a person belongs to and can borrow books from only one library—they cannot borrow books from other libraries in the group. An extract from the current unnormalized flat file database is shown in Table 10. Each time a person borrows a book, the library data and the borrower information is stored. If a book is borrowed more than once, the book data is also duplicated.

Table 10 Unnormalized library database extract

Library Name	Location	Borrower ID	Borrower Name	Book	Book Number	Fiction	Date Borrowed	Date Returned	Fees Due
Central	City Centre	121	James	Shark tales	3939-2929	Yes	23/06/25	08/08/25	10
		121	James	Fishing for beginners	3958-6873	No	23/06/25	08/08/25	
		122	Elisabeth	The crow lane	8628-4920	Yes	23/06/25	08/08/25	
Alliance	Evergrove	134	Anjana	The glass card	2847-2933	Yes	24/06/25	10/08/25	
		134	Anjana	Copenhagen – a guide	2810-4958	No	24/06/25	10/08/25	
		134	Anjana	The silver river	5849-1920	Yes	24/06/25	10/08/25	
University	University Quad	144	Filippo	Python for starters	9201-4830	No	23/06/25	10/08/25	20
				Java 101	1092-4920	No	27/06/25	20/09/25	
		145	Portia	Psychology for starters	4839-5802	No	30/08/25	20/09/25	
				Python for starters	9201-4830	No	30/08/25		

Table 11 shows the normalization process for the library database. First normal form removes the borrower, book, and borrowing information into a new table. Second normal form removes the book information as it does not rely on the whole of the composite key. In third normal form, the borrower information is removed as there are transitive dependencies within the loan table.

Table 11 Normalization process for the library database

UNF	1NF
Library_Name Location Borrower_ID Borrower_Name Book Book_Number Fiction Date_Borrowed Date_Returned Fees	LIBRARY(Library_Name Location) LOAN (BorrowerID Borrower_Name Fees Book Book_Number Fiction Date_Borrowed Date_Returned Library_Name*)
2NF	**3NF**
LIBRARY(Library_Name Location) LOAN (Book_Number* Date_Borrowed BorrowerID Borrower_Name Fees Date_Returned Library_Name*) BOOK (Book_Number Book Fiction)	LIBRARY(Library_Name Location Borrower_ID*) LOAN (Book_Number* Date_Borrowed BorrowerID* Date_Returned Library_Name*) BOOK (Book_Number Book Fiction) BORROWER (Borrower_ID Borrower_Name Fees)

A3.2.7 Evaluate the need for denormalizing databases

The first thing to note about denormalized databases is that these databases have been previously normalized to a satisfactory level and denormalization has occurred because this will optimize the database performance.

When a database that has been normalized is being queried heavily, the use of joins to link the data can become very intensive and slow down the database significantly. One way you can overcome this is to unnormalize the database, meaning that you keep some of the data together despite this meaning some data will be repeated. For example, in the online shopping database in Table 9, the artist details may be stored with the item details as these are often presented together, rather than having to join the tables each time the data is needed.

Advantages and disadvantages of denormalizing a database

Advantages

- Simpler queries as the database has to look at fewer entities to collect all the data.
- Faster data retrieval as the database has fewer joins to complete.

Disadvantages

- More challenging updates and inserts as some of the data is repeated.
- Updating code can be difficult to write as the data is in multiple places.
- There may be inconsistencies as there are duplicate copies of data.
- The fact there are many copies of some data requires more storage.

When deciding to denormalize a database, you need to balance having a speedier, easier-to-query database against the challenges of storing multiple copies of the data, leading to update and insertion challenges. Data warehouses often use denormalized databases.

TOK

When developing databases, there are many things to consider.

Imagine you are asked to develop a database for a ferry company. You need to consider:

- Ship: Name, IMO number, registration, capacity, number of passengers, number of vehicles, colour, captain.
- Passengers: Name, address, passport, destination, age, nationality.
- Vehicle: Vehicle type, license plate, size, manufacturer, model, colour.
- Route: Start, end, duration, weather, areas crossed.

You **can** store many types of data, but do you **need** to store everything?

How can the abstraction of the data help to model the real world within a database?

Activity

Normalize the databases in Table 12 and Table 13 to third normal form.

Table 12 Extract from unnormalized playlist database

Playlist ID	Playlist Name	Date Created	Song Name	Song Release Date	Artist Name	Artist Type	Active Since
9222	Dad Music	21/01/25	Sorry to see you go	17/08/1978	The Punkettes	Alternative Rock	1970
			Bye Bye	06/06/1977	The Punkettes	Alternative Rock	1970
			Imma See	23/05/1985	Jamie and the Rocks	Alternative Rock	1981
4829	Happy Study	21/03/25	The Key Concepts	18/10/24	Old Gregg	Upbeat	2022
			More or Less	12/10/23	MSTCRW	Upbeat	2022
			VR Pyramid	12/10/23	De Marseille	Upbeat	2021
4841	Rage Tunes	22/03/25	Trust Me	29/12/98	Stereosound	Metal	1992

Table 13 Extract from unnormalized match ticket database

Member ID	Name	Address	Seat	Stand	Price	Ticket Number	Opponents	Opponent Colours	Match Type
ST85	Jasper Harrison	49 North Wharf	34A	L	75	39202	Sunnyside	Red & White	Cup
			45B	M	60	95842	Olten	Blue & Black	League
			123C	L	70	30101	Clay hill	Orange	Cup
ZA49	Zanib Saltman	58 Le Marche	99B	A	120	49302	Sunnyside	Red & White	Cup
			93B	A	125	29292	Clayhill	Orange	Cup

Practice questions

6. Explain what is contained in a logical database schema. [3 marks]
7. Explain how ERDs are used to model a database. [6 marks]
8. Describe two reasons why a database should be normalized. [4 marks]
9. State suitable data types for the following attributes.
   a. Price of ticket.
   b. Whether a playlist is downloaded to a device or not.
   c. Name of a candidate.
   d. Description of candidate. [4 marks]
10. Normalize the database in Table 14 to third normal form. [5 marks]

Table 14 Extract from unnormalized student visit database details

Trip Name	Destination	Max Places	Leader	Leader Contact	Student ID	Name	Grade	Age
Sustainable Denmark	Copenhagen	10	Alice Croft	+38029183392	39101	Eamon Green	10	16
					201921	Lovisa Wright	10	16
					94030	Kirsty Porritt	11	17
					40393	Becky Black	11	17
Creativity	Venice	10	Lawerence Debiton	+3829101011	392001	Martina Hayes	11	17
						Nabiha Ansari	10	16
						Soraya Azad	11	17
						Assad Khan	11	17

A3.3 Database programming

Syllabus understandings

A3.3.1 Outline the difference between data language types within structured query language (SQL)

A3.3.2 Construct queries between two tables in SQL

A3.3.3 Explain how SQL can be used to update data in a database

AHL

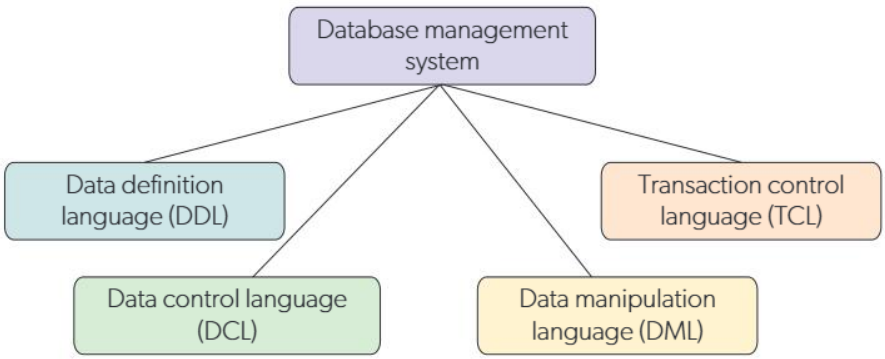
A3.3.4 Construct calculations within a database using SQL's aggregate functions

A3.3.5 Describe different database views

A3.3.6 Describe how transactions maintain data integrity in a database

A3.3.1 Outline the difference between data language types within structured query language (SQL)

Database management systems use four different types of language to create databases, manipulate databases, and enable safe interaction with the database. The database management system can be thought of as an umbrella for each of the languages. This is shown in Figure 12.



▲ **Figure 12** An overview of the database languages

Data definition language

The data definition language (DDL) is used to create the database structures. It is used to create the schema, the tables, and constraints within the database. Using DDL statements, you can create the outline of the database.

Commands in the DDL include the following.

- `CREATE` Used to create database objects.
- `ALTER` Used to change the structure of the database.
- `DROP` Used to delete objects from within the database structure.

- `TRUNCATE` Used to remove all records from the entity within the database structure.
- `RENAME` Used to rename an object within a database structure.
- `COMMENT` Used to add a comment to the database structure.

Data manipulation language

The data manipulation language (DML) is used to access the data within the database and to manipulate the data within the database. For example, querying the database for information, updating records, and deleting records.

Commands in the DML include the following.

- `SELECT` Used to retrieve data from the database.
- `INSERT` Used to add data into the entity.
- `UPDATE` Used to update the existing data within an entity.
- `DELETE` Used to delete all records from an entity (but not the entity itself).
- `CALL` Used to call an SQL operation.
- `LOCK TABLE` Used to aid concurrency in the database.

Data control language

The data control language (DCL) is used to control access to the database. The DCL helps to maintain security in the database as it allows user to have access to the database or it can revoke data from the database.

Commands in the DCL include the following.

- `GRANT` Allows user access privileges in a database.
- `REVOKE` Allows the removal of privileges in a database.

Transaction control language

If you are studying at Higher Level, you will learn more about this in section A3.3.6.

The transaction control language (TCL) is used to complete the changes in a database.

Commands in the TCL include the following.

- `COMMIT` Used to complete queries and save them within the database.
- `ROLLBACK` Used if there is an issue in an operation to restore the database back to its last `COMMIT` statement.

TOK

The *Oxford English Dictionary* defines language as:

> The system of spoken or written communication used by a particular country, people, community, etc., typically consisting of words used within a regular grammatical and syntactic structure; (also) a formal system of communication by gesture, esp. as used by deaf people.
>
> (*Oxford English Dictionary*, 2023)

Using the above definition to help you, to what extent do you believe that DDL is a language?

Using SQL to develop a database

The following SQL statements are used to create data structures in the database. For each example, the SQL command has been given and, when appropriate, examples provided from an SQLite database running in the DataGrip IDE.

Create a database

You can create a database using the statement in Table 15.

Table 15 SQL code to create a database

Syntax	Example
`CREATE DATABASE database_name;`	`CREATE DATABASE testDB;`

Create a table

You can use the statements in Table 16 to create a table within a database.

Table 16 SQL code to create entities within a database

Syntax	Example with descriptions

Without key
`CREATE TABLE table_name(`
`Attribute1 datatype,`
`Attribute2 datatype,`
`Attribute3 datatype,`
`Attribute4 datatype,`
`)`
With primary key
`CREATE TABLE table_name(`
`Attribute1 datatype NOT NULL PRIMARY KEY,`
`Attribute2 datatype,`
`Attribute3 datatype`
`);`
	`CREATE TABLE Customer(`
	`CustomerID int NOT NULL PRIMARY KEY,`
	`CustomerEmail varchar(255),`
	`CustomerAddress varchar(255)`
	`);`
	`NOT NULL PRIMARY KEY` indicates there is a constraint on this data. The value cannot be empty and it must be unique as it is a primary key.

With primary and foreign key
`CREATE TABLE table_name(`
`Attribute1 datatype NOT NULL PRIMARY KEY`
`Attribute2 datatype;`
`Attribute3 datatype;`
`Attribute4 datatype;`
`FOREIGN KEY (Attribute4) REFERENCES table_name(Attribute4)`
`);`
	`CREATE TABLE Item (`
	`ItemID int NOT NULL PRIMARY KEY ,`
	`ItemDesc varchar(255),`
	`ItemCost real,`
	`ArtistID int,`
	`FOREIGN KEY (ArtistID)`
	`REFERENCES Artist(ArtistID)`
	`);`
	In the Item entity there is also a foreign key. The attribute is created and then identified as a foreign key, creating the link between the Artist and Item entities.

Syntax	Example with descriptions

With concatenated key
`CREATE TABLE table_name(`
`Attribute1 data_type,`
`Attribute2 data_type,`
`Attribute3 data_type,`
`Attribute4 data_type,`
`PRIMARY KEY(Attribute1, Attribute2)`
`);`
	`CREATE TABLE Purchase(`
	`ItemID int NOT NULL ,`
	`Quantity int,`
	`Date date NOT NULL ,`
	`CustomerID int NOT NULL ,`
	`PRIMARY KEY (ItemID, Date, CustomerID),`
	`FOREIGN KEY (ItemID) REFERENCES Item(ItemID),`
	`FOREIGN KEY (CustomerID) REFERENCES Customer(CustomerID)`
	`);`
	In this entity there is a concatenated key. These are also foreign keys.

Delete table

Sometimes you want to be able to delete tables as they are no longer required in the database. This is shown in Table 17.

Table 17 SQL code to delete an entity

Syntax	Example
`DROP TABLE Table_name;`	`DROP TABLE delivery;`

Alter table

Sometimes it is necessary to make changes to tables as they require new attributes. This is shown in Table 18.

Table 18 SQL code to alter an entity

Syntax	Example with descriptions
`ALTER TABLE Table_name` `ADD attribute data_type;`	`ALTER TABLE Customer ADD deliveryAddress varchar(255);` A delivery address attribute is added to the Customer entity.
`ALTER TABLE Table_name` `DROP COLUMN attribute;`	`ALTER TABLE Customer DROP deliveryPreference` Removes delivery preference from the Customer entity.

Modifying data in a table

Once you have created a table, it is useful to be able to add data, modify data and delete data from the table. This is shown in Table 19.

Table 19 SQL code to modify the data in an entity

Syntax	Example
`INSERT INTO table_name (Attribute1, Attribute2, Attribute3…)` `VALUES (value1, value2, value3….)`	`INSERT INTO Customer (CustomerID, CustomerEmail, CustomerAddress, deliveryAddress) VALUES('2','Christopher@mail.com','34 Maple Crescent, Midtown','Unit A Industry Road, Midtown');`

`UPDATE table_name`
`SET Attribute1 = value1, Attribute2 = value2`
`WHERE condition;`
	`UPDATE Item`
	`SET ArtistID = '101'`
	`WHERE ItemID = '2940';`

`DELETE FROM table_name WHERE condition;`	`DELETE FROM item WHERE ItemID = '49393';`

A3.3.2 Construct queries between two tables in SQL

Queries are used in databases to extract data and provide context for data. Data within a database has little relevance unless it is put into context. Queries can be used for filtering, which means finding the data you want to know from the database, for example, all the customers in the database who own a specific brand of car. Queries can be used for pattern matching—searching for patterns in the data when you don't know exactly what you are looking for, for example, all customers who have bought a red item. Finally, queries can be used to order data. For example, you may wish to find the top ten customers, and ordering the data will enable you to see this information.

Thinking skills

Developing commands to extract data from databases requires algorithmic thinking. Developing SQL statements requires you to think algorithmically.

Construct SQL statements to complete these database tasks.

1. Create the database called Artefacts.
2. Within the Artefacts database, add the following entities with the shown attributes:

```
CURIO (Curio_ID (PK), Curio_name, Curio_Type)

CABINET (Cabinet_ID (PK), Cabinet_Description,
Cabinet_location)

COLLECTION (Cabinet_ID (PK/FK), Curio_ID(PK/ID))
```

Key term

Query An instruction developed in SQL to extract information from the database.

TOK

Social media sites use databases to store masses of data points identifying users' likes and dislikes. Online shops use databases to store numerous data points identifying consumer likes and dislikes. To access this data and make it useful, companies need to manipulate the data using specialist language.

To what extent do a shared code and language, such as those in the SQL language, help or hinder the pursuit of knowledge on a global level?

Like programming in SQL, in section B2.3.2 you will construct code to select data when specific conditions are met.

Tables 20 to 32 detail some useful SQL commands to extract data from a database.

Table 20 SQL Select code

Command	Explanation of the command
`SELECT`	The Select operator returns selected attributes from a table. If you want to return all attributes from a table, you use the `*` wildcard. You can also use the `%` wildcard to search for a specific substring within a text field.

Syntax

```
SELECT Attribute1, Attribute2, Attribute3 FROM Table_name;
SELECT * FROM Table_name;
```

Example

```
SELECT ArtistID, ArtistName FROM Artist;
```

Output

ArtistID	ArtistName
101	Evie Ceramics
102	Rojas Artwork
103	Heather Needlework
104	Queenies Prints
105	Alejandro Jewellery

Example

```
SELECT * from Artist;
```

Output

ArtistID	ArtistName	ArtistEmail
101	Evie Ceramics	EvieCeramic@mail.com
102	Rojas Artwork	Rojas@mail.com
103	Heather Needlework	HNW@mail.com
104	Queenies Prints	Queenie@mail.com
105	Alejandro Jewellery	Ale@mail.com

Table 21 SQL Select distinct code

Command	Explanation of the command
`SELECT DISTINCT`	The Select distinct operator only returns unique values from the entity. For example, you may use it if you wanted to find all the different types of items sold in a shop.

Syntax

```
SELECT DISTINCT Attribute1, Attribute2, Attribute3
FROM table_name;
```

Example

```
SELECT DISTINCT ItemID FROM Purchase;
```

Output

	ItemID
1	2921
2	4832
3	4944
4	5849
5	9283
6	9399
7	9481
8	9482

Table 22 SQL From code

Command	Explanation of the command
FROM	The From operator indicates the table where the data should be extracted from. This is important, as if this is from one table, you do not need a join, but if the results are from two or more tables, a join is required. You use this when you want to extract data from a table.

Syntax

```
SELECT * FROM Table_name;
```

Example

```
SELECT * from Customer;
```

Output

	CustomerEmail	CustomerAddress	deliveryAddress
1	Emilia@Mail.com	87 The Grove, Midtown	87 The Grove, Midtown
2	Christopher@mail.com	34 Maple Crescent, Mid…	Unit A Industry Road,
3	Ian@mail.com	Woodside, Lakeview	Woodside, Lakeview
4	jordan@mail.com	Wordsworth way, Winder…	Main Street Lakeview
5	Vereena@mail.com	Apple Grove, Lakeview	The Wynd, Winderview
6	Zhi@mail.com	98 Westside, Midtown	98 Westside, Midtown

Table 23 SQL Where code

Command	Explanation of the command
WHERE	The Where operator indicates there is a condition that has to be met when completing the select statement. Similar to an If statement in programming, the data is only shown from the database where the condition is met. You may use this to find all items over a certain price, or all customers located in a certain country.

Syntax

```
SELECT Attribute1, Attribute2, …. FROM table_name
WHERE condition;
```

Example

```
Select * from Item WHERE ItemCost > 300;
```

Output

ItemID	ItemDesc	ItemCost	ArtistID
4944	Abstract Wall Art One	384	102
9383	Abstract Wall Art Two	332	102
2921	Bracelet - fine	450	105

Table 24 SQL Between code

Command	Explanation of the command
`BETWEEN`	The Between operator selects values between a range (inclusive of the start and end values). The Between operator works on numbers, text or dates. You may use this if you want to find all items in the database within a certain price range.

Syntax

```
SELECT * FROM table_name
WHERE attribute1 BETWEEN value1 AND value2;
```

Example

```
Select * from Item WHERE ItemCost BETWEEN 100 AND 200;
```

Output

ItemID	ItemDesc	ItemCost	ArtistID
2992	Self portrait	192	102
9282	Shell Necklace	120	105

Table 25 SQL Order by code

Command	Explanation of the command

`ORDER BY`
`ASC`
`DESC`
	The Order by operator is used to order the results either in ascending or descending order. The data can be sorted on as many attributes are required. You may use this if you have lots of items and you want to be able to find them quickly.

Syntax

```
SELECT * FROM table_
name
ORDER BY attribute ASC
or DESC;
```

Example

```
SELECT * from item
ORDER BY itemCost ASC;
```

Output

1	9481	Lighthouse Print	15	104
2	3935	Decorative Drum	15	102
3	7685	Troll - gold and black	15	101
4	3402	Glass candle holder	15	104
5	5893	Personalised Tote Bag	15	103
6	5849	Priory Print	17	104
7	9399	The Bridges	25	104
8	5839	Mountain Scene	29	103
9	9283	The waves	29.75	103
10	9482	Tile featuring mountain scene	34.1	101
11	4849	Birds on the beach	45	103
12	2940	Vase featuring face	45.05	101
13	4832	Floor vase with leaves	45.48	101
14	9282	Shell Necklace	120	105
15	2992	Self portrait	192	102
16	8593	Necklace with Rose	234	105
17	5859	Sculpture featuring two cats	239	101

Example

```
SELECT * from item
ORDER BY itemCost DESC,
ItemDesc ASC;
```

Output

10	4849 Birds on the beach	45	183
11	9482 Tile featuring mountain scene	34.1	181
12	9283 The waves	29.75	183
13	5839 Mountain Scene	29	183
14	9399 The Bridges	25	184
15	5849 Priory Print	17	184
16	3935 Decorative Drum	15	182
17	3482 Glass candle holder	15	184
18	9481 Lighthouse Print	15	184
19	5893 Personalised Tote Bag	15	183
20	7685 Troll - gold and black	15	181

Table 26 SQL Group by code

Command	Explanation of the command
GROUP BY	The Group by operator groups the rows that have the same values into summary rows. You may use it to find the number of customers in a specific country.

Syntax

```
SELECT Attribute1, Attribute2
FROM table_name
WHERE condition
GROUP BY Attribute
ORDER BY Attribute
```

Example

```
SELECT * FROM Customer
GROUP BY CustomerCountry
ORDER BY CustomerCountry DESC;
```

Output

	CustomerEmail	CustomerAddress	deliveryAddress	CustomerCou
1	Emilia@Mail.com	87 The Grove, Midtown	87 The Grove, Midtown	UK
4	jordan@mail.com	Wordsworth way, Winderview	Main Street Lakeview	Italy
3	Ian@mail.com	Woodside, Lakeview	Woodside, Lakeview	France

Example with Count

```
SELECT COUNT(CustomerID),
CustomerCountry
FROM Customer
GROUP BY CustomerCountry ;
```

Output

"COUNT(CustomerID)"	CustomerCountry
2	France
2	Italy
2	UK

Table 27 SQL Having code

Command	Explanation of the command
`HAVING`	The Having operator is similar to the Where operator. It is a condition that has to be met before the data is displayed. However, since Where cannot be used with the aggregate functions, the Having clause must be used instead.

Syntax

```
SELECT COUNT (Attribute), Attribute
FROM table_name
GROUP BY attribute
HAVING COUNT (Attribute) condition;
```

Example

```
SELECT COUNT (CustomerID),  CustomerCountry
FROM Customer
GROUP BY CustomerCountry
HAVING COUNT (CustomerID) > 2;
```

Output

	"COUNT (CustomerID)"	CustomerCountry
1	4	France
2	3	Italy

Table 28 SQL Join code

Command	Explanation of the command
`JOIN`	The Join operator is used to combine rows from different tables when there is a common attribute between them. The common attribute is known as the foreign key. There are several joins you need to know. • (Inner) `JOIN` Returns all the records that have matching values in both tables. • `LEFT` (outer) `JOIN` Returns all the records from the left table that have matched records on the right table. • `RIGHT` (outer)`JOIN` Returns all the records from the right tables that have full matched records on the left table. Use this to collate information from across entities.

Syntax

```
SELECT table_name1.attribute1, table_name1.attribute2, table_name1.attribute3,
table_name2.attribute2
FROM table_name1
INNER JOIN table_name2 WHERE table_name1.attribute1 = table_name2.attribute2;
```

Example

```
SELECT Item.ItemID, Item.ItemDesc, Item.ItemCost, Item.ArtistID, Artist.ArtistName,
Artist.ArtistEmail
FROM Item
INNER JOIN Artist ON Item.ArtistID = Artist.ArtistID
WHERE ItemCost > 100;
```

Output

ItemID	ItemDesc	ItemCost	ArtistID	ArtistName	ArtistEmail
5859	Sculpture featuring two cats	239	101	Evie Ceramics	EvieCeramic@mail.com
4944	Abstract Wall Art One	384	102	Rojas Artwork	Rojas@mail.com
9383	Abstract Wall Art Two	332	102	Rojas Artwork	Rojas@mail.com
2992	Self portrait	192	102	Rojas Artwork	Rojas@mail.com
2921	Bracelet - fine	450	105	Alejandro Jewellery	Ale@mail.com
8593	Necklace with Rose	234	105	Alejandro Jewellery	Ale@mail.com
9282	Shell Necklace	120	105	Alejandro Jewellery	Ale@mail.com

Table 29 SQL Like code

Command	Explanation of the command
`LIKE`	The Like operator is used within the Where operator to look for non-exact matches. This is known as **pattern matching within the attribute**. There are two wildcards used within the database system. • `_` represents one single character. • `%` represents any number of characters (zero, one, or many). Use this when you do not know how many characters you are looking for. This operator can be used to extract all data when you are not sure of the exact match. The example below shows all artists containing the word "work". A join has been used to match the Artist information from the Artist table to all products from the Product table containing the word "work".

Syntax

```
SELECT Attribute1, Attribute2
FROM table_name
WHERE Attribute LIKE pattern;
```

Example

```
SELECT * from Artist
WHERE ArtistName LIKE '%work';
```

Output

ArtistID	ArtistName	ArtistEmail
102	Rojas Artwork	Rojas@mail.com
103	Heather Needlework	HNW@mail.com

Example

```
SELECT Item.ItemCost, Item.ItemDesc, Item.ArtistID, Artist.ArtistName
FROM Item
JOIN Artist on Artist.ArtistID = Item.ArtistID
WHERE Artist.ArtistName LIKE '%work';
```

Output

ItemCost	ItemDesc	ArtistID	ArtistName
384	Abstract Wall Art One	102	Rojas Artwork
332	Abstract Wall Art Two	102	Rojas Artwork
192	Self portrait	102	Rojas Artwork
29	Mountain Scene	103	Heather Needlework
45	Birds on the beach	103	Heather Needlework
29.75	The waves	103	Heather Needlework
15	Decorative Drum	102	Rojas Artwork
15	Personalised Tote Bag	103	Heather Needlework

Table 30 SQL And code

Command	Explanation of the command
AND	The And operator is used where there are two or more conditions that must be true before the data is displayed. For example, the customer must be based in Italy **and** the customer must have ordered an item worth more than 100 euros.

Syntax

```
SELECT Attribute1, Attribute2,
FROM table_name
WHERE condition1 AND condition2;
```

Example (without Join)

```
Select itemID, ItemDesc, ItemCost
FROM Item
WHERE ItemCost > 40 AND ItemDesc LIKE '%featur%';
```

Output

ItemID	ItemDesc	ItemCost
2948	Vase featuring face	45.85
5859	Sculpture featuring two cats	239

Example (with Join)

```
SELECT Purchase.ItemID, Purchase.Quantity, Purchase.Date, Customer.CustomerID,
Customer.CustomerEmail
FROM Purchase
INNER JOIN Customer on Purchase.CustomerID = Customer.CustomerID
WHERE Customer.CustomerCountry = 'Italy' AND Purchase.Date <'2025-07-01';
```

Output

ItemID	Quantity	Date	CustomerID	CustomerEmail
2921	2	2025-06-06	4	jordan@mail.com

Table 31 SQL Or code

Command	Explanation of the command
OR	The Or operator is used where there are two or more conditions that can be true and the data will be displayed. For example, the customer must be based in Italy **or** France.

Syntax

```
SELECT Attribute1, Attribute2
FROM table_name
WHERE condition1 OR condition2;
```

Example (without Join)

```
SELECT * from Customer
WHERE CustomerCountry = 'Italy' OR CustomerCountry = 'France';
```

Output

	CustomerEmail	CustomerAddress	deliveryAddress	CustomerCoun
1	3 Ian@mail.com	Woodside, Lakeview	Woodside, Lakeview	France
2	4 jordan@mail.com	Wordsworth way, Winderview	Main Street Lakeview	Italy
3	5 Vereena@mail.com	Apple Grove, Lakeview	The Wynd, Winderview	France
4	6 Zhi@mail.com	98 Westside, Midtown	98 Westside, Midtown	Italy
5	7 Miriam@mail.com	14 Rainton ave, Lakeview	14 Rainton ave	France
6	8 Siobhan@mail.com	5a Wisteria Ave, Hawthone	5a Wisteria Ave	France
7	9 Stella@mail.com	Via Roma 3, Vasso	Via Roma 3	Italy

Example (with Join)

```
Select Purchase.Date, Purchase.CustomerID, Customer.CustomerEmail, Customer.
CustomerAddress, Customer.CustomerCountry
FROM Purchase
INNER JOIN Customer on Purchase.customerID = Customer.CustomerID
WHERE Purchase.date BETWEEN '2025-07-28' AND '2025-07-31'
AND Customer.CustomerCountry = 'Italy' OR Customer.CustomerCountry = 'France';
```

Output

Date		CustomerEmail	CustomerAddress	CustomerCountry
2025-07-28	3	Ian@mail.com	Woodside, Lakeview	France
2025-07-28	4	jordan@mail.com	Wordsworth way, Winderview	Italy
2025-07-28	3	Ian@mail.com	Woodside, Lakeview	France
2025-07-28	4	jordan@mail.com	Wordsworth way, Winderview	Italy
2025-07-29	5	Vereena@mail.com	Apple Grove, Lakeview	France

Table 32 SQL Not code

Command	Explanation of the command
NOT	The Not operator is used to exclude certain data. If you know what you want to exclude rather than what you want to include, then this is the best option. This could be used if you want to **exclude** all customers from Italy and France.

Syntax

```
SELECT Attribute1, Attribute2
FROM table_name
WHERE NOT condition;
```

Example (without Join)

```
SELECT * from Customer
WHERE NOT CustomerCountry = 'Italy' AND NOT CustomerCountry = 'France';
```

Output

	Cus..	CustomerEmail	CustomerAddress		CustomerCountry
1	1	Emilia@Mail.com	87 The Grove, Midtown	87 The ..	UK
2	2	Christopher@mail.com	34 Maple Crescent, Midtown	Unit A ..	UK
3	10	Ryan@Mail.com	45 Ivy Street, Lakeland	Unit A ..	UK

Example (with Join)

```
SELECT item.ItemDesc, item.ItemCost, item.ArtistID, Artist.ArtistName
from Item
JOIN Artist on Artist.ArtistID = Item.ArtistID
WHERE NOT Artist.ArtistName LIKE '%Work%'
  AND NOT Artist.ArtistName LIKE '%Ceramics%';
```

Output

	ItemDesc	ItemCost	ArtistID	ArtistName
1	Lighthouse Print	15	104	Queenies Prints
2	The Bridges	25	104	Queenies Prints
3	Priory Print	17	104	Queenies Prints
4	Bracelet - fine	458	105	Alejandro Jewellery
5	Necklace with Rose	234	105	Alejandro Jewellery
6	Shell Necklace	120	105	Alejandro Jewellery
7	Glass candle holder	15	104	Queenies Prints

A3.3.3 Explain how SQL can be used to update data in a database

Indexes in databases are used to speed up queries. They provide a method to quickly look up the data. Rather than having all the data in the table, an index is a pointer to the data in the table—similar to an index in a book.

Indexes are required in very large databases due to the volume of data that is being processed at any given time. For example, imagine you are in a library but the information and books are not organized in any particular way. You would have to spend a long time looking for the information or book. In a normal library, all the data is indexed using a classification system. You can use the index to locate information quickly. Databases are similar. They contain thousands, millions, or billions of records, potentially containing petabytes of data. Being able to access data quickly is essential for companies to run efficient queries. Indexes help to serve as lookup tables to quickly retrieve data.

Databases store data in rows organized into tables. Each row has a primary key that uniquely identifies the data in the rows. These keys are stored in indexes to access the data quickly. Each time a new data item is created, the index needs to be automatically updated, and this can take time.

There are three update operations available in a database: adding, modifying, and removing. Every time each one of these updates is made, indexes need to be rebuilt and reorganized, which adds a significant overhead to the operation.

TOK

There are vast data repositories on the internet.

- Worldwide.espacenet.com is a database storing all the patents that have been filed and granted in the European Union. When querying the database, you can see all ideas that are protected and cannot be used without permission when developing new inventions.
- The Library of Congress in the USA which stores information about current laws in the USA, the USA copyright office, and the prints and photographs office. When querying this database, you can discover what is and is not within copyright, and you can find information regarding laws and how the laws are applied in specific situations.
- The gapminder.org database from the World Health Organization allows you to search development indicator metrics for all countries across the world and then download the data as a spreadsheet to manipulate. This data can show you factors, such as birth rate, GDP or the number of years students spend in school, for one specific country or for a whole region.

You have access to all of this information. When developing knowledge claims in TOK, consider: how can the data retrieved from databases be used to support the knowledge claims you develop?

The three main modification operations are outlined in Table 33.

Table 33 SQL code to add and modify records in a database

Operation	SQL command	Description

New record
Syntax
`NSERT INTO table_name (attribute1, attribute2, attribute2)`
`VALUES (value1, value2, value3)`
	`INSERT INTO`
	Example
	`INSERT INTO Artist (ArtistID, ArtistName, ArtistEmail) VALUES ('106', 'Clodagh Traditional', 'Trad@mail.com')`
		This operation is used to add a new record into an entity.
		ArtistID / ArtistName / ArtistEmail
		1 — 101 — Evie Ceramics — EvieCeramic@mail.com
		2 — 102 — Rojas Artwork — Rojas@mail.com
		3 — 103 — Heather Needlework — HNW@mail.com
		4 — 104 — Vantage Prints — Vantage@mail.com
		5 — 105 — Alejandro Jewellery — Ale@mail.com
		6 — 106 — Clodagh Traditional — Trad@mail.com

Operation	SQL command	Description

Modify record
Syntax
`UPDATE table_name`
`SET attribute1 = value`
`WHERE condition;`
	`UPDATE SET`
	Example
	`UPDATE Artist SET ArtistName = 'Vantage Prints', ArtistEmail = 'Vantage@mail.com' WHERE ArtistName = 'Queenies Prints';`
		The Update statement is used to update existing records in an entity.
		Output
		ArtistID ▽ : ArtistName ▽ : ArtistEmail ▽ :
		101 Evie Ceramics EvieCeramic@mail.com
		102 Hojas Artwork Hojas@mail.com
		103 Heather Needlework HNA2@mail.com
		104 Vantage Prints Vantage@mail.com
		105 Alejandro Jewellery Ale@mail.com

Remove record
Syntax
`DELETE FROM table_name`
`WHERE condition`
	`DELETE`
	`DELETE FROM Customer WHERE CustomerID = '4';`
		The Delete statement is used to delete specific records in an entity.
		Output

CustomerID :	CustomerEmail ▽ :	CustomerAddress ▽ :	deliveryAddress ▽ :
1	Emilia@mail.com	87 The Grove, Midtown	87 The Grove, Midtown
2	Christopher@mail.com	34 Maple Crescent, Midtown	Unit A, Industry Road, Midtown
3	Ian@mail.com	Woodside, Lakeview	Woodside, Lakeview
5	Vereena@mail.com	Apple Grove, Lakeview	The Wynd, Winderview
6	Zhi@mail.com	98 Westside, Midtown	98 Westside, Midtown

AHL

A3.3.4 Construct calculations within a database using SQL's aggregate functions

Aggregate functions allow you to perform calculations on the data in the database. Databases are used to support strategic decision-making within a company. They can also be used by small independent users to perform data analytics on their own data, such as: the average number of items sold, the maximum sales, the sum of the sales, and so on.

Analytic functions you can perform include average, count, max, min and sum.

Average: This finds the average of a value in the database.

Table 34 SQL code to perform averaging calculations on data in a database

Syntax	Example and description
`SELECT AVG(Attribute)` `FROM Table_Name;`	`SELECT AVG (Purchase.Quantity) from Purchase;` **Output** "AVG (Purchase.Quantity)" ▽ : 2.5454545454545454 This shows the average number of items purchased by each person in the database in the purchase table.

`SELECT AVG(Attribute)`
`FROM Table_name`
`WHERE Attribute = value;`
	`SELECT AVG (Purchase.Quantity)`
	`from Purchase`
	`Where CustomerID = 3;`
	Output
	
	
	This shows the average number of items purchased by the customer with the CustomerID 3.

Count: This returns the number of rows that match a specific criterion.

Table 35 SQL code to perform count calculations on data in a database

Syntax	Example and description

`SELECT COUNT (Attribute)FROM Table_Name;`	`SELECT Count (ItemID)`
	`FROM Purchase;`
	Output
	
	
	This operation counts the number of ItemIDs that appear within the purchase entity. This does include duplicates.

`SELECT COUNT (Attribute)`
`FROM Table_Name`
`WHERE condition;`
	`Select Count(ItemID)`
	`FROM Purchase`
	`WHERE Quantity>1;`
	Output
	
	
	This operation counts the number of ItemIDs that appear within the purchase entity when more than one item has been ordered. This does include duplicates.

`SELECT COUNT (DISTINCT Attribute)`
`FROM Table_Name;`
	`Select Count(DISTINCT ItemID)`
	`FROM Purchase;`
	Output
	
	
	This operation counts the number of itemIDs ItemIDs that appear within the purchase entity. However, this only counts the unique itemIDs, ItemIDs not the duplicates.

AHL

Max: This returns the highest value in an entity for a selected attribute.

Table 36 SQL to find the maximum values within data in a database

Syntax	Example and description

`SELECT MAX (Attribute)`
`FROM Table_Name;`
	`SELECT MAX(ItemCost), ItemID, ItemDesc`
	`FROM Item;`
	Output
	
	This operation selects the maximum value based on the item cost. The ItemID and ItemDesc are also displayed.

`SELECT MAX (Attribute)`
`FROM Table_Name`
`WHERE condition;`
	`SELECT MAX (ItemCost), ItemDesc, Item.ItemID`
	`FROM Item`
	`WHERE ArtistID = 103;`
	Output
	
	This operation selects the maximum value based on the items cost if the ArtistID is 103.

Min: This returns the smallest value in an entity for a selected attribute.

Table 37 SQL code to find minimum values within data in a database

Syntax	Example and Description

`SELECT MIN (Attribute)`
`FROM Table_Name;`
	`SELECT MIN (ItemCost), ItemDesc, Item.ItemID`
	`FROM Item;`
	Output
	This operation selects the minimum value based on the item cost. The ItemID and ItemDesc are also displayed.

`SELECT MIN (Attribute)`
`FROM Table_Name`
`WHERE condition;`
	`SELECT MIN (ItemCost), ItemDesc, Item.ItemID`
	`FROM Item`
	`WHERE ArtistID = 101;`
	Output
	
	This operation selects the minimum value based on the item cost if the ArtistID is 101.

Sum: This returns the total sum of a numeric attribute within an entity.

Table 38 SQL code to return the sum of data within a database

Syntax	Example and description

`SELECT SUM (Attribute)`
`FROM Table_Name;`
	`SELECT SUM(ItemCost)`
	`FROM Item;`
	Output
	
	
	This operation sums the item cost of all values in the item table. This shows the value of all items in the database.

`SELECT SUM (Attribute)`
`FROM Table_Name`
`WHERE Condition;`
	`SELECT SUM(ItemCost)`
	`FROM Item`
	`WHERE ArtistID = 105;`
	Output
	
	
	This operation sums the item cost of all values in the item table if the ArtistID is 105.

A3.3.5 Describe different database views

Databases are only useful when you can view the data within them. Database developers understand how to develop databases and construct queries to view the data. But database users are not necessarily developers, and constructing queries would add a layer of complexity that would not be user-friendly. **Views** help to bridge the gap between database developers and users.

Views in SQL databases are virtual tables. These tables do not store data—they are tools to show data from multiple tables in an organized way. Queries are used to gather data from multiple tables. This data is then placed into the virtual table to display the information to the user. These virtual tables constructed from the queries are known as views. When the query has been executed and data added to the virtual table, it is known as a materialized view.

Key term

View A view is a virtual table used to show information from the database to non-expert users.

ATL Social skills and thinking skills

To understand what database users need to see to make their job easier, you need a good understanding of other databases and how they work. This can then help to abstract the correct data into the views.

People interacting with a database are often not subject experts. For example, a school database includes all the information about the school, including pupils, teachers, non-teaching staff, curriculum subjects, school resources, and finances. The person using the database could be the administration assistant in the school office. They access the database to send messages to parents and to find teachers for cover. They need to see parent contact details and teacher timetables, but not grading information or pay information.

Imagine you have been asked to develop a database to store information for an online business selling gifts and greetings cards. Consider three different database users.

- A person shopping at the online store. They buy cards and gifts for their family and friends.
- A person working in the purchasing office. They make decisions about what cards and gifts to buy from suppliers.
- A person working in the customer services department. They deal with customer queries.

How could you discover what information would be useful for each type of database user? How could you design a view to help them see only what they need to see?

What should the view include? What should the view exclude?

Table 39 shows some advantages of materialized views.

Table 39 Advantages of materialized views

Advantages	Description
Views can be used to hide the complexity of the data	Database information is stored across multiple tables. Bringing the data together requires joins and complex queries. A view allows information to be presented so that the queries, joins and aggregations are hidden from the user. These results can also be tailored to show or hide columns of information dependent on the users' requirements. So, views hide the complexity of the database from the users to make their interactions with the database simpler and more user-friendly.
Views can be used to provide data consistency	Data consistency ensures data is accurate and up to date across the database. For a database to function correctly, data must be consistent. This maintains integrity, quality and reliability. If the data in a database is not consistent, then issues and anomalies arise. Views provide a single point of access to view data and ensure that all data is derived from the same place. If the underlying data changes in the entities, the results in the view will change. This ensures all users accessing the view have accurate, up-to-date data.
Views make use of independence	Database views are separate from the database schema, so the information users can access using the database views is separate to the database schema. The schema underlying the view is used to collate and display the data, but the view does not have access to this. This is a useful feature as it enables the database schema to be altered without affecting the structure and accessibility of the views. So, if any changes need to be made to the schema, the users will not experience any difference.
Views can be used to enhance database performance	Database queries make use of data from multiple tables. This is inclusive of joins, aggregations and complex calculations. Running a query can use significant processing power and slow down the database. A view pre-processes the complex calculations and aggregations and so can enhance the performance of a database. By constructing a view, the complex tasks are pre-completed, speeding up the processing and the overall performance of the database.
Views can be used to simplify queries	Relational databases rely on complex queries across multiple tables. They make use of joins, aggregations and calculations to provide key information from the database to the user. Using views, you can abstract the data but hide the details of the database schema away from people who do not need to see it. The results of the query are saved in a view that can be reviewed at any time.
Views can be read-only or updatable	Database views are usually read-only. The changes to the underlying data should only be made using the correct methods, which enable the database to maintain integrity. In very specific cases, views can be used to update a database. This is only possible if there is a one-to-one relationship between the view and the table underlying the view. Views cannot be updatable if there are aggregate functions, such as grouped by, or if they have used union or made use of subqueries. Changes to the underlying function will not be reflected in the view unless the view is recreated. As views are read-only, or updatable only in very specific circumstances, using views to access data helps to maintain the database integrity.
Views can be used to provide security	Companies have many different employee levels. Higher-level management can usually see all data within a database, while employees at an administrative level often have very restricted access to data. For example, users at a senior level may see details of employees' salaries, performance records and personal information. Employees at an administrative level may only need to see contact details. However, it is not practical to have separate databases for each level of employee in the system. Views provide a solution to this problem. A view can be created to gather all the data into one place, with confidential data hidden from the view so that each employee can only access the data they are allowed to access. Views limit access to the entities they have drawn the data from, ensuring confidential data remains confidential.

AHL

TOK

Programmers develop code within databases to help with decision-making. Examples of these decisions include whether someone can get insurance, whether someone is eligible for a loan, or whether someone is accepted into a school.

To what extent are programmers ethically responsible to share how these decisions are being made?

TOK

Abstraction hides the complexity from the end user. This means that the user does not see how the data in the view has been created. They do not see the raw data, the calculations that have been performed, or how the data has been filtered. This can be helpful if you only need to use the data provided, but it can limit your understanding of the data.

To what extent does abstraction in the database views hinder or support the production of knowledge?

A3.3.6 Describe how transactions maintain data integrity in a database

Consider a database that is selling train tickets. There may be an entity containing customer information, an entity containing train information, and a transaction table showing the tickets that have been sold and the customer who purchased them. If someone emailed the company to change their reservation, more than one employee could read the email and change the data at (roughly) the same time. If two people change the data, how do we know what data is correct?

User one edits

Transaction_ID	Customer_ID	Journey_ID	Date	Quantity
293884	8593020	BAT2902	05/06/2025	10
293885	5589022	BAT2902	05/06/2025	1
293886	5730201	NAI4910	05/06/2025	21
293887	5883901	NAI3939	05/06/2025	3
293888	8992020	NAI3939	05/06/2025	43

User two edits

▲ **Figure 13** An illustration of the problem when two people try to access the same data

You could solve the problem by only letting one person have access to the database, but this is not an effective solution as databases are only useful when they are multi-access. To solve the issue of multi-user access, databases have transactions that make use of ACID properties.

A **transaction** is a single operation that accesses, deletes or modifies the data within a database. Transactions access data using read, write and modify operations. To maintain consistency before, during and after the transaction, ACID properties are observed by the database management system.

Figure 14 shows the ACID properties.

Atomicity: The entire transaction must be completed, or the transaction should not happen at all. There can be no partial transaction.

Consistency: The database moves from one consistent state to another consistent state after the transaction occurs. For example, values must remain the same after the transaction occurs. Primary keys remain unique after the transaction has occurred. Foreign key relationships are maintained after the transaction has occurred.

Key term

Transaction A single operation in a database used to read, write, or modify data.

Atomicity	**C**onsistency
Isolation	**D**urability

▲ **Figure 14** ACID properties

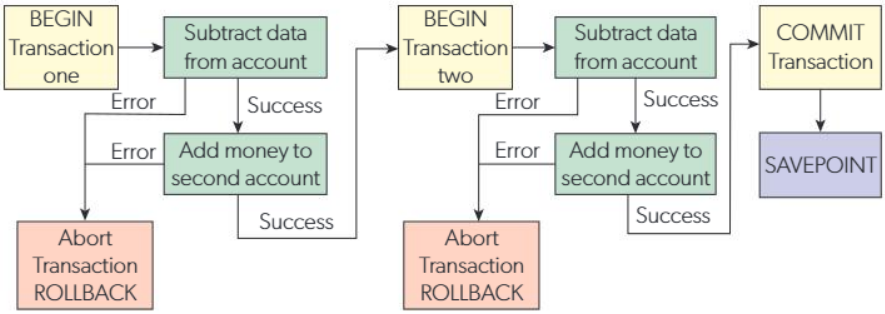
Isolation: Maintains the independence of database transactions. All transactions are isolated until all previous transactions have been committed. This ensures that consistency is maintained despite concurrent transactions occurring at the same time.

Durability: Provides guarantees for committed transactions. The system makes sure that all changes are committed even if there is a system failure.

The uses of ACID properties are as follows.

- Maintaining the integrity of the database as outlined by the database schema.
- Ensuring the overall consistency of the database after each transaction. This maintains the integrity of the database so all data in the database remains reliable.
- Eliminating partial and unsuccessful operations from the database, ensuring database integrity.
- Restoring the data to the last known consistent state using the rollback function, in case of database failure or system failure.

Databases are used to store details of banking transactions. This includes what money has been used for, how much money to debit from an account, how much money has been paid to someone, and how much money to credit an account. With many transactions happening on bank accounts it is important that the transactions are carried out correctly and that the data within the database can be relied upon. Bank account transactions are a classic example of how ACID properties function.



▲ **Figure 15** A bank account transaction

Bank account transactions make use of the transaction control language (TCL) and ACID properties to ensure transactions happen correctly.

- The transaction begins. The bank account that the money is being subtracted from is isolated. No other transaction can use this bank account. (ACID property: isolation)
- The money is removed from the account.
- The second bank account, that the money is being added to, is isolated. No other transaction can use this bank account. (ACID property: isolation)
- The money is added to the second bank account successfully. (ACID property: consistency)

- The transaction is successful and the change committed. (ACID property: atomicity)
- The records are released from isolation and available for use by other transactions. (ACID property: durability)

TCL is used to apply the ACID properties in a relational database system. TCL commands make sure the data is isolated and transactions occur correctly before the transaction is committed. If the transaction fails at any point, the rollback action occurs.

TCL is a subset of commands within SQL. There are four commands that you will find helpful within TCL.

Begin transaction: Identifies that the transaction has begun and the ACID properties should be observed.
SQL command: `START TRANSACTION;`

Commit: Used to save all transactional operations to disk. This means that the operations are permanent and can be viewed by all users of the database system.
SQL command: `COMMIT;`

Rollback: When carrying out a transaction and everything executes correctly, the transaction is committed and the changes saved. However, errors do occur, and when they do, the database can be rolled back (restored) to previous save points. The mistakes are rectified and the data is in a consistent state.
SQL Command: `ROLLBACK TO savepoint_name;`

Savepoint: Database operations can be divided into logical blocks.
For example, all insert operations can be completed at the same time and all delete operations can all be completed at the same time. The save point allows the database to be saved in logical increments.
SQL command: `SAVEPOINT savepoint_name;`

Advantages and disadvantages of using ACID properties in a relational database

Advantages

- ACID ensures **data remains consistent** before and after the transactions occur. This is important because databases are used to make decisions. It is essential the data is consistent at all times.
- ACID properties **maintain the integrity of the data**. If ACID properties were missing from a relational database, the data in the database could be lost. Unreliable databases lead to unreliable data, and the database would not be able to support the operations it was designed for.
- ACID properties **enable multiuser access to the database**. The isolation properties enable transactions to be completed concurrently. Having many people being able to access the database supports businesses using the database.
- ACID properties **enable recovery**. Savepoints and rollbacks enable data to be restored, even if there is a system failure or a database crash. As the data integrity can be guaranteed even after a system failure, ACID properties help to maintain accuracy of data.

TOK

ACID properties are used to make sure only one transaction at a time can access a record. This prevents conflicting actions happening within the database. When ACID properties are in place, the database is in a stable form.

To what extent do ACID properties ensure the data returned by a database query is true?

Disadvantages

- Maintaining the ACID properties on a database can cause **performance overhead** in the system. The transactions required additional processing to maintain the ACID properties. In small databases this is not a big problem, but in larger databases it becomes more of an issue.
- ACID properties cause some **scalability challenges**. When multiple transactions occur concurrently over large databases, this can cause added complexity.
- Implementing ACID properties **increases complexity** in a system. Expertise is required when developing and maintaining the database, so more resources are required overall.
- Ensuring data integrity in the database using ACID properties requires **increased management time**.

Although the disadvantages are significant, they are usually outweighed by the advantages of having a fully integral database.

TOK

One of the most important functions of a database is to support decision-making. Data stored in medical databases helps doctors to make decisions regarding treatment options for patients. Data in banking databases helps bank staff to decide what credit and saving options to offer to their customers. Data in university databases helps administrators to make decisions about current and future student offers.

Considering the importance of decisions being made using data in databases, what ethical responsibilities do developers have to ensure the data represented in the database is up to date and accurate?

Practice questions

11. Construct an SQL statement to add the following entity to a database:
`Crime (Crime_ID(PK), location, type, witness)` [3 marks]
12. Construct an SQL statement to find the average number of products ordered in the following entity:
`ORDER (Order_ID(PK), Item_ID, Item_name, number_of_products_ordered)` [3 marks]
13. Describe two advantages of using views in a database. [4 marks]
14. Describe the role of ACID in maintaining the integrity of database transactions [4 marks]
15. Describe two disadvantages of maintaining ACID properties in a database. [4 marks]

A3.4 Alternative databases and data warehouses

AHL

Syllabus understandings

A3.4.1 Outline the different types of databases as approaches to storing data

A3.4.2 Explain the primary objectives of data warehouses in data management and business intelligence

A3.4.3 Explain the role of online analytical processing (OLAP) and data mining for business intelligence

A3.4.4 Describe the features of distributed databases

AHL

A3.4.1 Outline the different types of databases as approaches to storing data

A relational database is one model of storing databases. However, there are several other types of databases you need to know about in this course. You need to be able to know how they work and in which situations they may be used.

NoSQL model

In contrast to relational databases storing data in multiple related tables, **NoSQL** databases store data using documents. For example, instead of storing book information in a book table which is related to the author table and then the publisher table, all books would be stored in separate documents combining all the book information together. NoSQL databases make use of flexible schemas and can scale well so are best to use with large volumes of data and high user loads.

Examples of flexible schemas include the following.

- Embedding information that is related to a document inside the document. For example, if you want to track employees within departments you can store department data within the employee document.
- You can use documents to store reviews. If you have a website, you can embed recent reviews but store older, less relevant reviews in a different model.
- When displaying products on a website, users want to know many different things about the product. Using a document, you can store all the attributes together.

Platform as a service (PaaS) model

The **platform as a service (PaaS)** model is commonly referred to as a **cloud-based implementation**. A cloud-based database is a database system that follows a traditional structure but is hosted on a remote computing platform. This enables the database to be accessed across the globe and, depending on user requirements, it can scale dynamically. As with all cloud-based implementations, accessing a database through a network may potentially create a bottleneck (latency) and thus slow down service.

TOK

Originally, databases started as index cards where users hand wrote the information and stored them in a filing cabinet. This moved to storing all the information within one table and then normalized databases. Now we have normalized, unnormalized, flat file, cloud, and distributed databases.

How can we know that the development of new database models is an improvement on past models?

See section A1.1.9 to review the different types of services in cloud computing.

When you use a cloud database, the database is hosted online and networks are utilized to speed up access to the database across the globe. Cloud databases are often used to facilitate data access over wide geographical areas. For example, if a bank used a database to house financial data, the further away the access point is from the network, the longer it would take to access the database. Using a cloud-based database enables the database to be distributed around the world, speeding up the process.

Cloud-based databases allow users to:

- dynamically scale (expand or contract) quickly and easily when necessary
- span the globe effectively using distributed data centres
- manage the database themselves or through a third party
- easily back up data.

Data stored on cloud-based databases is more likely to be professionally managed. This means that robust backup and restore features are in place, and if issues occur, there will be less downtime.

Spatial database model

Spatial databases are optimized to store information about geographical data such as points, lines and polygons. They can also handle complex 3D objects and topological data. The purpose of spatial databases is specifically to store and manipulate this geographical information. They are usually known as geographical information systems (GIS). GIS databases allow SQL statements to query, analyse and manipulate data with several enhancements specific to geographics.

Examples of these include the following attributes.

Measurement: The ability to calculate line length and distances between boundaries.

Geoprocessing: Modify current features to create new ones.

Query geographic relationships: For example, do two boundaries overlap?

Construct new geometrics: For example, by specifying boundary nodes.

Query specific information about a feature: For example, finding the centre point.

In-memory database model

As the name suggests, an **in-memory database** keeps all of the stored data within the database, in the RAM of the computer system. Traditional database systems keep the data stored on the system back-up storage: it is retrieved on demand. As in-memory databases store all the memory in RAM, they are faster to run than traditional databases, and fewer read/write instructions are required by the CPU. As a result, in-memory databases are used for real-time, faster processing. The downside of in-memory databases is that if the database crashes, all of the data will be lost. Non-volatile random access memory (NVRAM) can help, but the number of read/write cycles to NVRAM are limited.

In-memory databases are commonly used for:

- real-time processing applications such as medical devices, machine learning or real-time banking
- online interactive gaming
- processing sensor data
- e-commerce applications
- geospatial processing.

A3.4.2 Explain the primary objectives of data warehouses in data management and business intelligence

Businesses work best when they understand the behaviours and requirements of their clients. For example, supermarkets often offer loyalty cards to customers to encourage customers to use that supermarket brand more often than their competitors. However, the use of these points cards also enables supermarkets to collect millions of data points about consumers. This data can be used to identify new trends in spending, changes in the behaviours of shoppers, and products that are losing appeal over time.

By analysing the data within their business, supermarkets can identify the products that customers want the most and buy and sell these at lower prices as the higher volume of sales protects their profit margins. The supermarket can offer niche products at a higher price point to maintain their profit margin on lower-volume goods. Offering discounts on everyday products and personalizing deals for consumers ultimately boosts the supermarket's profits. **Data warehouses** enable this level of analysis to occur.

Key term

Data warehouse An append-only data repository, used to house historical data.

Data warehouses are database management systems that are solely interested in performing analytical functions. They are not involved in transactional processing. Data warehouses only work with historical data that has come from a wide range of sources, including log files and transactional files. As data warehouses store large amounts of data from many different sources, they are very good for aiding decision-making.

ATL Thinking skills

Identifying patterns in data helps developers to understand the links between items and make effective business decisions: what products to invest in, what products to retire, what geographical locations are experiencing growth and need more resources.

Imagine you have been asked by a local business owner to develop a database to help them understand their business in more detail. The business is a fruit and vegetable shop. It gathers produce from local farms to sell at its shop in the local market town. It also offers a fruit and vegetable subscription service: customers sign up and get vegetable boxes delivered to their door.

With a partner, analyse the data and complete the following tasks.

- Identify the data that needs to be stored. You may find it useful to use an ERD.
- Identify the relationships between the different entities.
- Identify the business decisions that can be made using the information you will collect.

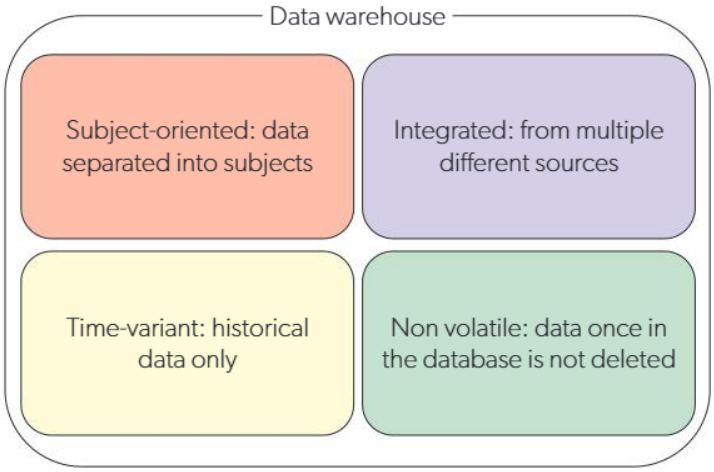
Compare your work with another group. What can you learn from their activities? What can they learn from you?

As a bigger group, consider to what extent companies should tell their customers what data they collect and how this data is being used to make decisions.

Companies use data warehouses as they offer the following advantages.

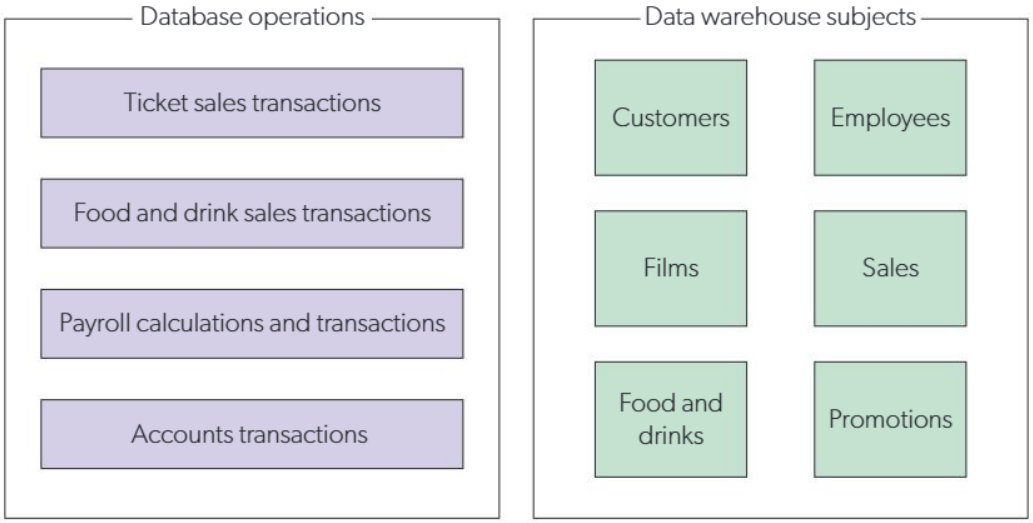
- They enable integration of many different sources of data.
- Data in the warehouse does not change, it is only historical.
- Data can be analysed about one particular subject or area of the business.
- They can analyse changes over time.

Data warehouses are used in companies to help build reports and complete analysis. They are there to maintain the organization's historical information and be used as a source for decision-making.



▲ **Figure 16** The different components of the data warehouse

Data warehouses are subject-oriented: Databases that are not data warehouses are often used to carry out transactions and complete queries. These databases store information in many different entities. When transactional databases are queried, the query draws data from different entities using joins. Data warehouses separate data into subjects, so you can analyse by subject rather than by transactional data. This is shown in Figure 17. In this example, you can see the difference between transactional operations and data warehouse subjects in a cinema database.



▲ **Figure 17** The different operations and subjects within a cinema database

By separating the data into subjects, businesses can perform analytics on different areas of the business. By identifying trends in different areas, they can determine where best to invest and make adjustments to improve the business.

Data warehouses are integrated: They gather data from multiple different databases and logs across the system. To gather all the data safely in a data warehouse, the data needs to go through a data cleaning process and get transformed into data that is readable by the data warehouse. This is known as the extract, transform and load (ETL) process.

- **Extract:** Data is gathered from many different sources across the system.
- **Transform:** Data is transformed into a form that the data warehouse can read. This stage includes data cleaning and data filtering.
- **Load:** The prepared data is loaded into the data warehouse.

Data cleaning is required before extracting. Data cleaning techniques that may be utilised at the extract phase include the following.

- **Data normalization:** Data is organized into a consistent format matching the data in the warehouse. (This is not the same as extracting data into different tables.)
- **Data cleansing:** Errors that are apparent in the database are identified and corrected.
- **Data filtering:** Unwanted data in the data set is removed.
- **Data validation:** Data is checked to make sure it meets the restrictions in place in the data warehouse. This is to minimize inconsistency in the database.

Data warehouses are time-variant: Data warehouses are append only. This means you can write to them but not delete from them. Historical data is kept in a data warehouse meaning you can view files from one month ago, six months ago, one year ago, or even longer. This is different to transactional databases, which only keep the more recent transactions. Keeping historical data enables companies to look for trends in data, which is valuable when it comes to strategic decision-making.

Data warehouses are non-volatile: A data warehouse is physically separate to the databases it gathers data from (the source databases). The source databases are constantly being read from, amended, and having data deleted. They are volatile (changeable) data sources. In data warehouses, the data is loaded into the warehouse, then added to. No changes are made to the data once it has been added to the warehouse. In this sense, the data warehouse is non-volatile, meaning it is non-changeable.

For businesses, there are many benefits of using a data warehouse.

- Understanding the patterns and trends within the data, which allows better future predictions (forecasting) and planning.
- Managing the enormous volumes of data that a company generates.
- Data warehouses are more user-friendly than traditional databases for business analytics and decision-making queries.
- Allowing multi-user access to data, as the data cannot be changed once added to the data warehouses.

Data warehouses are used for analytical processing and data mining, which you will learn more about in section A3.4.3.

TOK

Data is collected and placed into data warehouses. Analytics are then performed on the data. This data is used to make decisions within businesses.

Consider a health insurance company. It collects data from its customers about which healthcare professionals they work with and what health devices they use or wear. The insurance company can use this data to provide analytics about customer lifestyles, possible health problems, and possible future medical needs.

Companies collect data and use the information in data warehouses to make decisions. Based on ethical considerations, what kind of decisions should not be made using data in data warehouses?

AHL

Key term

Online analytical processing (OLAP) Database analysis technology used to query, extract and study data summarized from a database.

ATL Thinking skills

OLAP is a key tool used for making decisions when using databases. Many of these decisions are made using AI tools.

To what extent should you trust the data being produced by AI-enhanced OLAP tools? When using AI-enhanced OLAP tools, what features of the data produced would allow you to trust the data? What features would make you wary of it?

Share your findings with a partner or a small group. Do they agree with you?

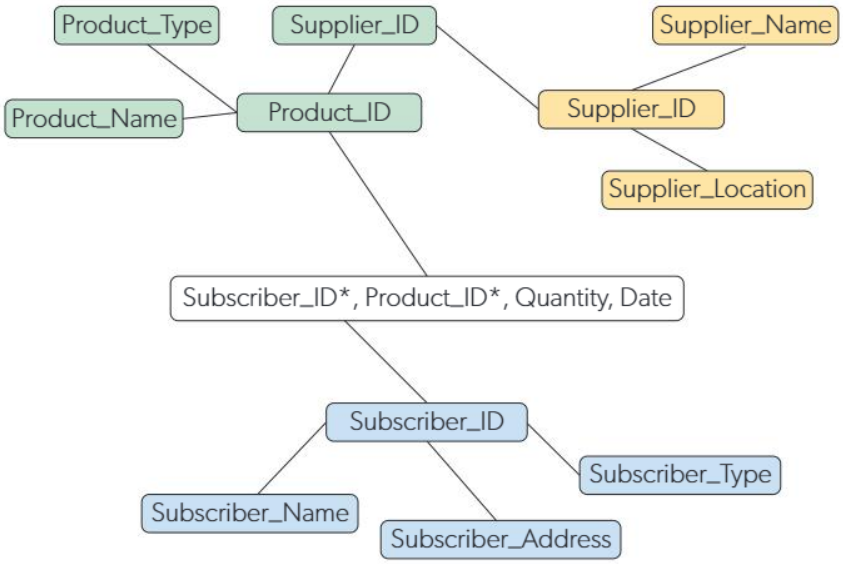
A3.4.3 Explain the role of online analytical processing (OLAP) and data mining for business intelligence

What is online analytical processing?

Online analytical processing (OLAP) is a key tool for identifying information within a data warehouse. OLAP uses a star or snowflake schema to represent data in a multi-dimensional table, known as an OLAP cube.

Storing the data in a cube format allows the data to be queried in multiple ways. The data can be queried as a whole, or along any single axis (X, Y or Z). Examining the data from multiple angles can provide the user with many different interpretations. These different interpretations can then be used to support decision-making.

To create a OLAP cube you need to link the data together. This can be completed using primary keys and foreign keys within the database schema. For example, consider the following database. There is a transaction entity storing data regarding the sales within a company. The transaction entity employs foreign keys to link it to the data about the product, the supplier and the subscriber who purchased the product. The foreign keys are used to gather the additional attributes from the related table to bring all the data together in a multi-layers format (the OLAP cube). This is shown in Figure 18.

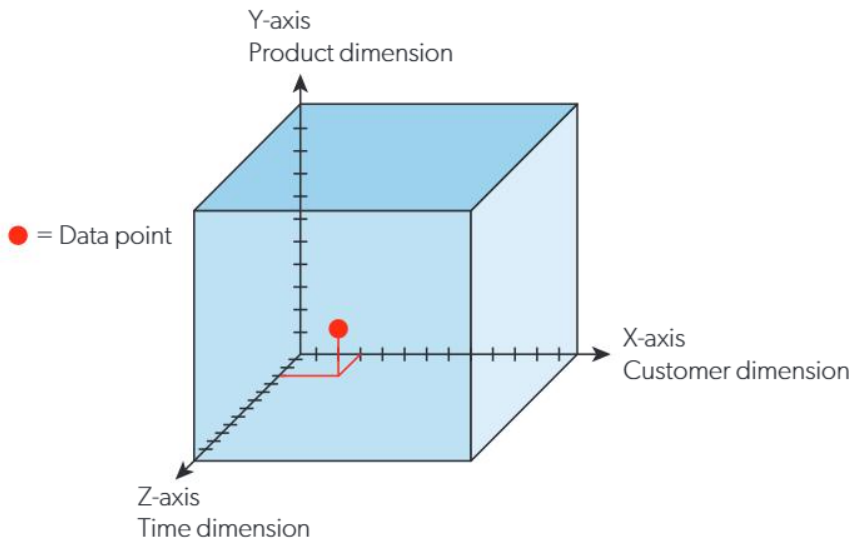


▲ **Figure 18** Visualization of an OLAP cube

The OLAP cube is central to the OLAP system. It is important that you understand how this data has been collated when you think about data analytics within the OLAP framework. OLAP is pre-processed and organized before analytics occurs. This contrasts with data mining, which only makes use of raw data. OLAP and data mining can be used in tandem. Data mining tools are used to analyse user behaviours from raw data within a data warehouse. OLAP software is then used to inspect and draw conclusions by looking at the data from different angles. Using both data mining and OLAP in combination supports effective decision-making with a company.

How do OLAP systems work?

Data in data warehouses is collected from multiple sources. Data in the data warehouse is cleaned and, when using OLAP systems, pre-processed and organized into data cubes (known as OLAP cubes). An OLAP cube is a cube of data organized on different axes. One aspect of the data will be organized on the X-axis, one on the Y-axis, and another (usually time) on the Z-axis. An example is shown in the cube in Figure 19.



▲ **Figure 19** Example of an OLAP cube

Each cube is organized on multiple axes and the cubes are then organized into different sections known as dimensions. Business analytics make use of the different dimensions to query different areas that the data warehouse contains. This gives the decision-makers access to many different perspectives on the data. Companies might use the following dimensions.

Customer data dimension: Storing all the data about the company's customers, including demographics, geographic location, browsing and purchasing history.

Geographical data dimension: Storing all the information about the geographic location where purchases are being made, including location, demographics, purchases in the area.

Time period data: Storing information including what purchases were being made, what adverts were being run, and what was trending during a given time period.

By pre-curating data into these categories from different tables and multiple sources, the data warehouse queries are more efficient.

OLAP uses five different operations to find information from the data warehouse. These operations provide information that supports effective decision-making within companies. There are five types of OLAP operations that can be performed on data.

Roll-up: This function analyses and summarizes the data within the cube. Using the hierarchical structure, the data at the bottom of the cube is summarized and then the data on the next level is added to the summary. For example, a count of an item on the bottom layer is rolled over to the next year and the final count from the cube is presented to the user.

Drill-down: This function allows analysts to look closely at the dimensions of the data. Analysts use this function to search for specifics. For example, looking across time periods to understand the growth or decrease in demand for a product over time.

Slice: Data in the cube is organized in a logical manner. A cube containing all the information about location may contain one location on the first slice, a second location on the second, and so on. This is helpful if you want to look at trends across all locations. However, the slice function allows you to look at one specific slice and examine specific trends in that area.

Dice: This allows analysts to select multiple dimensions across the cube. As the cube is organized with all data from a specific domain, we can look carefully into the cube for specific information. A cube may contain all the items for sale in one location, so using the dice operation on the data in the cube could allow us to find all the sales for one specific item in a specific region.

Pivot: This allows analysts to get a new insight into the data by viewing it from a different dimension, by rotating the data cube on a different axis.

OLAP is used to support effective decision-making within businesses. The main way OLAP can support this is by providing the information to support these activities.

Sales reporting: Companies that have sales as a core business can use sales reporting to help make decisions. The more sales a company makes, the better its profits will be. OLAP can help to identify which products are selling in different markets. By identifying the products that are selling well in each market, a company can ensure that it has enough supply in that location to meet the demand.

Marketing: Successful marketing campaigns can generate more custom and so boost company profits. OLAP enables companies to analyse their marketing campaigns, looking at potential successes and areas for improvement. Understanding the marketing that works best for the company can help to improve the overall performance of the business.

Management reporting: Business decisions are often finalized by management teams. Data is usually required to support any strategic decisions. The analytical tools provided by OLAP can be used to develop reports for management, showing the different trends in the company and possible outcomes of any new trends.

Processing management: Effective businesses rely on processes working correctly to make sure their products and services are in the right place at the right time. Understanding OLAP can be used to identify weaknesses in processes. The information learned from this can be used to adjust the process, improving the overall efficiency of the company.

Budgeting: Ensuring cost effectiveness can help a business maximize its profit. Using the insights from OLAP can help businesses to analyse weaknesses and strengths in their marketing costs, production costs, and logistics costs. Ensuring that the right products are in the right place at the right time helps to manage budgets.

Forecasting: This is one of the most powerful business tools. Forecasting is the process of using past data to try to predict future consumer behaviour. Using past trends identified by the OLAP system allows businesses to forecast future trends, consumer habits, and logistical requirements. This allows them to identify how best to invest in the future to maximize their profit.

Data mining

In OLAP, the data is pre-curated into cubes of data based on different subjects. This allows for more efficient data querying. Data mining requires no pre-processing of the data. The raw data is used to identify patterns and trends within the data warehouse, so that useful information can be extracted. Companies use data mining to make business decisions, helping them to develop effective marketing campaigns, increase their sales and minimize costs, ultimately increasing their profits.

Data mining programs use data warehouses to analyse patterns and relationships in data. Companies can use data mining to discover browsing trends, purchasing trends and engagement trends organized by location, demographic and product type.

TOK

You may have learned in your TOK class that the interpretation of data can be open to bias. Bias from the person interpreting the data or bias from the person interpreting the results. OLAP cubes allow data to be examined from many different perspectives, helping to identify previously unknown links between data points.

Can the information produced by OLAP change established values or beliefs?

This problem in practice

Have you ever opened up your social media accounts and wondered how the adverts knew exactly what you were thinking? This could be a result of data mining. Social media accounts are very valuable in terms of consumer data. The accounts can track who you engage with, what posts you are interested in, and what your likes (and dislikes) are. They can see where you have been and who you have been with. Although you cannot see the links immediately, the data mining behind social media can discover common patterns between your actions and the actions of those you interact with. Using analytical tools, an algorithm can then determine what your current interests may be and then send you adverts to sell you products related to this interest. If you have been talking about buying new trainers with some of your friends, your friends might have searched for different trainers… and now you are seeing adverts for trainers!



▲ **Figure 20** Data collected through social media can be used to determine your interests

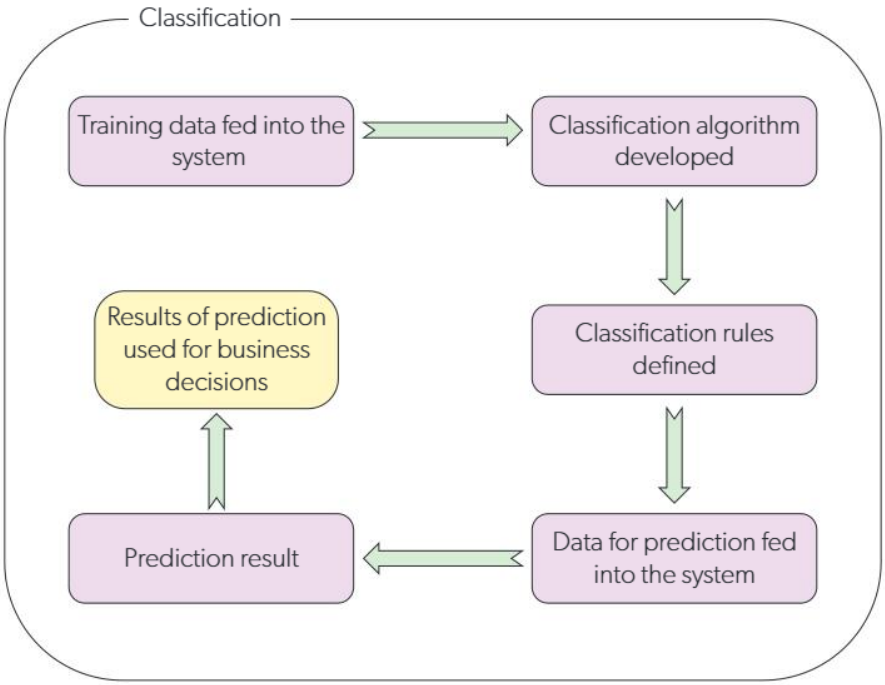
Data mining makes use of the following techniques to uncover these hidden patterns and trends: classification, clustering, regression, association rule discovery and sequential pattern discovery.

See topic A4 for more information about the mechanics behind data mining techniques.

Classification

Classification is using information about a product to label an item in a specific way on a superficial level. This is similar to classifying each item on your desk using the labels writing tool, technology tool, reading material, and so on. Within the classification process, a model is developed using known items. Each item is fed into the system and labelled correctly. This information is then used to train the system. Once the system is fully trained, unknown items can be added to the data set and these will be assigned the correct label by the computer system.

Labels can be used to match items to consumers. The classification process extracts data from the warehouse to develop a model, placing the data in different segments and then using this data to predict future trends. A simplified model of the classification process is shown in Figure 21.



▲ **Figure 21** A simplified model of the classification process

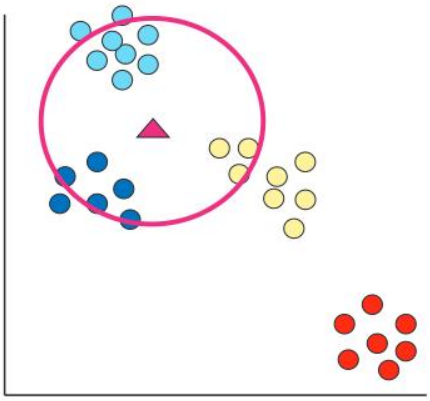
Companies use classification along with data warehouses to use past data to inform future successes. Training data is used to classify items, and consumers can also be classified in a similar way. These classifications can be used to match items with potential customers. When a new item is added to the model and classified, this information can be used to match the item with potential customers. Marketing campaigns can then be created specifically for these markets. Another example could be past marketing campaign information that can be fed into the system along with how successful they were in certain demographics. New marketing ideas can be fed into the system and classification can determine the best way to get this campaign to consumers, for example, whether to use social media or to run traditional adverts on television.

Clustering

Clustering is a technique used to group different objects together based on seen or unseen patterns. Clustering items together allows you to discover which items are similar and which are different from each other.

Using this information can help companies to determine potential customers for products. Figure 22 shows a visual representation of clustering. An unknown item (shown here as a red triangle) is added to the data set. The clusters it most closely aligns with are most like the product (that is, the data within the circle).

Clustering can help a company identify where a new product or service may sit among its current customer base. A new product could be introduced to the data and this will be clustered within the current data. The clusters closest to the new product will indicate possible target markets for the new product. This can be cross-referenced with other techniques to plan the best marketing strategy for the product.

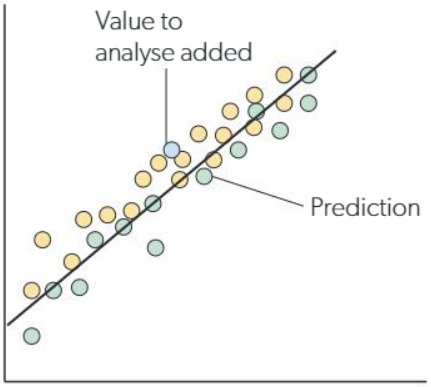


▲ **Figure 22** Unsupervised machine learning used to develop clusters

Regression

Regression is a supervised learning technique used to predict the numerical value of one item based on another item. This works for any continuously valued attribute. Regression is usually used for trend analysis and financial forecasting. Financial forecasting is important for companies as it helps them to plan for the future. Research and development budgets, staffing, marketing budgets and similar are all decided based on financial and future financial planning.

Companies can use regression when trying to make a prediction about profits. Using past data from sales and the profits at that time, companies can introduce different sales data to predict the profits that would be made if the sales figures were reached.



▲ **Figure 23** An image showing the linear regression trend

Association rule discovery

Association rule discovery is when the data is explored to find associations, rules and patterns between the items in the data. These rules could be expressed as "if X, then Y is likely". These rules are important as they can help to discover previously unknown trends within data.



▲ **Figure 24** The association rule discovery process

Companies can use association rule discovery to determine the likelihood that a product will be bought. By using association rules, companies can target special offers to increase the likelihood the products will be purchased.

TOK

Association rule discovery uses information in data warehouses to show links between products. Sometimes this is correct—if a person buys car cleaning products then it may be good to recommend car air fresheners as it is likely they are keeping their car clean. However, data mining tools may not account for cultural factors. For example, during sporting events, people may buy party food and celebration decorations to show support for their team. But this is not an association that exists all the time, so constantly marketing decorations containing country flags and party food may not be a good idea.

What features of information produced by data mining can have an impact on its reliability?

AHL

This problem in practice

Association rule discovery in supermarkets

Supermarkets use information gathered from loyalty cards and loyalty apps to tailor special offers to customers, making them more likely to purchase a given product. Association also helps to determine prices. If you sell two products at a lower cost knowing the customer is likely to buy the third, you can increase the price on the third. For example, in the summer, supermarkets might put small discounts on sausages and barbeque sauce but increase the price of charcoal for barbecues. Customers think they are getting a good deal and remain loyal to the brand but profits remain unchanged.



▲ **Figure 25** A supermarket loyalty app, used to track user shopping habits

Sequential pattern discovery

You may understand a sequence as a list of logically ordered items. The sequential pattern discovery tool in machine learning uses algorithms to discover unknown sequences within data warehouses. This allows companies to predict the next movement of customers.

	Link 1	Link 2	Link 3	Link 4	Link 5	Link 6	Link 7	Link 8	Link 9	Link 10	Link 11	Link 12
User One	o	x	x	x	x	o	o	o	x	x	x	x
User Two	x	x	x	x	o	o	o	o	x	x	o	o
User Three	x	x	x	x	x	o	o	o	x	x	o	o
User Four	x	x	x	x	x	o	o	x	x	x	x	x
User Five	o	x	x	x	o	o	o	x	x	x	o	o
User Six	x	x	x	x	o	o	o	o	x	x	o	o

▲ **Figure 26** The sequential pattern discovery process

This problem in practice

Fraud detection

When you use a bank card to spend money, your bank can see where you spend the money and the amount you spend. If you suddenly start spending money in a new, different location, this may be flagged as suspicious activity. If you continue to spend in your home location and a different location at the same time, this is an indication of fraud and your bank may call you.

However, if your location changed from your home location to a train station and onto a new location, the bank's pattern-matching algorithm may not flag this as suspicious activity. This pattern sequence suggests that activity in a train station or airport indicates a move in location is likely.



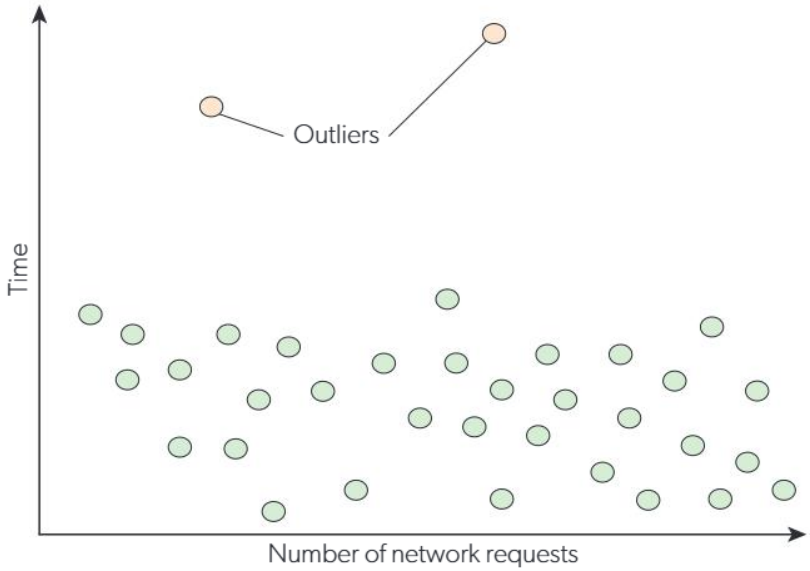
▲ **Figure 27** Withdrawing money from an ATM

Other uses of sequential pattern discovery include the following.

Website navigation: Pattern sequencing can be used to analyse clicks on a website. Areas of high activity tend to have more engagement, and companies can use this information to understand where to place information they want users to see. Pattern sequencing can also identify what websites people are likely to visit next based on their behaviour.

Social media analysis: As social media contains a wealth of data regarding subscribers and their habits, companies can use sequence pattern discovery to analyse trends in consumer behaviour, discover content people find engaging, and learn how people navigate the internet. This can lead to more targeted marketing and products more suited to people's lifestyles.

Anomaly detection: Often, outliers in a data set signify something abnormal is happening within the data. If outliers are identified then investigations can be made to determine the cause.



▲ **Figure 28** Outliers within a data set

TOK

These are four types of bias that can occur when data mining.

Selection bias: When choosing the training data you do not select a sample accurately representing the population.

Confirmation bias: The data mining technique has been used to confirm a preconceived theory rather than being allowed to discover theories freely.

Measurement bias: Inaccurate data collection measurements have been used to gather the data leading to incorrect results in the final product.

Overfitting bias: The results from complex models and have been forced to fit too tightly into the generalization.

To what extent is bias inevitable when using data mining tools? What could you do to counteract this bias?

This problem in practice

Anomaly detection in manufacturing

When medication is manufactured, each stage is carefully monitored to ensure the product is within the accepted limits and therefore suitable for use. If the product is not within the limits then an outlier is detected—this could indicate an issue with the machinery, or contamination. The current batch of the manufacturing process is not released to the public, keeping consumers safe.



▲ **Figure 29** An example of a manufacturing process

AHL

TOK

Many tools help individuals and companies make decisions using data within databases. They all work in different ways to produce results.

How important are tools that produce information from data mining in the production or acquisition of knowledge?

TOK

With data mining tools, machines can analyse data finding patterns that were previously unknown. They can be used to analyse data to maximize sales, find fraud, understand human behaviour, and predict human behaviour. They are able to do this faster than humans and, in some cases, without the users understanding how the decisions have been made.

How have the technological tools within data mining extended human capacity to make observations and judgements about each other and phenomena in the natural world?

One of the key requirements for a distributed database is a stable network that is fit for purpose. See section A2.1.2 for more on this.

Key term

Distributed database A database that has been segmented, with the segments housed in different locations.

Anomaly detection can also be used to uncover problems.

Finance: Like sequential pattern recognition, if someone is spending money in one location within their usual pattern of spending but at the same time a large amount of money is removed from the account in another location, this is an outlier and would be flagged as possible fraud. Another outlier could be if there is suddenly a large number of people placing money in the same stock on the stock market, outside their normal pattern of investment.

Cybersecurity: Data usually travels around the network normally. Data flow is generally steady and adheres to the given protocols. However, a significant increase in data movement or a sudden change in the protocols being used would show as an outlier. This could indicate a potential data breach or the presence of malware.

Applications of data mining

Marketing: Successful marketing campaigns understand their target market and what they are looking for. When companies understand the target market responses to media campaigns, they can tailor adverts for them. Data mining helps to understand the market, what they respond to and what they ignore.

Sales: One of the key uses of a data warehouse is to understand trends in sales. What products are selling well and in which location? What products are not selling in certain locations? Using this data, companies can make sure the correct products get to the correct place when required.

Fraud detection: Data mining uses techniques to map links between data to look for information. If there is no link to be found then it is an indicator to the company to further investigate the data to look for possible fraud.

Human resources (HR): HR departments have lots of data about employees including training records, performance and, if appropriate, why they left the company. Mining this data can help companies put together better packages for people entering the company to encourage them to stay.

Customer service: Companies rely on their reputation but, unfortunately, things can go wrong. Data mining can be used to find common solutions to problems and use these to populate answers on websites and chat bots to help resolve queries quickly.

A3.4.4 Describe the features of distributed databases

Databases are often used to house data used to support businesses across the globe. Traditionally, companies operated in one geographical area with minimal data. Now, companies operate worldwide with an increasing reliance on data. Transferring data from one location to another can be inefficient as the more data you have to transfer, the slower the transactions become. Therefore, **distributed databases** are used. A distributed database is a database that is not limited to one single system. The system is split over different sites. The distributed databases can be housed on multiple computers, sometimes housed on physical machines or sometimes housed on the cloud.

The key thing for distributed databases is that they need to look and feel like one single database.

There are two types of distributed databases: homogenous and heterogenous.

Homogeneous database: A database in which all different sites use the same database schema, house the same data, and use the same operating system and database management system. The databases in each location are the same and are therefore slightly easier to manage.

Heterogeneous database: A database in which the sites may use different schemas. The operating systems and database management systems used may also be different. This can lead to challenges with queries and transactions, and often an intermediate translation system may need to be used.

Databases can be separated using fragmentation or replication.

Fragmentation: The database is split into smaller parts and the fragments are stored at different sites. The fragments can be brought back together to recreate the whole.

Replication: The entire database is stored at each site. This means that all sites have copies of all data. This enables the availability of all data at all sites. However, it does come with a processing overhead as concurrency control has to be very carefully managed in a distributed database.

This problem in practice

Consider a large hotel chain with many different hotels located all over the globe. Different users of the system need different data.

- The hotel group management are the team who make strategic decisions based on data across the globe. The hotel group management would be interested in knowing which locations have a growth market and which locations are not experiencing growth, as well as profit or loss across the whole chain.
- The management and employees of a single hotel will only care about that one specific hotel, the rooms that are occupied, upcoming bookings, and any events that are going to occur.

Distributed databases can be useful in this situation. Each individual hotel could have its own section of the overall company database, for data relating to bookings and events at that specific hotel. In this case, a fragmented database is most likely to be used, where each hotel manages its own fragment of the database.

Additional uses for a distributed database include military control systems, corporate management systems, air traffic control systems and manufacturing control systems.



▲ **Figure 30** A large hotel chain may use a fragmented database

AHL

TOK

Distributed databases are used to store information around the world. This includes hotel chains storing information about employees and customers, and pharmaceutical companies storing information about illnesses and treatments. With both these examples, data is stored on different servers in different countries by many different parts of the business.

Who owns the knowledge on distributed databases? Who has legislative power over the knowledge?

Features of distributed databases

Concurrency control

Using distributed databases requires the data to be consistent across all sections of the databases. Concurrency control mechanisms ensure the databases do not violate ACID properties. There are two types of concurrency control mechanisms used to achieve this in a distributed database.

Pessimistic concurrency control (PCC): PCC assumes that all transactions will try to access the same resource at the same time. To overcome this before any operation is performed on the data, the resource is locked.

Optimistic concurrency control (OCC): OCC assumes that no transactions will access the resource at the same time. OCC does not lock the data but instead checks for conflicts at run time.

Data consistency

A data consistency model is required to guarantee data is consistent across all sections of the database in the system. Without consideration of consistency, the data will be unreliable across the distributed database and therefore the database will be unable to provide reliable data. There are three models of consistently in a distributed database.

Strong consistency: This ensures that data is updated across all sections of the database immediately. After each write update, the operation is reflected in all sections of the database, ensuring any subsequent read operation has access to the correct data. This is the strictest version of consistency and requires a lot of processing.

Eventual consistency: This allows for temporary inconsistencies in a distributed database. This model determines that given a long period of time without any updates, the data in the database will eventually reach the same state. This model does not guarantee that the read operation reflects the latest write operation. Eventual consistency does sacrifice accuracy but also has lower processing overheads.

Causal consistency: If there is a strong causal connection between two data points, then causal consistency guarantees the state is preserved across all sections of the database. If one operation depends on another, priority will be given to updating the state. All non-causal operations will be updated using the eventual consistency model.

Data partitioning

This involves separating databases into smaller, more manageable sections of a database. Data partitioning improves query performance as the data set is smaller, the storage requirements are smaller, and scalability is improved as you can add subsequent partitions to the database. There are many ways to partition a database, but the four most common are outlined here.

List partitioning: Dividing the data based on specific values within a primary key column. For example, dividing by IDs.

Hash partitioning: Dividing the data based on a hash function applied to a key column. This is usually used when random distribution of values is required or there is no clear partitioning options for the data.

Range partitioning: Dividing the data based on values in a key column. For example, date ranges or ranges within a specific number range.

Round-robin partitioning: Dividing the data equally across all partitions using a round robin algorithm. This is used when there are no obvious partitioning options available.

Data security

Security is essential in all database systems but especially for distributed database systems. There are several ways to improve security within a distributed database.

Authentication: Using usernames and passwords to allow users access to the database system. A further measure is to use two-factor authentication when users have to enter a username and password and then confirm their login attempt using an additional device.

Data encryption: Data entered into the database system is automatically encrypted when the data is entered into the database and decrypted when read. Usually this is a built-in capacity of the database.

Validated input: A security measure ensures the data matches the validation rules of the database, as if the data does not match the validation rules, errors may occur in the database.

Distribution transparency

Transparency in the distributed databases refers to separating databases across different sections but masking the implementation from the user so the database seems to be one cohesive system. Different transparencies that are useful to know when discussing distributed databases include the following.

Access transparency: Ensuring that operations function the same at both local and remote resources. The distribution of the different resources is hidden from the user but the user should not be able to tell.

Location transparency: Enabling access to resources no matter where they are in the network. This means that any movement of files is reflected across the whole system, ensuring the files are always available.

AHL

Concurrency transparency: Allowing multiple transactions to make use of resources without affecting each other. As concurrent user use is essential in a distributed database system, being able to access and use data concurrently is necessary.

Replication transparency: Enabling the existence of different data sets across the database systems. This will improve reliability because if anything happens to one section of the distributed database other sections can be used to restore the data. However, users should be unable to see that the file systems exist in other places.

Failure transparency: Faults do occur on distributed systems, as there are multiple parts and therefore many parts faults can occur. When components fail, failure transparency enables a backup system to take over without the users being aware of the fault.

TOK

As with all databases, distributed databases are subject to anomalies and mistakes if not managed correctly. If the database does not follow the ACID properties data access and updates may cause issues. Databases designers work hard to ensure this does not happen but distributed databases are more susceptible to these issues than other databases.

With this in mind, to what extent is certainty possible when dealing with data that has been produced from distributed databases?

Fault tolerance

As distributed database systems are connected as a single system, faults can occur. However, because there is built-in redundancy and data duplication across the system, the system can tolerate faults and still function. Fault tolerance is the ability of the system to function properly even when there is a fault within the database system. Fault tolerance is required to provide the following features in a distributed database.

Availability: The system needs to be ready for use at all times.

Reliability: The system can work without failure.

Safety: The data within the system is protected from unauthorized access.

Maintainability: How easy it is to repair faults.

Global query processing

As the data is spread across different database sections across a network when developing a query, the query needs to be optimized on a local and global level. This works by the query being entered into the database that is then evaluated at a local level. The schema at the local level determines the data that can be added from the local segment of the database and what needs to come from a global level. The query is then run at a global level and the query is run where the different fragments are within the database. The global optimizer then brings all the data together and processes the results. The local query and global queries are brought together and shared with the user.

Replication

This is the process of storing data at more than one site across the distributed database. This improves the availability of data even if there is a problem at one site within the database. There are different types of replication.

Full replication: The whole database is copied to every site within the distributed system. The advantages to this are that data is available at all sites and global queries can be processed at each site. Disadvantages to full replication are concurrency is challenging and updating is slow.

Partial replication: Important fragments of the database are replicated at each site but there is local data at each site. This model has the advantages of both full replication and no replication.

No replication: Each fragment of the database is only stored at one site. The advantages of this include fewer concurrency issues, as there is no data replication, but this comes at a cost—queries are very slow.

Replication can be completed in one of four ways.

Master–slave replication: One database is the master server. This receives all transaction data, which is passed out to the rest of the database sections.

Multi-master replication: All servers are masters. Any transactions carried out on a server are replicated to all connected servers.

Peer-to-peer replication: Each server can act as both a master and a slave. Data is shared between each server.

Single source replication: A single database is used to store all transactions. This database is then replicated to other databases.

Advantages and disadvantages of distributed databases

Advantages

- Faster processing, because the data and processing power are distributed across a network, so performing requests and actions on the data is more efficient.
- Expansion of the database is easier as you can add more sites to the database to house the data.
- The database is easier to share as the database is separated into different locations. It is also easier to provide local autonomy to each section of the database as you can fragment the data across the network.
- In a distributed database the data is more reliable. This is because the data is spread over different systems. If one section of the system develops a fault, different areas of the system can be used as a backup.

Disadvantages

- The system can very quickly become complex and, therefore, challenging to manage.
- The ACID properties are important in a distributed database to ensure the consistency of data across the database network.
- Translation is needed if not all databases are using the same system, which can lead to additional processing.
- Security is a challenge with many different nodes on a distributed database.

Role of ACID in transactions

When implementing ACID transactions in a distributed system, the data is separated across multiple partitions, adding a layer of complexity to completing transactions that meet the ACID properties across the distribution. This is overcome through the transactions being completed in two phases: the execute phase and the commit phase.

In the execute phase, a query is added to the system and the data is identified in the closest data partitions. A time stamp is placed in the transaction log to show the point when the transaction began. This can help with updating partitions. A lock is placed on the data in the partitions and the query completed.

TOK

There is no expiration date on data storage. Digital archives allow data to be stored—theoretically—forever. Digitalization of archives mean the world has access to more information than ever before. Without regulation, data that is held about you could be saved forever.

The European Union has a law called the general data protection regulation (GDPR) that explains what companies can and cannot do with your data, and what your rights are with regard to your data, including how long the data can be kept for. Many other countries have similar laws.

In pairs or in small groups, spend some time looking at the GDPR or equivalent legislation in your country, then answer the following questions.

- What does the law state about how long data can be kept?
- Should data within databases have an expiry date?
- What consequences may there be to a person if all data every collected about them was kept?

If the query is successful, the commit phase is entered. As well as the commit process being completed, the transaction time is added to the log. Once this stage is reached, the update is passed to all partitions within the distributed database.

In distributed database systems, one server becomes a coordinator. The coordinator keeps track of participating servers known as workers. The role of the coordinator is to ensure all transactions are completed consistently using ACID properties. The role of the workers is to ensure the transactions happen correctly and report the outcome of their operation to the coordinator. In a distributed database, an atomic commit is required before a transaction is committed. For an atomic commit to be completed, the following requirements should be met.

- All participant servers must reach the same conclusion.
- If any participant decides to commit then all other participants must have voted to commit.
- If all participants vote yes, all participants must commit.
- If any participant votes no, rollback must occur on all participants.

By following the atomic commit, distributed databases maintain integrity.

Practice questions

16. Describe an OLAP cube. [2 marks]
17. Explain the difference between clustering and classification. [6 marks]
18. Describe two applications of data mining. [4 marks]
19. Explain why a spatial database is a suitable database for storing information about virus outbreaks. [3 marks]
20. Explain why the atomic commit is necessary in distributed databases. [3 marks]

Linking questions

1. What processes are needed to store data in database structures so that they can be used in machine learning (A4)?
2. How does database programming in SQL differ from programming computationally in a high-level language (B2)?
3. To what extent is the effectiveness of the distributed database determined by the network that connects the various tables (A2)?
4. How could machine learning be applied to databases (A4)?
5. How do programming languages interact with databases to store, retrieve and manipulate data (B2)?

End-of-topic questions

Topic review

1. Using your knowledge from this topic, A3, answer the guiding question as fully as possible:
   What are the principles, structures and operations that form the basis of database systems? [6 marks]

Exam-style questions

2. Describe one anomaly that could occur in an unnormalized database. [2 marks]
3. Describe two features of data in a data warehouse. [4 marks]
4. Explain one tool that could be used to support decision-making in a data warehouse. [3 marks]
5. Consider the following database which stores data regarding customers and their purchases at an online store.

User_ID	Name	Contact	Item_ID	Item_Name	Manufacturer_ID	Location
G943	Grahame	Grahame@mail.com	49493	Dinner set	4930	Italy
			49402	Cutlery	3023	France
			49403	Fruit bowl	4940	Italy
L0493	Lizzie	Lizzie@gmail.com	49493	Dinner set	4930	Italy
			40933	Salt and pepper shaker	5055	Austria

   a. Construct an SQL statement to display all customers who have visited Italy. [3 marks]

   b. Using the data in the view, normalize the database to third normal form. [5 marks]

6. Explain why a database may utilize views when displaying data to users. [3 marks]
7. Describe the role of ACID properties when updating a record in a database. [4 marks]
8. Explain why spatial databases are used to store data with geographical information. [3 marks]
9. Identify the purpose of the MAX aggregate function. [2 marks]
10. Outline two forms of data security in a distributed database. [4 marks]
11. Explain how associative rule discovery can be used to encourage consumers to buy more items. [5 marks]

AHL



A4

Machine learning

What principles and approaches should be considered to ensure machine learning models produce accurate results ethically?

Machine learning models are becoming central to accessing information, communication, and—increasingly—decision making. Making these models must include providing meaningful explanations of how models reach their conclusions and include human-in-the-loop approaches.

As computer scientists, you must deliberately, carefully and consciously consider the process of creating, tuning and evaluating the machine learning models you design and construct. What is the best way to do this?

A4.1 Machine learning fundamentals

Syllabus understandings

A4.1.1 Describe the types of machine learning and their applications in the real world

A4.1.2 Describe the hardware requirements for various scenarios where machine learning is deployed

Machine learning is a subfield of artificial intelligence which focuses on the development of algorithms and statistical models that enable computers to perform specific tasks without using explicit instructions. Machine learning is commonly used to make predictions, classify, categorize and identify patterns in data. By training on data sets, machine learning models can learn from past experiences and improve their accuracy over time.

A4.1.1 Describe the types of machine learning and their applications in the real world

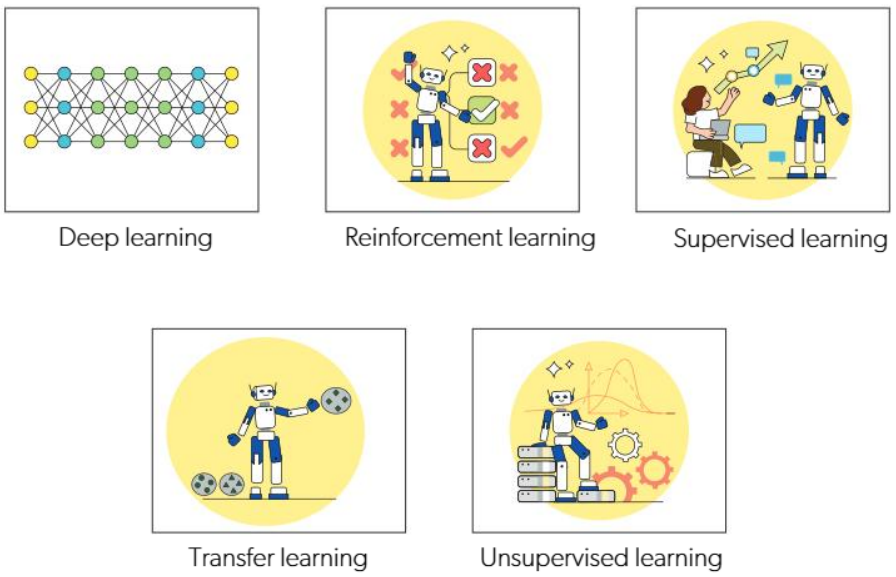
Different approaches for machine learning algorithms

A machine learning algorithm is a set of rules and statistical techniques that allows computers to learn patterns and make decisions from data without being explicitly programmed. These algorithms adjust their output by learning from previous outcomes and optimizing their output over time based on input data.

Imagine you are teaching a robot how to play your favourite game. You explain the rules and show it how to play a few rounds. At first it might make mistakes, but as it plays more, it improves.

That is roughly how machine learning works. It is like a set of instructions for a computer program to learn from experience. It is fed numerous examples and it figures out patterns and relationships in the data. The more data it sees, the smarter it gets, just like your robot improving with practice.

Instead of just following a fixed set of rules, machine learning algorithms can adapt and adjust their behaviour based on what they have learned. It is like they are constantly tweaking their strategy to get the best possible results.



▲ **Figure 1** The five types of machine learning

Deep learning

Deep learning (DL) is a subset of machine learning where artificial neural networks—algorithms inspired by the human brain—learn from large amounts of data. Deep learning architectures such as deep neural networks, convolutional neural networks (CNNs), and recurrent neural networks (RNNs) have been applied to fields including computer vision, speech recognition, natural language processing, and audio recognition. The "deep" in "deep learning" refers to the number of layers in the neural networks, with each layer representing higher-level features defined in terms of lower-level ones.

Reinforcement learning

Reinforcement learning (RL) is an area of machine learning concerned with how intelligent agents ought to take actions in an environment to maximize the notion of cumulative reward. The term cumulative reward in the context of reinforcement learning refers to the total sum of rewards an agent receives over the course of an episode or throughout its interaction within an environment. The agent learns to achieve a goal in an uncertain, potentially complex environment. In reinforcement learning, the agent decides its actions based on its past experiences (exploitation) and new choices (exploration), which is ideal for problems including robotic controls, game playing, and decision-making in finance.

Supervised learning

Supervised learning (SL) is a type of machine learning algorithm that uses a known data set (called a training data set) which includes input data and response values (labels) to learn a function that can be used to predict the output associated with new data. Supervised learning is further categorized into **classification** tasks and regression tasks. In classification, the outputs are categories, while in regression, the outputs are continuous values.

Key term

Classification Categorizing data into predefined groups or classes.

Transfer learning

Transfer learning (TL) is a machine learning method where a model developed for a particular task is reused as the starting point for a model on a second task. It is a popular approach in deep learning where pre-trained models are used as the starting point on computer vision and natural language processing tasks, given the vast compute and time resources required to develop neural network models on these problems and the enormous jumps in skill that they provide on related problems.

Unsupervised learning

Unsupervised learning (UL) is a type of machine learning algorithm used to draw inferences from data sets consisting of input data without labelled responses. The most common unsupervised learning method is cluster analysis, which is used for exploratory data analysis to find hidden patterns or grouping in data. The algorithms, therefore, explore the data to find any kind of structure or pattern, without prior training on data. Other approaches in unsupervised learning include dimensionality reduction, density estimation, and market basket analysis.

Labelled data refers to data sets where each instance is tagged with one or more labels that identify certain features or classifications of the data.

Real-world applications of machine learning

Real-world applications of machine learning may include market basket analysis, medical imaging diagnostics, natural language processing, object detection and classification, robotics navigation, and sentiment analysis.

Market basket analysis

Market basket analysis identifies associations and patterns between item purchases. By analysing transaction data, algorithms such as Apriori or FP-Growth (which are unsupervised learning algorithms) can predict which products are likely to be purchased together. This analysis is invaluable for cross-selling strategies in retail.

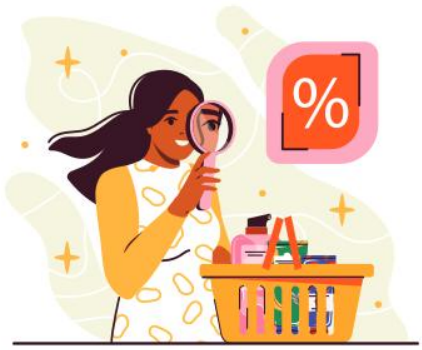
The learning paradigm which best fits this problem is unsupervised learning.

Why unsupervised learning?

In market basket analysis, the goal is to uncover relationships between items in a data set where no specific outcomes (like class labels or continuous targets) are provided. The analysis seeks to find which items frequently co-occur in transactions without prior knowledge of any relationships. Unsupervised learning algorithms such as Apriori and FP-Growth are designed to identify frequent itemsets in transactional data. These algorithms work by exploring combinations of items that appear together more frequently than would be expected by chance. This pattern discovery is central to unsupervised learning, where the focus is on identifying structure from unlabelled data. The core of market basket analysis is to generate association rules, which have implications in the form of $X \Rightarrow Y$, meaning "X implies Y", where X and Y are itemsets. For instance, if a customer buys X, they are likely to buy Y. This type of analysis does not require a training phase with known outputs, which is typical of supervised learning. Instead, it focuses on deriving rules from the inherent properties of the data.

Are you more likely to shop online or in person? If you regularly shop online or often use the same supermarket, you probably have a loyalty card or a customer account. Supermarket loyalty cards may give you discounts on the products you buy most often. Customer accounts can save you time by remembering your details, so you do not have to complete the same information every time you buy an item.

These services are also useful for the business. They help the business to track how quickly products sell, to adjust prices. They can also help companies to understand the demographics of people using their business.



▲ **Figure 2** Market basket analysis

TOK

- To what extent is big data changing what it means to know your customers?
- What are the moral implications of possessing large amounts of information about consumer behaviour?



▲ **Figure 3** An image of a teenager's brain Notice the red parts: these areas of the brain are active while the teenager thinks about food

Medical imaging diagnostics

In medical imaging, machine learning models, particularly deep learning models such as convolutional neural networks (CNNs), are trained on large data sets of imaging data to identify and diagnose conditions from MRIs, X-rays or CT scans. Transfer learning is often employed here to leverage pre-trained models on similar tasks to improve performance even with smaller data sets.

The learning paradigms which best fit this problem are supervised learning, deep learning and transfer learning.

Why supervised learning?

Medical imaging diagnostics typically involves classifying images (such as MRIs, X-rays or CT scans) into diagnostic categories or detecting the presence of specific medical conditions. Supervised learning is ideal because it requires a labelled data set where each image is tagged with a diagnosis or finding, which the model learns to predict. In supervised learning, models are trained on a data set where the input images (features) and their corresponding diagnoses (labels) are known. The goal is for the model to learn the relationship between the image data and the output labels so it can accurately predict the diagnosis for new, unseen images.

Why deep learning?

Deep learning, particularly using CNNs, is highly effective for image recognition tasks because these networks can automatically detect intricate patterns and features in images without the need for manual feature extraction. CNNs, which are a class of deep neural networks, are designed to process pixel data and learn hierarchies of features by building complex patterns on top of simpler ones. This ability makes them exceptionally good at interpreting medical images, where the diagnosis may depend on subtle visual cues within the images.

Why transfer learning?

In the medical imaging field, acquiring large, labelled data sets can be challenging due to privacy issues, rarity of certain conditions, and the need for expert annotation. Transfer learning addresses this by allowing a model developed for one task to be repurposed for a second related task. Transfer learning typically involves taking a model that has been pre-trained on a large data set (often a general image recognition task with vast amounts of data) and fine-tuning it to perform a specific medical diagnostic task. This method leverages the learned features (like edges, textures and patterns common across general images) that are relevant to medical images, enhancing learning efficiency and improving performance even with smaller, specialized data sets.



▲ **Figure 4** Natural language processing

Natural language processing

Natural language processing (NLP) involves processing and understanding human language using models. Techniques such as neural networks, particularly transformers, are used for tasks such as translation, sentiment analysis and chatbots. Transfer learning is especially prominent in NLP, where models such as bidirectional encoder representations from transformers (BERT) are pre-trained on large collections of text and then fine-tuned for specific tasks. The learning paradigms which best fit this problem are deep learning, supervised learning and transfer learning.

Why deep learning?

NLP tasks involve understanding complex language structures and semantics, which require models capable of handling and interpreting vast arrays of sequential and contextual data. Deep learning models, especially those with deep architectures such as recurrent neural networks (RNNs) and transformers, are well suited for this because they can process sequences of data (words, sentences) and capture long-range dependencies within the text. Neural networks, particularly transformers, are structured to manage and learn from sequence data effectively. Transformers, for instance, use mechanisms such as attention to weigh the importance of different words irrespective of their position in the text, making them extremely effective for a range of NLP tasks such as language translation, text summarization and question answering.

Why supervised learning?

Many NLP tasks, such as sentiment analysis, text classification and machine translation, rely on labelled data sets where inputs (text) are paired with outputs (sentiment labels, categories, translated text). Supervised learning is especially effective because it involves training models on these labelled data sets so that the model learns to predict the correct output based on the input text. In supervised learning for NLP, a model is presented with text data and the corresponding target outputs. The model's task is to learn the mapping from the input text to the output during training so that when new, unseen text is provided, the model can predict the appropriate output based on learned patterns.

Why transfer learning?

Transfer learning has become a cornerstone of modern NLP due to the expansive size of language data and the complexity of language-based tasks. It allows for leveraging models pre-trained on vast amounts of general language data, which can then be fine-tuned on smaller, task-specific data sets. In NLP, transfer learning uses models such as BERT, which are pre-trained on large collections of text in a self-supervised manner using tasks like masked language modelling. This pre-training helps the model understand general language contexts and structures. The model can then be fine-tuned with smaller amounts of labelled data specific to a particular task (for example, sentiment analysis for movie reviews), significantly improving performance and reducing the need for large, labelled data sets in every specific task.

Summary

- Deep learning allows for the handling of complex, hierarchical language data effectively.
- Supervised learning ensures that models are accurately trained on specific linguistic tasks using labelled data.
- Transfer learning maximizes the utility of large-scale pre-training for performance boosts across various NLP tasks, making sophisticated NLP capabilities accessible with relatively minimal task-specific data.

Object detection and classification

Object detection and classification involves identifying objects within images and classifying them into predefined categories. Deep learning models, particularly CNNs, are commonly used here. These models are trained using labelled images where the objects have been marked and classified by humans.



▲ **Figure 5** Human and object recognition

The learning paradigms which best fit this problem are supervised learning and deep learning.

Why supervised learning?

Object detection and classification requires the model to predict specific outcomes, such as the presence of an object and its category. Supervised learning is ideal for this because it involves training a model on a data set where the input images and their corresponding outputs (for example, types of objects and their locations) are clearly labelled. In supervised learning for object detection and classification, each training image is annotated with labels that define what objects are present and where they are located (usually with bounding boxes, see Figure 5 for example). The model learns to map the raw image data to these labels, enabling it to predict both the presence and the position of objects in new, unseen images.

Why deep learning?

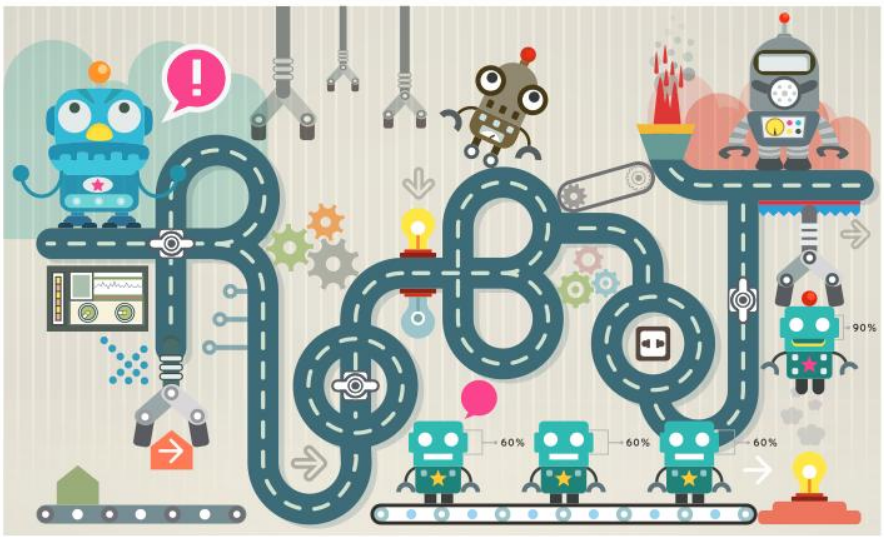
Deep learning, especially with CNNs, is suited to image processing tasks. CNNs are designed to recognize spatial hierarchies in data, which means they can identify complex patterns in images, such as shapes and textures, that are integral to understanding what objects are present. CNNs, and more sophisticated architectures derived from them, such as R-CNN (region-based CNN) or YOLO (You Only Look Once), are particularly adept at handling the multi-scale and multi-aspect ratio nature of real-world objects in images. These networks process images through multiple layers of filters, gaining the ability to recognize features at various levels of abstraction—from simple edges to complex objects.

Summary

- Supervised learning ensures that the models are accurately trained to identify and categorize objects based on clear, predefined labels, making it possible to measure and optimize model performance against real-world requirements.
- Deep learning provides the technological foundation for processing and interpreting image data effectively. By learning from extensive labelled data sets, these models develop the ability to discern and classify diverse objects in varied environments and lighting conditions.

Robotics navigation

Robots learn to navigate and manipulate their environment effectively. Reinforcement learning is pivotal here, as robots learn to make sequences of decisions, receiving feedback through rewards. Supervised learning can also be used for specific tasks such as obstacle recognition using sensor data.



▲ **Figure 6** Robotics navigation

The learning paradigms which best fit this problem are reinforcement learning and supervised learning.

Why reinforcement learning?

Robotics often involves interaction with unpredictable and continually changing environments. Reinforcement learning is ideal for these scenarios because it enables robots to learn optimal behaviours through trial and error, using feedback from their actions in the form of rewards or penalties. In reinforcement learning, a robot or agent learns to make decisions by performing actions and receiving rewards (positive or negative) based on the outcomes. This learning process is guided by a policy which is refined over time to maximize the cumulative reward. This is particularly useful in navigation tasks where the robot must learn to avoid obstacles and reach targets efficiently.

In addition, reinforcement learning is well suited for tasks requiring a series of decisions that depend on both the current state and the sequence of preceding actions. This sequential decision-making is important for robots that need to plan paths or strategies over time, considering potential future states and rewards.

Why supervised learning?

While reinforcement learning is excellent for learning from interaction, supervised learning is effective for more narrowly defined tasks that require recognizing patterns or classifying data. In robotics, tasks such as object recognition, obstacle detection and classification of terrain are critical for safe navigation. Supervised learning involves training a model on a data set where the inputs (sensor readings, camera images) and the desired outputs (labels indicating obstacles, safe paths) are predefined. The robot uses this trained model to interpret its sensors and make immediate decisions about its environment.

Robots are typically equipped with various sensors that provide data for understanding and interacting with their surroundings. Supervised learning models can be trained on this data to accurately predict necessary outputs, such as the presence of an obstacle or the type of surface. This capability is essential for the robot to perform specific navigation tasks where immediate and accurate response is based on learned experience.

Summary

- Reinforcement learning helps robots develop strategies for navigation and manipulation by learning from the consequences of their actions, ideal for complex environments where predefined rules might not suffice.
- Supervised learning enables robots to quickly and reliably recognize patterns and make classifications based on training data, which is indispensable for responding to immediate environmental cues.

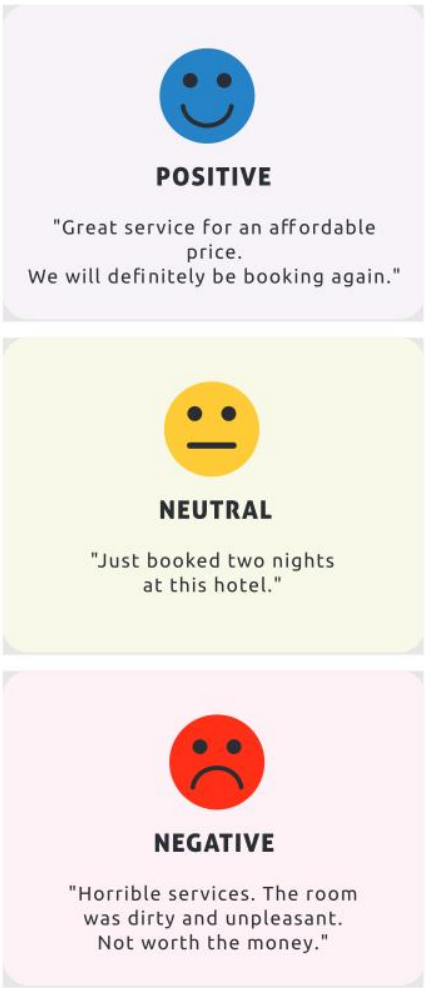
Sentiment analysis

This process involves analysing text data from reviews, social media, and so on, to determine the sentiment expressed (positive, negative, neutral). Deep learning, particularly long short-term memory (LSTM) networks or transformers, can be trained on labelled data sets to recognize and predict sentiments. Transfer learning allows these models to adapt to specific domains or types of text quickly.

The learning paradigms which best fit this problem are supervised learning, deep learning and transfer learning.

Why supervised learning?

Sentiment analysis typically requires the classification of text into predefined categories (such as positive, negative or neutral). Supervised learning is well suited for this task because it involves training models on data sets where both the inputs (text data) and the desired outputs (sentiment labels) are clearly defined. In supervised learning for sentiment analysis, a model is trained on a corpus of text data where each piece of text is annotated with a sentiment label. The model learns to correlate specific features of the text, such as word choice and sentence structure, with the sentiment, enabling it to predict the sentiment of new, unseen text.



▲ **Figure 7** Sentiment analysis

Why deep learning?

Deep learning models, particularly those using LSTM networks and transformers, are capable of processing sequential data, such as text, where understanding context and the order of words is important. LSTMs are a type of recurrent neural network (RNN) that are adept at learning from sequences of data with long-range dependencies, making them ideal for texts where context spread across sentences influences sentiment. Transformers, on the other hand, utilize self-attention mechanisms to weigh the relevance of all parts of the text simultaneously, which is highly effective for capturing complex contextual relationships between words in sentences.

Why transfer learning?

Sentiment analysis often needs to be tailored to specific contexts or domains (such as analysing sentiments in social media versus product reviews) which might not always have adequate labelled data available for training effective models from scratch. Transfer learning involves taking a model that has been pre-trained on a large data set (usually a general language processing task) and fine-tuning it on a smaller, domain-specific data set. This approach leverages the learned features from the broader data set, such as basic language understanding, which can be effectively adapted to more specialized tasks with relatively minimal additional training. This method significantly reduces the need for large, labelled data sets in each specific domain, making it possible to achieve high performance in sentiment analysis even with limited domain-specific data.

Summary

- Supervised learning ensures that sentiment analysis models are precisely trained on how sentiments are expressed in specific data sets, making them reliable for accurate sentiment prediction.
- Deep learning techniques, particularly LSTMs and transformers, provide the technical means to understand the nuances and complexities of human language.
- Transfer learning allows these models to be quickly adapted to new domains or types of text, enhancing their versatility and applicability across different contexts.

TOK

- To what extent does the process of training machine learning models reflect the human ways of learning and adapting?
- How can ethical considerations in machine learning model development influence the reliability and trustworthiness of these models?
- In what ways does the application of different machine learning paradigms (such as supervised, unsupervised, reinforcement, deep learning and transfer learning) enhance our ability to solve complex real-world problems?

This problem in practice

Jefferson High, a progressive public school, is experimenting with machine learning to revolutionize its educational approach. With a diverse student body exhibiting a wide range of learning styles and abilities, the school has sought to implement a system that personalizes learning experiences, thus optimizing educational outcomes for every student.

The school's IT department, in collaboration with the computer science faculty, has initiated a pilot program utilizing several machine learning models to address different educational challenges.

Supervised learning for predictive analytics: Early identification of students who might need additional help or are at risk of dropping out. Historical data such as grades, attendance records and test scores are fed into a supervised learning algorithm. The model predicts students' future performance, allowing early intervention by educators.

Natural language processing (NLP) for language skills development: Enhancing reading comprehension and writing proficiency. NLP techniques are used to analyse students' written assignments for grammar, style and content coherence. Feedback is generated automatically, providing students with immediate guidance on how to improve their writing skills.

Reinforcement learning for adaptive learning pathways: Personalized learning experiences based on individual student responses. An adaptive learning platform uses reinforcement learning to adjust the difficulty level of tasks in real-time, ensuring that each student remains engaged and challenged without being overwhelmed.

Deep learning for visual learning aids: Creation of customized educational content, especially in subjects requiring visual learning like geometry and biology. Deep learning algorithms analyse students' interaction with visual content and adaptively modify it to enhance understanding and retention of complex concepts.

The integration of machine learning at Jefferson High has resulted in significant improvements in student engagement and academic performance. Early interventions have reduced dropout rates, while personalized learning pathways have seen an increase in student proficiency in core academic skills. Teachers are now able to focus more on facilitating learning rather than spending extensive time on assessments and grading.

To ensure ethical use of machine learning, Jefferson High adheres to several principles.

Transparency: Students and parents are informed about how data is used and the purpose behind it.

Control and consent: Data collection is conducted with explicit consent, and students have some control over their data.

Bias mitigation: Regular audits of machine learning models are carried out to identify and eliminate potential biases, ensuring fair treatment of all students.

Jefferson High's case illustrates the power of machine learning to transform educational environments, making them more inclusive, efficient, and responsive to the needs of all students. It serves as a model for other institutions aiming to integrate technology into their educational practices responsibly and effectively.

A4.1.2 Describe the hardware requirements for various scenarios where machine learning is deployed

To understand hardware requirements for machine learning, you need to understand the different scenarios you will be considering.

Table 1 Machine learning scenarios

Scenario	Description
Development and testing	This is the phase where you build and refine your machine learning models. It involves writing code, experimenting with different algorithms, and testing them to see how well they perform. You usually start with smaller, manageable data sets and simpler models to ensure everything is working correctly.
Data processing and feature engineering	In this stage, you prepare your data for the models. This includes cleaning the data (removing errors or irrelevant information), transforming it into a format the algorithm can use effectively, and creating new data attributes or "features" that can help improve model accuracy. It is about making the data meaningful and ready for analysis.
Model training and deep learning	Here, you take your prepared data and use it to teach the machine learning model how to make predictions. This process involves adjusting the model's parameters to improve its predictions based on the data it sees. Deep learning is a part of this, using complex models (neural networks) that learn from vast amounts of data.
Large-scale deployment and production	Once the model is trained and tested, it is time to use it in real-world applications. This could mean integrating the model into a company's software to automate tasks or make decisions based on data. It requires the model to be reliable, fast and scalable, meaning it can handle growing amounts of data or users without breaking down.
Edge computing	If the use case requires, this scenario involves placing computation close to where data originates (such as in a smartphone or a car) rather than in a distant data centre. It is useful for tasks that need immediate responses, like facial recognition in a smartphone or real-time decision-making in autonomous vehicles.

Hardware configurations for machine learning can vary widely, from standard laptops to advanced infrastructure, based on the scale of the task, the complexity of the model, the size of the data set, and the desired speed of processing.

Each type of hardware setup is chosen based on the specific requirements of the machine learning tasks, balancing factors such as cost, performance and operational needs.

Table 2 Hardware configurations for different machine learning scenarios

Scenario	Processing	Storage	Scalability
Development and testing	Standard laptops or desktops with multi-core CPUs.	SSDs with 256 GB to several TB capacity to facilitate faster data access and processing speeds.	Ideal for developing and testing small to medium-sized models or for educational purposes.
Data processing and feature engineering	High-performance workstations or servers with powerful CPUs.	Large SSDs (1 TB or more) or multiple HDDs in RAID configuration for extensive data storage and redundancy.	Capable of handling larger data sets and more complex tasks, but less scalable than cloud-based solutions.
Model training and deep learning	Dedicated GPU servers with one or more high-end GPUs.	High-capacity SSDs for rapid data access and storage of large data sets and model checkpoints.	Highly scalable with additional GPUs and clustered environments for parallel processing.
Large-scale deployment and production	High-end servers or cloud-based solutions with scalable CPUs and GPUs/TPUs.	Enterprise-level storage solutions, often distributed file systems or cloud storage with high data throughput.	Extremely scalable, supports high-load, continuous operations and real-time processing on a large scale.
Edge computing	Compact, energy-efficient devices with integrated CPUs and GPU.	Smaller SSDs or flash storage sufficient for operating systems and application code.	Scalable to a large number of devices but limited by individual device capabilities.

Table 3 Hardware configurations for machine learning

Hardware	Use case	Configuration
Standard laptops and desktops	Ideal for beginners, students or developers working on small-scale projects or learning the fundamentals of machine learning.	Typically equipped with consumer-grade CPUs, a moderate amount of RAM (8–16 GB), and possibly low-end GPUs. These machines can run basic models and handle small to moderate data sets.
High-performance workstations	Suitable for professional developers and researchers working on more complex models that require substantial computational resources but do not yet need a full server setup.	Equipped with high-end CPUs, 32-128 GB of RAM, and one or more mid-range to high-end GPUs. These systems can handle larger data sets and more computationally intensive training sessions.
GPU-enhanced systems	Necessary for deep learning and large-scale machine learning tasks that involve complex neural networks and massive data sets.	Features multiple high-end GPUs which support parallel processing capabilities that significantly reduce training and inference times.
Dedicated AI servers	Used in industry and academia for high-demand, continuous machine learning tasks, including real-time data processing and training very large models.	These servers might include multiple high-performance GPUs or TPUs, extensive RAM (up to 1 TB or more), and specialized hardware such as high-speed networking interfaces and large-scale storage systems.
Edge devices	For deploying machine learning models directly into consumer devices or IoT (Internet of Things) environments where real-time processing is needed at the edge of the network.	Typically equipped with energy-efficient CPUs (such as ARM-based processors), 1–8 GB of RAM, flash storage (a few GBs to tens of GBs), and possibly GPUs or specialized accelerators such as FPGAs or ASICs for intensive tasks. Operate on low-voltage power or batteries.

Computer hardware, memory, and CPU speeds improve incredibly quickly. These configurations are accurate in 2024, but the authors recognize that they are likely to be out of date very soon.

Advanced infrastructure components

Machine learning can require advanced infrastructure because of the inherent high computational demands, the focus on speed and efficiency, scalability, and specialized operations for training models.

Application-specific integrated circuits

Application-specific integrated circuits (ASICs) are custom-designed circuits tailored to execute specific tasks efficiently. In machine learning, ASICs are often designed for high-speed, low-power data processing. They are used in consumer electronics and embedded systems, including mobile devices where power efficiency is paramount.

Edge devices

Edge devices refer to hardware that processes data at or near the source of data generation rather than relying on a central data-processing facility. These devices often incorporate ASICs or small-scale GPUs. This hardware is ideal for real-time applications, such as facial recognition or autonomous vehicle navigation, where low latency is critical.

Field-programmable gate arrays

Field-programmable gate arrays (FPGAs) are semiconductor devices that can be reconfigured after manufacturing, allowing developers to customize the hardware post-production to suit specific needs. FPGAs are used in scenarios where flexibility is required and for accelerating specific workloads, including machine learning inference and data flow processing.

GPUs

Originally designed for rendering graphics, GPUs are highly effective at handling the parallel processing tasks that are prevalent in machine learning and deep learning. They are essential for training deep learning models due to their ability to perform multiple calculations concurrently, significantly speeding up the training process.

See section A1.1.2 for a deeper discussion about GPUs.

Tensor processing units

Developed specifically for neural network machine learning, **tensor** processing units (TPUs) are Google's custom ASICs that accelerate tensor calculations TPUs provide significant acceleration for applications using TensorFlow, Google's machine learning framework, especially in large-scale and cloud environments.

Cloud-based platforms

These platforms provide virtualized and scalable resources on-demand, including CPUs, GPUs and TPUs, allowing users to flexibly scale their machine learning projects according to varying workloads. Cloud-based solutions are used by organizations that prefer not to maintain physical infrastructure. They are essential for handling bursty data loads, experimental projects, or varying computational demands.

Key term

Tensor A way to represent data in different dimensions. It can be thought of as an array of numbers, like a list or a table, but it can extend to many dimensions. For example:

- A single number is a 0-dimensional tensor.
- A list of numbers (a line) is a 1-dimensional tensor.
- A table of numbers (like a grid) is a 2-dimensional tensor.

Tensors can go beyond these to handle even more dimensions of data, which is useful in machine learning and deep learning tasks.

High-performance computing centres

High-performance computing (HPC) centres consist of thousands of processors working in parallel to perform large-scale computation tasks. They often include clusters of servers, each equipped with multiple high-performance CPUs and GPUs. They are ideal for extremely large-scale machine learning tasks, such as training complex models across vast data sets, or detailed simulations that require immense computational resources.

TOK

- To what extent does the process of training machine learning models reflect the human ways of learning and adapting?
- How can ethical considerations in machine learning model development influence the reliability and trustworthiness of these models?
- In what ways does the application of different machine learning paradigms (such as supervised, unsupervised, reinforcement, deep learning and transfer learning) enhance our ability to solve complex real-world problems?

Practice questions

1. a. Define machine learning. [1 mark]
   b. Describe how machine learning differs from traditional programming. [2 marks]
2. a. Describe two real-world applications of machine learning. [2 marks]
   b. Identify the learning paradigms used in the real-world applications described in part a. [2 marks]
3. Distinguish between deep learning and reinforcement learning. [3 marks]
4. Describe the concept of transfer learning and its significance in machine learning. [3 marks]
5. Identify one ethical consideration that must be addressed when implementing machine learning models. [1 mark]

A4.2 Data preprocessing

Syllabus understandings

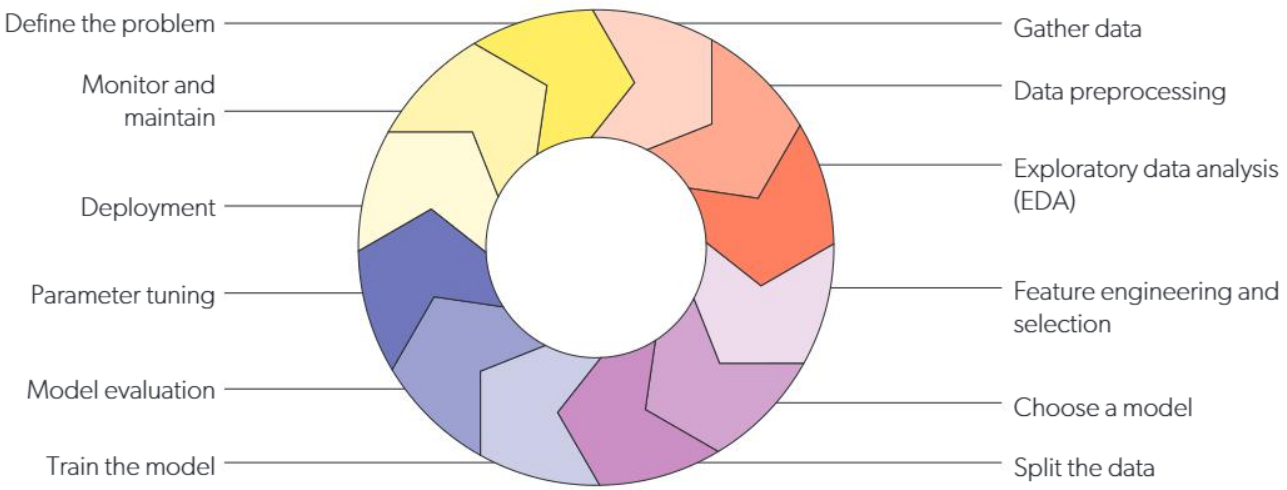
AHL

A4.2.1 Describe the significance of data cleaning

A4.2.2 Describe the role of feature selection

A4.2.3 Describe the importance of dimensionality reduction

Data preprocessing is a step in the machine learning workflow aimed at improving the quality of data and making it suitable for building a model. The goal is to enhance the performance and accuracy of the machine learning algorithms.



▲ **Figure 8** The machine learning workflow

A4.2.1 Describe the significance of data cleaning

Data cleaning standardizes data so it is consistent, error-free, accurate, and complies with regulatory and legal requirements. When data is first considered for machine learning (training), it is often disorganized and full of errors. The term "noisy" or "noise" refers to data with unnecessary and inaccurate data.

Before data cleaning

Customer ID	Age	Email Address	Income	Last Purchase Date
001	35	john.doe@domain.com	54000	01-02-2020
002	−25	jane_smith@domain.com	58000	
003	45	n/a	62000	12/15/2019
004	thirty	annetom@domaincom	−5000	20-10-2019

▲ **Figure 9** Initial data

Issues in the data include the following.

- Invalid age values (negative number).
- Missing email information ("n/a") and invalid email format.
- Missing or inconsistent date formats in the Last Purchase Date column.
- Missing data in the Last Purchase Date column for customer 002.
- Negative income value, which is not plausible.
- Inconsistent entry in age ("thirty").

After data cleaning

Customer ID	Age	Email Address	Income	Last Purchase Date
001	35	john.doe@domain.com	54000	2020-01-02
002	25	jane_smith@domain.com	58000	Data Not Available
003	45	Data Not Available	62000	2019-12-15
004	30	annetom@domain.com	50000	2019-10-20

▲ **Figure 10** Clean data

Cleaning actions taken are as follows.

- Negative age corrected, from −25 to 25 (assuming typo).
- Replaced "n/a" with "Data Not Available" for clearer data absence indication.
- Standardized the date format to YYYY-MM-DD.
- Inputted missing date data with a placeholder text to indicate unavailability.
- Corrected the email syntax and removed invalid characters.
- Normalized the income value by converting negative income to positive.
- Converted textual numeric data ("thirty") to its numeric form (30).

The impact of data quality on model performance

The impact of data quality on model performance in machine learning cannot be overstated. Data quality directly influences the accuracy, reliability and robustness of the predictive models.

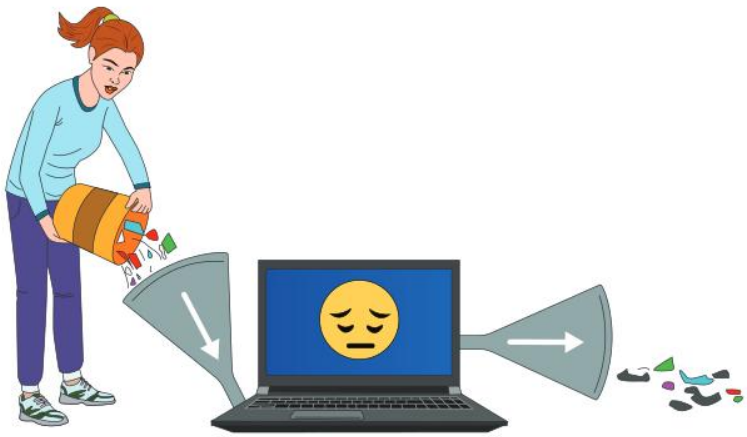
High-quality data that is accurate and complete enables models to make more precise predictions. Conversely, poor-quality data can lead to inaccurate and unreliable outcomes because the model learns from flawed or misleading information. There are other important impacts of data quality on model performance, but none of them are as important as this:

good data = good predictions

In addition, a model trained on high-quality data is better able to generalize from the training set to unseen data. This means it can perform well in real-world scenarios, which is ultimately the goal of most machine learning projects. Data that is noisy, incomplete or non-representative can lead the model to overfit to the noise and anomalies in the training set, thereby performing poorly on new data.

Finally, ensuring data is representative and unbiased prevents models from perpetuating or amplifying biases. High-quality data must be diverse and inclusive to avoid ethical issues and comply with legal standards.

There is an old proverb in computing: "garbage in, garbage out". This proverb helps us to understand that the data we put into a system directly impacts the data we get out of the system.



▲ **Figure 11** Garbage in, garbage out

Worked example 1

A company wants to develop a machine learning model to screen job applications. The data set they use to train this model contains résumés from the past 10 years. This data set consists of résumés from predominantly male applicants, reflecting the company's historical gender imbalance in certain roles.

1. How would this model perform?
2. How could the company improve the model's performance?

Solution

1. The model would most likely be biased to male candidates, even when a female candidate might be a much better fit.
2. The company could increase the data set to give more female applicants and résumés, or change the weighting of different criteria.

Managing outliers, duplicates, errors, and irrelevant data

Handling outliers

An outlier is an observation in a data set that is distant from other observations. Statistically, you can use interquartile range (IQR) scores, Z-scores, or visualization tools like box plots to identify outliers. Values that fall beyond a defined threshold can be considered outliers. Once you have identified an outlier, you might do the following.

- Trim the outliers by removing them from the data set entirely.
- Cap the outliers by replacing them with the nearest value that falls within an acceptable range.
- Transform the outliers by applying transformations like log or square root to reduce the effect of extreme values.

Removing or consolidating duplicate data

Duplicate data refers to instances within a data set where certain records are repeated. These repetitions can be exact copies, where every field in a record is identical to another, or near duplicates.

Techniques for managing duplicate data include the following.

- Using software or database queries to identify and remove duplicate entries based on specific keys or a combination of fields.
- Manually reviewing in some cases, especially where duplicates are not exact or the data set is small.
- When duplicates contain partially varying data, consolidating the information into a single record, often by averaging numerical values or choosing the most frequent category.

Identifying and correcting incorrect data

Incorrect data has errors or inaccuracies. Techniques to manage incorrect data include the following.

- Implementing data validation rules based on known ranges, data formats, or other criteria to identify anomalies that may indicate incorrect data.
- Cross-referencing data by using external or additional data sources to validate and correct questionable entries.
- Employing machine learning techniques to detect anomalies that could indicate data inaccuracies.

Filtering irrelevant data

Irrelevant data refers to any information in a data set that does not contribute to or is unnecessary for the specific analysis or modelling task at hand. Techniques for filtering include the following.

- Utilizing feature selection techniques like correlation matrices, backward elimination, and random forest importance to identify and remove irrelevant or less important features.
- Consulting domain experts to understand which data elements are likely to be irrelevant to the problem being solved.

Transforming improperly formatted data

Improperly formatted data refers to information within a data set that does not adhere to an expected or standardized format required for processing or analysis. Techniques for managing include the following.

- Applying parsing techniques to reformat data into a usable structure, such as converting strings to standardized date formats or splitting a full name into first and last name.
- Using regular expressions patterns to identify and transform data formats.
- Applying a uniform format across similar data types, such as converting all currency values to a single currency and format.

Handling missing data

Missing data refers to the absence of data values in a data set where they are expected. There are three techniques which can be considered to manage missing data: imputation, deletion, and predictive modelling.

Imputation

- Replace missing values with the mean, median or mode of the column.
- Use the K-NN algorithm to impute missing values based on the similarity of entries in parameter space (parameter space is synonymous with feature space).

Deletion

- Remove entire records where any data is missing (listwise deletion).
- Use available data while ignoring any instances where data is missing (pairwise deletion).

Predictive modelling

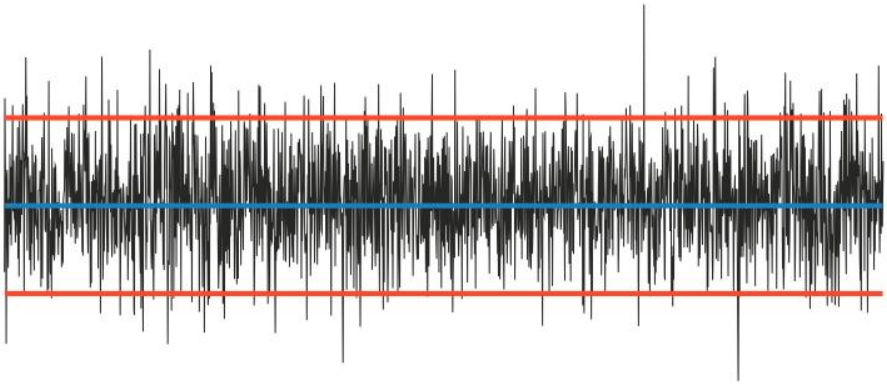
- Use regression models to predict missing values based on other data in the data set.
- Use algorithms like decision trees or neural networks to predict missing values.

Normalization and standardization

Normalization and standardization are two data preprocessing techniques used to adjust the scale and distribution of variables in a data set before applying machine learning algorithms. Both methods aim to transform the data to be more suitable for modelling, but they do so in slightly different ways.

Normalization

Normalization (also known as min–max scaling) involves rescaling the features to a specific range, typically 0 to 1, or –1 to 1. This transformation is beneficial for algorithms that are sensitive to the scale of input data, such as gradient descent-based algorithms, and it helps in speeding up their convergence. Normalization ensures that all features contribute equally to the results and it prevents models from misinterpreting the data due to the scale of the variables.



▲ **Figure 12** Normalizing sound so that it is not too loud or too quiet

Standardization

Standardization (also known as Z-score normalization) involves rescaling the features so that they have the properties of a standard normal distribution with a mean of zero and a standard deviation of one. Standardization is useful for data with unknown minimum and maximum values or when there are outliers that would distort min–max scaling. It maintains useful information about outliers and makes the algorithm less sensitive to them. Finally, standardization is suitable for techniques that assume data is normally distributed, such as logistic regression.

Key differences between normalization and standardization

Normalization changes the range of the data to [0, 1] or [–1, 1], while standardization transforms data to have mean 0 and standard deviation 1, without bounding values to a specific range. Normalization can be significantly affected by outliers since they will compress the majority of the data to a very small interval. Standardization is less sensitive to outliers because it is based on the mean and standard deviation, which are inherently influenced by extreme values but to a lesser extent.

Key term

Feature A common term in machine learning and statistics that can refer to:

- an individual attribute (a single measurable property or characteristic of something you are observing; for example, in a data set about houses, features could be price, number of bedrooms, floor area, and so on)
- an input variable (used by machine learning models to make predictions or classifications)
- a column in a data set (in a tabular data set, each column usually represents a feature).

A4.2.2 Describe the role of feature selection

Feature selection is a process in machine learning used to identify and retain the most informative attributes of a data set while removing those that are redundant or irrelevant. The benefits of feature selection include enhanced model accuracy, reduced overfitting, faster training and improved interpretability. The process of feature selection is important because you do not want to remove an attribute which could be helpful, but you also do not want to keep an attribute which is not helpful to your model.

The goal of feature selection is not just to remove data but to retain the most informative attributes. These attributes are those that provide the most utility in predicting the output variable. The retained features should be those that result in the highest possible performance of the machine learning model according to some criteria, such as accuracy, precision or recall.

Think about feature selection for building a predictive model using a data set of real estate sales. The data set contains various features of properties such as location, size, price, number of bedrooms, number of bathrooms, age of the property, proximity to schools, proximity to highways, crime rate and property tax rate. The goal is to predict the price of a property based on these features.

Initial data set:

Location: City or neighbourhood of the property.

Size (m^2): The floor area of the property.

Price: Selling price of the property.

Bedrooms: Number of bedrooms.

Bathrooms: Number of bathrooms.

Age of property: Number of years since the property was built.

Proximity to schools: Distance to the nearest school.

Proximity to highways: Distance to the nearest highway.

Crime rate: Crime rate in the neighbourhood.

Property tax rate: Annual property tax rate.

Feature selection aims to reduce the number of features to simplify the model without significantly impacting the accuracy of price predictions.

Feature selection strategies

Filter methods

Filter methods evaluate the relevance of features by their intrinsic properties, using statistics such as correlation coefficients with the output variable, Chi-square tests, ANOVA, or mutual information scores. These methods are usually fast and effective in reducing the feature space before employing any machine learning algorithms.

Filter methods and predicting the price of a home

When preparing to predict home sale prices, a real estate data set might include features such as floor area, number of bedrooms, age of the property, proximity to schools, and postal codes. By applying filter methods such as correlation coefficients, you can quickly identify which features have the strongest relationships with home prices. For instance, floor area and number of bedrooms may show high positive correlations with the sale price, suggesting they are important predictors and should be retained. In contrast, the postal code, while useful for categorical analysis, might not show a strong direct correlation with price and could be considered for removal if it does not significantly change the model's performance.

Wrapper methods

Wrapper methods use a subset of features and train a model using them. Based on the model performance, they then decide to add or remove features from your model. This is done iteratively, such as in forward selection, backward elimination, or recursive feature elimination (RFE). These methods can be computationally expensive but often provide better performance as they evaluate features in the context of the model.

Wrapper methods and predicting the price of a home

Using a wrapper method such as RFE, you start with all features and iteratively remove the least important ones based on the performance of a regression model. The process might start with (floor area, number of bedrooms, age, proximity to schools, and postal code, and through iterative training and evaluation, it could determine that age and postal code contribute the least to predicting sale price. These features would then be eliminated, leaving behind a model focused on floor area, number of bedrooms and proximity to schools as the core predictors.

Embedded methods

Embedded methods perform feature selection as part of the model training process and are specific to certain algorithms that have their own built-in feature selection methods. For example, Lasso and Elastic Net add a penalty to the loss function during training that can shrink some feature coefficients to zero, effectively performing feature selection by keeping only the significant features.

Embedded methods and predicting the price of a home

When employing an algorithm like Lasso regression for predicting home sale prices, the regularization parameter in Lasso helps to penalize the coefficients of less important features, effectively reducing them to zero. Thus, during the training process, Lasso might determine that features such as proximity to schools and number of bedrooms have non-zero coefficients, indicating significant influence on sale prices. Simultaneously, it might reduce the coefficients for features such as age of the home, or certain less impactful location descriptors, to zero, effectively selecting the most relevant features without separate feature elimination steps.

Removing redundant or irrelevant features

Identifying redundant features

Redundant features are those that provide no additional information because they are duplicates of other features or are highly correlated with other features. For instance, if two features are highly correlated, one can be removed without substantial loss of information. This reduction in redundancy helps to focus the model's learning on unique attributes.

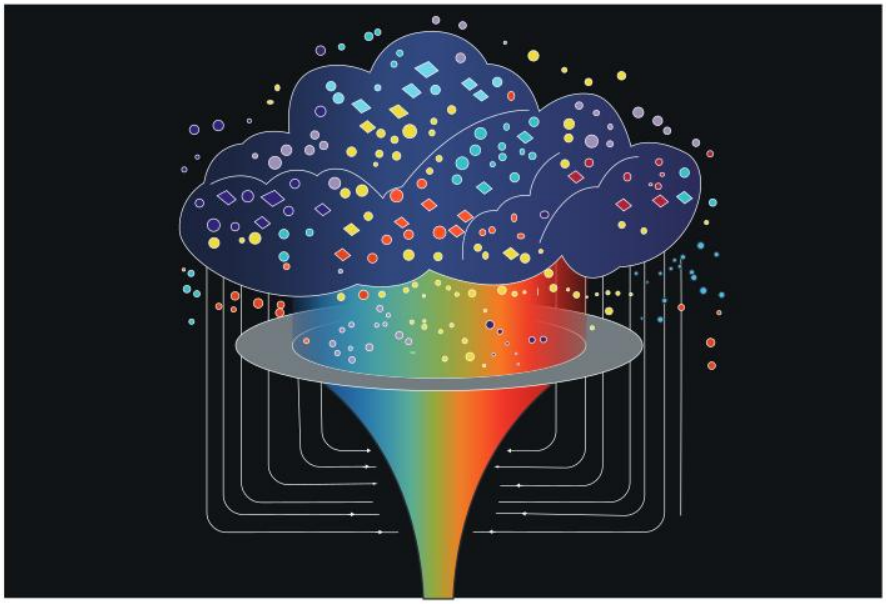
Eliminating irrelevant features

Irrelevant features do not contribute to or may decrease the accuracy of the predictive model because they have no relationship with the output variable. Removing these features prevents the model from considering noise during the learning process, which can improve predictive performance and reduce the likelihood of overfitting.

A4.2.3 Describe the importance of dimensionality reduction

A dimension in data simply represents a feature, attribute or variable. For example:

- in customer data, dimensions could be age, income, location, purchase history, and so on
- a medical image could count each pixel in an image as a dimension
- within text data, word frequencies in a document form dimensions.



▲ **Figure 13** A representation of dimensionality reduction

Dimensionality reduction strategically reduces the number of features in a data set. In this process, you identify and retain a set of significant features, derived from the original features through a transformation process. These significant features capture the most important information from the original data set. The goal is to reduce the overall dimensionality, thus simplifying models, speeding up computations, reducing noise and minimizing the risk of overfitting, while still preserving as much relevant data variance as possible.

Curse of dimensionality considerations

The "curse of dimensionality" refers to various phenomena that arise when analysing and organizing data in high-dimensional spaces (often with hundreds or thousands of dimensions) that do not occur in low-dimensional settings. Considerations may include overfitting, computational complexity, data sparsity, distance metrics effectiveness, data visualization, sample size increases and memory usage. This term, coined by Richard Bellman, encapsulates several challenges that grow exponentially with the increase in dimensions.

Suppose you are building a machine learning model to predict the sale price of homes based on a data set with various features (dimensions). These features might include the number of bedrooms, number of bathrooms, floor area, lot size, type of flooring, age of the property, proximity to schools, and many other characteristics.

Scenario: Without the curse of dimensionality

If you only use a few important features, such as number of bedrooms, floor area, and location, your model can more easily discern patterns that directly affect home prices. Fewer dimensions allow for simpler model training, faster computations, and typically require less data to effectively train the model without overfitting.

Scenario: With the curse of dimensionality

Now, imagine including hundreds of features—extending to less critical ones such as the brand of appliances, detailed descriptions of each room's decor, types of plants in the garden, and the colour of the walls. Determining which features are actually important becomes more challenging as the dimensionality increases, potentially obscuring key factors that are more predictive of home prices.



▲ **Figure 14** House sales

Overfitting

In high-dimensional spaces, models often have too many parameters relative to the number of observations, which leads to overfitting. The model learns not just the underlying patterns but also the noise in the training data, which harms its performance on new, unseen data. The model's ability to generalize decreases as dimensionality increases, unless significantly more data is provided.

Computational complexity

Many algorithms that involve computations over the feature space see their complexity increase exponentially with the number of dimensions. This can be due to the increased number of calculations or the more complex data structures needed to handle high-dimensional data. Algorithms become slower, requiring more computational resources, which can make processing impractical for very large data sets.

Data sparsity

As the dimensionality increases, the volume of the space increases so quickly that the available data become sparse. This sparsity is problematic because it becomes difficult to find closely related samples for predictions, as each point in the space tends to be far away from all others. Models become less reliable and may require exponentially more data to achieve statistical significance.

Distance metrics effectiveness

In high dimensions, all points tend to be equidistant from each other. Common metrics such as Euclidean distance lose meaning, making it hard to distinguish between near and far points, which impacts clustering, nearest neighbour classification, and other algorithms relying on distance calculations.

Techniques that rely on distance measurements become less effective and sometimes provide misleading results.

Data visualization

Visualizing high-dimensional data is inherently challenging because human perception is limited to three dimensions. Directly visualizing data without dimensionality reduction techniques such as PCA or t-SNE is impossible, making it harder to detect patterns, outliers or clusters visually.

Sample size increases

To adequately cover a high-dimensional space with data, the sample size needed grows exponentially with the number of dimensions. This phenomenon is related to the concept of data sparsity. Practical data collection becomes infeasible, as the amount of data required to maintain the same level of performance grows exponentially.

Memory usage

Storing and processing high-dimensional data requires significantly more memory. Each additional dimension can increase the storage requirement linearly, which becomes problematic with thousands of dimensions. Higher memory usage necessitates more powerful hardware and can limit the scalability of data-driven applications.

Example: Predicting home sale prices in a high-dimensional feature space

Imagine you are a real estate agent using a machine learning model to predict the sale price of homes. The sale price is based on various attributes of the home. Initially, you start with a comprehensive data set with basic features such as number of bedrooms, number of bathrooms, floor area, age of the home, and location.

Initial scenario

In a lower-dimensional setting, each data point (home) has a number of near neighbours in the feature space that are similar along these basic dimensions. This proximity allows the model to make accurate predictions because it can easily find and learn from comparable examples.

High-dimensional expansion

Now, consider that you decide to vastly expand the number of features in your data set to include detailed characteristics such as:

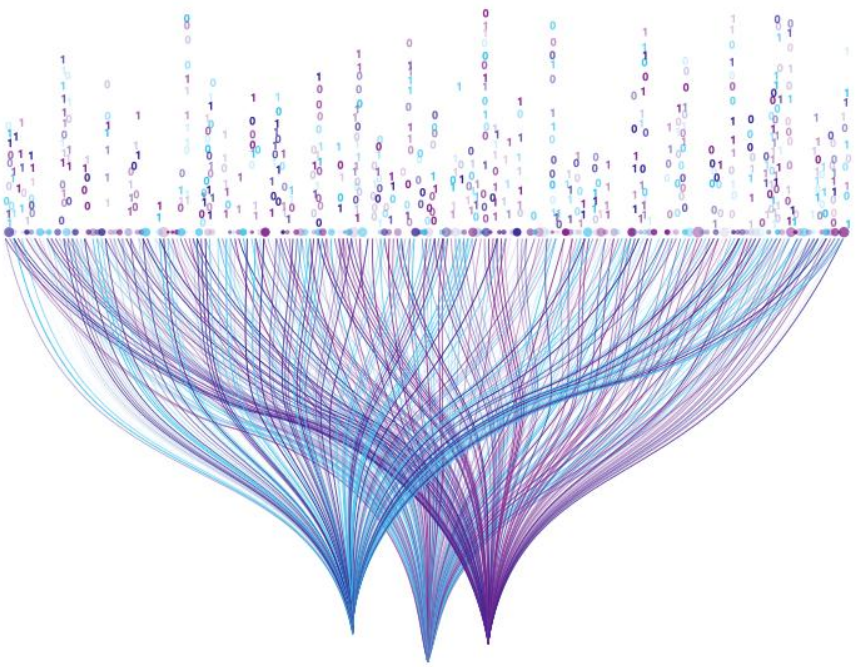
- type of door knobs
- brand of kitchen appliances
- wallpaper patterns
- type of window treatments (curtains, blinds, and so on)
- detailed landscaping specifics
- decorative colour schemes in each room
- type of wood in the built-in cabinetry
- history of all maintenance activities.

In this high-dimensional space, the feature space grows exponentially as you add more dimensions. Consequently, the volume of the space expands far beyond the number of data points you have. With so many dimensions, each home in your data set becomes an isolated point, far from others in this expanded feature space. Essentially, every home appears unique, with few obvious "near neighbours".

The implication here is that it is difficult for your model to find similar homes to any given one because each is distant from the others in many dimensions of the feature space.

As homes are far apart in high-dimensional space, the model's predictions based on nearby examples become less reliable. The comparisons the model attempts to make may hinge on irrelevant or noisy features.

To achieve reliable predictions and overcome the sparsity, you would need exponentially more data points to cover the expanded feature space adequately. This requirement often is impractical and expensive.



▲ **Figure 15** Which of these data points are most helpful to predict the price of a home?

Dimensionality reduction of variables

Note: Statistical techniques such as principal component analysis (PCA) and linear discriminant analysis (LDA) are beyond the scope of this course.

Dimensionality reduction is a group of techniques in data analysis and machine learning that involves reducing the number of variables or features in a data set while preserving as much of the relevant information as possible. This process is essential when dealing with high-dimensional data, which can lead to various problems such as increased computational cost, overfitting, and the curse of dimensionality.

How to reduce dimensions

Imagine you are working on a project to predict the most popular lunch meals in the school cafeteria. You might start by collecting data such as the colour of the food, the temperature outside, the price, how long the lunch line was, and what day of the week it is.

Think about what features (or types of information) are most likely to influence the outcome. For the lunch meals, maybe the type of food and its price are more important than the colour of the food.

Group similar features together. For instance, if two features tell you almost the same thing, you might only need one of them. If both the length of the lunch line and the waiting time tell you how busy the cafeteria is, maybe you just keep one.

Try making predictions with different sets of features to see which combinations give good results without being too complex.

Purpose of dimensionality reduction

The primary goal of dimensionality reduction is to simplify the data without losing critical information. Simplification makes the data easier to explore and visualize and can improve the performance of machine learning algorithms. The key benefits include the following.

Enhancing data visualization: Making it feasible to plot high-dimensional data in two or three dimensions.

Improving model performance: Reducing the risk of overfitting by eliminating noise and less informative details, thereby helping models to generalize better on new, unseen data.

Reducing computational resources: Decreasing the time and memory required to store and process data.

Facilitating data analysis: Making pattern recognition and correlation detection between features more manageable.

Practice questions

6. Describe the role of data preprocessing in the machine learning workflow. [2 marks]
7. Describe the significance of data cleaning and the impact of poor-quality data on model performance. [3 marks]
8. Describe three techniques used in data cleaning. [3 marks]
9. a. Define the concepts of normalization and standardization. [2 marks]
   b. Outline why they are important in data preprocessing. [2 marks]
10. a. Define the concept of feature selection. [1 mark]
    b. Outline its importance in the context of machine learning. [2 marks]

TOK

- To what extent does the quality of data influence the reliability and accuracy of knowledge derived from machine learning models?
- How do the processes of data cleaning and feature selection reflect human methods of filtering and prioritizing information?
- In what ways does the concept of dimensionality reduction in machine learning illustrate the challenges and benefits of simplifying complex information?

ATL Communication skills

Check that your responses demonstrate a clear understanding of the data preprocessing steps and their importance in the machine learning process.

Provide detailed examples and explanations to illustrate the techniques and their impacts on model performance.

Where relevant, discuss the ethical and practical implications of data quality and preprocessing decisions.

A4.3 Machine learning approaches

Syllabus understandings

AHL

A4.3.1 Explain how linear regression is used to predict continuous outcomes

A4.3.2 Explain how classification techniques in supervised learning are used to predict discrete categorical outcomes

A4.3.3 Explain the role of hyperparameter tuning when evaluating supervised learning algorithms

A4.3.4 Describe how clustering techniques in unsupervised learning are used to group data based on similarities in features

A4.3.5 Describe how learning techniques using the association rule are used to uncover relations between different attributes in large data sets

A4.3.6 Describe how an agent learns to make decisions by interacting with its environment in reinforcement learning

A4.3.7 Describe the application of genetic algorithms in various real-world situations

A4.3.8 Outline the structure and function of ANNs and how multi-layer networks are used to model complex patterns in data sets

A4.3.9 Describe how CNNs are designed to adaptively learn spatial hierarchies of features in images

A4.3.10 Explain the importance of model selection and comparison in machine learning

AHL

A4.3.1 Explain how linear regression is used to predict continuous outcomes

Linear regression is a statistical method used to model and analyse the relationships between a dependent variable and one or more independent variables.

A continuous outcome refers to a variable that can take on an infinite number of values within a given range. These values are measurable and can be expressed on a continuous scale, meaning they are not restricted to discrete or separate categories, but rather can vary gradually.

The goal is to find a linear equation that best predicts the dependent variable based on the values of the independent variables.

Linear regression assumes that the relationship between the dependent and independent variables can be described using a linear equation, which in its simplest form (single variable) is:

$$Y = \beta_0 + \beta_1 X + \in$$

- Y is the dependent variable.
- X is the independent variable.
- β_0 is the intercept of the line (the value of Y when X is 0).
- β_1 represents the slope of the line, which indicates how much Y changes for a one-unit change in X.
- $\in$ is the error term, accounting for the variation in Y not explained by X.

The relationship between the independent (predictor) and dependent (response) variables

Independent (predictor) variables are the variables that are used to predict or explain variations in the dependent variable. They are considered "independent" because their values are presumed not to be influenced by other variables in the analysis but are thought to influence the dependent variable.

The dependent (response) variable is what the model aims to predict or explain. It is dependent because its values are assumed to depend on the independent variables.

The significance of the slope and intercept in the regression equation

In linear regression, the equation typically takes the form:

$$Y = \beta_0 + \beta_1 X + \in$$

Here, β_0 (the intercept) and β_1 (the slope) are parameters that define the statistical relationship between the dependent variable Y and the independent variable X.

In linear regression, the equation typically involves two key components: the intercept and the slope. These elements define how the dependent variable (what you are trying to predict) is related to the independent variable (what you are using for the prediction).

The intercept

The intercept is the expected value of the dependent variable when the independent variable is zero. It is where the regression line crosses the *y*-axis.

The intercept represents the starting point of the dependent variable before any effect from the independent variable is considered. For example, it could show you the basic cost of a product before additional features are added. It helps position the regression line correctly, ensuring the model aligns well with the actual data.

Sometimes the intercept does not make practical sense, such as predicting a physical measurement at zero time or zero distance, but it is still a necessary part of the model's structure.

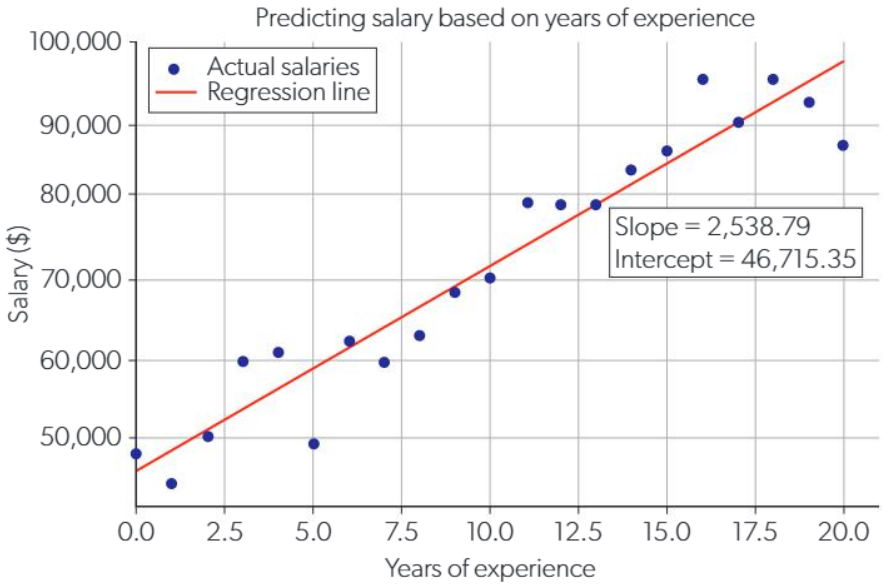
The slope

The slope indicates how much the dependent variable changes for each unit increase in the independent variable. It describes the steepness and direction of the regression line.

The slope tells you how the dependent variable responds as the independent variable increases. If the slope is positive, the dependent variable increases with the independent variable. If the slope is negative, the dependent variable decreases. The size of the slope shows the extent of the impact that changes in the independent variable have on the dependent variable. The slope directly measures how much influence the independent variable has on the dependent variable, assuming all other factors remain constant.

Example: Predicting salary based on years of experience

A model predicts an employee's salary based on their years of experience.



▲ **Figure 16** Scatter graph predicting salary earned based on experience

In the salary prediction model, the slope and intercept are clearly annotated. The slope is approximately 2,539 and the intercept is approximately 46,715. This means that the starting salary (with zero years of experience) is estimated at about $46,715, and for each additional year of experience, the salary is predicted to increase by about $2,539. These values are shown on the graph with the regression line, providing a visual representation of how salary escalates with years of experience based on the **synthetic data**.

The intercept could represent the starting salary of a new employee with zero years of experience. It reflects the baseline salary that one might expect when entering the job market, potentially including entry-level wages offered by companies. The slope indicates how much the salary increases for each additional year of experience. A positive slope means that more experienced employees earn more, highlighting the value of experience in the job market.

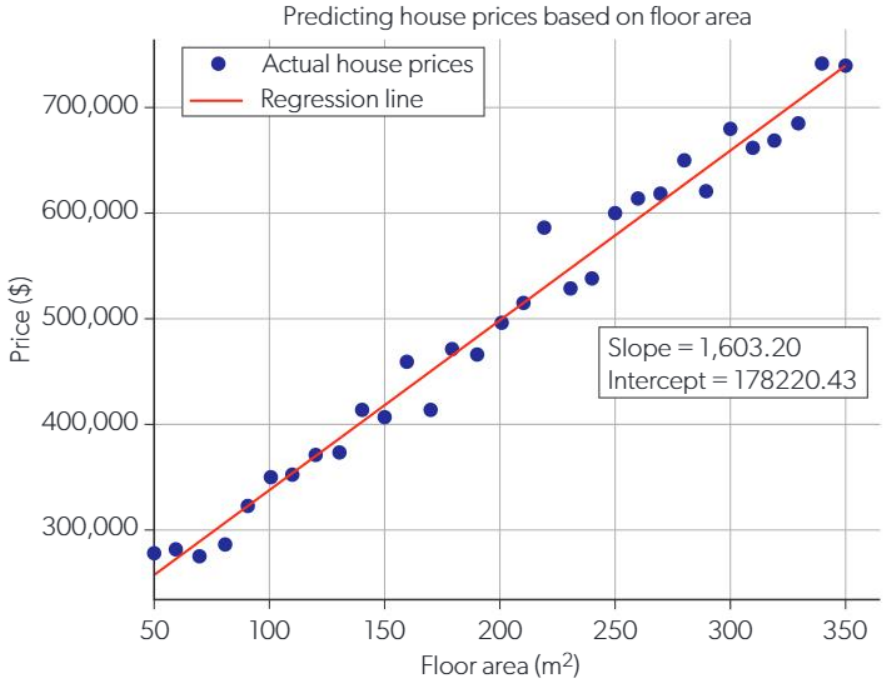
This model can help understand how career progression impacts earning potential. It allows employers and employees to project salary increases over time based on experience, aiding in career planning and compensation negotiations.

Key term

Synthetic data Artificially generated data that mimics the characteristics of real-world data but does not directly copy it. It is often used for testing, training machine learning models, and validating algorithms.

Example: Predicting house prices based on size

In this regression model, the intercept might represent the starting price of houses before considering the size based on floor area (m^2). For example, the intercept could indicate the base cost of the smallest types of properties or other fixed costs associated with house pricing that do not depend on size. The slope represents how much the price of the house increases with each additional square metre of area. If the slope is positive, it suggests that larger houses are more expensive. The slope tells you the additional cost per square metre, quantifying how property size influences its market value.



▲ **Figure 17** Scatter graph predicting house prices based on size

For the house price example, the slope and intercept values are clearly annotated. The slope is approximately 1,603 and the intercept is approximately 178,220. This means that the starting price of the smallest house (50 square metres) is predicted to be about $178,220. For each additional square metre of floor area, the price is expected to increase by about $1,603. These values provide insights into how changes in house size affect the price.

Assessing how well the model fits the data

The fit of a regression refers to how well a model's predictions correspond to the actual observations. It is a measure of the effectiveness of a model in capturing the underlying patterns or relationships between variables in the data set. Essentially, it provides an indication of how well the data points fit the regression line or model.

A commonly used measure to assess this fit is the coefficient of determination, known as r^2 (R-squared). The formula for r^2 is:

$$r^2 = 1 - \frac{SS_{res}}{SS_{tot}}$$

- SS_{res} is the sum of the squares of the model residuals (the differences between observed and predicted values).
- SS_{tot} is the sum of the squares of the differences from the observed values to the mean of the observed data.
- An r^2 value of 0 indicates that the model explains none of the variability of the response data around its mean.
- An r^2 value of 1 indicates that the model explains all the variability of the response data around its mean.

Key terms

Classification techniques A subset of algorithms designed to assign categorical labels to items based on input data.

Discrete categorical outcomes the type of variable in statistics and data analysis that can take on a **limited, fixed number** of possible values, where each value represents a category or class.

You learned about supervised learning in section A4.1.

A4.3.2 Explain how classification techniques in supervised learning are used to predict discrete categorical outcomes

The K-Nearest Neighbours (K-NN) and decision trees algorithms

The K-Nearest Neighbours (K-NN) and decision trees algorithms are both used to categorize new data points based on patterns learned from existing labelled data, but they operate in fundamentally different ways.

K-Nearest Neighbours

K-NN is a **non-parametric**, **instance-based** learning algorithm.

- Non-parametric methods are those that do not assume a fixed form or a specific distribution model for the underlying data. In other words, non-parametric means flexible. K-NN does not have a strict rulebook to follow. It adapts to whatever data you give it.
- Instance-based learning is a category of machine learning algorithms that base their predictions on specific examples from the training data rather than deriving general rules.

K-NN classifies new cases based on a **similarity measure**, a metric used to determine how alike two data objects are. K-NN is called "lazy learning" because it does not explicitly learn a model. Instead, it memorizes the training instances which are then used as "knowledge" for the prediction phase.

How K-NN works

1. When a new data point needs to be classified, K-NN calculates the distance from this point to all other points in the training set.
2. It then selects the K nearest points, where K is a user-specified number, based on these distances. The nearest points are also called nearest neighbours.
3. Classification is performed by a majority vote to its nearest neighbours. The new data point is assigned the class most common among its K nearest neighbours.

K-NNs are simple and effective, make no assumptions about the data (making it versatile, working with any number of classes), and are easy to implement for multi-class problems. K-NNs can be computationally expensive because they need to calculate the distance for each instance and sort all the instances during the classification phase. They are generally not suitable for large data sets or data sets with high dimensionality without dimension reduction. Finally, they are sensitive to noisy data, missing values, and outliers.

Worked example 2

Classifying fruits using K-NN

You have a data set with information about different fruits. These are classified either as Apple or Banana. Each fruit in our data set is described by two features: weight (grams) and colour code (numeric).

Table 4 The training data set

Fruit	Weight (grams)	Colour code	Label
Fruit1	150	1	Apple
Fruit2	170	1	Apple
Fruit3	130	2	Banana
Fruit4	180	2	Banana

You receive a new fruit with the following features: Weight is 160 grams and the colour code is 1.

Classify the new fruit based on your model. Use K = 3.

Solution

Step 1: Apply K-NN with K = 3.

Decide whether this new fruit is an apple or a banana based on the three nearest neighbours in the data set.

a. Distance calculation: Calculate the Euclidian distance from the new fruit to each fruit in our data set. The Euclidean distance between two points (x_1, y_1) and (x_2, y_2) is given by the formula:

$$\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

Let the weight values be x and the colour codes be y. Using the Euclidian distance formula, the distance from the new fruit (160, 1) to Fruit1 (150, 1) is $\sqrt{(160-150)^2 + (1-1)^2} = \sqrt{100-0} = 10$.

Repeat the calculation for all the other fruits in the data set.

This gives:

Distance to Fruit1 = 10

Distance to Fruit2 = 10

Distance to Fruit3 = 30.41

Distance to Fruit4 = 20.62

b. Nearest neighbours: The three closest fruits are Fruit1, Fruit2, and Fruit4 based on the calculated distances (Fruit1, Fruit2, Fruit4).

Step 2: Majority voting

Consider the three nearest neighbours: are they labelled as Apples or Bananas? They may not all be the same, but there will be a majority. Use this majority to decide how to classify the new fruit.

1. Fruit1: Apple
2. Fruit2: Apple
3. Fruit4: Banana

Since two out of the three nearest neighbours are labelled as Apple, the new fruit is classified as an Apple.

By examining the majority label among the closest data points (neighbours), K-NN classifies the new fruit. This simple example demonstrates how K-NN uses distance measurements and majority voting among nearest neighbours to classify new data points based on existing labelled data.

AHL

Decision trees

Decision trees are a flowchart-like tree structure where an internal node represents a feature (or attribute), the branch represents a decision rule, and each leaf node represents the outcome. The topmost node in a tree is called the root node. It breaks down a data set into smaller and smaller subsets while at the same time an associated decision tree is incrementally developed.

How decision trees work

1. Begin at the root node and **split** the data on the feature that results in the most homogeneous child nodes (subsets) based on some splitting criterion (for example, Gini impurity, entropy).
2. Apply the splitting process **recursively** to each child until one of the termination conditions is met (for example, all the instances at a node belong to the same class, no remaining attributes to split on, or reaching a maximum depth).
3. Optionally, reduce overfitting by **pruning** the tree to remove splits that have little importance.

Decision trees are easy to understand and interpret. In addition, they can handle both numerical and categorical data and they require relatively little data preparation (no need for feature scaling, for example).

Worked example 3

Classifying fruits using decision trees

Here is a hypothetical fruit data set.

The features in this data are:

- Weight (grams): numerical
- Colour code: numerical (1 = Red, 2 = Yellow, 3 = Orange)
- Sweetness level: numerical (scale of 1–10).

Labels:

1. Apple
2. Banana
3. Orange

(Note the use of some synthetic data for the sake of this example.)

```
fruit_labels = np.array([0, 0, 1, 1, 2, 2, 0, 1, 2])  # Labels for the fruits
```

Classify the new fruit based on your model.

Solution

Create and plot the decision tree.

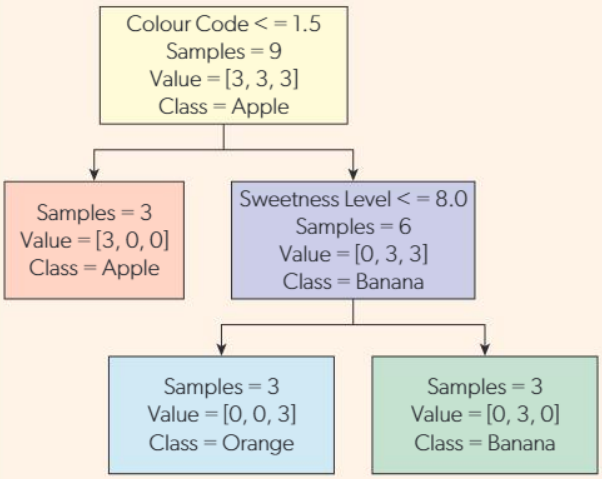
To create a simple decision tree classifier to learn from this data you would do the following.

```
# Features: Weight (grams), Colour Code, Sweetness Level
# Labels: 0 = Apple, 1 = Banana, 2 = Orange
    fruit_array = np.array([

        [150, 1, 8],  # Apple
        [100, 1, 7],  # Apple
        [120, 2, 9],  # Banana
        [130, 2, 10], # Banana
        [110, 3, 6],  # Orange
        [160, 3, 7],  # Orange
        [140, 1, 8],  # Apple
        [135, 2, 9],  # Banana
        [145, 3, 7]   # Orange

    ])
```

1. **Define the data:** Create arrays for features and labels.
2. **Train the decision tree:** Fit the model to the data.
3. **Visualize the decision tree:** Use `plot_tree` to visualize the decisions made by the tree.



▲ **Figure 18** A decision tree that categorizes fruit based on weight, colour and sweetness level

Figure 18 shows the decision tree based on your synthetic fruit data set. The tree uses features such as weight, colour code, and sweetness level to classify fruits into categories: apple, banana, and orange. Each node in the tree makes decisions based on these features, guiding to a classification at the leaves. This visualization illustrates the decision-making process the tree follows to determine the type of fruit based on the given attributes.

Decision trees can be prone to overfitting, especially with complex trees. If some classes dominate, you can create biased trees. Decisions are also based on axis-aligned splits, which might not capture the true decision boundaries.

While K-NN is based on feature similarity, decision trees use a hierarchical decision-making process. The choice between K-NN and decision trees depends on the size of the data set, the nature of the problem, and the complexity of the relationships in the data.

Real-world applications of K-NN

Collaborative filtering (CF) recommendation systems recommend items by finding similarities between users or items based on past interactions. K-NN can be utilized in both user-based and item-based collaborative filtering.

User-based CF implements a similarity calculation. K-NN is used to find users who are similar to the target user based on their ratings or interactions with items. Items liked or highly rated by these similar users are recommended to the target user, assuming that users with similar tastes in the past will have similar preferences in the future.

Item-based CF analyses similarity among items. K-NN helps in finding items similar to those that the target user has liked or rated highly. The algorithm recommends these similar items to the user, based on the idea that a user will likely enjoy items similar to those they have liked before.

An example of user-based CF with book recommendations

This is an example of user-based collaborative filtering using the K-NN algorithm, focused on a book recommendation system for an online bookstore. This example will detail how you can recommend books to users based on the similarity of their reading patterns to those of other users.

The goal is to provide personalized book recommendations by finding users with similar reading interests and using their preferences to suggest books.

Step 1: Data collection

For this example, data collected includes:

- user interactions, where each user's interactions are tracked, including which books they have read, their ratings for these books (on a scale of 1 to 5), and other actions such as reviews or bookmarks
- book metadata, such as genre, author and publication year, although primary recommendations are based on user behaviour rather than content.

Step 2: Data preprocessing

Construct a user–book matrix where rows represent users and columns represent books. Entries in this matrix are the ratings given by users to books. This matrix is likely sparse, as not all users rate all books.

Step 3: User-based K-NN collaborative filtering

Calculate the similarity between users based on their book rating patterns. Then, identify the K nearest users (neighbours) to a target user—those who have the most similar taste in books based on the ratings.

Step 4: Generating recommendations

Choose a user for whom recommendations are to be made. For books that the target user has not rated, predict potential ratings by averaging the ratings of these books by the nearest neighbours, weighted by their similarity to the target user. Finally, create a list of books with the highest predicted ratings that the target user has not read yet.

Example implementation

Users 1, 2 and 3 have rated books A, B and C.

Table 5 Sample data

	Book A	Book B	Book C
User1	4	5	-
User2	5	3	4
User3	-	4	5

Recommendations will be generated for User1. First, calculate how similar User2 and User3 are to User1 based on their ratings for Books A and B. Next, consider both Users 2 and 3 since K = 2. Predict User1's rating for Book C using the ratings from Users 2 and 3, weighted by their similarity to User1.

This user-based collaborative filtering method using K-NN helps the bookstore offer personalized recommendations that are likely to appeal to the user, based on similar users' preferences. This system can significantly enhance user experience by helping them discover books that align with their tastes, potentially increasing user engagement and sales.

An example of item-based CF with book recommendations

This approach focuses on finding relationships between items (books in this case) based on the user ratings, and then recommending items similar to those a user has liked.

The steps for data collection, data preprocessing, K-NN collaborative filtering and generating recommendations are identical to the example above.

Example implementation

Suppose we have the following user–book matrix (entries are ratings).

Table 6 Sample data

	User1	User2	User3	User4
Book A	4	5	-	3
Book B	3	-	2	5
Book C	5	3	5	-
Book D	-	3	4	4

Calculate similarities and make recommendations

Calculate the similarity between Book A and all other books based on user ratings. Assume User1 rated Book A highly. Calculate the similarity scores for Book A with Books B, C and D. Identify the books most similar to Book A (let's say Books C and D have higher similarity scores). Check if User1 has not rated Books C and D. Recommend Book C and Book D to User1 based on their high similarity to Book A, which User1 liked.

Real-world applications of decision trees

Decision trees can be practically applied in the context of diagnosing medical conditions based on a patient's symptoms.

An example of decision trees with medical diagnosis

Imagine a healthcare system where medical practitioners use decision trees to assist in diagnosing diseases based on symptoms presented by patients. This can be especially useful in triage systems, primary care settings, or remote areas where resources are limited.

How decision trees are applied

Step 1: Data collection

For this example, data collected includes:

- patient history, including information such as age, sex, medical history and lifestyle factors
- recorded symptoms experienced by the patient
- results from laboratory tests, which might include blood work, imaging studies, and so on.

Step 2: Decision tree model

The decision tree is trained on historical data of patients, including the symptoms and outcomes diagnosed by medical professionals. Each node in the tree represents a symptom or a test result, and branches represent possible values or outcomes of these tests or symptoms. The leaves of the tree represent potential diagnoses.

Application in diagnosis

When a new patient presents with a set of symptoms, the decision tree is used to navigate through these symptoms and/or test results.

Starting from the root of the tree, decisions are made at each node based on the patient's specific symptoms or test results, moving through the branches until reaching a leaf node which suggests a potential diagnosis.

Benefits of using decision trees in medical diagnosis

Interpretability: One of the significant advantages of decision trees is their ease of interpretation. Medical professionals can easily understand the diagnostic path taken by the tree, which aids in trust and transparency.

Speed: Decision trees can quickly process new patient data, providing rapid diagnostics that are essential in acute medical situations.

Cost-effective: They can reduce the need for unnecessary tests by identifying probable diagnoses based on symptom presentation alone.

Consistency: Decision trees provide standardized diagnosis procedures that ensure consistency in patient care, irrespective of the individual healthcare provider.

Example of using decision trees in medical diagnosis

Suppose a decision tree is developed for diagnosing respiratory illnesses. It might start with symptoms such as cough and fever as top-level nodes, then delve deeper based on other symptoms and factors.

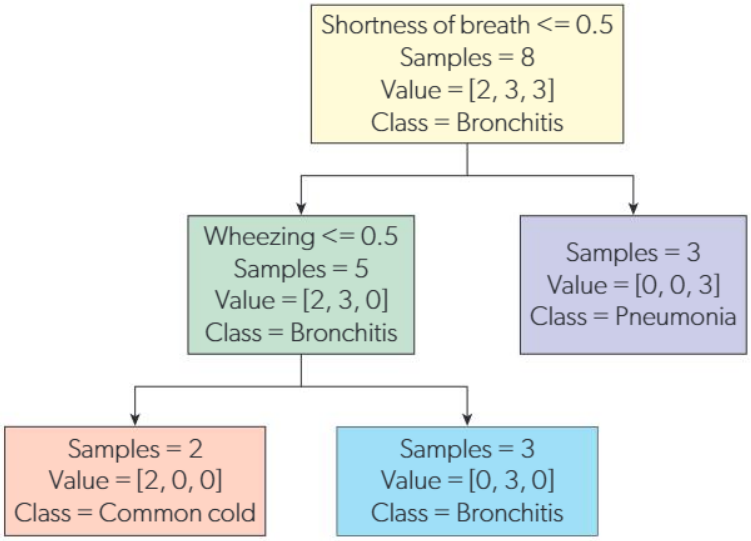
A branch might explore the presence of additional symptoms like shortness of breath or wheezing.

Further branches might differentiate between types of coughs (dry or productive), fever duration, and other accompanying symptoms or pre-conditions such as asthma or smoking.

Ultimately, the tree could help distinguish between conditions such as bronchitis, pneumonia, or common colds, guiding the medical professional on potential treatment plans or further tests needed.

```
# Synthetic data set
# Features: [cough, fever, shortness of breath, wheezing]
# cough, fever, shortness of breath, wheezing: 0 = no, 1 = yes
X = np.array([
[1, 1, 0, 0],  # Common Cold
[1, 1, 0, 1],  # Bronchitis
[1, 1, 1, 1],  # Pneumonia
[0, 1, 1, 0],  # Pneumonia
[1, 0, 0, 0],  # Common Cold
[1, 1, 1, 0],  # Pneumonia
[1, 0, 0, 1],  # Bronchitis
[1, 1, 0, 1],  # Bronchitis
])

# Labels: 0 = Common Cold, 1 = Bronchitis, 2 = Pneumonia
y = np.array([0, 1, 2, 2, 0, 2, 1, 1])
```



▲ **Figure 19** A decision tree for diagnosing respiratory illnesses

A4.3.3 Explain the role of hyperparameter tuning when evaluating supervised learning algorithms

Hyperparameter tuning involves adjusting the settings of an algorithm that are fixed before the learning process begins and cannot be learned from the data. These settings, known as hyperparameters, significantly influence the performance of machine learning models. Understanding and effectively tuning these hyperparameters can lead to more accurate, efficient and robust models.

Accuracy, precision, recall and F1 score as evaluation metrics

Accuracy, precision, recall and the F1 score are metrics used to evaluate the performance of classification models in machine learning. Each metric provides insights into different aspects of model performance.

Accuracy

Accuracy measures the overall correctness of the model and is defined as the ratio of correctly predicted observations to the total observations.

Worked example 4

A model predicts whether an email is spam (positive class) or not spam (negative class). Out of 100 emails, the model correctly identifies 90 emails (both spam and not spam).

Work out the accuracy of the model.

Solution

The accuracy is a model is given by the formula:

$$\text{accuracy} = \frac{\text{number of correct predictions}}{\text{total number of predictions}}$$

For this model:

$$\text{accuracy} = \frac{90}{100} = 0.90 \text{ or } 90\%.$$

Thus, the accuracy is 90%.

Precision

Precision is the ratio of correctly predicted positive observations to the total predicted positives. It measures the accuracy of positive predictions.

Worked example 5

The model from Worked example 4 predicts 50 spam emails, of which 40 are actually spam.

Calculate the precision of the model for predicting spam.

Solution

Precision is calculated by the formula:

$$\text{precision} = \frac{\text{true positives (TP)}}{\text{true positives (TP)} + \text{false positives (FP)}} = \frac{40}{50} = 0.80 \text{ or } 80\%$$

So, the precision of this model for predicting spam is 80%.

Recall (sensitivity)

Recall measures the ability of a model to find all the relevant cases within the positive class. It is defined as the ratio of correctly predicted positive observations (true positives) to all observations that are actually in the positive class.
This includes both the true positives and the observations that were positive but were incorrectly classified (false negatives).

Worked example 6

There are 60 actual spam emails in a sample, and the model correctly identifies 40 of them as spam.

Calculate the recall of the model.

Solution

The calculation for the recall of a model is:

$$\text{recall} = \frac{\text{true positives (TP)}}{\text{true positives (TP)} + \text{false positives (FP)}} = \frac{40}{60} = 0.67 \text{ or } 67\%$$

So, the recall of the model is 67% (correct to 2 significant figures).

Note: the calculations for precision and recall are very similar. Make sure you know the difference and use the correct formula.

F1 Score

The F1 Score is the weighted average of precision and recall. This score takes both false positives and false negatives into account. It is particularly useful when the class distribution is uneven. The F1 score is the harmonic mean of precision and recall.

Worked example 7

Using the precision and recall from the previous worked examples, calculate the F1 score of the model.

Solution

The calculation for the F1 Score uses the formula:

$$\text{F1 Score} = 2 \times \left(\frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}} \right)$$

$$= 2 \times \left(\frac{0.80 \times 0.67}{0.80 + 0.67} \right) = 0.73 \text{ or } 73\%$$

These metrics provide a more holistic view of model performance beyond simple accuracy.

Table 7 Model performance measures

Measure	Usage scenario
Accuracy	Useful when the target classes are well balanced.
Precision	Helpful when the cost of a false positive is high (for example, in spam filtering, labelling an important email as spam).
Recall	Important when the cost of a false negative is high (for example, in disease screening, failing to identify a sick patient).
F1 Score	Useful when you need a balance between precision and recall, especially if there is an uneven class distribution.

The role of hyperparameter tuning on model performance

A hyperparameter is a parameter whose value is set before the learning process begins and cannot be learned from the data during training. Hyperparameter tuning involves systematically searching for the optimal set of hyperparameters that governs the learning process of an algorithm. These hyperparameters significantly influence how well a model can fit the data and generalize to new data.

An example of a hyperparameter in K-NN would be the number of nearest neighbours to consider when making a prediction. For a decision tree, a hyperparameter would be the maximum depth of the trees, or the minimum number of samples required to split an internal node.

Overfitting and underfitting when training algorithms

Overfitting

Overfitting occurs when a model is too closely fit to a limited set of data points and captures the noise along with the underlying data distribution. As a result, while the model performs exceptionally well on its training data, it performs poorly on new, unseen data.

Worked example 8

Imagine training a complex decision tree to classify whether animals are dogs or cats based on a series of features (for example, size, weight, ear shape).

What are potential problems in the training process which could lead to inaccurate results?

Solution

If the tree is allowed to grow deep without constraints, it might start making decisions based on irrelevant features (such as a specific collar colour seen frequently on the training dogs). Such a model might perform perfectly on the training set (100% accuracy) because it has essentially memorized the data set, including noise and outliers, but it will likely perform poorly on new examples of dogs and cats, especially those that do not fit the specific characteristics of the training set.

Addressing overfitting requires the following.

- Simplify the model by selecting a less complex model or reducing the number of parameters.
- Use regularization techniques which penalize overly complex models.
- Include more training data to cover a broader range of the input space.
- Use techniques like cross-validation to ensure the model's ability to generalize.

Underfitting

Underfitting occurs when a model is too simple to learn the underlying pattern of the data. Consequently, it performs poorly on the training data, and this poor performance also extends to new, unseen data.

Worked example 9

Can you train a very simple model to predict whether an animal is a dog or a cat using only one feature? What are potential problems in the training process or training data which could lead to inaccurate results?

Solution

For example, consider training a model using only the size of the animal. Such a model might hypothesize that all large animals are dogs and all small animals are cats. This overly simplistic approach fails to capture the complexity of the data (for example, there are small dogs and large cats), resulting in high error rates on both the training data and new examples.

Addressing underfitting involves the following.

- Increase the complexity of the model by adding more parameters or using more sophisticated learning algorithms.
- Reduce or remove regularization, which might be constraining the model too much.
- Feature engineering: adding more relevant features that could help the model make better predictions.

AHL

A4.3.4 Describe how clustering techniques in unsupervised learning are used to group data based on similarities in features

Clustering is a technique used in data analysis and statistics that involves grouping a set of objects in such a way that objects in the same group (or cluster) are more similar to each other than to those in other groups. It is a method of unsupervised learning, and it's commonly used for statistical data analysis used in many fields, including machine learning, pattern recognition, image analysis, information retrieval and bioinformatics.

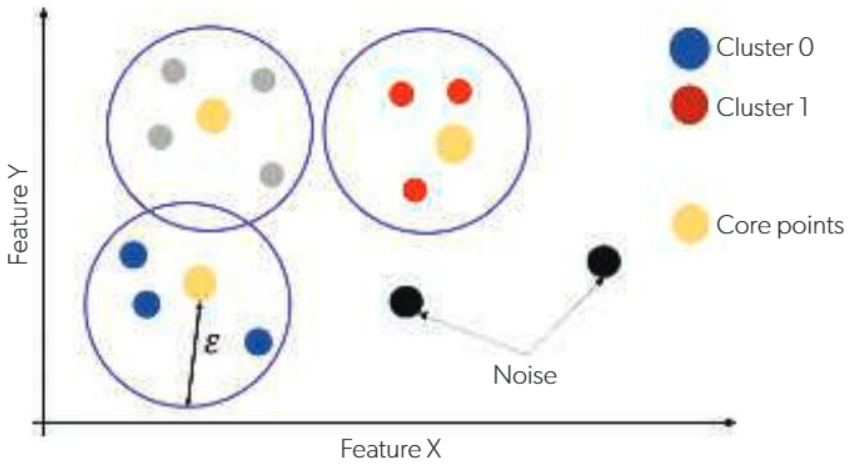
Clustering techniques in unsupervised learning group data

In unsupervised learning (unlabelled data), clustering techniques identify distinct groups or clusters from large data sets, where each group represents data points with similar characteristics.

K-means clustering

K-means clustering is an algorithm which partitions the data set into k distinct, non-overlapping clusters. It assigns each data point to the closest cluster centre (**centroid**), minimizing the within-cluster sum of squares (variance). K-means clustering is ideal for quick preliminary analyses, suitable for large data sets.

A centroid is typically calculated as the mean position of all the points in a cluster, and represents the average location of all data points within that cluster.



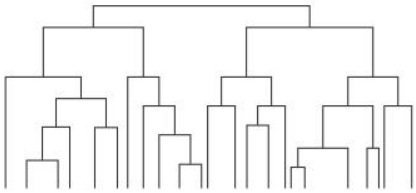
▲ **Figure 20** Example of data classification using centroids and k-means

The basic steps for k-means clustering are as follows.

1. Initialize k centroids randomly.
2. Assign each point to the nearest centroid.
3. Recalculate the centroid of each cluster.
4. Repeat steps 2 and 3 until the centroids stabilize.

Key term

Centroid In the context of clustering algorithms, the centre of a cluster.



▲ **Figure 21** A dendrogram

Hierarchical clustering

Hierarchical clustering builds clusters by either a divisive method (top-down) or agglomerative method (bottom-up). It creates a tree of clusters called a dendrogram, offering a visualization of data groupings at different scales. This type of clustering is useful for data sets where tree-like relationships are important, such as in taxonomy creation.

The basic steps for hierarchical clustering are as follows.

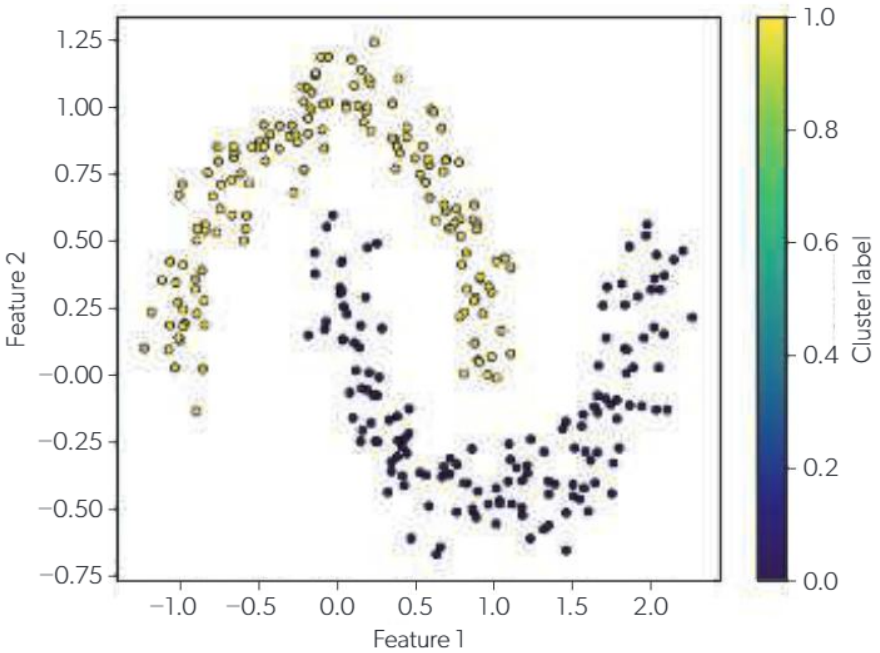
1. Treat each data point as a single cluster.
2. Merge the closest pair of clusters into one.
3. Recalculate distances between clusters.
4. Repeat until all points are merged into a single cluster.

Density-based spatial clustering of applications with noise

Density-based spatial clustering of applications with noise (DBSCAN) clusters points that are closely packed together, marking as outliers the points that lie alone in low-density regions. It does not require the number of clusters to be specified beforehand. This type of clustering is especially effective for complex geometrical data and is resistant to outliers.

The basic steps for DBSCAN are as follows.

1. Define two parameters, ε (eps) and the minimum number of points required to form a dense region (minPts).
2. Mark all points within ε distance of a point as neighbours.
3. Create a cluster if a point has at least minPts neighbours.
4. Expand the cluster by adding all density-reachable points.
5. Repeat the process for all unvisited points.



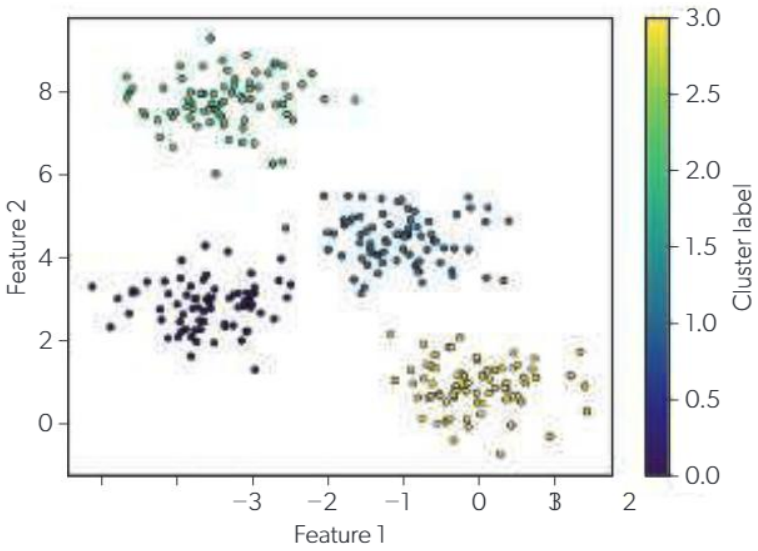
▲ **Figure 22** DBSCAN clustering

Mean shift clustering

Mean shift clustering aims to discover blobs in a smooth density of samples. It is a centroid-based algorithm, which works by updating candidates for centroids to be the mean of the points within a given region. Mean shift clustering works well with clusters of arbitrary shapes and sizes. It does not require specifying the number of clusters in advance (unlike k-means).

The basic steps for mean shift clustering are as follows.

1. Start with an initial estimate for the centroid location.
2. Compute the mean of the points within a sliding window centred at the centroid.
3. Translate the centroid to the mean location.
4. Repeat until convergence.

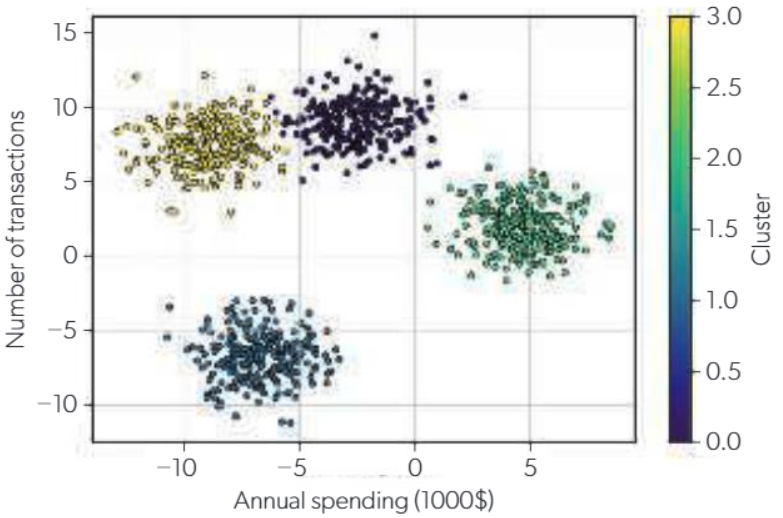


▲ **Figure 23** Mean shift clustering

Real-world applications of clustering

This visualization shows the segmentation of customers into distinct groups, each potentially representing a different type of buyer behaviour, such as:

- high spenders with many transactions (possibly premium customers)
- low spenders with few transactions (possibly occasional shoppers)
- customers with high spending but fewer transactions (possibly bulk buyers)
- customers with lower spending but higher transactions (possibly regular, small amount buyers).



▲ **Figure 24** Customer segmentation

Clustering is extensively used across various industries to segment data into meaningful groups. One of the real-world applications of clustering involves using purchasing data to segment a customer base. This process, commonly known as customer segmentation, helps businesses understand their customers better, tailor marketing strategies, enhance customer service, and optimize product offerings.

Market segmentation

Market segmentation identifies different groups of customers based on their purchasing behaviour, demographics and preferences. Clustering algorithms such as k-means, hierarchical clustering or DBSCAN are used to group customers who exhibit similar purchasing patterns. Marketers can tailor campaigns that are specifically designed for each segment, improving the efficiency of marketing efforts and increasing customer satisfaction.

Personalized marketing

Personalized marketing develops marketing strategies that cater to the specific needs and desires of different customer segments. It works by analysing clustered data to identify the preferences and needs of each group, such as frequent purchases, preferred product types, and price sensitivity. The goal is to create personalized promotions, targeted advertisements and custom product recommendations that resonate with each distinct group, leading to higher conversion rates and customer loyalty.

Pricing strategy

Another real-world application of clustering is to develop pricing strategies. The goal is to appeal to various customer segments. Clustering allows businesses to identify segments that may be more sensitive to price changes or willing to pay a premium for certain features. Dynamic pricing strategies can be applied where prices are adjusted based on the purchasing power and behaviour of each segment, maximizing revenue while maintaining customer satisfaction.

A4.3.5 Describe how learning techniques using the association rule are used to uncover relations between different attributes in large data sets

Association rule learning is a technique used to discover interesting relations between variables in large databases. It establishes "interestingness" by using specific techniques such as support and confidence, which measure the strengths of the associations. Association rule learning is used in unsupervised learning (meaning the data is unlabelled). It focuses on identifying rules that describe how often items co-occur in a given data set.

While market basket analysis is a common application, association rule learning has applications in various domains including fraud in financial transactions or user behaviour analysis on websites.

Table 8 Concepts and terms in association rule learning

Concept/term	Definition
Itemsets	A collection of one or more items. These are collections of one or more items in the data set.
Support	An indication of how frequently an itemset appears in the data set. This measure gives the proportion of transactions that include a particular itemset. It helps in identifying the most significant relationships in the data set

Concept/term	Definition
Confidence	A measure of the reliability of the inference made by an association rule. This measure provides the likelihood of finding a given association. It is defined as the ratio of the support of the entire rule to the support of the antecedent (items on the left side of the rule).
Lift	A measure of how much more often items in a rule A → B occur together than expected if they were statistically independent. This is the ratio of the observed support to that expected if the two rules were independent. A lift value greater than one suggests that the presence of one item in a transaction increases the likelihood that another item will also appear in the same transaction.

Association rule mining

Association rule mining is a technique used to find patterns, correlations or associations among sets of items in transactional or relational data sets.

1. After you have received and prepared the data to be analysed, identify frequent itemsets. Using an algorithm like Apriori will identify itemsets that appear frequently in the data set. This step involves defining a minimum support threshold, which is the minimum frequency (as a percentage of the total) an itemset must have to be considered further.
2. From the frequent itemsets, generate association rules that predict the occurrence of an item based on the presence of other items in the transaction. This step requires setting a minimum confidence threshold, which is the minimum probability with which a rule must hold true.
3. Evaluate the generated rules using metrics such as lift, conviction and leverage to determine their effectiveness and strength.

The Apriori algorithm

The Apriori algorithm is a popular data mining method used for mining frequent itemsets and relevant association rules.

1. Set a minimum support and confidence. Start by defining the minimum thresholds for support and confidence that an itemset and a rule must meet to be considered significant.
2. Generate candidate itemsets. Start with one-item itemsets (frequent items). These are single items that meet the minimum support threshold.
3. Determine frequent itemsets. For each potential itemset, calculate its support by counting how often it appears in the data set. Compare this with the minimum support.
4. Create higher order itemsets. Once all frequent one-item itemsets are found, pairs of these frequent itemsets are merged to form two-item itemsets. This process is repeated to create three-item itemsets, four-item itemsets, and so on, until no more frequent itemsets can be generated.

5. Generate association rules from the frequent itemsets. For each itemset that meets the minimum support threshold, generate all possible rules. Calculate the confidence of each rule and compare it to the minimum confidence threshold.
6. Rule pruning: Rules that do not meet the confidence threshold are pruned (discarded).

Worked example 10

A supermarket wants to analyse customer buying habits using Apriori in market basket analysis. What might its findings be using Apriori?

Solution

Using the Apriori algorithm, the supermarket might create the following results.

- Itemsets such as {bread}, {milk}, and {butter} have high support individually.
- Combining them, they find that the itemset {bread, milk} also has high support.
- From this, they generate rules like {bread} → {milk} if the confidence is high enough.
- Further analysis could reveal that {bread, milk} → {butter} is a frequent and reliable rule with a high lift, indicating that customers who buy both bread and milk are much more likely to buy butter than randomly selected customers.

Worked example 11

1. Describe the process the police might follow to find associations between different types of crimes, across various neighbourhoods.
2. After running association rule learning techniques, it is revealed that:

   Frequent Itemset: {Vandalism, Theft} with high support.

   What are the implications of this for law enforcement? How might they use this information to improve policing strategies?



▲ **Figure 25** How can the police connect crimes and respond effectively?

Solution

1. The police might follow to find associations between different types of crimes, across various neighbourhoods would follow the steps below.

   Step 1: Data preparation

   Gather data: collect crime reports from multiple neighbourhoods over several years. Each report includes details like the type of crime (for example, theft, vandalism, assault) and the location. Treat each neighbourhood's crime report over a defined period (for example, a month) as a single transaction. Each transaction contains items, which in this case are the types of crimes reported.

   Step 2: Define minimum support and confidence

   Set a minimum level of support to identify frequent sets of crimes that occur together in the same neighbourhoods. Also set a minimum confidence level to determine the strength of the association between the crimes.

   Step 3: Generate candidate itemsets

   Start with individual crimes as one-item itemsets to see which crimes occur frequently.

   Step 4: Determine frequent itemsets

   Calculate the support for each itemset. For example, find the percentage of neighbourhoods where theft alone or vandalism alone was reported in a month.

Step 5: Create higher order itemsets

Combine frequent one-item itemsets to form two-item itemsets (e.g., {theft, vandalism}). Continue this process to identify frequent combinations of different crimes.

Step 6: Generate association rules

From the frequent itemsets, generate rules such as {vandalism} → {theft}.

Calculate the confidence of each rule to ensure it meets the threshold, indicating that if vandalism is reported in a neighbourhood, there is a high likelihood of theft also being reported.

Step 7: Rule pruning

Discard rules that do not meet the minimum confidence level.

2. You could create an association rule where {Vandalism} → {Theft} with a confidence of 75% and a lift significantly above 1. This indicates that theft is much more likely in areas where vandalism is reported compared to areas where it is not.

   This information can be used by police departments to focus resources and patrols in neighbourhoods where vandalism has been reported to prevent potential thefts, allocate more investigative resources to areas with high rates of these co-occurring crimes to reduce overall crime rates, and implement community outreach programs targeting vandalism and theft prevention.

A4.3.6 Describe how an agent learns to make decisions by interacting with its environment in reinforcement learning

In reinforcement learning (RL), an agent refers to the entity or algorithm that is tasked with making decisions in a specific environment. The agent's objective is to learn how to behave or choose actions that maximize some notion of cumulative reward through interactions with its environment. This learning process involves a series of steps where the agent observes the state of the environment, takes actions and receives feedback in the form of rewards or penalties based on the outcomes of its actions.

Table 9 Examples of some agents

Agent	Why is this an agent?
Robotic vacuum cleaner	A robotic vacuum cleaner is an agent because it autonomously senses its environment using sensors and performs actions like moving and cleaning. Its actions are driven by the goal of keeping the floor clean. Receives rewards for positive actions (such as collecting dirt or cleaning efficiently) and learns to refine its cleaning strategies over time to maximize these rewards.
Autonomous drone (for delivery or surveillance)	Environment: Geographic terrain, weather conditions, obstacles, regulatory constraints. Objective: Complete delivery or surveillance tasks efficiently and safely. Actions: Navigate routes, avoid obstacles, optimize delivery schedules. Feedback: Delivery times, energy consumption, avoidance of accidents.
Game-playing AI	An AI trained to play games like chess, Go, or Atari titles is an RL agent. Its environment is the game board, and its actions are the moves it selects.
Stock trading bot	A software system designed to autonomously buy and sell stocks in the financial market can function as an RL agent. The environment for this agent is complex and dynamic, influenced by various factors like real-time stock prices, news events, economic indicators, and overall market trends. The agent's actions involve buying or selling specific stocks at particular times. The reward signal in this case could be the profit earned from these trades. Through continuous interaction with the financial market, the RL agent learns to identify patterns, analyse trends, and make informed trading decisions with the goal of maximizing its returns.

In reinforcement learning, the principle of cumulative reward and the foundational concepts of agent–environment interaction are critical.

Table 10 Reinforcement learning terms

Term	Definition
Agent–environment interaction	An agent interacts with an environment over time through a series of steps. At each step, the agent performs an action, and the environment responds by presenting a new state and a reward. This interaction is central to how learning occurs.
States	A state represents the current situation of the environment. It is what the agent observes and uses to make decisions. The set of all possible states is called the state space.
Actions	Actions are the set of all possible moves or decisions the agent can make in a given state. The collection of all actions available to the agent is known as the action space.
Rewards	A reward is a feedback from the environment that evaluates the efficacy of the agent's last action. Rewards can be positive or negative, providing a scalar signal to the agent to assess its performance.
Policies	A policy is a strategy employed by the agent, mapping states to actions. It determines the action that an agent should take in a given state. In RL, policies can be deterministic, where a state directly corresponds to an action, or stochastic, where an action is chosen based on a probability distribution.
Cumulative reward	Also known as the return, cumulative reward is the total amount of reward an agent accumulates over time. The goal in many RL problems is to maximize this cumulative reward. It is often calculated as the sum of rewards collected over steps, potentially discounted by a factor at each step to prioritize immediate rewards over distant ones.

The exploration versus exploitation trade-off

The exploration versus exploitation trade-off is a concept that addresses how an agent should balance the decision between exploring new actions and exploiting known actions to maximize its cumulative reward.

Exploration

Exploration is the process by which an agent tries new actions to discover their potential rewards. It involves choosing actions with less certainty about their outcomes to gather more data about the environment. Exploration allows the agent to improve its understanding of the environment, potentially discovering more rewarding actions that were previously unknown. Too much exploration can lead to suboptimal short-term results as the agent might choose inferior actions in the process of learning.

Exploitation

Exploitation involves choosing actions that the agent already knows to yield the highest reward based on its current knowledge. The idea is to leverage the information the agent has accumulated to make the best possible decision in a given state. By exploiting known good actions, the agent maximizes its immediate reward, effectively utilizing its existing knowledge. Solely focusing on exploitation can lead the agent to potentially miss out on discovering better options. This could result in a suboptimal policy if better states or actions remain unexplored.

Balancing exploration and exploitation

Common strategies include the following.

Epsilon-greedy strategy: An agent exploits the best-known action most of the time but explores random actions with a small probability (epsilon). This probability can decay over time as the agent becomes more confident in its knowledge.

Upper confidence bound (UCB): A method which uses statistical confidence bounds to balance the trade-off, choosing actions that maximize an upper confidence limit of the expected reward.

Thompson sampling: Maintains a probability distribution of how good each action might be. Based on these distributions, the agent randomly samples an expected reward for each action and chooses the action with the highest sampled reward.

The optimal balance depends on the specific characteristics of the environment and the learning task. The key challenge in real life is to devise a policy that adequately balances exploration with exploitation, thus maximizing long-term rewards by efficiently learning the environment's dynamics.



▲ **Figure 26** A game-playing agent interacts with its environment in a platform game

A game-playing agent example

With a game-playing agent, the environment is a virtual world in a platform game featuring various terrains, obstacles (such as gaps and enemies) and rewards (such as coins and power-ups). The agent is a character controlled by the AI in the game.

Table 11 Key factors for game-playing agents

Agent factor	Description
State	Can include the character's position on the screen, the types of terrain immediately around the character, the presence of obstacles or enemies nearby, and any collectibles within reach.
Actions	The agent can move left, move right, jump, duck, or perform a special action such as attacking or using an item.
Rewards	The agent receives positive rewards for collecting items, defeating enemies or completing levels. It gets negative rewards for losing health or falling off platforms.
Policy	This is the strategy or set of rules that the agent follows to decide its actions based on the current state.

Learning process

The agent begins with limited knowledge, typically operating under a naive or exploratory policy. It might randomly choose actions when it starts playing. As the agent takes actions within the game, it observes the new states and the immediate rewards or penalties associated with its actions. The agent's goal is to maximize its total score (cumulative reward) for a game or a series of games. This score is influenced by both immediate rewards and long-term benefits of reaching new levels or achieving bonus scores.

Learning and policy update

The agent assesses the results of its actions through the rewards received and updates its strategy accordingly. This is the agent's **feedback loop**. Techniques like Q-learning or Deep Q-Networks (DQN) are often used, where the agent updates a value function estimating the expected future rewards for each action taken in each state.

Exploration vs exploitation

Exploration is especially important in complex games where the agent must experiment with different strategies to understand various game mechanics. The agent might choose suboptimal paths to discover hidden areas or test different enemy handling strategies.

As the agent learns which actions yield the highest rewards, it begins to prefer (exploit) those actions more frequently, leveraging its accumulated knowledge to maximize its score.

Convergence

After sufficient gameplay and learning, the agent develops a robust policy that effectively balances short-term rewards and long-term game progression strategies, leading to consistently higher scores and more successful game completions.

This iterative learning process allows the game-playing agent to continually improve its ability to navigate the game world, effectively handle challenges, and optimize its actions for maximum score, embodying the core principles of reinforcement learning in a dynamic and interactive setting.

An agent might need hundreds of thousands of iterations to win a level in this example. In order to find all the secrets and earn the highest possible score, it may require over a million iterations.

A4.3.7 Describe the application of genetic algorithms in various real-world situations

Genetic algorithms (GAs) are a type of optimization technique. They find optimal solutions to complex problems through mechanisms inspired by natural selection, such as inheritance, mutation, selection and crossover.

In Table 12, key terms for genetic algorithms are defined and illustrated in the context of how a wind farm might be organized.



▲ **Figure 27** A wind farm

Table 12 Genetic algorithms terms, illustrated in the context of organizing a wind farm

Term	Description	Example (Wind farm layout)
Population	A set of potential solutions to the problem. In genetic algorithms, individuals in the population are encoded representations of solutions.	Might consist of different configurations of wind turbines placed within a defined area.
Fitness function	A function that quantifies the optimality of a solution (individual) in the population, guiding the selection process.	Could be calculated as a combination of total energy production minus the costs associated with turbine placement.
Selection	The process of choosing individuals from the population for reproduction based on their fitness. This can be done using methods like roulette wheel or tournament selection.	Selecting configurations with the highest energy output per cost ratio to form a mating pool.
Crossover	A genetic operator used to combine the genetic information of two parents to generate new offspring. This is an analogue of reproduction and biological crossover.	Combining the turbine placements from two high-fitness configurations, ensuring no overlap in turbine locations.
Mutation	A genetic operator that introduces variations by randomly altering the genetic makeup of offspring. This helps maintain genetic diversity within the population.	Randomly changing the position of one or more turbines within a configuration to explore new layout possibilities.
Evaluation	The process of calculating the fitness of each individual in the population. This step assesses how well each solution solves the problem.	Computing the energy production and cost for each configuration to apply the fitness function.
Termination	The criterion to stop the algorithm, which could be a set number of generations, a time limit, or when a satisfactory fitness level is achieved.	Stopping the algorithm after 200 generations or when a configuration exceeds a predetermined efficiency threshold.

Application of genetic algorithms to route planning

A real-world application of GAs is in optimization problems, such as route planning, which includes the travelling salesperson problem (TSP). The TSP poses the challenge of finding the shortest possible route to visit a list of cities and return to the origin city, without re-visiting a city.

Representation (Encoding): Each possible route (solution) is typically represented as a sequence or a permutation of numbers that correspond to the cities' indices.

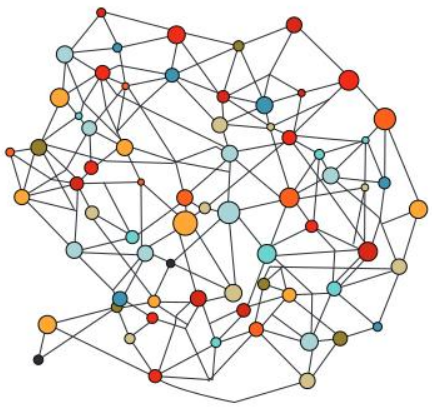
Initialization: Start with a randomly generated population of routes.

Evaluation: Calculate the total distance of the route (the fitness function) for each individual in the population.

Selection: Select individuals to reproduce based on their fitness.Those with shorter routes (lower total distances) have a higher chance of being selected.

Crossover (Recombination): Combine parts of two routes to create a new route. Techniques like ordered crossover ensure that the offspring are valid routes (for example, no city is visited twice).

Mutation: Introduce small random changes to an individual's route, which helps to explore new areas of the solution space and avoid local minima.



▲ **Figure 28** The travelling salesperson problem (TSP) Each node represents a city What is the most efficient path to visit each city?

Replacement: Replace the least fit individuals in the population with new ones.

Termination: Repeat the process until a stopping criterion is met, such as a maximum number of generations or a satisfactory fitness level.

This method is effective for the TSP because it can efficiently search a very large solution space and adapt to changes, making it ideal for dynamic or complex routing problems. Genetic algorithms are used in logistics, scheduling, and network design problems. They are an excellent approach in solving optimization problems.

Overview of genetic algorithm method applied to the TSP

Population initialization

Start with a population of potential routes, where each route is a different sequence of cities. These routes can be generated randomly, and the population size is predefined (for example, 50 different routes).

Fitness function

Each route in the population is evaluated using a fitness function. For the TSP, the fitness function is typically the inverse of the total distance of the route (shorter routes have higher fitness).

For example, if a route covers 500 km, its fitness score might be $\frac{1}{500}$ (higher fitness = shorter route).

Selection

Select a subset of routes from the current population to create new routes for the next generation.

Selection is based on the fitness of each route: routes with shorter distances (higher fitness) have a higher chance of being selected.

Common methods of selection include:

- Roulette wheel selection—routes are selected with a probability proportional to their fitness.
- Tournament selection—several routes are randomly selected, and the one with the shortest distance is chosen.

Crossover (combining routes)

After selection, pairs of routes are combined to create new routes. Crossover takes two existing routes (parents) and mixes their city sequences to form new routes (offspring).

For example, if one parent route visits cities in the order [A, B, C, D, E], and another visits [C, A, E, D, B], a new route could inherit part of the sequence from each parent while maintaining valid city sequences.

Mutation (altering routes)

To maintain diversity and avoid getting stuck with similar routes, a mutation step is applied. In the context of TSP, mutation could mean randomly swapping two cities in a route.

For example, if a route visits cities [A, B, C, D, E], after mutation, it might visit [A, D, C, B, E].

Evaluation

After crossover and mutation, the new set of routes (offspring) is evaluated using the same fitness function as before. Each route's fitness is calculated based on its total distance, and shorter routes are given higher fitness scores.

Termination

The algorithm repeats the steps of selection, crossover, mutation, and evaluation for several generations. The algorithm stops when a termination condition is met, such as:

- reaching a fixed number of generations (for example, 100 generations)
- finding a route with a sufficiently short distance
- no significant improvement in route distance over a number of generations.

Final solution

Once the algorithm ends, the best route in the current population (that is, the route with the shortest total distance) is returned as the solution.

Table 13 gives a detailed breakdown of what each part of the genetic algorithm does.

Table 13 Significant parts of a genetic algorithm for the TSP

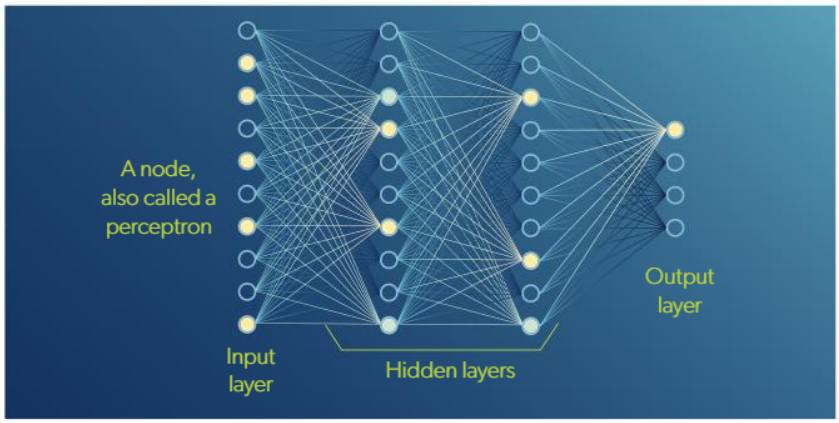
Step	Description
Representation (Encoding)	Generates a random permutation of city indices, representing a single route or solution where each city is visited exactly once. This function ensures that every possible route is a valid solution by considering all cities without repetition.
Initialization	Initializes the population by creating multiple routes. Each route is generated by the `encodeRoutes` function, resulting in a diverse initial population of potential solutions.
Evaluation	Calculates the fitness of a given route by summing the distances between consecutive cities in the route, plus the distance from the last city back to the first to complete the loop. The fitness is defined as the inverse of the total distance, making shorter routes have higher fitness values.
Selection	Selects individuals (routes) from the current population to breed the next generation. Routes are chosen based on their fitness scores, with higher chances given to routes with higher fitness. This is typically implemented using methods like roulette wheel selection or tournament selection.
Crossover (Recombination)	Generates a new route by combining parts of two parent routes. The function ensures that the new route is valid and contains no duplicate cities.
Mutation	Introduces random changes to a route, which helps to maintain genetic diversity within the population and allows the algorithm to explore a wider range of potential solutions. This is critical for preventing the algorithm from getting stuck in local optima.
Replacement	Merges the current population with the newly generated individuals and then selects the best individuals to form the new population. This step is essential for keeping the population size constant and focusing on better solutions over time.
Termination	Manages the overall process of the genetic algorithm, including repeatedly applying the evaluation, selection, crossover, mutation and replacement until a stopping criterion is met (for example, a maximum number of generations or a satisfactory fitness level).
Main function and execution	Sets up the problem context by defining the distance matrix for the cities, specifies the algorithm parameters like population size and number of generations, and runs the genetic algorithm. After completion, it finds and prints the best route found along with its fitness.

A4.3.8 Outline the structure and function of ANNs and how multi-layer networks are used to model complex patterns in data sets

An artificial neural network (ANN) is a computational model designed to mimic the human brain's interconnected neuron structure to process information. It consists of layers of nodes (called perceptrons) which are connected by pathways that transmit signals to other nodes.

Table 14 Components and terminology of a typical ANN

Component/Term	Definition
Activation function	A mathematical function applied to a node's output to determine whether it should be activated, influencing how the signal should be processed further through the network.
Bias	Each node has a bias, which is an additional parameter adjusted along with weights to optimize the neural network's performance. The bias in a node acts as an additional input along with the actual inputs multiplied by their respective weights.
Epoch	A term used to describe one complete pass through the entire training data set. This includes both forward propagation and back propagation.
Hidden layers	One or more layers that process the inputs by performing computations and then transmitting the results to the next layer. These layers are called "hidden" because they do not directly interact with the external environment (neither input nor output).
Input layer	Receives the initial data for processing
Link (or connection)	The actual pathway through which signals are transmitted from one node to another. Each link between two nodes in a network has an associated weight.
Node (also called neuron, or perceptron)	A computational unit that receives inputs, processes them, and produces outputs. Each node can perform simple calculations on its inputs, which are then passed on to another node.
Output layer	The final layer that produces the output of the network
Propagation	The process by which input data moves through the network, from the input layer, through one or more hidden layers, to the output layer. There are different types of propagation.
Weight	The value that influences the strength or importance of a signal being passed between nodes in the network. It modifies how much impact one node's output will have on another node's input. Weights are adjusted during the training process. Do not call a link a weight—they are different.



▶ **Figure 29** Elements of an ANN Each node in these layers processes the input it receives and passes the output to the next layer

Table 15 Components of an ANN to simulate interconnected nodes (neurons) to process and learn from input data, enabling tasks such as classification, regression and pattern recognition

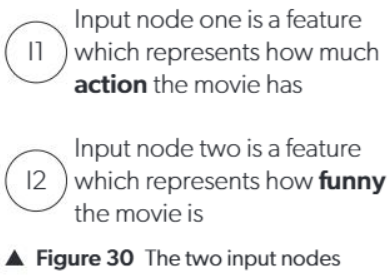
Component	Definition
Forward propagation	Data is inputted through the input layer, and each subsequent layer receives data from the previous layer. Nodes process these inputs using their weights and biases, typically summing the weighted inputs and then adding the bias. This sum is passed through an activation function, which determines the output of each node. Activation functions introduce non-linear properties to the network, allowing it to learn more complex patterns.
Back propagation	The learning phase, where the ANN adjusts its weights and biases. The process involves calculating the error in the output (difference between the actual output and the predicted output) and then going back through the network to adjust the weights to decrease this error. Adjustments are made according to the gradient of the error with respect to each weight, using techniques such as gradient descent.
Training cycle	The cycle of forward propagation, calculating loss, back propagation, and updating weights is repeated across many epochs (an epoch is a full pass through the training data) until the network performs satisfactorily.

Classification example

Imagine you are using a neural network to decide if a movie is good based on two features: how much action there is (this will be labelled input 1, or I1) and how funny it is (this will be labelled input 2, or I2). Note that this is a simplified model—most models have many more features.

Step 1: Draw and label a diagram to show the inputs.

1. Label input nodes with I (upper case i: not L or one): I1, I2, I3... I*n*
2. Label hidden nodes with H: H1, H2, H3... H*n*
   Hidden nodes often have multiple layers: these would be labelled H1N1 (hidden layer 1, node 1), H1N2 (hidden layer 1 node 2), H2N2 (hidden layer 2 node 2)...
3. Label output nodes with O (upper case o: not zero): O1, O2, O3... O*n*



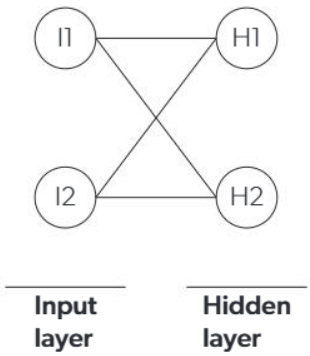
▲ **Figure 30** The two input nodes

Step 2: At the input layer, the model is given inputs (I1 = 3, I2 = 1.5). Suppose I1 = 3 represents a moderate level of action, and I2 = 1.5 represents it being somewhat funny.

The first node (H1) in the hidden layer multiplies the action score (I1) by 0.5 and the humour score (I2) by 0.3. These are the weights for action and humour for this particular node.

The second node (H2) in the hidden layer does something similar but with different numbers (0.4 for action, 0.9 for humour). These are the weights for action and humour for this particular node.

Machine learning models multiply input values (such as the action score and humour score in this example) by weights (such as 0.5 for action and 0.3 for humour) to model the relative importance and influence of each input on the output decision. H1 is designed to favour action over humour. H2 very heavily favours humour over action.



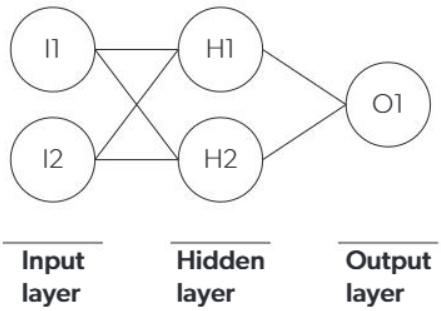
▲ **Figure 31** The hidden nodes process the inputs by performing computations

Step 3: After the hidden nodes have calculated the initial weights, they will calculate the bias. Adding a bias term allows a node to output non-zero values even if the inputs are all zero, which could correspond, for example, to a base level of likability of a movie due to factors not included as inputs (such as marketing, star actors, and so on). The bias can adjust the threshold at which the node activates, thereby incorporating a kind of baseline expectation into the model's predictions.

Step 4: Activation function. After calculating the weighted sum and bias, this sum is then passed through an activation function. The activation function defines how the input signal (the weighted sum of inputs and bias) is transformed before being passed to the next layer. Some common activation functions are given in Table 16.

Table 16 Common activation functions

Activation function method	Description
Sigmoid	Transforms the input into a range between 0 and 1, making it useful for probabilities or binary classifications. It is often used in the output layer of binary classification networks.
ReLU (rectified linear unit)	Outputs the input directly if it is positive, otherwise it outputs zero. This function is popular in hidden layers because it introduces non-linearity while keeping computation simple and reducing the likelihood of vanishing gradients.
Tanh (hyperbolic tangent)	Transforms the input to range between −1 and 1. It is similar to sigmoid but can provide stronger gradients since outputs are zero-centred.
Linear activation function	Does not transform the input it receives; it simply outputs the input as it is.
Step function	Outputs 1 if the weighted sum is greater than a threshold, 0 otherwise.



▲ **Figure 32** The output node produces a final decision

Step 5: Passing to the next layer (in the example, it is to the output layer, or output layer decision). The output layer aggregates inputs from the hidden layers, possibly applies additional transformations, and produces a final decision (for example, thumbs up or thumbs down for the movie). Output layers often have a threshold (often 0.5 for binary classification) to decide the final class based on the sigmoid output. If the output is greater than 0.5, the movie receives a thumbs up; if less, a thumbs down.

By the time the input data reaches the output layer, it has been adjusted in a useful way based on the input scores and the rules each node used. So, the final score should tell us something valuable—such as whether most people would enjoy the movie. The network learns the best multipliers and biases (the ways each student tweaks their scores) during its training phase, where it gets lots of examples of movies and how much people liked them.

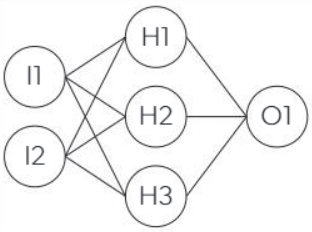
Regression example

A regression task is a type of predictive modelling problem where the objective is to predict a continuous outcome variable based on one or more input variables. The continuous outcome variable means that the prediction can take any value within a range, in contrast to classification tasks where the outcome is categorical (like yes/no).

The goal is to predict the price of a car based on two features: the age of the car (in years) and its mileage (in thousands of kilometres).

Example setup

Table 17 How a regression example is set up

Features and output	Input features (I1 and I2) are the **age** of the car in years (I1) and the **mileage** of the car in thousands of kilometres (I2). The output layer (O1) is the price of the car in USD.	
Neural network model	The input layer receives the two features of the car (age and mileage). The hidden layer contains a certain number of neurons (3 for simplicity) that process the inputs. The output layer produces the predicted price of the car.	
Architecture overview		The input layer has 2 nodes (corresponding to 2 features). The hidden layer has 3 nodes. The output layer has 1 node.
Activation functions	In this example, the hidden layer uses the ReLU activation function for introducing non-linearity, allowing the network to learn more complex patterns. The output layer uses a linear activation function, which is suitable for regression as it allows the output to assume any real value.	

Forward propagation steps

1. Input to hidden layer: Each node in the hidden layer calculates a weighted sum of the inputs plus a bias. The sum is then passed through the ReLU activation function.
2. Hidden layer to output layer: The output of each node in the hidden layer is again multiplied by weights and summed up along with a bias in the output layer's node.
3. The sum passes through a linear activation function to produce the final output.

Example calculation

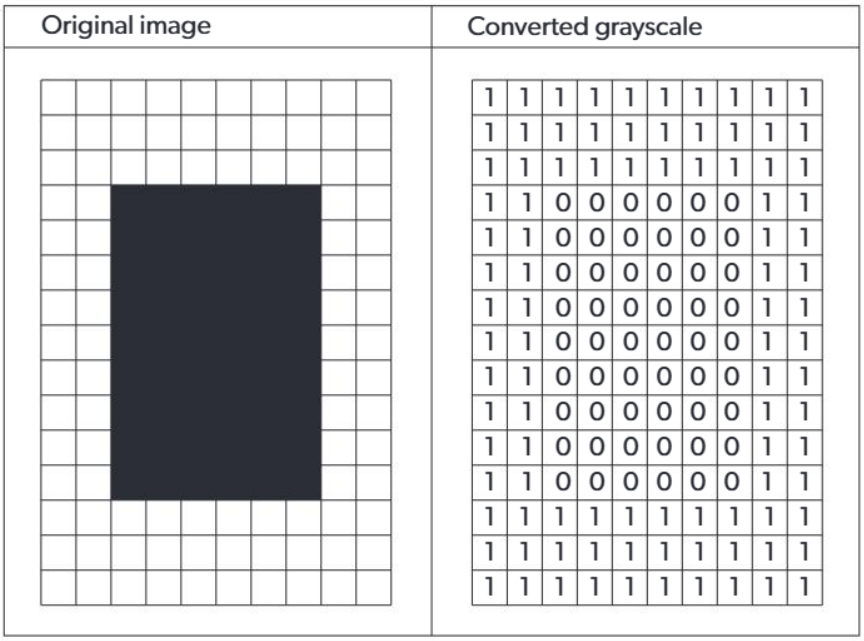
- Assume simple values:
  Inputs I1 (Age) = 5 years and I2 (Mileage) = 50 thousand kilometres.
- Weights and biases: Random initial values.
- Calculations: Inputs are processed through the hidden layer with ReLU.
- The outputs from the hidden layer are combined in the output layer to predict the price.

This neural network example, with just one hidden layer, illustrates the basic principles of ANNs in a regression context, showing how layers, neurons, weights, biases and activation functions come together to process inputs and make predictions. The ability to learn complex patterns from data makes ANNs particularly useful for predicting outcomes like car prices based on their features.

Basic example related to pattern recognition

In this example, the goal is to train a neural network to distinguish between hand-drawn images of circles and squares. This is a binary classification task, a common type of pattern recognition problem in machine learning.

The input features (I) are pixel values of the images. Each image is converted into a grayscale grid of pixels (e.g., 28 × 28 pixels), where each pixel's intensity is represented as a value between 0 (black) and 255 (white). Alternatively, a scale between 0 (black) and 1 (white) can may be used.



▲ **Figure 33** Original image and converted grayscale

The output (O) is class labels for the images, where 0 might represent a circle and 1 might represent a square.

The neural network model

1. The input layer receives the flattened pixel values from the images.
2. The hidden layer contains several nodes (assume 16 for simplicity) that process the inputs.
3. The output layer produces a binary output indicating the class (circle or square).

Architecture overview

The input layer consists of 784 nodes if using a 28 × 28 pixel image, corresponding to each pixel in the image. The hidden layer has 16 nodes. The output layer is simply 1 node, using a sigmoid activation function to output a probability that the input image is a square.

Activation functions

The hidden layer uses the ReLU activation function to output any positive integer, and zero for anything else. The output layer employs a sigmoid activation function to produce a probability between 0 and 1, indicative of class membership (circle vs square).

Training the model

In the last two examples, you saw simple neural networks with forward propagation only. In this example, you will learn about back propagation. Back propagation is where the "learning" happens in a neural network. Back propagation is a method for iteratively adjusting the weights of the network to minimize the difference between the predicted output and the actual output. The process occurs after the network has attempted to make predictions, a stage known as forward propagation, where data moves forward from the input to the output layer.

Table 18 Key steps in back propagation

Calculating error	After the forward pass (forward propagation), the network assesses its accuracy by calculating the loss, which measures the difference between the network's prediction and the actual target values. The loss provides a quantifiable measure of how well the network is performing. A common loss function for classification tasks is binary cross-entropy, which is particularly effective for binary outcomes.
Running back propagation	Back propagation begins by taking the derivative (how much one quantity changes in response to a change in another quantity) of the loss function with respect to each weight in the network, which shows how much a change in each weight affects the overall error. This step informs the network in which direction and how much to adjust the weights to reduce the error. The errors are propagated backward through the network, from the output layer to the input layer, updating the weights along the way. This backward movement gives the technique its name.
Weight update via optimization algorithm	With the errors known, the network updates the weights to minimize the loss. This is typically done using an optimization algorithm like stochastic gradient descent (SGD). SGD updates the weights by slightly adjusting them in the opposite direction of the derivative of the loss function. The size of the adjustment is controlled by a parameter known as the learning rate. This step is repeated iteratively, using small batches of data at a time, which helps improve the robustness and generalization of the network.
Iteration and convergence	The process of forward propagation, loss calculation, back propagation, and weight adjustment is repeated for multiple iterations, often called epochs, over the entire data set. With each epoch, the weights are further refined, ideally leading to a decrease in the loss and an improvement in model predictions.

One epoch: Forward propagation steps

Step 1: Input to hidden layer

Each node in the hidden layer calculates a weighted sum of the pixel inputs plus a bias. The sum is then passed through the ReLU activation function, which outputs the value itself if it is positive, and zero otherwise.

Step 2: Hidden layer to output layer

The outputs from the hidden layer are input to the output layer node. The output node calculates a weighted sum of these inputs plus a bias and then applies the sigmoid function. The sigmoid function transforms the sum into a probability between 0 and 1, indicating the likelihood that the image is a square.

One epoch: Back propagation steps

Step 3: Calculating the error

After the output layer provides the prediction, the next step is to calculate the error of the prediction. The error is determined by comparing the predicted probability (output of the sigmoid function) with the actual label of the image (0 for circle, 1 for square) using the loss function.

A common choice of loss function in binary classification tasks like this is the binary cross-entropy loss, which is effective at measuring the difference between the predicted probabilities and the actual binary outcomes.

Step 4: Back propagation of error

The calculated error is then used to find out how much each weight contributed to the error. This involves calculating the derivatives of the loss function with respect to each weight in the network.

Think of it like this: after the network makes its guesses and sees how wrong or right it was, it then figures out how much each part of its thinking (each weight) led to any mistakes. This step involves looking closely at the errors and tracing them back through the network to see where adjustments are needed.
This process, where we calculate and follow the errors to make corrections, is back propagation. It is like the network is learning from its mistakes to do better next time.

Starting from the output layer, the error is propagated back through the network. This involves computing the gradient of the loss function with respect to each weight by applying the chain rule of calculus. The chain rule lets us work out how each weight in the network needs to change to reduce the error—by understanding how each part of the network's calculation contributes to the final output.

Step 5: Updating the weights and biases

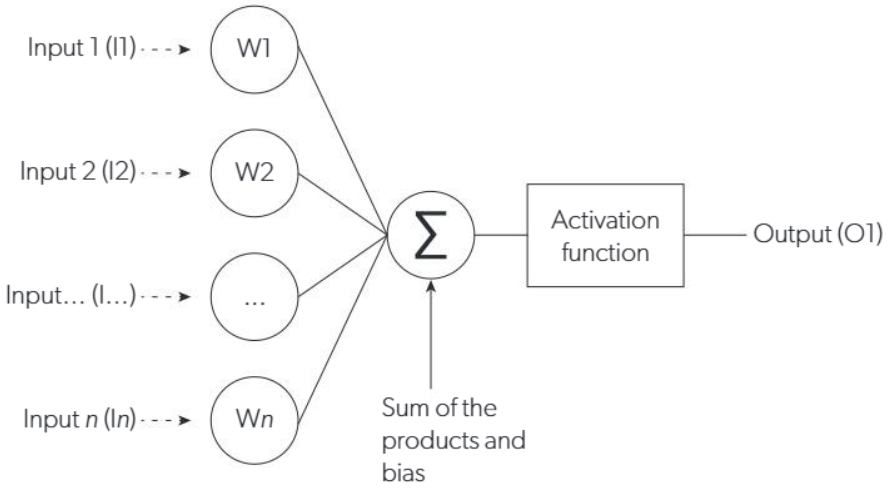
Once the gradients are calculated, the weights and biases are updated to minimize the loss. This update is typically done using an optimization algorithm like stochastic gradient descent.

In stochastic gradient descent, weights are updated by subtracting a fraction of the gradient from the current weights. This fraction is determined by the learning rate, a small number that controls how much the weights are adjusted during each iteration.

The combination of forward propagation to make predictions and back propagation to learn from errors makes up the fundamental training cycle for a neural network in pattern recognition tasks. This process allows the network to adjust its internal parameters (weights and biases) based on the feedback from its performance on known training data, thereby improving its ability to classify new data accurately.

Sketch of a single perceptron

Example 1



▲ **Figure 34** Sketch of a single perceptron

Example 2

Table 19 A perceptron

I1	→ multiplied by →	W1	Sum the products and	Add to the **bias** and →	activation function transforms the data from the bias and →	output
I2	→ multiplied by →	W2				
…	→ multiplied by →	…				
I*n*	→ multiplied by →	W*n*				

- Inputs (I1, I2, … I*n*): These represent the data or features fed into the perceptron. Each input has a corresponding weight within the perceptron.
- Weights (W1, W2, … W*n*): Weights determine the importance of each input feature. They are multiplied with their corresponding inputs.
- Bias (b): The bias is a constant value added to the weighted sum of inputs. It helps shift the decision boundary of the perceptron.
- Weighted sum: The inputs are multiplied by their corresponding weights and summed together with the bias.
- Activation function (f): The activation function defines how the input signal (the weighted sum of inputs and bias) is transformed before being passed to the next layer. Common examples include:
  - Step function: Outputs 1 if the weighted sum is greater than a threshold, 0 otherwise.
  - Sigmoid function: Maps the weighted sum between 0 and 1
  - ReLu: Outputs the weighted sum if it's positive, 0 otherwise.
  - Tanh: Transforms the input to range between –1 and 1. It is similar to sigmoid but can provide stronger gradients since outputs are zero-centred.

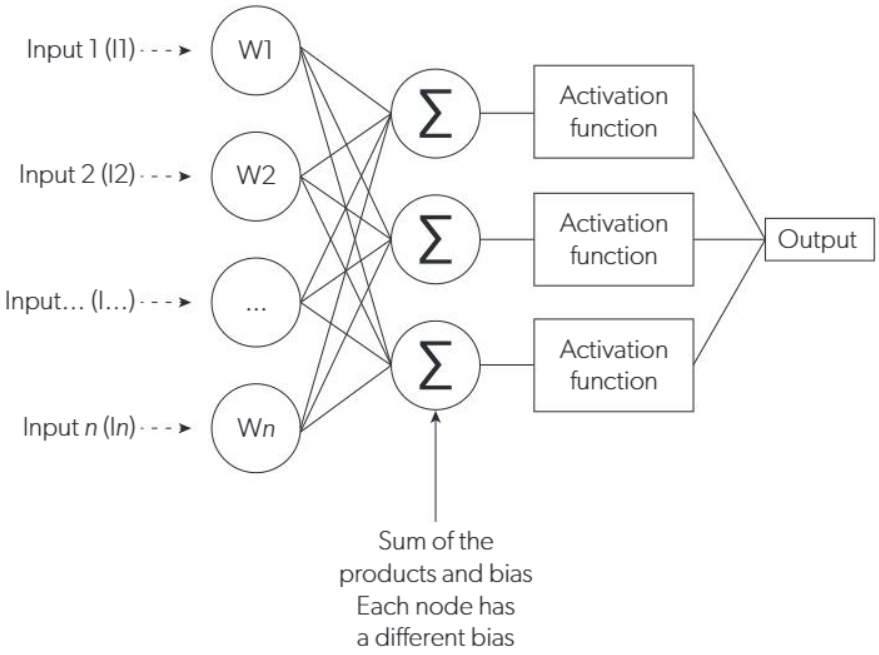
- Linear activation function: Does not transform the input it receives; it simply outputs the input as it is.

- Output (O): The final result produced by the perceptron. This output can be a classification (e.g., 0 or 1) or a continuous value depending on the activation function.

Points to remember

- A single perceptron is the most basic building block of neural networks.
- It can classify only linearly separable data. This means that you can imagine drawing a straight line to separate the different classes of data points.
- Perceptrons are the foundation for more complex neural network architectures that can learn non-linear patterns.

Sketch of a multi-layer perceptron (MLP)



▲ **Figure 35** Sketch of a multi-layer perceptron

A4.3.9 Describe how CNNs are designed to adaptively learn spatial hierarchies of features in images

A convolutional neural network (CNN) is a type of deep learning algorithm optimized for finding patterns in images and other types of data with grid-like structures (satellite imagery, audio spectrograms, and certain types of genomic data).

CNN basic architecture

Table 20 Overview of a CNN process

Input layer	Takes the raw pixel data of the image. Each pixel is a feature.
Convolutional layers	Utilize multiple filters to process the input data. A filter, also known as a kernel, is a small matrix used to detect specific features by performing convolution operations over the input. By applying each filter, the layer generates feature maps, which are outputs that highlight where and how intensely those specific features are detected within the input.
Activation functions	Introduced in each convolutional layer to introduce non-linear properties to the system, enhancing the network's ability to learn complex patterns. Commonly used functions include ReLU.
Pooling layers	Reduce the spatial size of the representation, which decreases the parameter counts and computation in the network, effectively summarizing the features detected in prior layers.
Fully connected layers	After several convolutional and pooling layers, the high-level reasoning in the neural network occurs here. Each neuron in these layers is connected to all activations in the previous layer.
Output layer	Typically uses a softmax activation function in classification tasks to output probabilities for the different class labels.

Table 21 CNN terms and concepts

Layer type	Definition
Activation functions	Functions applied after each convolution operation to introduce non-linearity into the model, enabling it to learn more complex patterns. Commonly used functions include ReLU.
Convolution	A mathematical operation used to extract features from input data. It involves taking a kernel (filter) and sliding it over the input data (like an image) to produce a feature map.
Convolutional layers	Layers that apply a set of learnable filters to the input. Each filter detects specific features at specific spatial locations.
Fully connected layers	Layers where every neuron is connected to every neuron in the previous layer. These layers are typically placed near the end of CNN architectures to perform high-level reasoning from the features extracted by convolutional and pooling layers.
Input layer	The initial data layer that receives the input features, typically in the form of an image or a multi-dimensional data array.
Kernel	A small matrix—also known as a filter—used to process data through the operation known as convolution.
Output layer	The final layer that outputs the prediction of the network. The output format is determined by the specific task (for example, classification, regression).
Pooling layers	Reduce the spatial dimensions (width and height) of the input volume for the next convolution layer. It helps reduce the computational complexity, and controls overfitting. Common approaches include max pooling and average pooling.

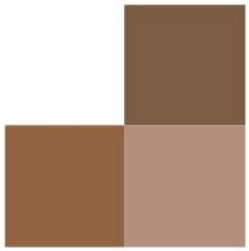
Example of a CNN

Step 1: Input

The input layer of a CNN takes an image—in this case, an image of a cat—as input. This image is typically represented as a matrix of pixel values. For coloured images, there are three such matrices corresponding to the RGB (red, green, blue) colour channels. Each pixel in these matrices holds a value ranging from 0 to 255, indicating the intensity of the colour at that specific pixel.



▲ **Figure 36** A pixelated cartoon cat



▲ **Figure 37** This might be a filter for the top-left ear. Of course, these would be pixel values instead of colours

Step 2: Feature detection and feature map

In the convolutional layers, multiple filters—each a small grid of numerical weights—move across the entire image of a cat. Each filter is engineered to identify distinct features by performing a convolution operation. The feature map is a new image that highlights the specific features detected by the filter. For example, one filter might focus on detecting the edges of the cat's ears by recognizing patterns of sharp contrast between light and dark pixels, resulting in a feature map that emphasizes these edges. Another filter might be tuned to capture the texture of the cat's fur by identifying repeated patterns of fine lines and shading, producing a feature map that reveals the detailed texture patterns. These feature maps collectively represent various aspects of the original image, each highlighting different elements important for further analysis and classification.

Step 3: Activation

After the feature maps are created by the convolutional layers, an activation function is applied to each one. One common activation function used is ReLU.

The ReLU function takes each value in the feature map and changes any negative values to zero while keeping all positive values the same. This step highlights the important features in the image by removing less useful information (anything less than zero). By doing this, ReLU allows the network to focus more on the significant features, enhancing its ability to analyse and learn from the data.

Step 4: Pooling

The pooling layer follows the activation function and serves to reduce the spatial dimensions (height and width, not depth) of each feature map. This reduction decreases the computational load and enhances the detection of features irrespective of their position in the input image. Commonly, max pooling is used, which reduces the size of the feature maps by taking the maximum value from a set of neurons in a window (for example, a 2×2 block) and outputting only that max value. This emphasizes the most prominent features, reducing the response to variations and noise.

Step 5: Integration

After several layers of convolution and pooling, the network uses fully connected layers to integrate all the features extracted by previous layers. This layer views the output of the previous layers as a single vector (flattening the feature maps into a one-dimensional vector), allowing it to learn non-linear combinations of the high-level features. These layers are called fully connected because every neuron in one layer is connected to every neuron in the next layer. The final fully connected layer collects all the essential information to make a final decision about the image.

Step 6: Classification

The last layer in the network is the output layer, which typically uses a softmax activation function in a classification task. This function converts the outputs of the network into probability values corresponding to each class (cat or dog).

The softmax function exponentiates (applies the exponential function to) each output and then divides each by the sum of all exponentiated outputs. This ensures that the output values are between 0 and 1 and sum to 1, representing them as probabilities.

Factors that affect how CNNs process input data and classify images

Number of layers

A network with fewer layers might only learn basic features like edges and simple textures. For instance, in our example, a shallow network might distinguish basic shapes but struggle with differentiating features between cat and dog faces.

On the other hand, a network with more layers can learn complex features at various levels of abstraction, such as fur texture, ear shape and facial expressions. A deeper network in the cat–dog classifier would better differentiate the subtle features that distinguish the two animals.

Kernel size and stride

Larger kernels cover a larger area of the input image, thus capturing more global information at once, such as an entire ear or eye. Smaller kernels focus on local features, such as fur texture or colour gradients. A large kernel of size 7×7 might help identify the outline of a dog's floppy ear, while a smaller kernel of 3×3 might focus on the detailed textures within the ear.

The stride determines the amount of overlap between the fields that the kernel processes. A stride of 1 means the kernel moves pixel-by-pixel across the image, leading to a very detailed feature map. A larger stride, like 2, skips pixels, making the feature map coarser but reducing computational load and memory usage.

Activation function selection

Activation functions introduce non-linearity to the network, enabling it to learn complex patterns.

ReLU is commonly used for its efficiency. It helps the network train faster and reduces the likelihood of the vanishing gradient problem. ReLU activates a neuron only if the input is positive, hence it is more biologically plausible than other activations. In a cat–dog classifier, ReLU could help the network learn to activate strongly on features specific to either category, ignoring negative values that represent non-useful information.

Loss function

The loss function guides the training process by quantifying the error between the predicted outputs and the actual labels.

Cross-entropy loss is popular in classification tasks because it measures the difference between two probability distributions—predicted probabilities versus actual distribution (ground truth). If the network incorrectly classifies a cat as a dog with high confidence, the cross-entropy loss would be high, signalling the network to make significant adjustments in its parameters.

A4.3.10 Explain the importance of model selection and comparison in machine learning

Model selection and comparison significantly influence the accuracy and efficiency of the predictions or insights derived from the data.

Table 22 Different machine learning algorithms

Algorithm	Description
Linear regression	Predicts a continuous output based on the linear relationship between input variables. It is simple and highly interpretable but assumes a linear relationship between inputs and outputs. Typical use includes economic forecasting and trend analysis.
Logistic regression	Used for binary classification problems. It predicts probabilities of class memberships based on a logistic function. Logistic regression might be used in email spam detection and disease diagnosis.
Decision trees	A tree-like model that makes decisions based on splitting rules from features. Easy to understand and interpret. Normally used in customer segmentation and credit risk assessment.
Random forest	An ensemble of decision trees that improves prediction accuracy by reducing overfitting through averaging multiple trees. Random forests are used for feature selection, classification and regression tasks in various fields.
Support vector machines (SVM)	Finds the hyperplane that best separates different classes in the feature space. Good for complex classification problems with clear margin of separation. Typically used in image classification, bioinformatics.
K-Nearest Neighbours (K-NN)	Classifies new cases based on a similarity measure (for example, distance functions) with known cases. K-NNs are used for recommendation systems and pattern recognition.
Neural networks	Composed of layers of interconnected nodes (neurons), capable of capturing non-linear relationships through activation functions. Used in speech recognition, image recognition.
Deep learning models	Involves more complex neural networks with multiple hidden layers that allow for high levels of abstraction and feature extraction. Deep learning models find application in autonomous driving and natural language processing.

How different algorithms can yield different results depending on the data and type of problem

The reason you might get different results from different machine learning algorithms is because each machine learning algorithm has varied underlying mechanics. It is important to use the best machine learning for the type of problem you might have.

Linear models, like linear regression and logistic regression, work well with data that have a linear relationship between the features and the target. Tree-based models such as decision trees and random forests are effective for handling non-linear data with complex patterns but can be prone to overfitting.
Neural networks offer high flexibility and capacity to model extremely complex relationships in large data sets but require substantial computational resources and data to train effectively.

The key point to remember is that each algorithm makes different assumptions about the data (like the distribution of the features or the relationship between features and labels), which can greatly affect their performance.

The reasons for selecting specific machine learning models over others

There is no perfectly clear path to choosing the right machine learning model. Think carefully about the following factors, and then make the best choice.

Table 23 Factors to consider when thinking about machine learning algorithms

Factor	Why to consider it
Nature of the problem	Classification problems might benefit more from different algorithms (for example, support vector machines, neural networks) than regression problems (for example, linear regression, regression trees).
Complexity of the model	Simpler models are faster to train and easier to interpret but might not capture complex patterns as effectively as more sophisticated models.
Data characteristics	The quantity, quality, and type of data available can dictate the choice of model. For instance, deep learning models generally require large amounts of data, whereas smaller data sets might call for models with fewer parameters, such as K-NN or linear regression.
Performance metrics	Different models may optimize different performance metrics. A model that maximizes accuracy might not minimize false positives.
Computational resources	Some models, especially those that are data-intensive and computation-heavy, require more powerful hardware, which can be a limiting factor.

The variability in algorithm performance based on the data's characteristics

The performance of a machine learning algorithm can vary widely depending on the characteristics of the data. Some models handle imbalanced data sets poorly, where the number of instances in different classes is disproportionate. The presence of outliers, noise, or non-normal distributions can affect model accuracy. Some models require normalization or standardization of features to perform well. Finally, algorithms vary in how they handle interdependent or correlated features. Some, like tree-based models, might inherently manage these well, while others might need modifications.

Model selection and comparison in machine learning are not just about finding the best tool for the job but also about understanding the intricacies of the data and the specific requirements of the problem at hand. This tailored approach ensures that the chosen model will not only achieve the highest performance but also align with the operational needs and constraints of the application.

TOK

- To what extent does the choice of machine learning algorithm influence the interpretation of data and outcomes in predictive modelling?
- In what ways do the assumptions made by different machine learning models affect their suitability for solving specific types of problems?
- How does the process of model selection and comparison in machine learning reflect broader principles of scientific inquiry and decision-making?

Practice questions

11. Explain why different machine learning algorithms may yield varying results when applied to the same data set. [4 marks]
12. Explain how the nature of the problem (classification or regression) influences the selection of a machine learning model. [4 marks]
13. Explain the role of data characteristics, such as the quantity and quality of the data, in determining the suitability of different machine learning models. [4 marks]
14. Explain how computational resources can impact the selection of machine learning algorithms, particularly when considering models like deep learning. [4 marks]
15. Explain how the variability in data characteristics, such as the presence of outliers or imbalanced data sets, can affect the performance of different machine learning algorithms. [4 marks]
16. Explain how classification techniques in supervised learning are used to predict discrete categorical outcomes. [4 marks]
17. Explain the process by which the K-NN algorithm classifies new data points using a similarity measure. [4 marks]
18. Explain how decision trees classify data points and why they may be preferred for certain types of classification problems. [4 marks]
19. Explain why K-NN is considered a non-parametric, instance-based learning algorithm. [4 marks]
20. Explain how decision trees can be applied to medical diagnosis. [2 marks]
21. Explain how K-NN algorithms can be applied to collaborative filtering recommendation systems. [2 marks]
22. Explain the relationship between the independent (predictor) and dependent (response) variables in linear regression. [3 marks]
23. Explain how linear regression can be applied to salary prediction. [2 marks]

A4.4 Ethical considerations

Syllabus understandings

A4.4.1 Discuss the ethical implications of machine learning in real-world scenarios

A4.4.2 Discuss ethical aspects of the increasing integration of computer technologies into daily life

Ethical considerations guide decision-making and actions to ensure they align with moral standards and societal expectations. Before you make a decision or take an action, you will consider the potential harms and benefits of the decision or action. Whenever thinking about ethics, you should balance the potential benefits against the potential drawbacks. Ethics is often about making a choice which serves the greater good.

A4.4.1 Discuss the ethical implications of machine learning in real-world scenarios

An **ethical implication** is a potential positive or negative consequence that a decision or action may have. These consequences can be far-reaching and affect various aspects of life, including well-being, justice, fairness, rights and freedom.

Types of ethical issue

Accountability is about determining who is responsible when AI systems make mistakes. Who should be held accountable: the developers, the users, or the AI itself? If there is a car accident, who should be held responsible? The car manufacturer, or the driver? The city who made the roads? The key point is that clear lines of accountability are essential—especially when systems are making decisions which impact human lives. Your teacher might use an AI to determine your final mark in computer science. Who should be accountable for that mark?

Algorithmic fairness raises the topic of how AI algorithms can perpetuate biases. This can lead to discriminatory outcomes in areas such as hiring, lending, and criminal justice. Ensuring fairness requires careful design, diverse training data and ongoing monitoring. For example, if your school uses training data for maths scores and girls traditionally scored higher than boys, the AI algorithm might favour girls over boys, despite an individual's maths ability.

Continuing the theme of **bias**, an AI system can inherit biases from the data they are trained on. This can lead to discriminatory outcomes and reinforce societal inequalities. If a specific group is unfairly treated now, how might that be carried into training data and an AI system? Mitigating bias requires awareness, diverse teams and rigorous testing.

The use of AI often involves collecting and analysing personal data. Obtaining informed **consent** from individuals about how their data will be used is important to protect privacy and autonomy. This is especially important in areas such as finance, health and education. Most of the time, training data should be thoughtfully sanitized and anonymized prior to being used in an AI system.

The **environmental impact** of training AI systems can be substantial. The development and deployment of AI systems consume significant energy and resources, contributing to environmental concerns. Ethical considerations include minimizing energy consumption and using sustainable practices. Some estimates project AI training could use over 85 terawatts per hour.

Privacy is a fundamental right. The right to privacy or private life is declared in the Universal Declaration of Human Rights (Article 12), the European Convention of Human Rights (Article 8), and the European Charter of Fundamental Rights (Article 7).

Protecting individual **privacy** is important because people have this fundamental right. Some technology companies give away a product for free and use and sell your private data. As you design AI systems, privacy rights should be considered.

AI systems can be vulnerable to cyberattacks, manipulation and misuse. Ensuring their **security** is critical to prevent harm and maintain public trust.

AI can have a profound **societal impact**. AI has the potential to disrupt industries, displace jobs, and alter social interactions. Ethical considerations involve mitigating negative impacts, ensuring equitable access to benefits, and preparing for workforce transitions.

Understanding how AI systems make decisions can be challenging due to their complexity. **Transparency** about algorithms, data sources and decision-making processes is essential for accountability and building trust.

These ethical issues are interconnected and require careful consideration and ongoing dialogue between stakeholders, including developers, policymakers, researchers and the public. The overriding goal is to ensure that AI serves humanity responsibly and equitably.

TOK

- To what extent should accountability be prioritized over innovation in the development and deployment of AI technologies?
- How can we reconcile the need for privacy with the benefits of data collection in AI systems?
- In what ways do biases in training data reflect broader societal inequalities, and how can addressing these biases in AI development contribute to social justice?

The challenges posed by biases in training data

Bias refers to the inclination or prejudice for or against one person or group, especially in a way considered to be unfair. Biases in training data pose significant challenges to the development and deployment of fair and equitable AI systems. These biases can stem from various sources, including societal prejudices, historical inequalities and data collection methodologies.

Challenges

Biased training data can lead AI systems to replicate and even amplify existing societal biases. For instance, if historical hiring data reflects gender or racial discrimination, an AI-powered recruitment tool trained on this data might unfairly favour certain groups over others.

Biased data can result in discriminatory outcomes in areas such as criminal justice, lending and healthcare. For example, banks might use an AI system to decide if a loan is a good risk. If the bank previously discriminated against people who lived in a certain neighbourhood, then that discrimination would be in their AI system's training data.

AI systems can inadvertently learn and perpetuate harmful stereotypes present in training data. A language model trained on biased text might generate responses that reinforce gender or racial stereotypes, perpetuating harmful narratives.

Training data that lacks diversity can result in AI systems that perform poorly for underrepresented groups. A medical diagnosis tool trained primarily on data from one demographic group may be less accurate for individuals from other groups, leading to potential misdiagnosis and health disparities.

Biases in training data can be subtle and difficult to detect. Even seemingly neutral data can contain implicit biases that can impact AI system behaviour. For example, a language model trained on news articles might inadvertently reflect the biases present in media coverage, perhaps favouring a liberal perspective over a conservative one.

Examples

A global ecommerce company had to stop using an AI recruitment tool when it was found to discriminate against female job applicants due to biases in the historical hiring data used for training.

The COMPAS algorithm used in the US criminal justice system was shown to have a higher false positive rate for Black defendants compared with White defendants, unfairly labelling them as higher risk for reoffending.

A group of banks using a credit scoring algorithm which relied on historical financial data unintentionally discriminated against certain demographic groups, such as racial minorities or people from lower socioeconomic backgrounds. This led to unequal access to financial services, including loans and credit, further exacerbating economic disparities.

The ethical concerns of using machine learning in online communication

Misinformation and disinformation

Misinformation is false or inaccurate information that is spread unintentionally. It may be due to mistakes, misunderstandings, or lack of knowledge. There is no deliberate intention to deceive or mislead others.

Disinformation is false information that is deliberately created and spread to deceive or mislead others. It is often used to manipulate public opinion, influence political discourse, or damage reputations. There is a malicious intent to deceive or harm.

Machine learning algorithms are increasingly used to generate and disseminate content, including text, images and video. However, they can also be exploited to create and spread false or misleading information, known as misinformation or disinformation. This can have serious consequences, such as manipulating public opinion, influencing elections, or inciting violence.

For example, deep fakes are synthetic media generated using machine learning algorithms that can make it appear as if someone said or did something they did not. Deep fakes have been used to spread false information about political figures and celebrities, potentially damaging their reputations and causing social unrest. In some cases, criminals have used deepfakes to steal money.

Bias and discrimination

Machine learning algorithms can perpetuate and amplify existing biases in data and society. This can lead to discriminatory outcomes in online communication, such as biased content recommendations, targeted advertising, or unfair moderation practices.

Online harassment and hate speech

Machine learning can be used to detect and filter harmful content such as hate speech and online harassment. However, these algorithms can also be misused to automate harassment campaigns or silence dissenting opinions. For example, autonomous agents (bots) powered by machine learning can be programmed to send abusive messages or spread hateful content, amplifying the impact of online harassment and creating hostile online environments.

Anonymity and lack of accountability

The anonymity afforded by online platforms can be a double-edged sword. While it allows for free expression and protects vulnerable individuals, it can also facilitate harmful behaviour such as cyberbullying, trolling and the spread of misinformation. Machine learning can be used to identify and track anonymous users, but this raises concerns about privacy and surveillance.

For example, some platforms use machine learning to identify and ban users who engage in harmful behaviour, but this can also lead to the wrongful suspension of accounts and the silencing of legitimate voices.

Privacy concerns

Machine learning algorithms rely on vast amounts of data to function, including personal information collected from users' online activity. This raises concerns about data privacy, surveillance, and the potential for misuse of personal information. For example, social media platforms use machine learning to analyse user data for targeted advertising, but this can lead to invasive tracking of online behaviour and the exploitation of personal information for commercial gain.

Addressing these ethical concerns requires a multi-faceted approach involving collaboration between tech companies, policymakers, researchers and civil society organizations.

A4.4.2 Discuss ethical aspects of the increasing integration of computer technologies into daily life

Emerging technologies often present entirely new ethical dilemmas that existing guidelines may not adequately address. For example, the rise of AI raises questions about accountability, bias and the impact on employment which require fresh ethical frameworks.

The importance of continually reassessing ethical guidelines as technology advances

The rapid advancement of technology necessitates the continual reassessment of ethical guidelines for several compelling reasons.

Ethical norms and societal values are not static but evolve over time. Guidelines must be adaptable to reflect changing societal expectations and ensure technology aligns with contemporary moral standards. For instance, the growing emphasis on privacy and data protection has prompted the development of stricter ethical guidelines for data handling practices, such as GDPR in the European Union.

Even with careful foresight, technological advancements can have unintended and unforeseen consequences. Regular reassessment of ethical guidelines allows for course correction when negative impacts become apparent. For example, the widespread adoption of social media platforms revealed issues related to cyberbullying and misinformation, necessitating the development of new ethical guidelines for online conduct.

Ethical guidelines that are outdated or fail to address contemporary concerns can erode public trust in technology and its developers. By actively reassessing and updating guidelines, organizations demonstrate their commitment to responsible innovation and build trust with stakeholders.

Continual reassessment ensures a dynamic balance between fostering technological innovation and upholding ethical principles. It allows for the identification of potential risks and the development of safeguards before harm occurs, ensuring that technology serves humanity responsibly.

Technological advancements often have global implications, requiring ethical guidelines that consider diverse cultural contexts and international norms. Regular reassessment ensures that guidelines remain relevant and effective in addressing global challenges, such as climate change and inequality.

The potential implications of emerging technologies on society, individual rights, privacy and equity

Emerging technologies such as quantum computing, augmented reality (AR), virtual reality (VR), and pervasive AI have the potential to revolutionize various aspects of society, but they also pose significant implications for individual rights, privacy and equity.



▲ **Figure 38** Augmented reality concept

Quantum computing

Quantum computing is a type of computing that uses quantum-mechanical phenomena, such as superposition and entanglement, to perform operations on data. Unlike classical computers, which use bits as the smallest unit of data (0 or 1), quantum computers use quantum bits, or qubits, which can represent and store information in both 0 and 1 simultaneously. This allows quantum computers to process vast amounts of data at unprecedented speeds, making them powerful tools for specific applications that are computationally intensive, such as cryptography and complex modelling.

The increase of computational power through the use of quantum computing has potential implications for drug discovery, materials science and financial modelling. Quantum computing could break current encryption standards, compromising sensitive data and communication. Economically, quantum computing could lead to job displacement in industries reliant on cryptography. Access to quantum computing resources could be concentrated among wealthy nations and corporations, exacerbating existing inequalities.

Augmented reality (AR)

Augmented reality (AR) is a technology that overlays digital information—such as images, text or sounds—onto the real world, enhancing the user's perception of reality. AR is experienced through devices such as smartphones, tablets and AR glasses, which use cameras and sensors to superimpose virtual elements in real-time. This technology is widely used in various applications, including gaming, education and professional training, to provide interactive and immersive experiences that integrate virtual content with the physical environment.

The potential of augmented reality is to enhance real-world experiences with digital overlays, transforming education, healthcare and entertainment. Possible implications include concerns about constant data collection and surveillance through AR devices, distraction and accidents in real-world environments, and being used to manipulate perception and spread false information.

Virtual reality (VR)



▲ **Figure 39** A virtual reality (VR) headset

Virtual reality (VR) is a technology that creates a completely immersive digital environment that replaces the user's real-world surroundings. This environment is experienced through a VR headset or head-mounted display, which isolates the user from the external world and presents highly interactive, three-dimensional virtual scenarios. VR is used extensively in fields such as gaming, training simulations, education and therapy, providing an intense and engaging way to simulate realistic scenarios or fantastic experiences.

The potential for VR is immersive experiences for entertainment, training, therapy and social interaction. Implications include the risk of users becoming overly immersed in virtual worlds, leading to social isolation. VR has been known to trigger anxiety, depression or post-traumatic stress disorder in some individuals. There are also concerns about the creation of misleading or harmful virtual environments.

Pervasive AI



▲ **Figure 40** Seamlessly embedded AI

Pervasive AI refers to the integration of artificial intelligence across various platforms, devices and environments, making it an intrinsic part of daily life and business operations. This form of AI is embedded seamlessly into background technologies, enabling smart automation and data-driven decision-making without explicit user interaction. This type of AI is often multi-modal, meaning you can use your voice, text, or a camera to interact with the AI agent. It enhances user experiences, optimizes processes, and improves efficiency in applications ranging from smart homes and healthcare to industrial systems and urban planning.

Potential benefits include integration of AI into everyday objects and systems, improving efficiency, convenience and personalization. Implications include job displacement, where automation of tasks could lead to significant job losses and economic disruption. AI algorithms could perpetuate or amplify existing biases in data, leading to unfair outcomes or discrimination. Pervasive AI could lead to increased surveillance and erosion of privacy.

Overall implications

Societal impact includes how these technologies could fundamentally alter the way people live, work and interact, with both positive and negative consequences. Balancing the benefits of these technologies with the protection of individual rights, such as privacy and autonomy, will be crucial. Ensuring equitable access to and distribution of the benefits of these technologies will be a major challenge.

Mitigating the risks

Mitigating risk refers to the process of identifying potential risks in a given scenario and implementing strategies to reduce their likelihood and impact. For example, when you drive a car, you drive slowly and carefully. This is a method of mitigating (reducing) the risk of an accident.

- Proactive regulation is important for developing ethical guidelines and regulations to govern the use and development of these technologies.
- Encouraging (and mandating) transparency and explainability will help ensure that AI algorithms mitigate bias and build trust.
- Educating the public about the potential risks and benefits of these technologies is important for informed decision-making.



Linking questions

1. How can machine learning be applied to optimize the management of network traffic (A2)?
2. How does database programming in SQL differ from programming computationally in a high-level language (A3, B2)?
3. To what extent are developments in machine learning ethical (TOK)?
4. How can larger models be processed using GPUs and cloud processing (A1)?
5. Can machine learning find and improve network security problems (A2)?

End-of-topic questions

Topic review

1. Using your knowledge from this topic, A4, answer the guiding question as fully as possible:
   What principles and approaches should be considered to ensure machine learning models produce accurate results ethically? [6 marks]

Exam-style questions

2. Outline the characteristics of each type of machine learning.
   a. Deep learning [2 marks]
   b. Reinforcement learning (RL) [2 marks]
   c. Supervised learning (SL) [2 marks]
   d. Transfer learning (TL) [2 marks]
   e. Unsupervised learning (UL) [2 marks]
3. Outline one real-world application of each type of machine learning.
   a. Deep learning [2 marks]
   b. Reinforcement learning (RL) [2 marks]
   c. Supervised learning (SL) [2 marks]
   d. Transfer learning (TL) [2 marks]
   e. Unsupervised learning (UL) [2 marks]
4. a. Describe the principle of reinforcement learning. [4 marks]
   b. Describe the application of reinforcement learning in robotics navigation. [3 marks]
5. Describe the hardware requirements for of machine learning during:
   a. development and testing [2 marks]
   b. large-scale deployment [2 marks]
   c. edge computing. [3 marks]
6. Describe the use in medical imaging diagnostics of:
   a. supervised learning [3 marks]
   b. deep learning [3 marks]
   c. transfer learning. [3 marks]

AHL

7. Describe the significance of data cleaning in the machine learning workflow. [4 marks]
8. Describe the techniques in the preprocessing stage for:
   a. handling outliers [3 marks]
   b. removing duplicate data [3 marks]
   c. dealing with missing data. [3 marks]
9. Describe the importance of normalization and standardization in data preprocessing. [4 marks]
10. a. Describe the role of feature selection. [2 marks]
    b. Outline the three different feature selection strategies. [6 marks]
11. a. Define dimensionality reduction. [1 mark]
    b. Describe the importance of dimensionality reduction in machine learning. [4 marks]
12. Explain how linear regression is used to predict continuous outcomes. [5 marks]
13. Describe the significance of the slope and intercept in the regression equation. [4 marks]
14. Discuss how the fit of a regression model is assessed using measures like r^2. [6 marks]
15. Compare linear regression to another machine learning model used for predicting continuous outcomes. [5 marks]
16. Evaluate the use of linear regression in predicting house prices based on floor area. [5 marks]
17. Discuss the ethical implications of machine learning in real-world scenarios. [6 marks]
18. Explain the challenges posed by biases in training data. [5 marks]
19. Describe the ethical concerns of using machine learning in online communication. [4 marks]
20. Discuss the importance of continually reassessing ethical guidelines as technology advances. [5 marks]
21. Evaluate the potential implications of emerging technologies such as quantum computing, augmented reality, virtual reality and pervasive AI on society. [6 marks]
22. Discuss the ethical considerations that must be addressed when implementing machine learning models in education, based on the case of Jefferson High (page 246). [4 marks]



B1

Computational thinking

How can we apply a computational solution to a real-world problem?

Computer scientists frame problems by applying computational thinking. Computational thinking encompasses a set of problem-solving skills including abstraction, algorithmic thinking, decomposition and pattern recognition. This way of thinking is very effective for clearly understanding and solving those problems.

Computer science requires careful thinking, careful listening, and careful understanding. Once you understand your problem, you can start to solve it.

B1.1 Approaches to computational thinking

Syllabus understandings

B1.1.1 Construct a problem specification

B1.1.2 Describe the fundamental concepts of computational thinking

B1.1.3 Explain how applying computational thinking to fundamental concepts is used to approach and solve problems in computer science

B1.1.4 Trace flowcharts for a range of programming algorithms

Computational thinking is a framework for thinking about problems and questions in various disciplines. It is a way of thinking about and examining problems where solutions can be represented computationally. Computational thinking helps you analyse how systems with multiple components work together. At the heart of computational thinking are four key ideas.

- Abstraction
- Algorithmic design
- Decomposition
- Pattern recognition

B1.1.1 Construct a problem specification

A problem specification consists of the following components.

1. The problem statement
2. Constraints and limitations
3. Objectives and goals
4. Input specifications
5. Output specifications
6. Evaluation criteria

The problem statement

A problem statement comprehensively and clearly defines a problem to be solved. Whenever a computer scientist encounters a problem, they ensure they understand every single word of the problem before thinking about how to approach the solution.

Problem statements should be concise and specific, and avoid ambiguity. It is helpful when problem statements focus on the "what" and "why" of the problem.

Problems statements often come from a client. However, as is the case with the internal assessment (IA), you can find any problem you want to solve or an area of computing you want to explore. No matter where a problem originates, the problem specification must be clear and concise.



▲ **Figure 1** Customer waiting time is specific and measurable

A poor example of a problem statement
"Our customer service processes are inefficient." This statement is far too broad and does not pinpoint the specific problem. The term "inefficient" could mean many things. For example, long wait times, lack of knowledge among staff, or poor issue resolution. The problem statement also lacks a clear "why". Why is inefficiency a significant problem? Does it lead to customer churn, lost revenue, or damage to the company's reputation?
A better example of a problem statement
"Customers wait an average of 25 minutes on hold before reaching a customer service representative, leading to a 15% increase in abandoned calls over the past quarter." This is a good example because it is focused, measurable, and highlights impact. It clearly identifies the problem as extended hold times, provides quantifiable metrics (25-minute wait, 15% increase) and links the problem to a negative consequence for the business (abandoned calls).

Another good example of a problem statement might be, "The limited address space of IPv4 is hindering the expansion of connected devices within an organization, leading to network scalability issues and potential security vulnerabilities as workarounds are used."

Constraints and limitations

A constraint is a restriction or boundary that impacts the solution. This could include resource limitations (such as time, budget or materials), technical constraints (such as software or hardware), or external factors (such as regulations or dependencies with other systems).

As with the problem statement, constraints must be clear and concise.



▲ **Figure 2** A budget states a clear limitation

A poor example of a constraint and limitation
"We don't have many resources to fix our customer service problem." This statement is vague and provides no guidance about what specific limitations exist. "Resources" is too broad: does it refer to money, people, time, technology, or something else? The impact is also unclear. How does this resource shortage affect the project's potential solutions?
A better example of a constraint and limitation
"The project budget is limited to $10,000, and you cannot exceed this amount due to funding restrictions." This is a strong constraint because it is specific, measurable, and states a clear limitation. It clearly states the budget is the resource constraint, it provides an exact dollar figure, and it states that exceeding this budgeted amount is not an option.

Objectives and goals

Outline the desired outcomes you want to achieve with the solution and differentiate between high-level objectives and specific goals.

High-level objectives represent the broad desired results you want to produce in a particular area of the customer service system. They are directional and aspirational. Specific goals are quantifiable targets that act as milestones toward achieving the high-level objective: they must be directly linked to the high-level objectives. Specific goals are directly measurable and usually time-constrained.

Worked example 1

A business wants to reduce waiting times for customers contacting it. Write an objective and goals for the business.

Solution

A high-level objective might be to "enhance the overall customer experience by minimizing wait times for support."

The specific goals must be directly linked to the objective. These might include decreasing average phone hold time by 20% within the next quarter and reducing average response time to email enquiries to under 12 hours within the next two months. The goals are measurable and time-constrained, so the business can check if it is meeting the goals.

Worked example 2

The business in Worked example 1 also wants to improve customer satisfaction. Write an objective and goals for it.

Solution

A high-level objective could be to "increase customer satisfaction with service interactions."

The specific goals would be to achieve an average customer satisfaction rating of 4 out of 5 stars on post-service surveys over the next six months and reduce the number of "highly dissatisfied" customer ratings by 30% within the next quarter. Both of these are time-constrained, quantifiable and measurable.



▲ **Figure 3** Specific goals are measurable and time-constrained

Input specifications

In the specification, describe the format, type and expected characteristics of the data or information fed into the solution. This ensures compatibility and proper processing.

Worked example 3

Describe the format, type and characteristics of a customer feedback system.

Solution

The input data is customer feedback, both qualitative and quantitative.

The formats might be open-ended text comments (from surveys, emails, social media), numerical ratings (1–5 stars, satisfaction scales), and net promoter score (NPS) responses. NPS rates the likelihood that a customer would recommend a company, product or service to a friend or colleague.

The expected characteristics of the input would include feedback directly related to specific interactions or aspects of the customer service experience, feedback collected as close to the service experience as possible, and, if possible, the ability to link feedback to a specific customer and their interaction history.



▲ **Figure 4** Customer satisfaction scale

Activity

Search online for news stories about businesses local to you. Write an objective and specific goals for the business to address the issue discussed in the news story. Use these worked examples to guide you.

Worked example 4

The company has decided to create a chatbot which accepts customer enquiries and requests in natural language format. State the formats and expected characteristics of the input.

Solution

The formats would be text-based input via a chat interface and, if the chatbot supports voice interaction, voice-to-text input.

Ideally, input would be clear, concise and unambiguous. Of course, this is not the case in the real world, but this should be your goal. In addition, the bot should be trained on a data set of frequently asked questions and common customer needs. Finally, the bot would have a wide variety of answers to handle diverse phrasing and ways of expressing the same intent (for example, "I have a problem with my bill" may need the same answer as "My invoice seems wrong").

Remember that having detailed input specifications helps your solution design and determines the type of data processing, storage and analysis the solution requires. Good input specification also guides the collection of feedback or enquiries to ensure they are useful and support the solution's goals. Finally, you can help ensure compatibility, and ensure the solution can handle real-world data from your customer base.

Output specifications

Define the format, content and presentation of the results generated by the solution. Be clear about how the solution will deliver the desired outcome.

Example: Customer feedback system

Output formats might include dashboards with visualizations (charts, graphs) highlighting trends in customer satisfaction ratings, wait times, and so on. There might also be detailed reports summarizing qualitative feedback and common themes. Finally, alerts for individual cases of highly negative feedback requiring immediate follow-up might be part of the output.



▲ **Figure 5** Input and output

The key metrics would probably include average satisfaction scores, NPS, wait times, volume of feedback by channel, changes in key metrics over time (week-over-week, month-over-month) and categorization of qualitative feedback such as common issues, praise or areas for improvement.

The presentation would need to have a clear, user-friendly interface suitable for customer service managers and team members and customizable reports allowing for filtering by date range, channel or topic.

Example: Customer service chatbot

Output formats might include direct responses to customer enquiries in text format, knowledge base articles or frequently asked questions (FAQs) linked to relevant topics and escalation to a live agent with a transcript of the chatbot interaction for seamless transfer.

Content probably includes accurate answers to common questions, step-by-step guidance for resolving simple issues, and the ability to recognize when a query is beyond its capabilities and redirect appropriately. The bot must appear friendly and use natural-sounding language, with an option to display timestamps for clarity.

Output specifications help ensure there is a high degree of clarity. They ensure the solution produces results aligned with the desired outcomes for improved customer service. Outcomes also help define what metrics and presentations you need to be able to track if the solution is effective. Finally, outcome specifications guide how the solution will interact with customer service staff.

Evaluation criteria

Define the benchmarks you will use to measure the success of the solution. Consider factors such as effectiveness, efficiency, accuracy, usability and maintainability.

Example: Customer feedback system

One of the first and best questions to ask when evaluating a solution is: how effective is it? In other words, does the solution solve the problem you set out to solve or answer the question you asked?

In this example, you would look for improved customer satisfaction scores, positive movement in average ratings, and/or NPS over time. You would also want to measure reduced negative feedback and measurable decrease in the volume of highly dissatisfied customers. Finally, the feedback system would start to identify trends and customer pain points (things that frustrate customers in the sales process), which would lead to targeted improvements.

In terms of efficiency, you would look at time savings and reduced time spent manually gathering and analysing feedback. Faster issue identification might include proactive flagging of critical customer concerns, allowing them to be resolved sooner.

Usability would measure user adoption, meaning that customer service staff find the dashboards and reports easy to use and integrate into their workflow. The interface should be intuitive, meaning that users require minimal training to navigate and extract meaningful insights.

In terms of maintainability, the adaptability of the system can be adjusted to accommodate new feedback channels or changing customer needs, and the system reliably safeguards feedback data.

Example: Customer service chatbot

To measure effectiveness, you should look at the task completion rate, which is the percentage of enquiries successfully resolved by the chatbot without needing human escalation. The customer satisfaction with chatbot interactions should also be evaluated by analysing positive ratings on post-chat surveys. Business users of the system would hope to have reduced strain on live agents and a measurable decrease in the volume of simple, repetitive enquiries handled by human staff.

The system should have 24/7 availability and the ability to provide support outside of regular business hours. You would also expect faster resolution times, so the average time taken to solve basic issues is shorter with the chatbot than through traditional channels.

Correctness of responses (accuracy) is important. You need to measure whether the chatbot provides factually accurate information and guidance. Customers will experience this as a question at the end of the query, asking if the chat (with the bot) has resolved their issue. Businesses also need to ensure that their chatbot understands customer intent and can correctly interpret different phrasings of the same request.

Finally, you would want to evaluate usability. Does the chatbot have a simple interface, clear prompts, and an intuitive flow that make it easy for customers to use?

A problem specification may include a problem statement, constraints and limitations, objectives and goals, input specifications, output specifications and evaluation criteria. All of these are of vital importance if you want to solve the right problem the right way. It does take some extra time to create a problem specification, but the result is a much clearer solution to a well-understood problem or question.

This problem in practice

Jana is a software engineer at a large hospital. She was asked to help manage a hospital unit with limited staff, beds, and specialized equipment. She needed to schedule these resources to accommodate incoming patients while maximizing efficiency and minimizing conflicts.

She started with **abstraction**, and identified core elements of the problem: staff (doctors, nurses, technicians with specific skills), equipment (operating rooms, specialized machines, and so on) and beds (general ward, ICU, and so on).

At first, she simplified the problem and ignored factors such as staff breaks or equipment maintenance schedules and she assumed all patients had equal priority.



▲ **Figure 6** Scheduling software helps hospitals to use resources efficiently

She then started to apply **pattern recognition** to this problem. She analysed historical data, looking at past records for peak times for certain types of patient cases (more trauma at night, elective surgeries in the morning). She also investigated patterns in the length of stay for different conditions. She analysed incoming patient data and realised incoming patient scheduling was based on urgency and resource needs. She also recognized a potential conflict when two patients needed the same specialized machine simultaneously.

Jana started to **decompose** the problem. She looked for subproblems where she could break down the large scheduling problem. She saw three subproblems.

1. Staff scheduling: match staff skill sets to predicted patient needs.
2. Equipment scheduling: align equipment availability with surgical procedures or treatment schedules.
3. Bed allocation: prioritize ICU beds for critical cases, track the expected discharge timeline for other patients.

Next, Jana started to consider how to solve this problem. She applied **algorithmic design** to the problem. She considered heuristic algorithms, which provide quick, "good enough" solutions. Her heuristic algorithm could assign staff based on availability and basic skill fit, then adjust as needed. She also considered optimization algorithms where she could search for the mathematically best solution. This is more computationally demanding but would provide a more stable and predictable schedule.

As Jana considered the possible algorithm to use, she realized she needed to consider the following.

1. Prioritization: how do you weigh urgency, resource scarcity and potential conflicts?
2. Flexibility: how does the algorithm adapt to unexpected events (ambulance arrival, equipment malfunction)?
3. Evaluation: how do you measure the "success" of the schedule (efficiency, patient wait times, resource utilization)?
4. Risk: the cost of poor scheduling may result in harm to a human being. What is the correct ethical approach to use when developing a solution for the hospital?

This process of computational thinking led to a high-quality effective solution for the hospital and the patients.

B1.1.2 Describe the fundamental concepts of computational thinking

Decomposition

Decomposition involves breaking down complex problems into smaller, manageable components. This makes problem-solving easier by focusing on individual pieces. The single responsibility principle (SRP) aids this process by ensuring each component has a single, clear task. While there is no universal algorithm for decomposition, asking key questions can help identify the fundamental parts of a problem.

The single responsibility principle was first used in the context of object-oriented programming (OOP), which you will learn about in topic B3.

Can I divide this problem into smaller, more manageable steps? Are there natural phases or sequences to follow?

Are there any repeated patterns or elements I can group together? Identifying repetitive processes often reveals opportunities for modularization.

Which parts of the problem are independent of each other? Does the order of solving some parts matter or can they be solved in isolation?

What does a picture of the problem look like? Can I sketch diagrams or mind maps to visually represent the problem and its parts?

If you had to explain this problem to a 5-year-old child, how would you do it? This is colloquially known as "Explain like I'm five" (ELI5) and forces you to simplify the problem and focus on the essentials.

Decomposing a bicycle

Using the five questions above, how would you decompose a bicycle?

1. Frame
2. Drivetrain
3. Wheels
4. Steering and control
5. Brakes
6. Seating
7. Mechanisms for changing gears
8. Suspension
9. Lights and reflectors
10. Fenders

Decomposing a smartphone

You can decompose a smartphone into constituent parts. (This is not an exhaustive list.)

1. Touchscreen
2. Display panel
3. Battery pack
4. Charging circuit
5. CPU
6. RAM
7. Storage (for example, internal storage, microSD card)
8. Buttons (for example, volume, power)
9. Ports (for example, USB, headphone jack)
10. Front camera
11. Rear camera
12. Flash
13. Accelerometer
14. Gyroscope
15. Proximity sensor
16. Ambient light sensor
17. Operating system and software

By breaking down the smartphone into these fundamental components, you can better understand and manage its complexity.

Decomposing AI behaviour

Imagine you want to design a non-playing character (NPC) that behaves realistically within a video game environment. How might you decompose this problem? You would start by thinking about the basic actions an NPC needs to complete.

Table 1 A method to decompose NPC actions

Component	Example
Sensing	Equip the NPC with the ability to perceive the player and the surroundings (sight, hearing).
Decision-making	Implement a system for the NPC to evaluate situations and pick actions (for example, a behaviour tree or a finite state machine).
Actions	Develop a set of actions the NPC can take (patrolling, attacking, fleeing, hiding).
Pathfinding	Ensure the NPC can navigate the environment to reach its goals.

Table 2 Alternative method to decompose NPC actions

Component		
Motivation	**Goals** Determine high-level goals or needs motivating the NPC's behaviour (for example, survival, resource gathering, territory defence).	**Personality** Establish traits that influence decision-making (for example, aggressive, cautious, curious).
Perception	**Sensory input** Define types of stimuli the NPC reacts to (visual, auditory, proximity).	**Filtering** Process raw sensory data to focus on relevant details (for example, recognizing threats, identifying valuable objects).
Decision-making	**Behaviour library** Create a modular set of potential behaviours the NPC can choose from (for example, investigate, pursue, attack, retreat, hide).	**Evaluation module** Develop a system (for example, utility-based AI, rule-based system) to evaluate behaviours based on motivation, personality and perceived information, and select the most appropriate action.
Action execution	**Animation control** Tie specific behaviours to corresponding animations (for example, play a "flee" animation when the retreat behaviour is chosen).	**Movement control** Use pathfinding and movement systems to execute chosen actions, ensuring the NPC's actions are physically possible in the world.

Pattern recognition

Pattern recognition is the ability to identify recurring similarities, trends or regularities within data, problems or solutions. Pattern recognition helps you to solve problems by identifying similar subproblems to solve. Solutions to similar subproblems can then be generalized or adapted, avoiding the need to reinvent solutions for each variation.

As with decomposition, there is no single foolproof algorithm to identify patterns. However, there are some questions you can ask yourself as you search for patterns

What elements or components make up the system? Are there specific parts that interact in predictable ways?

Are there repeated actions, processes or structures? Do things happen in the same way or a similar way over time?

How do things change within the system? Are there specific transformations, increases, decreases or transitions that happen regularly?

Worked example 5

Predict the next number in this sequence: 2, 4, 6, 8.

Solution

Applying pattern recognition, you will recognize that each number is 2 greater than the previous one. This is an increasing pattern with a constant difference. A computational solution is to represent this pattern with a simple formula:
next_number = current_number + 2

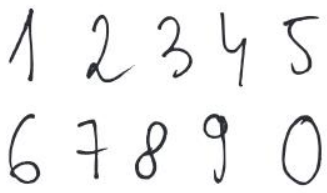
This pattern represents the concept of arithmetic progression, allowing you to generalize the solution to predict future numbers in any sequence with this constant difference. No matter how large this sequence becomes, you can find the next number.

Worked example 6

Build a system that can recognize handwritten digits (0–9) from images.

Solution

Start by considering pattern recognition. Each image can be represented as a grid of pixels, where each pixel has a brightness value (grayscale). These pixels become the raw features your solution needs to look for.
Then, look for the characteristic shapes and patterns of each digit. For example, "0" is often an oval shape, "1" is typically a vertical line, and "8" has two connected loops.
Develop a computational solution to look for these patterns. For example, any pattern which has a loop or oval is likely to be a 0, 6, 8 or 9.



▲ **Figure 7** Handwritten digits

Abstraction

Abstraction is the process of filtering out unnecessary details to focus on the essential elements or properties of a problem or system, representing them with a simplified model. Abstraction allows you to ignore distractions and create generalized solutions that can apply to other similar problems.

The challenge with abstraction is deciding what is necessary and what is unnecessary. Focus on the essential parts of the system. For example, the colour of a car is not essential but the type of engine is.

Ask yourself these questions as you abstract a problem:

What is the core purpose of the system? What is the ultimate problem it is trying to solve, or what main outcome does it produce?

What are the essential inputs and outputs? What minimum information does the system need to start, and what does it generate at the end?

Which parts of the system are truly necessary? If I remove a component, does the system still achieve its core purpose, or does it break down?

Can I simplify how components interact? Can I replace multiple steps or components with a single, higher-level concept?

Are there common functions performed by different parts? Can I group parts by what they do, regardless of their specific details?

Could I replace specific details with variables? Instead of focusing on concrete values, can I think of elements in terms of placeholders?

Worked example 7

Make a program to predict the outcome of a football match between your two favourite teams.

Solution

To create an abstraction of two football teams, ask yourself what are the essential parts of the teams that are likely to impact who may win. Consider, for example, the following.

Will the colour of a player's shoelaces matter? No. We can be confident that this will not affect the result.

Will a recent injury impact the game? Probably, but it will depend on the severity of the injury, how many players are injured, and their position(s) on the team.

Will the team's past record impact the game? Probably. If team A has beaten team B 127 times in the last 128 games, they are likely to feel very confident.

Will the age of the players matter? This is harder to say: older players have maturity and experience but younger players are usually faster and more agile. On balance, this may not be as important to the outcome.
So, an abstraction could be: "the chance of winning depends on (serious injuries) – (past record of wins)."

This is a simplification of a complex system. That is the essence of abstraction.

Worked example 8

Create an abstraction to predict the winner of a car race.

Solution

Work out what are the essential factors that will impact the outcome.

Will the colour of the car matter? Probably not.

Will the engine horsepower, tyre condition and aerodynamics of the car matter? Yes, absolutely.

Will the driver's favourite snack matter? Unlikely.

Will the driver's experience, reaction times and ability to handle the car under pressure impact the race? Yes, definitely.

Will the shape of the clouds matter? No, but the weather conditions are likely to be a factor.

Will the track layout (sharp turns versus long straights) and track surface impact the race? Yes, certainly.

Will unexpected things like mechanical failures or pit crew errors dramatically change the race? Yes, of course.

Your abstracted model might focus on these essential factors.

Car performance on a scale from 1 to 10. Assign a composite score based on things like engine power, aerodynamics and tyre wear.

Driver skill on a scale from 1 to 10. Assign a rating based on the driver's past record, qualifying times, and any known strengths or weaknesses.

Track suitability on a scale from 1 to 10. Assign a value based on how well the strengths of the car and driver match the demands of the specific course. A powerful car might excel on a track with long straights, while a skilled driver might have a greater advantage on a technical track.

Model mechanical failures. Assign a chance there is a mechanical failure for each component of the car. The brakes might have 50% chance of failure, a tyre might have a 75% chance of failure, and the engine might have a 10% chance of failure.

Remember that abstractions are simplifications. This model gives you a starting point for focusing on the elements most likely to determine the race winner. A real prediction model would be much more complex.

Algorithmic design

An algorithm is step-by-step instruction to solve a problem or achieve a desired outcome. Algorithms provide a logical and unambiguous roadmap for a computer (or human) to follow. Algorithms must be clear, precise and efficient.

Key term

Flowcharts Diagrams which logically and unambiguously represent the process (flow) of a program or system.

In this part of computational thinking, you need to design an algorithm. This can be in code, but it does not have to be. Many programmers use **flowcharts** to help them think through how the system functions. You will learn more about flowcharts in the next section.

There are some questions you should ask yourself as you design an algorithm.

What is the exact problem I am trying to solve? Can I state it clearly and concisely?

What are the inputs? What kind of data will my algorithm need to process?

What are the expected outputs? What should my algorithm produce, and in what format?

Are there any decision points in the algorithm?

Are there any places where there is a loop (or iteration)?

Can I break the problem into smaller, more manageable steps? Is there a natural sequence I can divide the process into?

The last question is especially helpful. Can you explain to a friend (who has limited knowledge of your solution) the step-by-step process of how your solution will function? This often helps you find missing bits of your algorithm.

You can look at the number pattern from Worked example 5 in a different way.

Worked example 9

Predict the next number in this sequence: 2, 4, 6, 8.

Solution

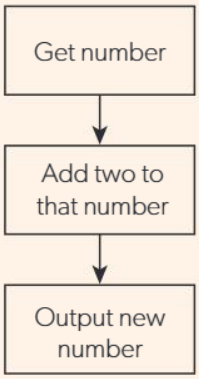
An algorithmic design might be as follows.

1. Get current number
2. Add 2 to current number
3. Output the new number

Or perhaps like this.

1. Get the current and assign it to a variable named NUMBER
2. Add 2 to the variable NUMBER, and store the answer in a variable NEXT_NUMBER
3. Output NEXT_NUMBER

Or, using a flowchart, it might look like this.



ATL Thinking skills

Computational thinking teaches you to identify the core elements of a problem and understand their relationships. Focusing on essential features while filtering out extraneous detail fosters creative solutions. Computational thinking encourages you to identify the underlying patterns and principles behind a problem. The process of designing step-by-step solution procedures reinforces a structured approach to problem-solving. This logical, systematic way of thinking can be transferred to other disciplines, allowing you to adapt your approach based on the nature of the problem.

For example, imagine you have multiple subjects to study for an upcoming exam, but you are unsure how to allocate your time effectively.

Applying computational thinking:

1. **Identify core elements** List the subjects you need to study, the topics within each subject, and the available time before the exam.
2. **Filter out extraneous detail** Focus on the most challenging topics or those with the most weight in the exam, ignoring minor topics that you are already confident in.
3. **Identify patterns** Notice if certain subjects or topics are interrelated (for example, maths principles that apply to physics).
4. **Design step-by-step solution** Create a daily study plan that allocates specific time slots to each subject based on priority. Include breaks and review sessions.
5. **Adapt approach** If you find one topic more difficult than anticipated, adjust your schedule to spend more time on it, shifting easier topics to later.

How can you apply computational thinking to a problem in your life?

B1.1.3 Explain how applying computational thinking to fundamental concepts is used to approach and solve problems in computer science

Computational thinking is a framework for thinking about problems. Using abstraction, algorithmic design, decomposition and pattern recognition, computer scientists can frame problems in such a way that they are computationally solvable. The habit and practice of computational thinking lends itself to becoming an excellent problem solver.

Software development

When creating a large-scale customer relationship management (CRM) system, the project is **decomposed** into several modules such as lead tracking, customer management, and communication tools. Developers identify common software bugs and document these **patterns** to speed up debugging in future projects. They then use object-oriented programming (OOP) to **abstract** complex operations into objects with methods that can be reused across the system without repeating code. **Algorithmic design** includes implementing functionality for automatic email scheduling, defining the precise steps and conditions under which emails should be sent to different user segments.

You will learn about object-oriented programming in more detail in topic B3.

Data analysis

During **decomposition**, a data analyst breaks down the process of analysing large sales data sets into specific tasks like data cleaning, normalization and analysis. In **pattern recognition**, the analyst looks at trends in seasonal sales data to predict future demands and optimize stock levels. **Abstraction** focuses on key performance indicators (KPIs) like monthly sales growth and customer acquisition cost, while filtering out less relevant data. Finally, in **algorithmic design** the analyst will create a data processing pipeline using scripts that systematically extract, transform and load data (ETL process).

Machine learning

Decomposition involves dividing a machine learning project into data collection, feature extraction, model training, evaluation and deployment phases. **Pattern recognition** includes identifying features in image data that are most relevant for classifying images into different categories using convolutional neural networks. **Abstraction** would represent complex data through a set of features and labels that serve as the input and output of machine learning models. Finally, **algorithmic design** would include designing a neural network architecture, specifying the number of layers, activation functions, and optimization algorithms.

Database design

Decomposition structures the database design process into conceptual design, logical design and physical design stages. **Pattern recognition** notices

common queries and structures the database schema to optimize these queries by indexing or denormalization. **Abstraction** defines tables and relationships in a way that represents real-world entities while ignoring irrelevant details. Finally, **algorithmic design** develops algorithms for efficient querying, updating and maintaining database integrity through transaction processes.

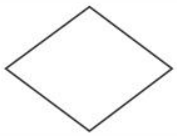
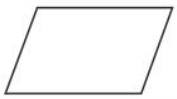
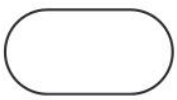
Network security

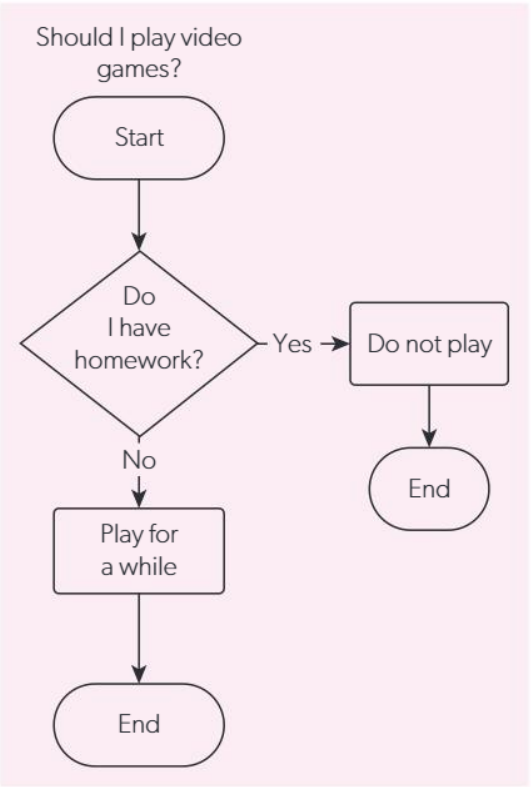
Decomposition analyses network security challenges by categorizing them into physical security, network infrastructure and application security. **Pattern recognition** identifies typical attack patterns like distributed denial of service (DDoS) or phishing to enhance predictive threat detection. **Abstraction** uses generic security models and protocols that provide a framework for implementing specific security measures without detailing the underlying complexities. **Algorithmic design** creates encryption algorithms and protocols like SSL and TLS to securely encrypt communications over the network.

B1.1.4 Trace flowcharts for a range of programming algorithms

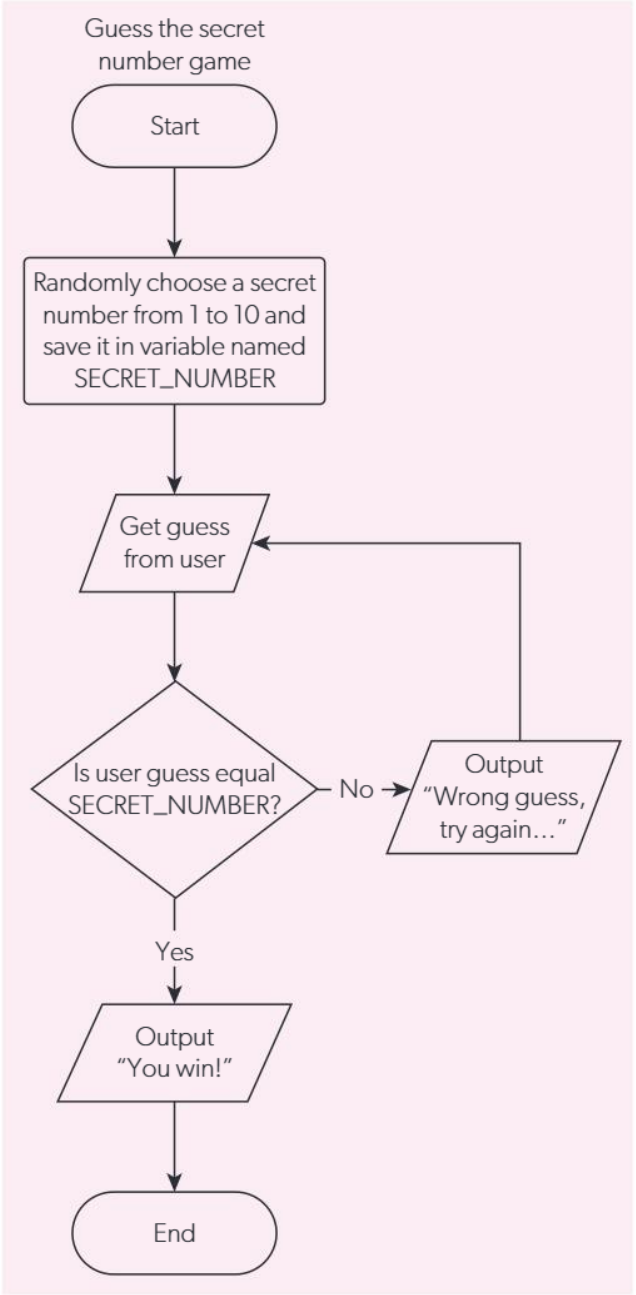
Flowcharts are used to depict processes, decisions, and flows of control. Generally, flowcharts flow from top to bottom and left to right.

Table 3 Standard flowchart symbols. Symbols are approved and standardized by the International Organization for Standardization (ISO) in ISO 5807

Symbol name	Symbol	Symbol definition and notes
Connector		If you have a complicated flowchart, you can use an on-page connector, which connects a part of flowchart without the need for drawing lines. This forms a shortcut between parts of the diagram shown on different parts of the page.
Decision		This symbol indicates a decision with only two possible answers. There should be a question in the symbol, which states which one of **two** paths a program will take. The question should only be a yes/no question or a true/false test.
Flowline		Shows the process's order of operation.
Input/Output		Represents a process of inputting and outputting data.
Process/Operation		Represents a set of operations that changes the value, form or location of data.
Start/End (or Terminal)		Indicates the beginning and ending of a program or sub-process.



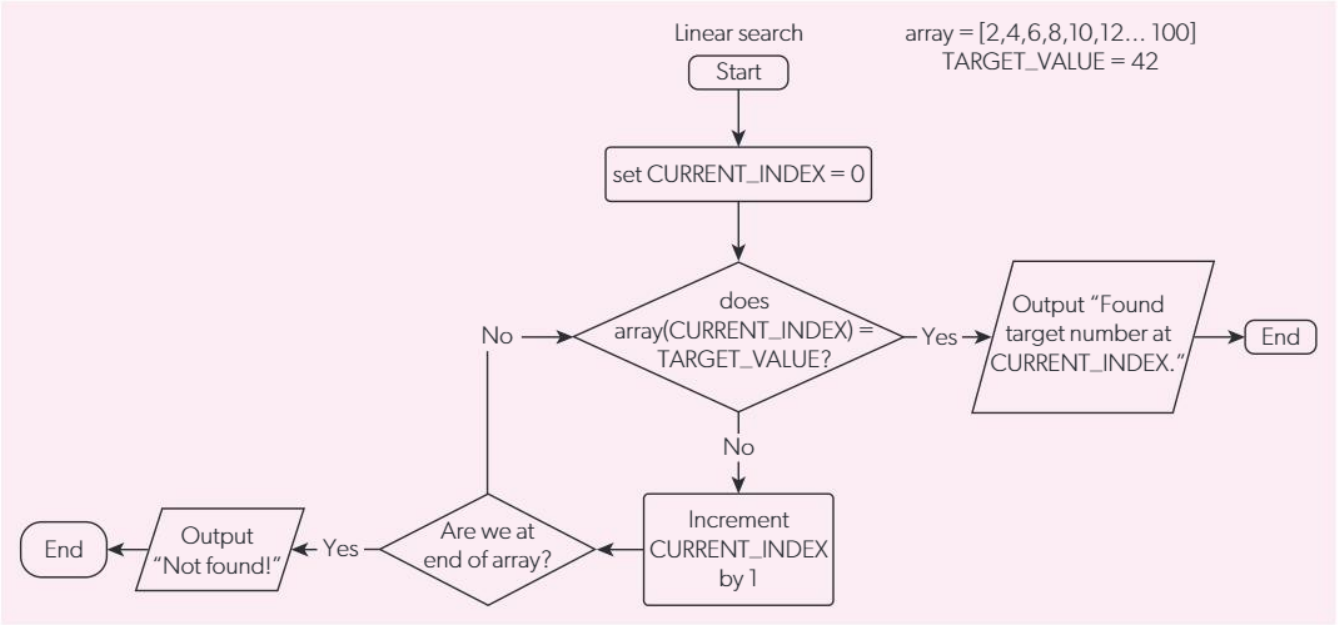
▲ **Figure 8** A simple example of a flowchart. It is helpful to label the flowlines from a decision process. Note that the decision process has only two possible states (outcomes)



▲ **Figure 9** A flowchart for a "guess the number" game. In this game, the player continues to guess until they get the correct answer

You will learn more about linear searches in section B2.4.2.

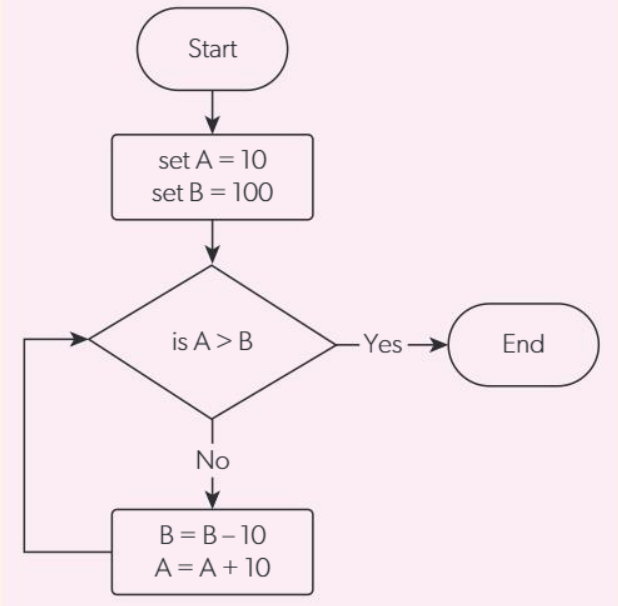
A flowchart can represent the logical process of a computer program. This can help you trace (or follow) the process. A linear search sequentially checks every element in an array, starting at the beginning and working through to the end of the array.



▲ **Figure 10** A linear search

Worked example 10

How many iterations are needed for this program to end?



Solution

After 1 iteration: A = 20, B = 90
After 2 iterations: A = 30, B = 80
After 3 iterations: A = 40, B = 70
After 4 iterations: A = 50, B = 60
After 5 iterations: A = 60, B = 50; program ends

TOK

Computational thinking is a powerful, effective way to understand and solve problems. But a reliance on computational thinking is often criticized for encouraging computer scientists to think about all the possible problems they **can** solve without putting enough thought into the ethical, environmental and social implications of the technology they are creating.

When you abstract a problem, how do you decide what to keep and what to ignore?

What are the implications for a solution if pattern recognition cannot account for important "edge cases"?

Can algorithms be biased?

Practice questions

1. Construct a problem statement for a software system designed to manage patient scheduling in a hospital. [4 marks]
2. Describe constraints and limitations for a mobile banking application, considering security, user interface and network reliability. [4 marks]
3. List a set of input specifications for an online customer feedback system. Include the types of data the system should collect and the expected data formats. [4 marks]
4. Construct a set of objectives and goals for a web-based e-commerce platform, differentiating between high-level objectives and specific measurable goals. [4 marks]
5. Construct evaluation criteria to assess the effectiveness and efficiency of an AI-based chatbot used for customer service, ensuring that both user satisfaction and task completion rates are considered. [4 marks]

ATL Self-management skills

Learning to use software to draw flowcharts neatly and accurately is important. This practical task helps solidify your understanding of flowchart symbols and your uses in depicting algorithmic processes.

1. Look for some flowcharts online and try to follow the logical flow.
2. Use the internet to investigate software that helps you create and trace flowcharts. Gliffy is one option that is easy to use and web-based. There are others, including Lucidchart and Microsoft Visio. You can also find the shapes in Google Slides and Microsoft PowerPoint.

If you do not have regular access to a computer, practice drawing the shapes. You can buy stencils to help you if you are not good at drawing.

3. Construct a flowchart to illustrate a process. This could be of your morning routine, or making a cup of tea, or something else. Include as many details as possible.
4. Ask a friend to explain your flowchart. Listen carefully for misunderstandings or errors.
5. Briefly reflect on how easy or difficult you find flowcharts. If any errors or misunderstandings were spotted in part 4, make a plan to address them.

Linking questions

1. How is pattern recognition used to identify different types of traffic flowing across a network (A2)?
2. How are the concepts of computational thinking used in code when designing algorithms (B2)?

End-of-topic questions

Topic review

1. Using your knowledge from this topic, B1, answer the guiding question as fully as possible:
   How can we apply a computational solution to a real-world problem? [6 marks]

Exam-style questions

2. Describe what a problem statement is and why it is critical in the problem specification process. [2 marks]
3. Explain the difference between a poor and a strong problem statement, giving examples of each. [4 marks]
4. a. Define the role of constraints and limitations in a problem specification. [1 mark]
   b. Outline an example of a well-defined constraint. [2 marks]
5. a. Define:
      i. high-level objectives
      ii. specific goals. [2 marks]
   b. Compare high-level objectives and specific goals in the context of computational problem-solving. [3 marks]
6. Discuss the role of input and output specifications in ensuring the effectiveness of a computational solution. [3 marks]
7. a. Describe the process of decomposition in computational thinking. [2 marks]
   b. Explain how decomposition can be applied to solve a real-life problem. [2 marks]
8. a. Describe the concept of pattern recognition. [2 marks]
   b. Explain the importance of pattern recognition in computational thinking, using a relevant example. [2 marks]
9. a. Describe how abstraction simplifies problem-solving in computational thinking. [2 marks]
   b. Explain how abstraction can be applied to solve a real-life problem. [2 marks]
10. Describe the process of algorithmic design, including the steps typically followed in creating an algorithm. [4 marks]
11. Explain how the four fundamental concepts of computational thinking (abstraction, algorithmic design, decomposition and pattern recognition) work together to solve problems in computer science. [4 marks]



B2

Programming

How can we apply computer programming to solve problems?

Throughout the computer science course and beyond you will be presented with problems to solve. By investigating the different elements of the problem and identifying the different tools you need to solve the problem, you can start to develop programs. For example, if you have to make a decision, you will make use of `IF` statements. If you have to repeat sections of the problem, you will make use of loops. This section of the book will introduce you to the tools you can use.

B2.1 Programming fundamentals

Syllabus understandings

B2.1.1 Construct and trace programs using a range of global and local variables of various data types

B2.1.2 Construct programs that can extract and manipulate substrings

B2.1.3 Describe how programs use common exception handling techniques

B2.1.4 Construct and use common debugging techniques

B2.1.1 Construct and trace programs using a range of global and local variables of various data types

A **variable** is a space in memory used to store data, referenced by a unique identifier. In Java, you need to tell the code what type of data you are storing. This is not necessary in Python, although it can be done.

Key term

Variable A space in memory that is used to store data, referenced by a unique identifier.

For example, to store a whole number, you would use this code in Java to assign a variable:



```
int myNumber = 5;
```

In Python, you would use:



```
myNumber = 5
```

This tells the program to create a space in memory called `myNumber` and that the starting value is 5. The = symbol is used to assign the value. In the Java code, `int` tells the program that the variable is an integer.

You need to be aware of several data types when programming. When declaring a variable of a certain data type, you are telling the variable what it is allowed and not allowed to store, as well as what operations can be performed on the data. Primitive data types are the data types that are built into the program and are used to develop other data types like strings.

To find out more about how data is stored, refer to subtopic A1.2 Data representation and computer logic.

Primitive data types have no methods associated with them. Strings are known as object data types. They have methods associated with them. If you are using Python, strings are classed as primitive data types but they do have methods associated with them.

By declaring data types (which is necessary in Java), you tell the compiler what you want to do with the data and how the stored data will behave. It is always better, for human readability, to show the variables you intend to use at the top of your program.

Table 1 A list of the data types you need to know about in this course

Data type	Description	Assignment in Java	Assignment in Python	Example
Boolean	A variable whose value can be true or false	`boolean x = false;`	`x: bool = False`	True, False
Char	A single character or single ASCII value	`char c = 'a';`	Python does not have a char data type. Use strings instead. `c: str = 'a'`	J, q, 4, %, !
Decimal	A number containing a decimal point	`float f = 7.99;` `double d = 7.99;`	`f: float = 7.99`	1.9283, 2345.12, −594.39393
Integer	A whole number	`int i = 0;`	`i: int = 0`	29399, 4, −2992
String	An alphanumeric collection of characters	`String s = "hello";`	`s: str = "hello"`	Hello, 45 Main Street, P@ssw0rd!

TOK

When you complete a form online, you may often need to complete many different fields, such as your name, your age and your address.

Programmers must decide what data type to use to store each piece of data. For names and addresses, the decision is easy: the data will be stored as a string. Age is different. If you use an integer, you only know the age of the person. If you use a double, you know the age of the person and the month they were born.

When programmers choose the data type, they also make decisions about the type and accuracy of the data that will be stored.

To what extent do you make assumptions when you choose a data type?

This problem in practice

Python does not require you to declare variables ahead of time. This is very convenient for programmers but there is a performance cost, so high-demand servers use Java. According to finance programmer Peter Lawrey, "Java is widely used by tier-one banks in the world for low latency persistence (around a microsecond) and messaging between microservices for trading systems."

However, Python is used by many financial traders, such as Agnes Poh, who works at the Hudson River Trading firm in Singapore. She uses Python to code quantitative trading programs because its flexibility enables her to make changes to algorithms quickly.



▲ **Figure 1** The Central Business District in Singapore

B2.1.2 Construct programs that can extract and manipulate substrings

Strings are a data type that have methods associated with them. This enables them to be manipulated using methods associated with the string data type. The methods associated with strings vary depending on the language you are using. In this course you are expected to be able to identify and extract substrings from given strings, as well as be able to alter, **concatenate**, or replace sections of a string.

Key term

Concatenate To join strings together.

Table 2 String methods in Java

Method	Purpose	Description	Example

`charAt (int index)`	Identify and extract substrings	Returns the character at a specific place in the string.	`String testString = "Old Town";`
			
			`System.out.println(testString.charAt(4));`
			
			Output: T

`contains (String sub)`	Identify and extract substrings	Returns True if the substring is present in the string.	`String testString = "Old Town";`
			
			`System.out.println(testString. contains("Old"));`
			
			Output: True

`indexOf (char c)`	Identify and extract substrings	Returns the index (place) of a given character in the string.	`String testString = "Old Town";`
			
			`System.out.println(testString.indexOf('w'));`
			
			Output: 6

`substring(int beginIndex, int endIndex)`	Identify and extract substrings	Returns a substring of the string.	`String testString = "Old Town";`
			
			`System.out.println(testString.substring(1, testString.length()-3));`
			
			Output: ld T

`replace(String target, String new)`	Alter and replace sections of strings	Replaces a given substring with another given character sequence.	`String testString = "Old Town";`
			
			`System.out.println(testString.replace("Old", "New"));`
			
			Output: New Town

`replaceFirst (String old, String new)`	Alter and replace sections of strings	Replaces the first instance of a substring with a new substring.	`String testString = "The Old Town is Old";`
			
			`System.out.println(testString. replaceFirst("Old", "New"));`
			
			Output: The New Town is Old

`replaceAll (String old, String new)`	Alter and replace sections of strings	Replaces all instances of an old substring with a new substring.	`String testString = "The Old Town is Old";`
			
			`System.out.println(testString. replaceAll("Old", "New"));`
			
			Output: The New Town is New

`ConCat(String new)`	Join strings together	Adds the given string to the end of the current string.	`String testString = "The Old Town";`
			
			`System.out.println(testString.concat (" is a bit dirty"));`
			
			Output: The Old Town is a bit dirty

Table 3 String methods in Python

Method	Purpose	Description	Example

`str[int index]`	Identify and extract substrings	Finds the character at a specific index in the string.	`testString = "Old Town"`
			`print(testString[4])`
			Output: T

`find(str c)`	Identify and extract substrings	Finds the index of a specific character in the string.	`testString = "Old Town"`
			`print(testString.find("T"))`
			Output: 4

`rfind(str find)`	Identify and extract substrings	Finds the last instance of the substring in a string.	`testString = "Old Town"`
			`print(testString.rfind("own"))`
			Output: 5

`slice(int start, int end)`	Identify and extract substrings	Returns a substring from a given string.	`testString = "Old Town"`
			`sub = slice (3,6)`
			`print(testString[sub])`
			Output: To

`replace(str target, str new)`	Alter and replace sections of strings	Replaces one substring with another.	`testString = "Old Town"`
			`print(testString.replace("Old", "New"))`
			Output: New Town

`split()`	Split strings up	Splits the string into a list.	`testString = "Old Town"`
			`x = testString.split()`
			`for i in x:`
			`  print(i)`
			Output:
			Old
			Town

Concatenate using +	Join strings together	Joins one or more strings together.	`testString = "Old Town" + " is a bit dirty"`
			`print(testString)`
			Output: Old Town is a bit dirty

Tables 2 and 3 include a selection of string methods that you may find helpful. Many more are available in the Java and Python **APIs**, which are available online.

Key term

Application programming interface (API) The set of rules the language follows to interact with other programs. The API also lists all methods in the language, and how they function.

Practical skills

Reading, writing and manipulating strings is an important skill when developing solutions. Many of the programs you develop will require user input. Input often involves strings. Being able to use strings within your code will make the program more meaningful for users.

Activity

One use of extracting and manipulating substrings would be to create an initial password for a person. This could be a substring of their family name combined with special characters. Try coding the initial password creator.

ATL Communication skills

Storing data

Identifying how to break down data and store it enables you to understand the type of data that will be collected from the users and how the data will be communicated. Understanding how data is to be communicated is part of the computational thinking process, specifically decomposition.

What data types would you recommend for the following? Justify the answer you have given.

- A telephone number
- Whether a person is vegan
- A password
- The cost of an item
- The number of glasses of water someone has drank in a day
- A person's middle initial

Key term

Exception handling There are many possible points of error in code. Exception handling enables us to deal with these appropriately.

B2.1.3 Describe how programs use common exception handling techniques

When developing code, it is essential that the code functions as expected and is able to handle potential errors caused by the user. When developing programs, there are many potential points in which the program could fail.

One of the first points of failure is unexpected input—when a program is expecting one kind of input but the user enters another. If this is not handled correctly then this could cause an error. Programmers should build in error handling to prevent such issues. This could include having a range check to check that a number falls within a certain value, using drop-down menus to prevent people from selecting something inappropriate, or using a contains method to check the entry contains what is required, such as an @ sign in an email address.

Another point of failure could be resource unavailability. Programs can make use of external resources such as libraries, files, external sensors, or actuators. If you try to make use of the resource but it is unavailable—for example, the file does not exist or the library has not been imported correctly—then you will receive an error message.

The final point of failure is logic errors. Even if you have no syntax errors in your code, logic errors may exist. Logic errors could include using brackets incorrectly within calculations, using the same variable name, data being incorrectly changed, creating an infinite loop, or incorrect use of Boolean operators. This is why you must test your program very carefully with lots of different data to ensure your program produces the correct results.

When there is a program error, such as the unexpected input or resource unavailable errors described above, an exception will occur. An exception is an unexpected behaviour that happens when the program is running. If exceptions are not handled correctly, this could cause the program to break. The exceptions may occur due to non-availability of resources or unexpected data input. In cases of exception errors, programmers may use the `try` and `catch` code blocks in Java, or `try` and `except` in Python.

Examples of these methods are shown below. In these examples, an array has been created to store common names of pets. The `try` section of the program attempts to print out an element that does not exist. Therefore Rather than causing an error, the `catch` statement prints a suitable error message. By surrounding code in `try` and `catch` or `try` and `except`, we can handle errors within our code.

Try and catch in Java

```
1.  String [] pets = {"Ben", "Leo", "Zeppo"};
2.  try {
3.      System.out.println(pets[5]);
4.  } catch (Exception e) {
5.      System.out.println("An error has occurred");
6.  }
```

Output: An error has occurred

Try and except in Python

```
1.  pets = ["Ben", "Leo", "Zeppo"]
2.
3.  try:
4.    print(pets[5])
5.  except IndexError:
6.    print("An index out of range error has occurred")
```

Output: An index out of range error has occurred

Using **try** and **catch** and **try** and **except** will allow you to handle any errors that occur, but it will not allow you to run any code after the exception has occurred. This is when the **finally** block is useful. The **finally** block of code will run regardless of whether an exception was thrown or not. This enables you to ensure no further errors can occur. For example, files can be closed within the **finally**. This also enables you to display additional messages such as "The program has ended safely". An example of code using the **finally** block is shown below.

Try, catch and finally in Java

```
1.  String [] pets = {"Ben", "Leo", "Zeppo"}
2.
3.  try {
4.    System.out.println(pets[5]);
5.  } catch (Exception e) {
6.  System.out.println("An error has occurred");
7.  } finally {
8.  System.out.println("The program has ended safely");
9.  }
```

Output: An error has occurred

The program has ended safely

TOK

There is a saying in computer science: "garbage in, garbage out". This means that if you put false information into a system, then you will receive false information out of the system.

One way to stop "garbage" getting into the system is by using exception handling. Exception handling can stop unexpected data from the user, stop incorrect files being read and ensure the program does not crash.

You rely on the knowledge produced by the computer to make decisions. To what extent does the reliability of this knowledge rely on exception handling?

Try, except and finally in Python

```
1.  pets = ["Ben", "Leo", "Zeppo"]
2.  try:
3.      print(pets[5])
4.  except:
5.      print ("An error has occurred")
6.  finally:
7.      print ("The program has ended safely")
```

Output: An error has occurred

The program has ended safely

ATL Communication skills

Communicating with the end user

Understanding how to help users and make developed solutions shows empathy and caring skills. As a developer, it is important to make your software solutions as user friendly as possible. So when developing algorithms, think about the end user.

Imagine you wanted to develop a program for teachers that enabled them to store data about their students. What information would teachers need to store? What features would the program need?

B2.1.4 Construct and use common debugging techniques

Key term

Debugging Using tools to find errors in your code.

There are several techniques a programmer can use to **debug** code. You may use them when you are programming or when you complete your internal assessment. Debugging code lets you check that your program works before moving onto the next stage of testing.

Trace tables

Trace tables are a way to trace the values of the variables to ensure that they are being changed correctly. Here is an algorithm to count the number of warm days (days over 20 degrees Celsius).

Trace tables in Java

```
1.  // an array temperature has been initialised to
    contain the following items
2.  [23.7, 19.9, 23.8, 18.8, 12.5, 24.0]
3.
4.  int warmDays = 0;
5.  for (int i = 0; i < temperature.length; i++) {
6.      if(temperature[i] > 20) {
7.          warmDays = warmDays + 1;
8.      }
9.  }
10. System.out.println("The number of warms days this
    month: " + warmDays);
```

Trace tables in Python

```
1.  temperature = [23.7,19,9,23.8,12.5, 24.0]
2.
3.  warmDays = 0
4.  for temp in temperature:
5.      if temp > 20:
6.          warmDays = warmDays + 1
7.
8.  print ("the number of warm days this month: ",
    warmDays)
```

The trace table to check this algorithm is working would look something like Table 4.

Table 4 Example trace table

Iteration	Value at i	warmDays count	Output
0	23.7	1	-
1	19.9	1	-
2	23.8	2	-
3	18.8	2	-
4	12.5	2	-
5	24.0	3	3

Breakpoint debugging

When programming, a developer may choose to make a breakpoint. A breakpoint is a point in the code where the algorithm will stop executing. If a programmer amends an algorithm, they may wish to create a breakpoint to review the values of the variables at that point to check that the changes have been effective. This is particularly useful if they wish to check that smaller sections of the code are working effectively in a large program.

```
1     for(int j = i+1; j< toSort.length; j++) {
2         System.out.println("In inner loop and the
          counter is " + j);
3         if (toSort[j]<toSort[highestIndex]) {
4             highestIndex=j;
5             System.out.println("In if statement and
          the highest index is " + highestIndex);
6         }
7     }
8     temp = toSort[highestIndex];
9     toSort[highestIndex]=toSort[i];
10    toSort[i]=temp;
11  }
12  for (int i = 0;i < toSort.length;i++)  {
13      System.out.println(toSort[i]);
14  }
```

▲ **Figure 2** Breakpoint debugging in Java

Practical skills

Debugging skills allow you to effectively find and correct errors in your code. Debugging is one of the key tools of producing knowledge in computer science. When developing algorithms, programmers need to be aware of the given inputs and expected outputs from the code they develop to check for accuracy. If the code is not accurate then debugging techniques need to be applied to fix this.

Print statements

Print statements can be used to print messages throughout the code. This allows programmers to understand what section of a decision they are in, what iteration of a loop they are in, or the value of a variable at a certain section of the code. This is essential, as it enables programmers to understand whether the program is executing in a way they are expecting and whether they need to make adjustments. An example is shown in Figure 3.

```
Scanner input = new Scanner(System.in);

int[]toSort={4,56,23,2938,29,485,2,394,58,
3838,485,9822};

int temp;

int highestIndex=0;

for(int j = i+1; j< toSort.length; j++) {

  System.out.println("In inner loop and
the counter is " + j);

  if (toSort[j]<toSort[highestIndex]) {

      highestIndex=j;

      System.out.println("In if statement
     and the highest index is " +
     highestIndex);

  }

}

temp = toSort[highestIndex];

toSort[highestIndex]=toSort[i];

toSort[i]=temp;

}

for (int i = 0;i < toSort.length;i++)  {

System.out.println(toSort[i]);

}
```

```
In inner loop and the counter is 4
In inner loop and the counter is 5
In inner loop and the counter is 6
In if statement and the highest index
is 6
In inner loop and the counter is 7
In inner loop and the counter is 8
In inner loop and the counter is 9
In inner loop and the counter is 10
In inner loop and the counter is 11
In outer loop and the counter is 1
In inner loop and the counter is 2
In inner loop and the counter is 3
In inner loop and the counter is 4
In inner loop and the counter is 5
In inner loop and the counter is 6
In inner loop and the counter is 7
In inner loop and the counter is 8
In inner loop and the counter is 9
In inner loop and the counter is 10
In inner loop and the counter is 11
In outer loop and the counter is 2
In inner loop and the counter is 3
In inner loop and the counter is 4
In if statement and the highest index
is 4
In inner loop and the counter is 5
In inner loop and the counter is 6
```

▲ **Figure 3** Print statements in Java

Step-by-step code execution

Step-by-step code execution usually provides the developer the chance to "Step in", "Step over", and stop the code. The integrated development environment (IDE) highlights the code that is being interpreted at the time. Step-by-step code execution also shows the values of the variables at any given time so you can see if they have the expected values. If the values of the variables do not match the expected values then you should revisit your code.

```
import java.util.ArrayList;
import java.util.Iterator;
import java.util.Scanner;

public class main {

  public static void main (String[] args) {
    Scanner input = new Scanner(System.in);

    int [] toSort =
   {4,56,23,2938,29,485,2,394,58,
   3838,485,9822};
    int temp;
    int highestIndex = 0;

    for (int i = 0; i < toSort.length - 1;
i++) {

      for (int j = i+1; j < toSort.length;
j++) {
          if (toSort[j]< toSort[highestIndex])
{
            highestIndex = j;
          }
      }
      temp = toSort[highestIndex];
```

Name	Value
no method return value	
args	String[0] (id=20)
Input	Scanner (id=21)
toSort	(id=27)
highestIndex	7
i	6
j	10

▲ **Figure 4** Step-by-step code execution in Java

ATL Thinking skills

Applying knowledge of testing and evaluation

Using the debugging techniques described will help you to become a more knowledgeable coder. You use testing and evaluation skills to investigate the problem to understand where it is going wrong and how you can fix it.

Which debugging techniques would you recommend for the following situations? Whenever possible, justify your answer.

- A program that is in development but is not calculating the correct results.
- A program that is in development but keeps skipping over a section of the code.
- A program that is in development but does not accept user input.
- A program that is in development but the loop is not functioning correctly.

Practice questions

1. Explain how a trace table can be used to identify errors within a section of code. [3 marks]
2. Describe the purpose of the `finally` block of code in exception handling. [2 marks]
3. Identify the correct data types for:
   a. the name of a plant [1 mark]
   b. whether a plant is poisonous [1 mark]
   c. the maximum height of a plant. [1 mark]
4. A school uses the first three characters of a student's name and the last three characters of a student's name along with @567 to develop an initial password. Construct an algorithm that takes a student's name and outputs their initial password. For example: Input "Boris Laurent", Output "Borent@567". [4 marks]

B2.2 Data structures

Syllabus understandings

B2.2.1 Compare static and dynamic data structures

B2.2.2 Construct programs that apply arrays and lists

B2.2.3 Explain the concept of a stack as a "last in, first out" (LIFO) data structure

B2.2.4 Explain the concept of a queue as a "first in, first out" (FIFO) data structure

B2.2.1 Compare static and dynamic data structures

Imagine you are selling tickets to a school concert. You have a fixed number of seats and you can only sell one ticket per seat. This is a fixed, unchangeable list, because if you sold more seats than you had available then you would have a lot of complaints. However, imagine you were also selling drinks and snacks at the concert. You do not know how many drinks and snacks you will sell. Therefore, you would have a variable number of items on your list. The number of items in this list needs to be changeable. This situation is comparable to the idea of static and dynamic data structures in programming.

A static data structure is a data structure that has a size which is set at compile time and which cannot be changed in the runtime environment. Compile time is the stage before the program is run, when the statements within the language are converted into binary instructions that the processor understands. Runtime is when the program is running in memory, which occurs after the compile stage.

For static data structures, when you declare the data structure you must give the data structure a size. Think about the concert seating analogy. You can declare the static data structure with a size equal to the number of seats available. When the program is run, the size of the array will be allocated in memory. An example of a static data structure in Java is an `array`, and in Python it is a `tuple`.

A dynamic data structure does not require you to set the size when you create the data structure. When the dynamic data structure is declared, a certain amount of memory is set aside to allow for elements to be added. If the list grows beyond this capacity, additional memory is allocated to enable these elements to be added to the list. When items are removed from a dynamic list, the memory used is deallocated, freeing up resources. An example of a dynamic data structure in Java is an `ArrayList`, and in Python it is a `List`. In Java, an `ArrayList` is given a data type and can only store data of that type. In Python, a `list` can store multiple instances of different data types.

Consider the following when choosing which type of data structures to use.

Flexibility: If you know the number of items you need, a static data structure is useful. If you have an unknown number of items, then a dynamic data structure might be more useful.

Memory usage: A dynamic data structure tends to be more efficient with memory as it only uses the memory it needs, whereas a static data structure has its memory set at runtime and uses all the space even if nothing is stored in the element.

Speed: A static data structure tends to be slightly quicker when performing actions as you can access elements directly using an index. A dynamic data structure is slightly slower as memory could be fragmented, and this can slow down the performance.

B2.2.2 Construct programs that apply arrays and lists

There are two forms of lists that you need to be aware of in this course: one-dimensional (1D) and two-dimensional (2D).

Each item has an index that you can use to access the data. You can visualize a one-dimensional list as a line of elements. For example, here is a one-dimensional list of integers.

Remember that, in programming, the first element in a list is **0**, not **1**.

Table 5 One-dimensional list

Index	0	1	2	3	4	5	6	7	8
Data	23	17	4839	606	583	484	34	1	1985

A two-dimensional list can also be visualized as a table. For example, here is a two-dimensional list of integers. Remember, the first element in a list is 0, not 1.

Table 6 Two-dimensional list

Index	0	1	2	3	4	5	6
0	82	1954	5	29	65	9003	1029
1	48	95	594	3920	9202	9583	821
2	598	9333	428	859	3847	4839	382

In this course, you need to be able to add, remove and traverse elements in one-dimensional and two-dimensional dynamic lists. Starting with one-dimensional lists, in Java, use `ArrayList` and in Python, use `List`.

First, you need to declare the list. The following code is used to initialize a list of strings.

Key term

List A data structure used to store multiple instances of the same data type under one variable name. In Python, you can store multiple data types in the same list.

Practical skills

Being able to create lists and manipulate data within lists is an essential skill for programming.

ArrayList Java

```java
import java.util.ArrayList;

public class Main {
  public static void main(String[] args) {
    ArrayList<String> myList = new ArrayList<String>();
  }

```

List Python

```
myList = []
```

To add to the list, use the following code.

Java

```
myList.add("Angelica");
myList.add("Ansh");
myList.add("Kai");
myList.add("Fabio");
myList.add("Michelle");
myList.add("Hana");
```

Current list: Angelica, Ansh, Kai, Fabio, Michelle, Hana

Python

```
myList.append("Angelica")
myList.append("Ansh")
myList.append("Kai")
myList.append("Fabio")
myList.append("Michelle")
myList.append("Hana")
```

Current list: Angelica, Ansh, Kai, Fabio, Michelle, Hana

To remove from a list, you have two options: remove by index or remove by value.

Remove by index in Java (the default)

```
myList.remove(2);
```

Current list: Angelica, Ansh, Fabio, Michelle, Hana

Remove by index in Python

```
del myList[2]
```

Current list: Angelica, Ansh, Fabio, Michelle, Hana

Remove by value in Java

```
myList.remove("Kai");
```

Current list: Angelica, Ansh, Fabio, Michelle, Hana

Remove by value in Python (the default)

```
myList.remove("Kai")
```

Current list: Angelica, Ansh, Fabio, Michelle, Hana

There are two ways to traverse a list. You can use a loop, which utilizes the index to access each item in the list, or you can use an iterator in Java or an enumerator in Python, which begins at the start of the list and selects the next item in the list until there are no more items to access.

Traversing a list using loop and indexes in Java

```
for (int i = 0; i < myList.size(); i++){
    System.out.println(myList.get(i));
}
```

Output:

Angelica

Ansh

Fabio

Michelle

Hana

Traversing a list using loop and indexes in Python

```
for name in myList:
   print(name)
```

Output:

Angelica

Ansh

Fabio

Michelle

Hana

Traversing a list with an iterator in Java

```
Iterator <String> it = myList.iterator();
while (it.hasNext()) {
    System.out.println(it.next());
}
```

Output:

Angelica

Ansh

Fabio

Michelle

Hana



Traversing a list with an enumerator in Python

```
for index, name in enumerate(myList):
  print(index, name)
```

Output:

0 Angelica

1 Ansh

2 Fabio

3 Michelle

4 Hana

Two-dimensional arrays in Java

In Java, there are two types of two-dimensional lists you need to be aware of:

- fixed size, or static, two-dimensional array
- dynamic two-dimensional array.

The following sections of code will explain to you how to create and use a static two-dimensional array and a dynamic two-dimensional array.

All code provided will be used to represent the following data set.

	[0]	[1]	[2]	[3]	[4]	[5]	[6]
[0]	21	23	24	22	23	24	29
[1]	22	20	19	18	18	17	20
[2]	24	28	29	31	32	31	33
[3]	23	25	21	26	22	20	19

To create a new static two-dimensional structure, you need to tell the structure what type of data will be held in it and how many rows and columns it has.



```
int [] [] temperature = new int [4][7];
```

This code initializes a two-dimensional array of integers that has four rows and seven columns.

To access a specific element in the array, type the name of the array followed by the row number and column number.

```
temperature [3][2] = 34;
```

To look at every element of the array, you need to have a double `for` loop. The element loops through each column in a row and then moves onto the next row.

Examples of all of this are outlined below. This program calculates the average temperature in a given month.

Two-dimensional array in Java

```
1.  public static void main(String args[])
2.
3.        int [][] temperature = new int [4][7];
4.        int average = 0;
5.        int count = 0;
6.
7.        for (int i = 0; i < temperature.length; i++) {
8.              for (int j = 0; j < temperature[0].
                length; j++) {
9.                average = average + temperature[i][j];
10.               count = count + 1;
11.             }
12.       }
13.       System.out.println("The average temperature
          this month is " + average/count);
14. }
```

A two-dimensional dynamic array structure enables a list that can change size. The number of rows can change, as can the number of columns. It is still recommended to tell the `ArrayList` what kind of data it is holding.

To initialize the two-dimensional `ArrayList` you use the following code.

```
ArrayList<ArrayList<Integer>> temperature = new
ArrayList<ArrayList<Integer>>();
```

To access a specific element of an array you need to type the index of the row and then the index of the column, as shown in the following code.

```
temperature.get(i).get(j);
```

This enables you to access each specific element. You can also use this code to add to the specific element.

```
temperature.get(2).add(25);
```

To look at every element of the array, you need to have a double `for` loop. The element loops through each column in a row and then moves onto the next row.

Examples of all of this are outlined below. This program also calculates the average temperature in a given month.



Two-dimensional dynamic array in Java

```
1.  public static void main(String args[]){
2.
3.     ArrayList<ArrayList<Integer>> temperature = new
       ArrayList<ArrayList<Integer>>();
4.
5.     temperature.add(new ArrayList<Integer>());
6.     temperature.add(new ArrayList<Integer>());
7.     temperature.add(new ArrayList<Integer>());
8.     temperature.add(new ArrayList<Integer>());
9.
10.    temperature.get(0).addAll (new
       ArrayList<>(Arrays.asList
       (21,23,24,22,23,24,29)));
11.    temperature.get(1).addAll (new
       ArrayList <>(Arrays.asList
       (22,20,19,18,18,17,20)));
12.    temperature.get(2).addAll (new
       ArrayList <>(Arrays.asList
       (24,28,29,31,32,31,33)));
13.    temperature.get(3).addAll (new
       ArrayList <>(Arrays.asList
       (23,25,21,26,22,20,19)));
14.
15.  temperature.get(1).add(24);
16.  temperature.get(2).add(25);
17.
18.  int average = 0;
19.  int count = 0;
20.
21.   for (int i = 0; i < temperature.size(); i++) {
22.       for (int j = 0; j < temperature.get(i).
          size(); j++) {
23.            average = average + temperature.get(i).
               get(j);
24.            count = count + 1;
25.       }
26.    }
27.    System.out.println("The average temperature this
       month is " + average/count);
28. }
```

Two-dimensional lists in Python

A two-dimensional dynamic array structure enables a non-fixed size list of temperatures. The number of rows can change, as can the number of columns.

In Python, a fixed sized list is a tuple. A tuple acts differently to a list: elements cannot be changed once added. All lists in Python are automatically dynamic.

To initialize a two-dimensional list in Python you use the following code.

Two-dimensional array in Python

```
temperature =
[[21,23,24,22,23,24,29],[22,20,19,28,28,17,20],[24,28,
29,31,32,31,33],[23,25,21,26,22,20,19]]
```

To access a specific element of an array you need to type the index of the row and then the index of the column, as shown in the following code.

```
print(str(temperature[2][3]))
```

This enables you to access each specific element. You can also use this code to add to the specific element.

```
temperature[2].append(25)
```

To look at every element of the array, you need to have a double `for` loop. The element loops through each column in a row and then moves onto the next row.

Examples of all of this are outlined below. Again, the program calculates the average temperature in a given month.

Temperature program in Python

```
1.  temperature =
    [[21,23,24,22,23,24,29],[22,20,19,28,28,17,20],[24,
    28,29,31,32,31,33],[23,25,21,26,22,20,19]]
2.
3.  temperature[1].append(24)
4.  temperature[2].append(25)
5.
6.  count = 0
7.  average = 0
8.
9.  for rows in temperature:
10.   print ("")
11.   for columns in rows:
12.     print(columns, end = " ")
13.     count = count + 1
14.     average = average + columns
15.
16. print ("The average temperature is", str(average/
    count))
```

Activity

A two-dimensional list can be used to store a seating plan for a classroom. Try to code a seating plan program.

Key term

Last in, first out (LIFO) data structure A data structure in which the last item added to the stack or queue is the first to be removed. You may also see this called a **first in, last out (FILO)** data structure: the two terms are interchangeable.

B2.2.3 Explain the concept of a stack as a "last in, first out" (LIFO) data structure

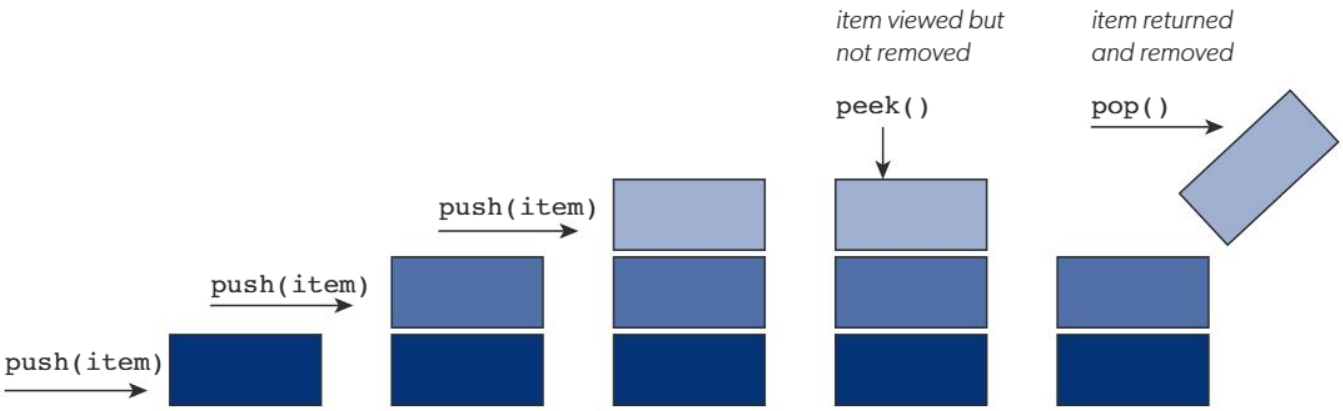
Imagine you have a pile of plates that you cleaning. You can only add and take away from the top of the pile. You can only clean the top plate on the pile. The plate at the bottom of the pile is the last item you will use. This is similar to a stack in a computer system.

A stack is known as a "last in, first out" (LIFO) data structure. A logical view of a stack is shown in Figure 5: you can see that three items have been added to the stack. If you want to add another item to the stack you use the `push()` method to add to the top. You can use the `peek()` function to look at, but not remove from, the top of the stack. Finally, to remove an item from the stack, you use the `pop()` method.

Methods that can be applied to a stack are outlined in Table 7.

Table 7 Stack methods

Method	Explanation
`isEmpty()`	Checks to see if the stack contains data.
`push()`	Adds data to the top of the stack.
`pop()`	Removes data from the top of the stack.
`peek`	Allows you to view the data at the top of the stack without removing the data.



▲ **Figure 5** Visual representation of a stack

Whenever the `push`, `pop` or `peek` methods are evoked, the operation is always O(1) as only the top element of the stack is being accessed at any given time and the size of the stack is irrelevant. Accessing the first item added to the stack is an O(n) operation, as you need to remove all items above it to reach that item. When a stack is created in programming it is of a fixed size. Stacks do not support dynamic allocation; therefore, if stacks become too big there is not enough space within memory and you will get a stack overflow error.

Big O(1) and Big O(n) are measurements of the efficiency of the algorithm. This is known as Big O efficiency. This will be explained further in section B2.4.1.

Stacks can be used in computer science to solve the following problems.

Recursive problems: Stacks are essential in recursive algorithms. Every time a recursive **call** is made the current state is pushed onto the stack. Once the base case is resolved the `push` method is used to resolve the calls.

Parsing: A stack is used to ensure that all of the brackets are closed. For example, in Java if the braces `{}` are closed, every time an open brace `{` is read it is pushed onto the stack and every time a closed brace `}` is read it is popped from the stack. An empty stack means the correct number of brackets.

Undo functions: A stack is used to keep a track of the operations completed by a user of a program. Every operation is pushed onto a stack. Each time they undo an action, the last action is popped from the stack and reversed.

Reversing a string: The characters from the string can be pushed onto the stack separately. When they are popped from the stack they are in reverse order.

Key term

Call When you call an algorithm, you are making a request for the program to perform the action detailed by that algorithm using the data provided.

Activity

A quadrat is a frame used in biology to count the number of organisms within a sample area.

Imagine you are using a quadrat to record the number of times you see a daisy. A stack could be useful as you could push the number of daisies per quadrat onto the stack. Try coding this program to help students using quadrats.

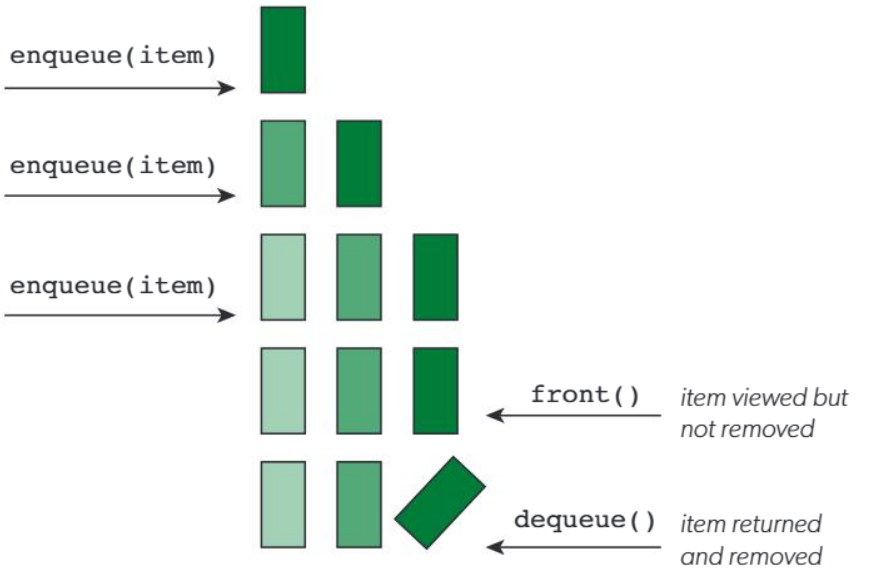


▲ **Figure 6** A queue of people

B2.2.4 Explain the concept of a queue as a "first in, first out" (FIFO) data structure

Imagine you are in a busy shop. You find the item you need and join a queue to pay. Because you join the back of the queue, the people in front of you will be served before you, in the same order they joined the queue. This is similar to a queue in a computer system.

A queue is known as a "first in, first out" (FIFO) data structure. A logical view of a queue is shown in Figure 7. In the figure you can see that three items have been added to the queue. If you wanted to add another item to the queue you would use the `enqueue()` method to add to the back of the queue. You could use the `front()` function to look at, but not remove from, the front of the queue. Finally, to remove, you use the `dequeue()` method.



▲ **Figure 7** Visual representation of a queue

Table 8 Queue methods

Method	Explanation
`isEmpty()`	Checks to see if the queue contains data.
`enqueue()`	Adds data to the back of the queue.
`dequeue()`	Removes the data from the front of the queue.
`front()`	Allows you to view the data at the front of the queue without removing the data.

To dequeue or access the front element of a queue is an O(1) efficiency. You are only accessing the first element, which is always at the front. To dequeue the last element in a queue is O(n) efficiency as you need to dequeue all elements in front of it. In programming languages, queues can use dynamically allocated memory. Therefore, if you have an unknown data set size, a queue can be more efficient in its use of memory than a stack.

Queues can be used in programming to solve the following problems.

Managing playlists in media: When you create a playlist for Spotify or YouTube, a queue is used to keep a track of what you add to your playlist, and when the current item you are listening to or watching is finished the **`dequeue()`** method is called to get the next item.

Printer queues: If you send many jobs to the printer, they are added to the print queue. If your printer is connected to your machine then once each print job has completed, the **`dequeue()`** method is called to get the next one.

Networking: Queues are used to manage traffic in routers and switches. Data packets are stored in the queue until they can be processed or serialized.

Activity

You could use a queue to model aeroplanes entering and leaving an airspace. Try coding this model. You can assume the planes arrive and leave the queue in the expected order.

ATL Thinking skills

Identifying suitable data structures

Using data structures allows you to become more knowledgeable. Think carefully: What does your program need to do? What data structure will support your algorithm? This is an essential part of the algorithmic thinking process.

Do you agree or disagree with the following data structure decisions?

- A list of animals in the zoo: one-dimensional list.
- A seating plan in a classroom: one-dimensional list.
- A list of planes waiting to land in an airport: stack.
- A line of people waiting to pay in a store: stack.
- A representation of a shelf in a convenience store: two-dimensional list.
- Jobs waiting to be printed by a network printer: a queue.

Compare your answers with a partner. If you got different answers, discuss the decision together to work out the correct answer.

TOK

Data structures are used in all aspects of everyday life. Lists can be used to identify a shopping list or a "to-do" list. Two-dimensional lists are similar to calendars and planners. If you wash dishes, you interact with stacks. Any time you wait in line to pay, perhaps in a supermarket or buying tickets for a movie, you are using a queue.

Abstract data types (ADTs) have set behaviours that determine how they act.

How can the abstract concepts of data structures be used to benefit real-life applications?

Practice questions

5. Outline three differences between a static data structure and a dynamic data structure. [6 marks]

6. A program stores the different transactions that have occurred in a bank in a day. Explain whether a static or dynamic data type would be suitable for this purpose. [3 marks]

7. A program stores the activities that occur monthly at a gym. Here is an extract from the program.

	[0]	[1]	[2]	[3]	[4]	[5]	[6]
[0]	8 – Pilates	18 – Boxfit	13 – Attack	9 – Pilates 19 – Boxfit	14 – Attack	8 – Pilates	8 – Pilates 10 – Boxfit
[1]	10 – Boxfit	9 – Pilates 17 – Attack	19 – Boxfit	9 – Pilates	9 – Baby Yoga	8 – Pilates	8 – Pilates
[2]	8 – Pilates	11 – Boxfit 17 – Yoga	18 – Pilates	12 – Boxfit	18 – Pilates 19 – Attack	8 – Pilates 10 – Boxfit	8 – Pilates
[3]	18 – Pilates	18 – Boxfit	19 – Pilates	19 – Boxfit	19 – Pilates	8 – Pilates	8 – Pilates

   a. Construct an algorithm to count how many Pilates classes there are during the month. [5 marks]

   b. Construct an algorithm that counts how many evening classes (classes happening at 18 or 19) there are. [5 marks]

   c. Construct an algorithm that returns a list of Boxfit classes including the week and day they occur on. Assume [0][0] is Monday week 1. [6 marks]

8. Describe a real-world application for a stack and a queue. [4 marks]

B2.3 Programming constructs

Syllabus understandings

B2.3.1 Construct programs that implement the correct sequence of code instructions to meet program objectives

B2.3.2 Construct programs utilizing appropriate selection structures

B2.3.3 Construct programs that utilize looping structures to perform repeated actions

B2.3.4 Construct functions and modularization

B2.3.1 Construct programs that implement the correct sequence of code instructions to meet program objectives



▲ **Figure 8** Cupcake recipe

When cooking, you often follow a recipe. If you follow the instructions in the correct order then you will make the food you expect. However, if you do not follow the instructions in the correct order, you may end up with something inedible. This is similar to developing algorithms. You need to carefully place the instructions in the correct order to get the correct functionality.

Placing the instructions in the correct order enables you to avoid errors such as infinite loops, deadlocks or incorrect inputs.

Infinite loops occur when the end condition of a loop is never met. For example, you do not increment a counter, or the base case is below the end condition so it is never met.

Deadlocks occur when two processes need access to the same resource at the same time. A non-programming example of a deadlock would be in a kitchen where one person has use of the cooker and another the counter top. If the person using the stove cannot free up the stove without using the counter top to place their item but the person using the counter top cannot move without having the stove free, neither can move and they are in a deadlock. Clear sequencing of instructions enables resources to be used at different times so that the deadlock can be avoided.

Incorrect inputs can break a program. It is important to consider this when you ask the user for inputs and to implement methods to prevent incorrect inputs breaking the program.

TOK

In computer science, you learn how to construct algorithms by developing solutions to problems. You can extend your knowledge by developing those algorithms. By carefully testing the algorithms, you can learn how they work, how to make them efficient, and how to incorporate them into different programs. This cycle of designing, developing and testing algorithms is one of the key methods of producing knowledge in computer science.

To what extent does the interpretation of the stages of the algorithm impact the knowledge produced by the algorithm?

ATL Thinking skills

Applying knowledge to real-world problems

Inquiring into real-world problems helps you to stay curious and think about computational processes in the real world. Identifying the different stages of real-world processes and the order they need to be completed enables you to start the pattern recognition process.

Consider the following process. What patterns are occurring? What actions need to occur to complete the task?

Without using code, write an algorithm to complete the task.

An escape room company has an overview of how they help customers to "escape". When the customers arrive, they are given 10 minutes to read through the instructions and the story of the room. Once they enter the room, they are left alone for 15 minutes. If they have not activated any triggers in 15 minutes, two clues are supplied. After 15 minutes, if there are no new triggers within 10 minutes, help is provided. At any given time, if the user requests help, they are given a clue. If the customers are particularly bad (for example, they have not activated a trigger in 10 minutes after two clues) a trigger is automatically activated.

B2.3.2 Construct programs utilizing appropriate selection structures

You make decisions every day, even if you are not consciously thinking about them. The games you play on your computer make decisions about what comes next based on the current performance in the game. If you fail a level and you have enough lives then the program will allow you to start again. If you do not then your game is over. This is known as **selection** in computer games.

There are two types of selection statements within programming you can make use of. You can use `if`, `if` ... `else` and `else` statements or you can use `case` statements. The most common selection statement in any programming language is an `if` statement. These `if` statements can be used with variables to help us control the flow through the program.

An `if` statement always follows the same structure.

If (condition)	Action that will occur if the condition evaluates to True
Optional else if (condition)	Action that will occur if the condition evaluates to True
Optional else	Catch-all action

A simple program saying what to do in your free time depending on your age is shown in the example that follows. The different options shown depending on age are controlled by `if` statements. For example, if you are under 5, the option displayed will be go to a soft play centre. If you are over 5 and under 12, the option to go to the park will be displayed. If you are 12 to 18 then the option to spend time with friends will be displayed. If you are over 18 then it will recommend you go for dinner. The program is going to assume that over 100 is an invalid age.

Key term

Selection Making decisions within the code, usually completed with an `if` statement.



▲ **Figure 9** Computer games make decisions based on performance

To understand what is happening when these statements are being evaluated, review subtopic A1.2 Data representation and computer logic.

Practical skills

Making decisions in code allows you to develop interesting programs.

TOK

There are several well-known examples of bias being accidentally built into algorithms. Bias occurs due to the data used to train the system or the way the program has been developed by programmers. You will learn more about this in topic A4 Machine learning.

Gender bias

In 2018, Amazon scrapped a hiring tool because it favoured male candidates. The hiring algorithm used historical data to develop the system. Since women traditionally held fewer STEM jobs, the data included a historical bias towards men. This resulted in female candidates being less likely to be offered roles they were equally or better qualified for than male candidates.

Racial bias

Many facial recognition systems are less effective with darker skin tones. This has often been because the algorithms have been trained exclusively with images of White people. If images of people of colour are not included in the training data, the algorithm does not know how to process this kind of data.

Is it possible for algorithms to be bias free, or is the bias of the programmer inevitable in the production of algorithms?

Selection statements in Java

```
1.  Scanner input = new Scanner(System.in);
2.
3.  System.out.println("What is your age? ");
4.  int myAge = input.nextInt();
5.
6.  if (myAge <= 5) {
7.    System.out.println("I recommend you go to the
      soft play centre");
8.  }
9.  else if (myAge > 5 && myAge <= 12  ) {
10.   System.out.println("I recommend a trip to the
      park");
11. }
12. else if (myAge > 12 && myAge <= 18) {
13.   System.out.println("I recommend meeting up with
      friends");
14. }
15. else {
16.   System.out.println("I recommend going to the
      movies");
17. }
```

Selection statements in Python

```
1.  myAge = int(input("What is your age? "))
2.
3.  if (myAge <= 5):
4.    print("I recommend the soft play centre")
5.
6.  elif(myAge > 5 and myAge <= 12 ):
7.    print ("I recommend a trip to the park")
8.
9.  elif (myAge > 12 and myAge <= 18):
10.   print ("I recommend meeting up with friends")
11.
12. else:
13.   print ("I recommend going to the movies")
```

You may have noticed the use of `&&` in Java and `and` in Python. These allow you to add more than one variable to the condition, for example, if it is sunny **and** it is warm. Use the notation || in Java and `or` in Python for "or"; for example, if it is sunny **or** it is warm. Use the notation `!` for "not" in both Java and Python; for example, if it is **not** raining. Examples of conditions using these operators are shown in Table 9.

Table 9 Selected operators

Operator	Description	Example

==	Checks for equality. Also works for strings in Python.	`variable x = 100`
		`x == 100`
		Output: True

>	Checks for greater than. Also works for strings in Python.	`variable x = 100`
		`x > 150`
		Output: False

<	Checks for less than. Also works for strings in Python.	`variable x = 100`
		`x < 150`
		Output: True

>=	Checks for greater than or equal to. Also works for strings in Python.	`variable x = 100`
		`x >= 100`
		Output: True

<=	Checks for less than or equal to. Also works for strings in Python.	`variable x = 100`
		`x <= 50`
		Output: False

Activity

You can make a text-based adventure game with `if` statements.

Give the user a question and two or three options. For example: "You walk into a banquet hall and you are faced with a talking wall. What do you do? Run the other way, talk back in a polite fashion, or challenge the wall to a riddle?" Use the user's answer and `if` statements to determine what happens next. Try coding a text-based adventure game.

TOK

Outliers in data sets are exceptions to normal data.

If you are developing a program about animals, you may want to store their height. The largest horse in the world was 2.13 m tall. You might not include this statistic when deciding the upper boundary for the height of a horse.

If you are developing a program about average salaries, you might choose to exclude data about multi-millionaires as their information would skew your data.

However, there are animals and people that do belong in these categories, and there could be reasons to include their data in your program.

When abstracting the information to develop selection statements, you must decide what data to keep and what data to ignore. What are the consequences of ignoring outliers in your data set?

Key term

Repetition Repeating sections of code, usually completed with a `for` loop or a `while` loop.

Operator	Description	Example

`!=`	Checks for not equal to. Also works for strings in Python.	`variable x = 100`
		`x != 50`
		Output: True

`.equals()`	Used to test equality of strings in Java.	`variable x = "chicken"`
		`x.equals("chicken")`
		Return: true

`.compareTo()`	Used to test if one string is less than (<0), equal to (0) or greater than (>0) another in Java.	`variable x = "chicken"`
		`x.compareTo("fox");`
		Return: >0

B2.3.3 Construct programs that utilize looping structures to perform repeated actions

Repetition allows you to repeat sections of code as many times as required. There are two types of loops you need to be aware of when programming: `while` loops and `for` loops.

`while` loops

A `while` loop repeats sections of code until a condition has been met. Real-life examples of `while` loops include playing music until the playlist ends or the user presses stop, a car moving forward while the driver has their foot on the accelerator (gas) pedal, or in gaming, continuing the level while you have lives left.

In programming, a `while` loop is used when a condition needs to be met but you do not know how many times the code needs to be repeated for that to happen. For example, a `while` loop allows you up to five attempts to enter a password correctly.

This program will run until the user enters the number 100. Note that you need an `if` statement inside the loop to determine whether the end condition is met or not.

While loop in Java

```
1.  Scanner input = new Scanner(System.in);
2.
3.  System.out.println("Enter a number");
4.  int guess = input.nextInt();
5.  boolean end = false;
6.
7.  while(!end) {
8.
9.    if(guess !=100) {
10.   System.out.println("you have not answered
      correctly");
11.   System.out.println("Enter a number");
12.   guess = input.nextInt();
13. }
14. else {
15.   end = true;
16. }
```

While loop in Python

```
1.  guess = int(input("enter a number"))
2.  end = False
3.
4.  while (end!=True):
5.    if (guess != 100):
6.      print("you have not answered correctly")
7.      guess = int(input("enter a number"))
8.    else:
9.      end = True
```

You can use a **while** loop to make a simple guessing game of "higher or lower?" The game works like this.

- User One enters a number.
- User Two guesses a number, trying to find the same number as User One.
- If they guess a higher number, print a message to tell them their number was too high.
- If they guess a lower number, print a message to tell them their number was too low.
- If they guess correctly, print "You have guessed correctly, well done!".
- Allow User Two five attempts to guess User One's number. If they do not guess the right number, print a message to say they did not guess correctly.

Guessing game in Java

```
1.  public static void main(String args[])
2.  {
3.    Scanner input = new Scanner (System.in);
4.    boolean end = false;
5.    boolean win = false;
6.    int number_of_lives = 5;
7.
8.    System.out.println("User One - What is the price
      of the item");
9.    int price = input.nextInt();
10.
11.   while (end == false) {
12.     System.out.println("User Two - please enter
        your guess");
13.     int guess = input.nextInt();
14.
15.     if(guess == price) {
16.       win = true;
17.       end = true;
18.     }
19.     else if (guess < price) {
20.       System.out.println("The guess is too low");
21.       number_of_lives = number_of_lives - 1;
22.     }
23.     else {
24.       System.out.println("The guess is too high");
25.       number_of_lives = number_of_lives - 1;
26.     }
27.
28.     if (number_of_lives == 0) {
29.       end = true;
30.     }
31.   }
32.   if (win == true) {
33.     System.out.println("Congratulations - You
        Win!");
34.   }
35.   else {
36.     System.out.println("Sorry you did not win this
        time");
37.   }
38. }
```

ATL Communication skills

Using programming to raise awareness

In your CAS activities, develop a game to help to engage your audience with your project and ideas. Simple games using repetition and selection statements are most effective. Games do not have to be complex to be engaging.

Write an idea for a game using your coding knowledge that will help young students (4–5 years old) understand when it is safe to cross the road.

Guessing game in Python

```
1.  end = False
2.  win = False
3.  number_of_lives = 5
4.
5.  price=input("UserOnepleaseenterthepriceoftheitem")
6.
7.  while (end == False):
8.    guess = input("User Two - Please enter your guess")
9.
10.   if (guess == price):
11.     win = True
12.     end = True
13.   elif (guess > price):
14.     print (guess, "is too high")
15.     number_of_lives = number_of_lives - 1
16.   elif (guess < price):
17.     print (guess, "is too low")
18.     number_of_lives = number_of_lives - 1
19.
20.   if (number_of_lives == 0):
21.     end = True
22.
23. if (win == True):
24.   print("Congratulations you win!!")
25. else:
26.   print("Sorry you did not win this time")
27. ,
```

To understand where loops can be used to control in the real world, refer to subtopic A1.3 Operating systems and control systems.

for loops

A **for** loop repeats the section of code for a set number of times. It could be used if something needs to be completed a set number of times. If you needed to make 20 pancakes, then you would repeat the cooking process 20 times. If you needed to calculate and display a times table, you may want to repeat the process 10 times.

For loop in Java

```
for(int i = 0; i < 11; i++){
  System.out.println(i + " * 10 = " + i*10);
}
```

Output:

0 * 10 = 0

1 * 10 = 10

2 * 10 = 20

3 * 10 = 30

4 * 10 = 40

5 * 10 = 50

6 * 10 = 60

7 * 10 = 70

8 * 10 = 80

9 * 10 = 90

10 * 10 = 100

For loop in Python

```
for i in range (0, 11):
  print (i*10)
```

Output:

0 * 10 = 0

1 * 10 = 10

2 * 10 = 20

3 * 10 = 30

4 * 10 = 40

5 * 10 = 50

6 * 10 = 60

7 * 10 = 70

8 * 10 = 80

9 * 10 = 90

10 * 10 = 100

Programmers use `for` loops to search, manipulate and display items within a list.

For loop in Java

```
1.  String [] team = new String []{"Heather", "Toni",
    "Francesca", "Ben", "Syed", "Wes", "Toni", "Ife",
    "Chidi", "Taro"};
2.
3.
4.  int count = 0;
5.
6.  for(int i = 0; i < team.length; i++) {
7.
8.    if (team[i].equals("Toni")){
9.      count = count + 1;
10.   }
11. }
12. System.out.println("The number of Tonis in the list
    is: " + count);
```

Output: The number of Tonis in the list is: 2

This program searches through the list using a linear search and counts the number of Tonis in the list.

For loop in Python

```
1.  team = ["Heather", "Toni", "Francesca", "Ben",
    "Syed", "Wes", "Toni", "Ife", "Chidi", "Taro"]
2.
3.  count = 0
4.
5.  for name in team:
6.    if name == "Toni":
7.      count = count + 1
8.
9.  print ("The number of Tonis in the list is: ", count)
```

Output: The number of Tonis in the list is: 2

This program searches through the list using a linear search and counts the number of Tonis in the list.

Practical skills

Repeating code within your programs allows you to develop more effective solutions.

Activity

You can make a pick up sticks game with loops. Usually, two users take turns removing 1, 2 or 3 sticks from a pile of 21 sticks. The loser is the person who runs out of sticks to pick. Try coding this.

B2.3.4 Construct functions and modularization

When you start programming, you make simple programs that are easy to read. As you become more competent, your programs will become more complex, containing many lines of code that become difficult to follow even with comments and meaningful names. This is when **modularization** becomes useful. For example, you could move a sequence of lines of code that calculate a bonus into a function or module called calculate function.

Key term

Modularization The process of separating lines of code that perform a task into different functions or modules. The same module can be used in many different programs.

There are several advantages to using modules.

Code is easier to read: Separating code into functions enables programmers to see immediately where the code is being used and how the code is functioning.

Code is easier to test: Functions and modules can be tested independently of the whole program, enabling them to be tested prior to the whole program being complete.

Code is easier to reuse: Functions and modules that perform tasks can be imported into other programs, which saves time and effort when coding new programs.

Code is easier to update: When the code is separated into functions and modules, updating the code is easier. As all of the code is contained within the function, any changes only need to happen within the function.

Here is an example of a function that returns the volume of a rectangle. The user inputs the height, width and depth of the rectangle. The function identifies the inputs by their parameters. The calculation is carried out and the resulting value, the volume of the rectangle, is sent back to the section of code that made the original call.



```
public static int rectangleVolume (int w, int h, int d){
// code not shown
}
```



```
def rectangleVolume(w, h, d):
# code not shown
```

A note about parameters

The function or method signature identifies the expected parameters. These are the values the function needs in order to function correctly.

The expected parameters are those inside the `( )` brackets. When you call this method or function from another class or from another method, you must include three values, in this case, integers. If you do not, then the function will not perform as expected.

The actual parameters are the real values we send into the method or function when it is called. These can either be values or variables holding the correct values.



```
int volume = rectangleVolume(width, height, depth); //
the variables representing the values
```

or

```
int volume = rectangleVolume (10, 20, 10):
```



```
volume = rectangleVolume(width, height, depth) # when the
variables represent the values
```

or

```
volume = rectangleVolume(10,20,10)
```

```
public static void main(String[] args) {

    Scanner input = new Scanner(System.in);
    System.out.println("Please enter the width");
    int width = input.nextInt();
    System.out.println("Please enter the height");
    int height = input.nextInt();
    System.out.println("Please enter the depth");
    int depth = input.nextInt();

    System.out.println(rectangleVolume(width, height, depth));
}
```

The program collects user input and stores the data in three variables.

The `rectangleVolume` method is called and the three variables are given in the method. The value is calculated in the method and returns the volume of the rectangle to print.

Integer will be returned.

Name of the method.

The method expects three integers to work.

```
public static int rectangleVolume(int w, int h, int d) {

    int volume = 0;
    volume = w * h * d;
    return volume;

}
```

This is a local variable. It only exists within the function.

A method has been created. The method tells us that an int will be returned and that it expects three integers to be able to complete the task.

▲ **Figure 10** Java parameters

Name of the method.

The method expects these parameters so it can work.

The function is defined and expects three variables to work correctly.

```
def rectangleVolume(w, h, d);
    volume = w * h * d;
    return volume;
```

Local variable only exists in this function.

Value returned to method from which it was called.

```
width = int(input("please enter the width"))
height = int(input("please enter the height"))
depth = int(input("please enter the depth"))
print(rectangleVolume(width, height, depth))
```

Data is collected from user and stored in three separate variables.

Data is sent to the method for the value to be calculated and then printed.

▲ **Figure 11** Python parameters

As outlined in Figures 10 and 11, local variables can only be accessed and used within the function where they are declared. Global variables are variables declared outside of a function. They can be accessed throughout the whole code.

Libraries are useful modules we can import into our code. Although they are not in the syllabus directly, they are useful to know. Libraries are pre-written, pre-tested pieces of code. You do not need to know how they work, only that they can save you time when developing a solution. To use a library, you normally need to import it before you can make use of any methods.

Figures 12 and 13 are examples of code that utilize the **random** library. The program will use a pseudorandom number to select a student from a list. A **random** library exists in both Java and Python.

```
import java.util.Random;
public class main {
  public static void main(String args[]{

    String [] myStudents = new String [] {"Angelina",
"Patrick", "Riley", "Leo", "Keani", "Dane", "Arham",
"Niko"};

    Random r = new Random();
    int choice = r.nextInt(myStudents.length-1);

    System.out.printIn("The chosen student is: " +
myStudents[choice]);
  }
}
```

Import the library.

Create instance of library ready for use.

Make use of the library method nextInt.

▲ **Figure 12** Random library in Java

```
import random

myStudents = ["Angelica", "Patrick", "Riley", "Leo",
"Keani", "Dane", "Arham", "Niko"]

print("The chosen student is: ",
random.choice(myStudents))
```

Import library for use.

Make use of library method.

▲ **Figure 13** Random library in Python

There are many libraries available for both Java and Python. Check the API documentation provided by Oracle and Python to see what is available for use. Table 10 gives a summary of some useful libraries.

Table 10 A summary of some useful libraries

Java library	Description
`Scanner`	Enables you to collect data from the console and read files.
`Random`	Enables you to generate pseudorandom numbers.
`time`	Enables you to make use of dates.
`math`	Enables you to make use of mathematical functions.

Python library	Description
`Random`	Enables you to generate pseudorandom numbers.
`OS`	Enables you to access the operating system.
`datetime`	Enables you to make use of dates.
`math`	Enables you to make use of mathematical functions.

Activity

In the game "Snake eyes", the user rolls two dice until each die shows the number one—this is called "snake eyes". The person that rolls "snake eyes" with the fewest throws wins. Try implementing a version of this in code.

Be careful **not** to give your programs the same name as a library. For example, do not call a file "random", because your computer will try to run that program rather than search the library.

Practice questions

9. Explain the difference between "and" and "or" in an `if` statement. [3 marks]
10. Outline two advantages of using a library. [4 marks]
11. A programmer is developing a program that will calculate the area of a circle using a function or method. Construct the code to calculate the area of the circle. [4 marks]
12. A programmer is developing a "Who am I?" game. The user is given five clues and they must guess who the celebrity is. The game ends when the celebrity's name is guessed.

    Construct an algorithm for this game. [5 marks]

Libraries make use of encapsulation. You can find out more about this in topic B3 Object-oriented programming.

TOK

Specific knowledge often belongs to a small group of users.

In Tanzania, farmers operate a unique rotational farming system using ridges and pits to prevent the destruction of arable land.

Pacific island communities use unique marine resource management techniques to protect the ecosystem of their islands.

Research scientists have specialized knowledge about the development of medications to aid specific illnesses, which are often kept secret until they are ready for market. When the new medications are available for sale, their formulas are kept secret, with only a few people knowing how they function.

Does the knowledge of producing code—for example, code in libraries or code you produce—only belong in the programming community?

B2.4 Programming algorithms

Syllabus understandings

B2.4.1 Describe the efficiency of specific algorithms by calculating their Big O notation to analyse their scalability

B2.4.2 Construct and trace algorithms to implement a linear search and a binary search for data retrieval

B2.4.3 Construct and trace algorithms to implement bubble sort and selection sort, evaluating their time and space complexities

AHL **B2.4.4** Explain the fundamental concept of recursion and its applications in programming

AHL **B2.4.5** Construct and trace recursive algorithms in a programming language

B2.4.1 Describe the efficiency of specific algorithms by calculating their Big O notation to analyse their scalability

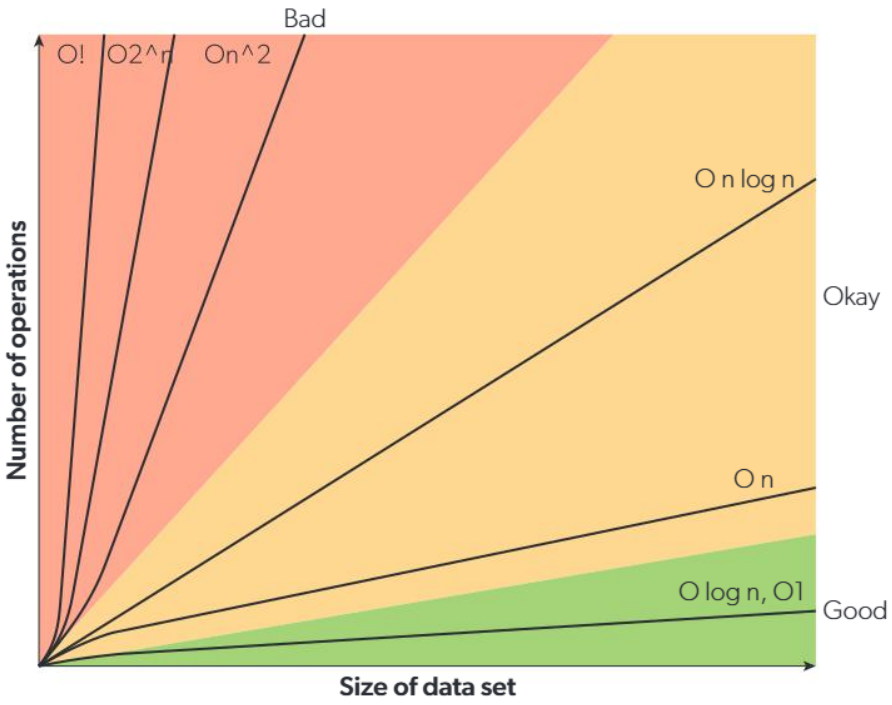
Programming is both about developing solutions and then evaluating solutions for their efficiency. One way we can determine the efficiency of algorithms is by analysing their **Big O efficiency**. This is a way to determine how well the program would function if it were bigger.

Big O notation and its effect on an algorithm is summarized in Figure 14.

Key term

Big O efficiency Measures the upper bound of an algorithm's efficiency, focusing on the worst-case scenario. It is one way to theoretically measure the efficiency of an algorithm in terms of number of operations required to complete a task.

To understand why it is important to consider algorithm efficiency think about the complex algorithms in subtopic A4.3 Machine learning approaches.



▲ **Figure 14** Big O efficiency for different algorithms

Big O values, descriptions and examples of them are shown in Table 11.

Table 11 Big O values, descriptions and examples

Big O	Notation	Description	Examples
Constant	O(1)	The number of operations required to complete the task will be the same regardless of the input size.	The `pop()` and `push()` methods for a stack. Adding to the end of a dynamic data structure. Accessing an element of a data structure using an index.
Logarithmic	O(log n)	The size of the data reduces by half each iteration of the operation. When the input size decreases each iteration it is said to have logarithmic time.	A binary search tree is a typical example of logarithmic Big O complexity as the search values decrease by half each iteration of the algorithm.
Linear	O(n)	Linear time complexity is when the time complexity increases linearly with the size of the input.	An example of this would be using a `for` loop to iterate through a data structure. The time it takes increases directly in line with the number of elements in the data structure.
Log Linear	O(n log n)	Log linear is slower than logarithmic. The size of the data does not decrease by half each iteration but it does decrease.	A quicksort traverses the array to find the pivot point, divides the array and swaps elements until sorted. Each iteration does not deal with the full array and does not lose half each time, but the size does decrease.
Quadratic	O(n^2)	Quadratic algorithms have loops within loops. This means that for each iteration of the outer loop a full iteration of the inner loop must be carried out. Adding to the data set exponentially increases the complexity.	Common quadratic complexity is present in the following sorting algorithms: bubble sort, selection sort and insertion sort. In these sorts, a double loop is present and therefore the complexities are quadratic.
Cubic	O(n^3)	Cubic algorithms have three nested loops. For each outer loop iteration the inner loop must be fully iterated but for each iteration of the first inner loop the second loop must be fully iterated.	Cubic complexity will only occur when you have three nested loops.
Exponential	O(c^n)	Exponential complexity is one of the worst complexities. The exponential complexity is constant to the power of the exponent.	Identifying the number of different pairings of students in a class. Assume the class has 20 students (constant 20) and the students have to be in pairs (exponent 2). Possible pairings = 400 different combinations. If the students were then put into trios (exponent 3), possible pairings = 8,000 different combinations. Real-world examples include brute force algorithms; calculating the Fibonacci sequence.
Factorial	O(n!)	Factorial algorithms increase factorially each time a new element is added to the data set. For example, if you wanted to find all the different permutations of an array, if you have three elements you have six permutations; four elements, 24 permutations. The same as factorial growth.	The travelling salesman problem (determining the most efficient way to visit each item on a route) is a factorial problem.

Calculating Big O complexity

To calculate Big O complexity, consider how the number of operations of the algorithm changes with the size of the input. Follow these steps.

1 Count the number of operations. Identify the main operations of the algorithm and use these as a base for your calculation as you scale up the data set.

2 Ignore anything that is constant or outside the method. For example, printing the result of the method (this only happens once) or assigning a variable (for example, `minIndex = 0`, which happens outside of the method). These do not change once the method begins.

3 Change the input size and count the number of operations. If the number has not changed, the complexity is O(1). If it has changed proportionally (linearly), the complexity is O(n). If it has changed quadratically, the complexity is O(n^2).

Worked example 1

Work out the Big O for this function.



```
1.    public static void main(String[] args) {
2.
3.        System.out.println(functionPercent(25,100));
4.
5.    }
6.    public static double functionPercent(double X, double Y) {
7.
8.        double result = X/Y * 100;
9.        return result;
10.
11.   }
```



```
1.    def calculatePercent(X, Y):
2.        percent = X/Y * 100
3.        return percent
4.
5.    print (calculatePercent(25,100))
```

Solution

Examine the function and follow these steps.

Step 1: Count the number of operations in the algorithm

Divide *x* by *y*		1
Multiply by 100		1
Total	=	2

Step 2: Consider the number of times the algorithm will run. This algorithm will only run when called as there are no loops. This means that each operation will only be counted once.

Step 3: As the number of steps in this algorithm never changes no matter how many times the method is run, the number of steps is constant. If the number of steps is constant, the algorithm has O(1) complexity.

Worked example 2

Work out the Big O for this function.

```java
public static void main(String[] args){

    int [] values = {2,5, 12, 56, 345, 2, 453, 6, 86, 765, 234, 123,1};
    System.out.println(findMax(values));
}
public static int findMax(int [] values) {

    int Highest = values[0];
    for (int i = 0; i < values.length; i++) {

      if (values[i]>Highest) {
        Highest = values[i];
      }
    }
    return Highest;

}
```

```python
values = [2,5,12,56,345,2,453,6,86,765,234,123,1]

def findMax():
    highest = values[0]

    for x in values:
      if x > highest:
        highest = x

    return highest

print(findMax())
```

Solution

Step 1: Count the number of operations in the algorithm

Assignment of *i*	1
Comparison of *i*	1
Incrementation of *i*	1
Comparison of current value and highest	1
Reassignment	1
Total =	5

Step 2: Consider the number of times the algorithm will run. This algorithm has a loop, so it will run the same number of times each loop. You must count the number of operations each time the loop is completed. If the loop executes 1 time there are 5 operations. If the loop executes 25 times, the number of operations is 125.

Step 3: The number of operations increases in direct proportion to the number of iterations or loop executions:
1 execution = 5 operations
25 executions = 125 operations
500 executions = 2500 operations.
This is a linear complexity: the number of executions is always multiplied by the same number to give the number of operations. This has complexity O(n).

ATL Thinking skills

Applying knowledge of Big O

Using Big O to calculate the scalability of algorithms allows you to become more knowledgeable and considerate about the solutions you produce. It will allow you to effectively evaluate the products you produce, encouraging you to consider how to make improvements.

For two of the programs you have previously developed, calculate the Big O efficiency of the algorithms you have used.

B2.4.2 Construct and trace algorithms to implement a linear search and binary search for data retrieval

Key term

Search To find data within a data structure.

Searching for data within a data set is an essential tool in programming. When you place items into a data set, you need to be able to find them again. Searches are used all the time in programs. You search music software to find artists you want to listen to. You search a streaming service to find a list of television shows or movies you want to watch. In programming, you can search for one or more items on a list, to find and use the related information.

You saw in the previous section of this chapter that the efficiency of your search algorithm matters when you have large data sets. Now compare two different search algorithms: the **linear search algorithm** and the **binary search algorithm**.

The linear search algorithm

The linear search algorithm looks at each item in the data set to find information.

Consider a list of songs contained in a playlist, as in Table 12. To look for the song "Little Red Lies", start at the beginning, look at each item in the list and—when you find the song—make a note of where you find it.

Table 12 Playlist

Index	[0]	[1]	[2]	[3]	[4]	[5]
Song	The Old Bridge	Down by the Fountain	Super Searchin'	Little Red Lies	Geescht	Watching Over You



Practical skills

When dealing with data sets, searching is an essential skill.

1 Set the found index to –1 (this means the song has not yet been found).

2 Start at the beginning on the list index 0.

3 If song name in this element matches the search song "Little Red Lies"
    set the index to the current index
  else
    move onto next index.

The code for this linear search is shown below.



Linear search in Java

```
1.  Scanner input = new Scanner(System.in);
2.  String [] playlist = new String [] {"The Old
    Bridge", "Down by the Fountain", "Super Searchin'",
    "Little Red Lies", "Geescht",  "Watching Over
    You"};
3.
4.
5.  int foundIndex = -1;
6.  System.out.println("What song are you looking for?");
7.  String searchTerm = input.nextLine();
8.
9.  for(int i = 0; i < playlist.length; i++) {
10.
11.     if (playlist[i].equals(searchTerm)) {
12.         foundIndex = i;
13.     }
14. }
15. if (foundIndex != -1) {
16.     System.out.println("The song was found at " +
        foundIndex);
17. }
18. else {
19.     System.out.println("The song was not found.");
20. }
```

Linear search in Python

```
1.  playlist = {"The Old Bridge", "Down by the
    Fountain", "Super Searchin'", "Little Red Lies",
    "Geescht", "Watching Over You"}
2.
3.  foundIndex = -1
4.  searchTerm = input("What song are you looking for?
")
5.
6.  for name in playlist:
7.    if (name == searchTerm):
8.      foundIndex = name.index
9.
10. if (foundIndex != -1):
11.   print("The song was found at " + str(foundIndex))
12. else:
13.   print("The song was not found")
```

Activity

When you are playing a game, it is often useful to know if you have an item in an inventory. Develop a program that stores an inventory of items and allows the user to search for the item. Display an error message if the user searches for an item they do not have in their inventory.

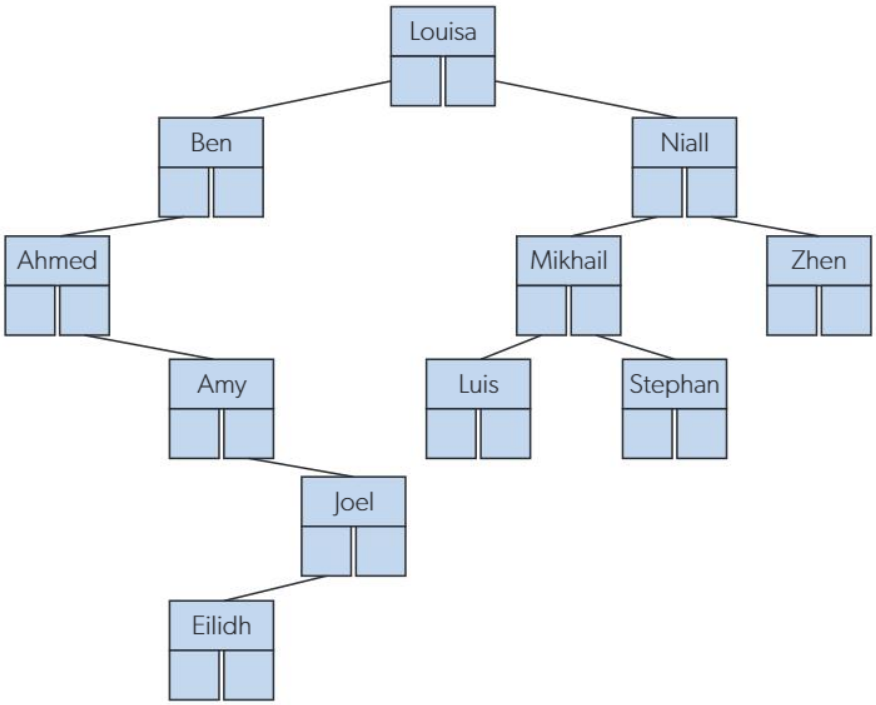
If you add more elements to the array, the number of operations will increase linearly with the number of elements added. The efficiency of this algorithm is described as O(n), which means it better for smaller data sets.

The binary search algorithm

For a binary search, the data set must be sorted. The root is the top of the data set. Everything lower (in value) than the root is placed to the left of the root. Everything higher (in value) than the root is placed to the right of the root.

Consider the following data set:
Louisa, Niall, Ben, Ahmed, Zhen, Mikhail, Amy, Joel, Stephan, Luis, Eilidh

Louisa will become the root and then each of the items will be added in the order as described above. The resultant tree can be represented logically as shown in Figure 15.



▲ **Figure 15** Binary tree diagram

To find if "Amy" exists in the data set you could follow the algorithm shown below.

1 Set the found variable to false (as in the beginning the term has not been found).

2 Set the root to the first root in the tree (Louisa) and make the comparison.

3 If current node (Louisa) is equal to the search term (Amy)

set found to true

else if current node (Louisa) is < Amy alphabetically

set first node on the left to current node

else

set first node on the right to the current node.

4 Repeat until the end of the data set is reached or data is found.

Binary search in Java

```
1.  public int binarySearch(int searchArray[], int search)
2.  {
3.    int left = 0, right =searchArray.length - 1;
4.    while (left <= right) {
5.      int mid = left + (right - left) / 2;
6.      // Check if the search term is present at the
        midpoint
7.      if (searchArray[mid] == search)
8.        return mid;
```

```
9.      // If search is greater than the midpoint,
        ignore left half
10.     if (searchArray[mid] < search)
11.       left = mid + 1;
12.     // If search is smaller than the midpoint,
        ignore right half
13.     else
14.       right = mid - 1;
15.   }
16.   // if not found return
17.   return -1;
18. }
19. //Main method
20. public static void main(String args[])
21. {
22.   main binarySearch = new main();
23.   int searchArray[] = { 10, 11, 12, 15, 20,25, 45,
      84, 129, 139, 483, 586, 685 };
24.   int search = 139;
25.   int result = binarySearch.
      binarySearch(searchArray, search);
26.   if (result == -1)
27.     System.out.println("The number is not in the
        Array");
28.
29.   else
30.     System.out.println("The number is in the array
        at index:   " + result);
31. }
```

Binary search in Python

```
1.  def binarySearch (searchArray, left, right,
    searchTerm):
2.    while left < right:
3.      midPoint = left + (right - left) // 2
4.      #check if searchTerm is present at the midPoint
5.      if searchArray[midPoint] == searchTerm:
6.        return midPoint
7.      #if searchTerm is greater than midPoint, ignore
        the left half
8.      elif searchArray[midPoint] < searchTerm:
9.        left = midPoint + 1
```

```
10.     #if searchTerm is smaller than midPoint, ignore
        the right half
11.     else:
12.       right = midPoint -1
13.
14. # if the searchTerm is not found return -1
15.   return -1
16.
17. searchArray = [10, 11, 12, 15, 20, 25, 45, 84, 129,
    139, 483, 586, 685]
18. searchTerm = 139
19.
20. result = binarySearch(searchArray, 0,
    len(searchArray)-1, searchTerm)
21.
22. if result != -1:
23.   print("Element is present at index " +
      str(result))
24. else:
25.   print("Element is not present in the array")
```

If you add more elements to the data set, the number of operations does not increase linearly. You lose half of the data set each time you make a comparison. So, the efficiency of this algorithm is O(log n). Binary searches are more efficient for larger data sets. You can find the data quicker as the data is already sorted (indexed). To search a large set of data—for example, all the items in a store—it is quicker to use a binary search as you halve the number of items to view with every comparison made. This makes it quicker to find the data.

B2.4.3 Construct and trace algorithms to implement bubble sort and selection sort, evaluating their time and space complexities

In a game, the leaderboard must have the leader at the top and the rest of the competitors in order after the leader. When a new person has a score good enough to be added to the leaderboard, they need to be added to the leaderboard in the correct space. This is known as **sorting**.

You need to be able to search for data and then put it in the correct order. There are many sorting algorithms available for programmers to use, from the highly efficient quick sort, with an efficiency of O(n log n), and merge sort, also with an efficiency of O(n log n), to the very inefficient logo sort O(n * n!) or gnome sort O(n^2).

The two sorts you need to be aware of for this course are **bubble sort** O(n^2) and **selection sort** O(n^2).

Key terms

Sort To place data in the correct order in a data structure.

Bubble sort A sort that compares pairs of elements and, if they are in the wrong order, swaps them. Although simple to implement this algorithm is inefficient on large data sets.

Selection sort A sort that utilizes indexing to sort data into the correct order. The starting value is compared to all other values, if a higher value is found the highest index is changed. At the end of the pass the value is moved into the correct space.

The bubble sort algorithm

The bubble sort algorithm uses pairs to check if the next element in the list is larger than the current element (assuming you are sorting lowest to highest). If the items are in the wrong order then they swap.

This is demonstrated below.

Unsorted array

10	8	349	3	39	1

Start of Pass One: Swap One

8	10	349	3	39	1

Swap Two

8	10	349	3	39	1

Swap Three

10	8	3	349	39	1

Swap Four

10	8	3	39	349	1

Swap Five

10	8	3	39	1	349

End of Pass One: 349 is in the correct spot

10	8	3	39	1	349

Start of Pass Two: Swap One

8	10	3	39	1	349

Swap Two

8	3	10	39	1	349

Swap Three

8	3	10	39	1	349

Swap Four

8	3	10	1	39	349

End of Pass Two: 39 is in the correct spot

8	3	10	1	39	349

Start of Pass Three: Swap One

3	8	10	1	39	349

Swap Two

3	8	10	1	39	349

Swap Three

3	8	1	10	39	349

End of Pass Three: 10 is in the correct spot

3	8	1	10	39	349

Start of Pass Four: Swap One

3	8	1	10	39	349

Swap Two

3	1	8	10	39	349

End of Pass Four: 8 is in the correct spot

3	1	8	10	39	349

Start of Pass Five: Swap One

1	3	8	10	39	349

End of Pass Five: 3 is in the correct spot

1	3	8	10	39	349

End of outer loop: all elements in the array are sorted

1	3	8	10	39	349

This is the algorithm for the bubble sort.

Bubble sort in Java

```
1.  int [] toSort = {4, 56, 23, 2938, 29, 485, 2, 394,
    58, 3838, 485, 9822};
2.  int temp;
3.
4.  for (int i = 0; i < toSort.length-1; i++) {
5.     for (int j = 0; j < toSort.length - i - 1; j++)
{
6.
7.        if(toSort[j] > toSort[j+1]) {
8.          temp = toSort[j];
9.          toSort[j] = toSort[j+1];
10.         toSort[j+1] = temp;
11.       }
12.    }
```

```
13. }
14. for (int i = 0; i < toSort.length; i++) {
15.       System.out.println(toSort[i]);
16. }
```

Bubble sort in Python

```
1.  toSort = [4, 56, 23, 2938, 29, 485, 2, 394, 58,
    3838, 485, 9822]
2.
3.  for i in range(0, len(toSort)):
4.    for j in range(0, len(toSort)- i -1):
5.      if toSort[j] > toSort[j+1]:
6.        temp = toSort[j]
7.        toSort[j] = toSort[j+1]
8.        toSort[j+1] = temp
9.
10. print(toSort)
```

Advantages of bubble sort	Disadvantages of bubble sort
• Easy to implement. • The elements are swapped in place: no extra memory requirements are necessary. • Minimal additional space required.	• Very inefficient for large data sets due to the number of swaps.

The selection sort algorithm

The selection sort algorithm uses a variable to track the index of the highest value element (sorting lowest to highest). At the end of the pass, the highest value is moved into the correct space.

This is demonstrated below.

Unsorted array

Highest Value = 10 // initial variable value

Highest Index = 0 // initial index value

10	8	349	3	39	1

Pass One: Search for the highest value to place in last element of array

Highest Value = 349

Highest Index = 2

Last element and highest index :2 swapped and values reset

10	8	1	3	39	349

Pass Two

Highest Value = 39

Highest Index = 4

Last element: 1 and highest index swapped and values reset

10	8	1	3	39	349

Pass Three

Highest Value = 10

Highest Index = 0

Last Element: 2 and highest index swapped and values reset

3	8	1	10	39	349

Pass Four

Highest Value = 8

Highest Index = 2

Last Element: 3 and highest index swapped and values reset

3	1	8	10	39	349

Pass Five

Highest Value = 3

Highest Index = 0

Last Element: 4 and highest index swapped and values reset

1	3	8	10	39	349

Outer loop ends and all elements are in the correct space

1	3	8	10	39	349

Here is the algorithm for the selection sort.



Selection sort in Java

```
1.  int [] toSort = {4, 56, 23, 2938, 29, 485, 2, 394,
    58, 3838, 485, 9822};
2.  int temp;
3.
4.  for (int i = 0; i < toSort.length - 1; i++) {
5.      int highestIndex = i;
6.      for (int j = i+1; j< toSort.length; j++) {
7.          if (toSort[j]< toSort[highestIndex]) {
8.              highestIndex = j;
9.           }
```



```
10.     }
11.     temp = toSort[highestIndex];
12.     toSort[highestIndex] = toSort[i];
13.     toSort[i] = temp;
14. }
15. for (int i = 0; i < toSort.length; i++) {
16.     System.out.println(toSort[i]) + " ");
17. }
```

Selection sort in Python

```
1.  toSort = [4, 56, 23, 2938, 29, 485, 2, 394, 58,
    3838, 485, 9822]
2.
3.  for i in range(0, len(toSort)-1):
4.      highestIndex = i
5.      for j in range(i + 1, len(toSort)):
6.          if (toSort[j]<toSort[highestIndex]):
7.              highestIndex = j
8.
9.      temp = toSort[highestIndex]
10.     toSort[highestIndex] = toSort[i]
11.     toSort[i] = temp
12.
13. print(toSort)
```

Advantages of selection sort	Disadvantages of selection sort
• Works well on small data sets. • No additional storage needed as elements swap in place. • The number of swaps required is minimized, compared with the bubble sort.	• Inefficient for large data sets.

Worked example 3

A local café is open seven days per week. It is considering closing for one or two days each week to save costs. How can it find out which days make the most money? Write a program to help the café work out which days to close.

Solution

Use a selection sort to sort the days into order of lowest sales to highest sales.

1. Create two lists, one containing the value of sales each day (named **dailySales**) and one containing the days of the week (named days).
2. Use the selection sort to sort the values in order from lowest to highest.
3. Use the **min_index** to complete the swap in both the **dailySales** list and the days list.
4. Use a **for** loop to print out the data in the correct order making it easier for the owner to compare days.



Java

```
1.    public static void main(String[] args) {
2.
3.      int [] dailySales = {324, 123, 100, 210, 378, 435, 345};
4.      String [] day = {"Sunday", "Monday", "Tuesday", "Wednesday", "Thursday",
        "Friday", "Saturday"};
5.
6.      for (int i = 0; i < dailySales.length - 1; i++) {
7.
8.        int min_index = i;
9.
10.       for (int j = i+1; j<dailySales.length; j++) {
11.         if (dailySales[j]< dailySales[min_index]) {
12.           min_index = j;
13.         }
14.       }
15.
16.       int temp = dailySales[min_index];
17.       dailySales[min_index] = dailySales[i];
18.       dailySales[i] = temp;
19.
20.       String dTemp = day[min_index];
21.       day[min_index] = day[i];
22.       day[i] = dTemp;
23.     }
24.
25.     for(int i = 0; i < dailySales.length; i++) {
```

```
        System.out.println("Day: " + day[i] + ", Sales: " + dailySales[i]);
    }
  }
```

Python

```
dailySales = [324, 123, 100, 210, 378, 435, 345]
day = ["Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday",
"Saturday"]

for i in range(0, len(dailySales)-1):
    min_index = i
    for j in range(i + 1, len(dailySales)):
      if (dailySales[j]<dailySales[min_index]):
        min_index = j

    dailySales[i], dailySales[min_index] = dailySales[min_index], dailySales[i]

for index in range (0, len(dailySales)):
      print("Day: ", day[index] , ", Value: " , dailySales[index])
```

Activity

Develop a leaderboard. When someone completes a level their score gets added to the leaderboard. The leaderboard is then sorted to show all scores in descending order.

Practical skills

Sorts are particularly useful when developing programs that use leaderboards and require the ranking of data.

Other useful algorithms

Here is a selection of other algorithms that you may find useful.

Summing

This will add all the values together in a list. Useful for if you are trying to total together sales, or add together the value of items.

Summing in Java

```
1.  int [] data = new int [] {1, 20, 39, 4, 12, 20, 4,
    34, 2, 29, 20, 5, 66};
2.  int sum = 0;
3.  for (int i = 0; i < data.length; i++) {
4.    sum = sum + data[i];
5.  }
6.  System.out.println("The sum of all values is " + sum);
```

Summing in Python

```
1.  data = [1, 20, 39, 4, 12, 20, 4, 34, 2, 29, 20, 5, 66]
2.  sum = 0
3.  for i in range (0, len(data)):
4.    sum = sum + data[i]
5.
6.  print("The sum of all values is ", sum)
```

Alternatively:

```
1.  data = [1, 20, 39, 4, 12, 20, 4, 34, 2, 29, 20, 5, 66]
2.  sum = sum(data)
3.  print("The sum of all values is ", sum)
```

Averaging

This will enable you to find the average of the values in a list.

Averaging in Java

```
1.  int [] data = new int [] {1, 20, 39, 4, 12, 20, 4,
    34, 2, 29, 20, 5, 66};
2.  int sum = 0;
3.  for (int i = 0; i < data.length; i++) {
4.      sum = sum + data[i];
5.  }
6.  System.out.println("The average of all values is "
    + sum/data.length);
```

Averaging in Python

```
1.  data = [1, 20, 39, 4, 12, 20, 4, 34, 2, 29, 20, 5, 66]
2.  sum = 0
3.  for i in range (0, len(data)):
4.    sum = sum + data[i]
5.
6.  print("The average of all values is ", sum/
    len(data))
```

Alternatively:

```
1.  data = [1, 20, 39, 4, 12, 20, 4, 34, 2, 29, 20, 5, 66]
2.  sum = sum(data)
3.  print("The average of all values is ", sum/
    len(data))
```

Count occurrences

Use this to count the number of times something features in a list. For example, you may want to count the number of people under 18 in a list or count the number of cars in a transporter.

Count occurrences in Java

```
1.  int [] data = new int [] {1, 20, 39, 4, 12, 20, 4,
    34, 2, 29, 20, 5, 66};
2.  int count = 0;
3.  int searchValue = 20;
4.
5.  for (int i = 0; i < data.length; i++) {
6.    if (data[i] == searchValue) {
7.      count = count + 1;
8.    }
9.  }
10. System.out.println("The number of times the value
    " + searchValue + " appears is: " + count);
```

Count occurrences in Python

```
1.  data = [1, 20, 39, 4, 12, 20, 4, 34, 2, 29, 20, 5, 66]
2.  count = 0
3.  searchTerm = 20
4.  for i in range(0, len(data)):
5.    if (data[i] == searchTerm):
6.      count = count + 1
7.
8.  print("The number of times ", searchTerm, " appears
    is " , count)
```

Finding the maximum or minimum

Use this to find the maximum or minimum value in a list. For example, you may want to find the worst test score or the oldest person. The code below returns the value and the index (in case you want to access the value directly).

Find the minimum in Java

```
1.  int [] data = new int [] {1, 20, 39, 4, 12, 20, 4,
    34, 2, 29, 20, 5, 66};
2.  int maximumValue = data[0];
3.  int maximumIndex = 0;
4.
5.  for (int i = 0; i < data.length; i++) {
6.    if (data[i] > maximumValue) {
7.      maximumValue = data[i];
8.      maximumIndex = i;
9.    }
10. }
11. System.out.println("The maximum value is " +
    maximumValue + " and appears at: " + maximumIndex);
```

Find the minimum in Python

```
1.  data = [1, 20, 39, 4, 12, 20, 4, 34, 2, 29, 20, 5, 66]
2.  maximumValue = data[0]
3.  maximumIndex = 0
4.  for i in range(0, len(data)):
5.    if (data[i] > maximumValue):
6.      maximumValue = data[i]
7.      maximumIndex = i
8.
9.  print("The maximum value is", maximumValue, "and is
    at index " , maximumIndex)
```

Alternatively:

```
1.  data = [1, 20, 39, 4, 12, 20, 4, 34, 2, 29, 20, 5, 66]
2.  maximumValue = max(data)
3.  print("The maximum value is", maximumValue)
```

AHL

Key term

Recursion An algorithm that calls itself with updated parameters until the base case is met.

B2.4.4 Explain the fundamental concept of recursion and its applications in programming

Imagine you have arrived in a strange city and you need to find your way to a specific address. You might ask someone how to get to the address, follow their instructions, and then ask someone else, continuing this process until you reach your destination. This could be represented in an algorithm like this:

direction (current location)

1 if at the address end the algorithm

2 else
  2.1 Ask for directions
  2.2 Follow the remembered directions
  2.3 Call *direction* (updated location)

Recursive algorithms

A **recursive** algorithm is an algorithm that calls itself with updated parameters until the base case is reached.

A classic example of a recursive algorithm is the Fibonacci sequence.

TOK

Inductive reasoning requires the observer to make a specific observation and then apply this to make general observations. For example:

"I saw puffins on a rocky cliff."

"My friends also saw puffins on rocky cliffs."

"All puffins live on rocky cliffs."

Recursion starts with a base case and then derives all additional values from this.

To what extent is recursion based on inductive reasoning?



▲ **Figure 16** Inductive reasoning involves generalizations derived from observations. For example, drawing conclusions about all puffins (a species of seabird) from a sample

Fibonacci in Java

```
1.  public static int fibonacci (int number) {
2.    if ((number == 0) || (number == 1)){
3.      return number;
4.    }
5.    else {
6.      return fibonacci (number - 1 ) + fibonacci
        (number - 2);
7.    }
8.  }
```

Fibonacci in Python

```
1.  def fibonacci (num):
2.    if (num == 0) or (num == 1):
3.      return num
4.    else:
5.      return fibonacci (num - 1) + fibonacci (num - 2)
6.
7.  print (fibonacci(10))
```

If the number passed into the method or function is not 0 or 1 (the base case), the method is called again with the parameter updated.

Reasons why you would use a recursive algorithm	Reasons why you would not use a recursive algorithm
• Recursive algorithms allow you to break down large, complicated problems into smaller, more manageable tasks. • A recursive solution, when appropriate, is often simpler and more elegant than an iterative solution.	• They contain a memory overhead as a stack is required to store the recursive calls. • If you do not have a good base case you may get into an infinite recursive loop which would create a stack overflow and crash the computer.

Another famous recursive algorithm is the Towers Of Hanoi, invented by French mathematician Édouard Lucas. Although toy versions only have 7 or 8 discs, there is a myth that if someone solved a version of the puzzle with 64 discs, the world would end. That's unlikely, but if you moved 1 disc per second, this would take 585 billion years.

AHL

One of the key algorithms to know that uses recursion is the recursive version of a binary search. As you can see, instead of being in a `while` loop, the algorithm makes use of recursive calls until the search term is found or there are no more elements to search:



Binary search in Java

```
1.  public int binarySearch (int [] searchArray, int
    left, int right, int searchTerm){
2.    if (left > right) {
3.      return -1;
4.    }
5.    int midPoint = (left + right)/ 2;
6.
7.    // If the element is present at the midPoint
8.    if (searchArray[midPoint] == searchTerm) {
9.      return midPoint;
10.   }
11.   // If element is smaller than mid, then ignore
      right subarray
12.   else if (searchArray[midPoint]>searchTerm) {
13.     return binarySearch(searchArray, left, midPoint
        - 1, searchTerm);
14.   }
15.   // Else the element is greater than mid, then
      ignore left subarray
16.   else {
17.     return binarySearch(searchArray, midPoint + 1,
        right, searchTerm);
18.   }
19. }
20. //Main method
21. public static void main(String args[])
22. {
23.   main binarySearch = new main();
24.   int searchArray[] = { 10, 11, 12, 15, 20, 25, 45,
      84, 129, 139, 483, 586, 685 };
25.   int search = 11;
26.   int result = binarySearch.
      binarySearch(searchArray,0, searchArray.length-1,
      search);
27.   if (result == -1)
```

```
28.     System.out.println("The number is not in the
        Array");
29.   else
30.     System.out.println("The number is in the array
        at index:   " + result);
31. }
```

Binary search in Python

```
1.  def binarySearchIternative (searchArray, left,
    right, searchTerm):
2.    while left < right:
3.      midPoint = left + (right - left) // 2
4.
5.      #check if searchTerm is present at the midPoint
6.      if searchArray[midPoint] == searchTerm:
7.        return midPoint
8.      #if searchTerm is greater than midPoint, ignore
        the left half
9.      elif searchArray[midPoint] < searchTerm:
10.       left = midPoint + 1
11.     #if searchTerm is smaller than midPoint, ignore
        the right half
12.     else:
13.       right = midPoint - 1
14.
15. # if the searchTerm is not found return -1
16.   return -1
17.
18. def binarySearch(searchArray, left, right,
    searchTerm):
19.   if (left > right):
20.     return -1
21.
22.   midPoint = left + (right - left) // 2
23.
24.   #check if searchTerm is present at the midPoint
25.   if searchArray[midPoint] == searchTerm:
```

```
26.     return midPoint
27.   #if searchTerm is greater than midPoint, ignore
      the left half
28.   elif searchArray[midPoint] > searchTerm:
29.     return binarySearch(searchArray, left, midPoint
        - 1, searchTerm)
30.   #if searchTerm is smaller than midPoint, ignore
      the right half
31.   else:
32.     return binarySearch(searchArray, midPoint + 1,
        right, searchTerm)
33.
34. searchArray = [10, 11, 12, 15, 20, 25, 45, 84, 129,
    139, 483, 586, 685]
35. searchTerm = 11
36. result = binarySearch(searchArray, 0,
    len(searchArray)-1, searchTerm)
37.
38. if result != -1:
39.   print("Element is present at index " +
      str(result))
40. else:
41.   print("Element is not present in the array")
```

Quicksort

Bubble sort and selection sort both have the same Big O efficiency. Recursion enables us to investigate another type of sorting algorithm: **quicksort**.

Quicksort makes use of a pivot point in the array. This can be any element in the array but commonly makes use of the last element.

Here is an outline of how the algorithm functions:

1. Choose an array element to be the pivot element.
2. Order the rest of the array so lower values are to the left of the pivot and higher values to the right.
3. Swap the pivot element with the first element of the higher values so it is between the two.
4. Repeat the operation until all elements are sorted (this is completed recursively).

Key term

Quicksort A sort that uses a pivot point and orders values compared to the pivot point, one side higher and one side lower. The pivot point is then changed and the process repeated until the data is sorted.

This is demonstrated below.

Unsorted array

Pivot point set to 1 as the last element in the array

10	8	349	3	39	1

Pass One: Search for the highest value to place in last element of array

All elements in the array are higher than 1 so must be on the right side of 1. Therefore swap 1 and 10

1 is in the correct spot; the pivot point is set as 10

1	8	349	3	39	10

Pass Two: Value 349 must be to the right of 10 and 3 to the left, so 349 and 3 swap

10 remains the pivot point

1	8	3	349	39	10

Pass Three: 10 must be between 3 and 349, so these elements are swapped

349 is now in the correct place

1	8	3	10	39	349

Pass Four: As 1 and 349 are in the correct place, choose 10 as the pivot point

8 and 3 are to the correct side of 10, as is 39, therefore 10 and 39 are in the correct spot

1	8	3	10	39	349

Pass Five: As 1, 10, 39 and 349 are in the correct spot, 3 is chosen as the new pivot point

8 should be to the right of the pivot, so these elements are swapped and the array is in order

1	8	3	10	39	349

Although in the worst case this algorithm is still O(n2), there are several things that can be put in place to bring the average to O(n log n). Algorithms exist that help to point towards the best pivot point, which makes the sort more efficient.

The algorithm is outlined below.

Java

```
1.  public class QuicksortMethod {
2.    public int partition(int array[], int low, int
      high) {
3.      int pivot = array[high];
4.      int i = (low - 1);
5.
6.      for (int j = low; j < high; j++) {
7.        if (array[j] <= pivot) {
8.          i++;
9.          int temp = array[i];
10.         array[i] = array[j];
11.         array[j] = temp;
12.       }
13.     }
14.
15.     int temp = array[i + 1];
16.     array[i + 1] = array[high];
17.     array[high] = temp;
18.
19.     return (i + 1);
20.   }
21.   public void quickSort(int array[], int low, int
      high) {
22.     if (low < high) {
23.       int pi = partition(array, low, high);
24.       quickSort(array, low, pi - 1);
25.       quickSort(array, pi + 1, high);
26.     }
27.     }
28.   }
```

Python

```
1.    def quicksort(arr):
2.        if len(arr) <= 1:
3.            return arr
4.        else:
5.            pivot = arr[0]
6.            less = [x for x in arr[1:] if x <=
              pivot]
7.            greater = [x for x in arr[1:] if x >
              pivot]
8.            return quicksort(less) + [pivot] +
              quicksort(greater)
9.
10.   # Example usage:
11.   arr = [10, 5, 2, 3, 7, 9, 1, 8]
12.   sorted_arr = quicksort(arr)
13.   print(sorted_arr)
```

Advantages of a quicksort

- Efficient for large data sets.
- Very little memory overhead required.

Disadvantages of a quicksort

- The worst case is still O(n2) if a poor pivot point is chosen.
- The sort does not work well for small data sets.

The rest of the array is ordered so that values lower than the pivot element are on the left and higher values are on the right. The pivot element is swapped with the first element of the higher values and this process is repeated until the array is sorted.

AHL

B2.4.5 Construct and trace recursive algorithms in a programming language

Consider the following algorithm for working out factorial numbers.

```
public int factorial (int n) {
  if (n <= 1) {
       return 1
  }
  else {
       return n * factorial(n – 1)
  }
}
```

If you were to run this program with the initial parameter of 4, you could trace it in a trace table (see Table 13).

Table 13 Factorial numbers trace table

Value of *n*	Return call	Return value	Explanation
4	4 * factorial (3)	24	As you are not at base case, you enter the else and substitute *n* for 4.
3	3 * factorial (2)	6	Your first call is placed on the stack and this one evaluated. You are still not at base case, so `push()` this call on to the stack and enter the else statement.
2	2 * factorial(1)	2	This is still not base case, so `push()` this call on to the stack and enter the else statement.
1	1	You now know factorial 1 is 1, so use this to evaluate the previous call.	You have reached base case, so the number 1 is returned. You must now `pop()` your items off the stack.

Another simple recursive algorithm is counting the sum of all natural numbers for a given number; for example:

Input: 7, Output: 28 because 1 + 2 + 3 + 4 + 5 + 6 + 7 = 28.

This is a good problem for recursion as it requires smaller and smaller problems to be put together to find a solution. This can be represented as an algorithm as follows.

Recursion in Java

```
1.  public int natural (int n) {
2.      if (n <= 1){
3.          return 1;
4.      }
5.      else {
6.          return n + natural(n - 1);
7.      }
8.  }
```

Recursion in Python

```
1.  def natural (num):
2.      if (num <= 1):
3.          return 1
4.      else:
5.          return n + natural(num - 1)
```

You can use Table 14 to test your calculations for the number 7.

Table 14 Sum of all natural numbers trace table

Value of *n*	Return call	Return value (after base case has been met)
7	7 + natural(6)	(7 + 21) = 28
6	6 + natural(5)	(6 + 15) = 21
5	5 + natural(4)	(5 + 10) = 15
4	4 + natural(3)	(4 + 6) = 10
3	3 + natural(2)	(3 + 3) = 6
2	2 + natural(1)	(2 + 1) = 3
1	---	1

Practice questions

13. Describe the term Big O efficiency. [2 marks]

14. Explain how Big O can be used to evaluate the efficiency of an algorithm. [3 marks]

15. Consider the following data structure:

SWIMMER

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]
Fabian	Zack	Martin	Alana	Blerina	Pawel	Pramiti	Oscar	Leone

Construct an algorithm that will sort the data structure into ascending alphabetical order. [5 marks]

AHL

16. Consider the following algorithm:

Java

```
public static int fact (int n){

   if (n <= 1){
      return 1;
   }
   else {
         return n * fact(n – 1);
   }
}
```

Python

```
def fact(n):

   if (n<=1):
      return 1
   else:
      return (n * fact(n - 1))
```

Trace the algorithm for `fact(4)`. [5 marks]

B2.5 File processing

Syllabus understandings

B2.5.1 Construct code to perform file-processing operations

B2.5.1 Construct code to perform file-processing operations

When programming, you store data in lists. However, when you stop running a program, all of your data is lost. When you store data in files, you can store the data after the program has been closed. One type of file you can use is a sequential file. A sequential file is a file that stores data one record after another. The files can only be accessed in a linear fashion.



Figure 17 Sequential file storage

To retrieve a record, the file needs to be read starting from the beginning until you reach the file you are looking for. The nature of the file means the data can only be read in order—you cannot skip a record.

Sequential files are not useful for finding data quickly but are useful when you need low overhead files. Example uses for sequential files include the following.

System log files: If a system generates log files, these can be stored sequentially in case they need to be looked at later.

Batch processing: In banks, many records are processed overnight. Data is processed in large volumes. Sequential files are useful for this.

Storing archives: If you have older data, storing it in sequential files is useful as you do not need to access them quickly.

Practical skills

If you want to make your solution effective in the long term, being able to write to a file is particularly useful.

File writing in Java

In Java, you can use the `FileWriter` class to write and append to a file.

If you open `FileWriter` with one parameter, for example:

```
FileWriter output = newFileWriter("TestFile.txt");
```

This will write to the file, writing over any previous contents.

If you open `FileWriter` with two parameters, for example:

```
FileWriter output = new FileWriter("TextFile.txt", true);
```

This will append the next text to the file, adding to the previous contents.

`.write(data);` will write data to the file.

`.close();` will close the file to prevent any errors occurring from unclosed resources.

Example overwriting file code



File overwrite in Java

```
1.  import java.io.File;
2.  import java.io.FileNotFoundException;
3.  import java.io.FileWriter;
4.
5.  public class FileInputOutputMain {
6.    public static void main(String[] args) {
7.      String [] test = new String [] {"Ana",
        "Cate", "Jamie", "Ian", "Seb", "Lisa"};
8.      String data = "";
9.
10.     for (int i = 0; i < test.length; i++) {
11.       data = data + ", " + test[i];
12.     }
13.     try {
14.       FileWriter output = new
          FileWriter("TestFile.txt");
15.       output.write(data);
16.       output.close();
17.     }
18.     catch (Exception e) {
19.       e.getStackTrace();
20.     }
21.   }
22. }
```

Example appending to file code



File append in Java

```
1.  import java.io.File;
2.  import java.io.FileNotFoundException;
3.  import java.io.FileWriter;
4.
5.  public class FileInputOutputMain {
6.    public static void main(String[] args) {
7.      String [] test = new String [] {"Ana",
        "Cate", "Jamie", "Ian", "Seb", "Lisa"};
```

```
8.      String data = "";
9.
10.     for (int i = 0; i < test.length; i++) {
11.       data = data + ", " + test[i];
12.     }
13.     try {
14.       FileWriter output = new
          FileWriter("TestFile.txt", true);
15.       output.write(data);
16.       output.close();
17.     }
18.     catch (Exception e) {
19.       e.getStackTrace();
20.     }
21.   }
22. }
```

File reading in Java

BufferedReader

In Java, you can use the `BufferedReader` class to read in a file using the `readLine()` method. It reads the file line by line so the data can be manipulated.

The code below shows that a `BufferedReader` object is created and the data is read in line by line. This is wrapped in `try` ... `catch` ... `finally` blocks. If there is an issue, an error will be shown. In the `finally` block, the file is closed to prevent any errors. It is important to close your files.



Buffered reader in Java

```
1.  import java.io.BufferedReader;
2.  import java.io.FileReader;
3.  import java.io.IOException;
4.  public class FileInputOutputMain {
5.    public static void main(String[] args) {
6.      BufferedReader myBufferedReader = null;
7.      String data = "";
8.
9.      try {
10.       String readInLine;
11.       myBufferedReader = new BufferedReader(new
          FileReader("TestFile.txt"));
12.
```

```
13.       while ((readInLine = myBufferedReader.
          readLine()) != null) {
14.         data = data + readInLine + "\n";
15.       }
16.     } catch (IOException ex) {
17.       ex.printStackTrace();
18.     } finally {
19.       try {
20.         if (myBufferedReader != null)
21.         myBufferedReader.close();
22.       } catch (IOException ex) {
23.       ex.printStackTrace();
24.       }
25.       System.out.println(data);
26.     }
27.   }
28. }
```

Scanner

The `Scanner` class enables you to read in the file line by line. This is shown in the code below. Again, notice the `try` ... `catch` ... `finally` blocks, to prevent errors and close the file appropriately.



Scanner in Java

```
1.  import java.io.File;
2.  import java.io.FileNotFoundException;
3.  import java.util.Scanner;
4.  public class FileInputOutputMain {
5.    public static void main(String[] args) {
6.      String data = "";
7.      Scanner fileReader = null;
8.
9.      try {
10.       File readInFile =  new File("TestFile.txt");
11.       fileReader = new Scanner(readInFile);
12.
```

```
13.         while (fileReader.hasNextLine()) {
14.           data = fileReader.nextLine();
15.         }
16.       } catch (FileNotFoundException ex) {
17.         System.out.println("The file does not
            exist");
18.         ex.printStackTrace();
19.       }
20.       finally {
21.         System.out.println(data);
22.         fileReader.close();
23.         System.out.println("The file has been
            closed");
24.       }
25.     }
26. }
```

File reading and writing in Python

In Python you can use the file read, write and append methods to read from, write to and append files. There are several methods you need to be aware of. These are explained below.

```
with open ("TestFile.txt", "w") as file1:
```

Opens the file with an intention to write (`"w"`). By using the `writelines()` you can add code to the file. Please note this means that anything previously in the file will be overwritten.

```
with open ("TestFile.txt", "a") as file1:
```

Opens the file with the intention to append (`"a"`). By using the code `writelines()` to write to the file you can keep the original data as well as add new data to the current file.

```
with open ("TestFile.txt", "r+") as file1:
```

Opens the file with the intention to read (`"r+"`). This method allows you to read data in from the file and store it in a variable ready for manipulation.

Activity

The mood of a written article could, in theory, be determined by the number of times certain words appear. For example, "celebrate" suggests a positive article and "guilt" could be used to suggest a negative article. Select an online news article or other written file. Read the article and count the number of instances of a given word. If the count is more than five, suggest the mood of the article.

File write in Python

```
1.  fileData = ["Today is a lovely day \n I may go out
    and eat an ice cream \n or go to the park"]
2.
3.  data = ""
4.
5.  with open ("TestFile.txt", "w") as file1:
6.    file1.writelines(fileData)
7.    file1.close()
8.
9.  with open ("TestFile.txt", "a") as file1:
10.   file1.writelines("update it began to rain")
11.   file1.close()
12.
13. with open ("TestFile.txt", "r+") as file1:
14.   data = file1.read()
15.   file1.close()
16.
17. print (data)
```

Linking questions

1. Does database programming in SQL require computational thinking (A3)?
2. Why is an understanding of variables and their scope important for effective memory management in computer systems (A1)?
3. Is algorithmic efficiency relevant to machine learning, where large data sets are processed and computational cost can be significant (A4)?
4. Are data structures like stacks and queues applicable in networking algorithms for packet routing and load balancing (A2)?
5. How can graph theory be applied to packet distribution in networks (A2, Mathematics A&I HL)?
6. How do graph algorithms and terminologies, such as vertices and edges, impact machine learning algorithms like network analysis (A4, Mathematics A&I HL)?
7. How can network traffic be used as an example of, or connection to, programming algorithms (A2)?
8. How could programming algorithms be applied to develop machine learning methods (A4)?

End-of-topic questions

Topic review

1. Using your knowledge from this topic, B2, answer the guiding question as fully as possible:

   How can we apply computer programming to solve problems? [6 marks]

Exam-style questions

2. State a suitable data type for the following variables.
   a. Your given name [1 mark]
   b. Your age [1 mark]
   c. The price of a cupcake [1 mark]
   d. Whether you wear glasses [1 mark]
   e. The name of your favourite game [1 mark]
3. Describe why `if` statements are necessary in programming. [3 marks]
4. Describe two differences between a `while` loop and a `for` loop. [4 marks]
5. Outline the purpose of a function. [2 marks]
6. A shop owner wants to record the number of items sold in their shop each day. This is what the owner records on one day:

Apples 10	Banana 3	Apples 2	Cherries 40	Grapes 20	Banana 4	Lemon 6

   a. Identify what type of data this is. [2 marks]
   b. Identify one limitation of storing the data in this way. [2 marks]
   c. Describe an alternative way to represent the data. [2 marks]
7. Compare a linear search and binary search. [4 marks]
8. Construct a function that will determine if someone is old enough to view a given movie based on the certificate of the movie. [4 marks]

9. Below is a mystery algorithm given both in Java and Python.

Java	Python

`public int mystery(int a, int b){`
`  if (b > a){`
`  int temp = a;`
`  a = b;`
`  b = temp;`
 	
`  }`
 	
`  while (b > 0){`
`  int r = a % b;`
`  a = b;`
`  b = r;`
`  }`
`  return a;`
`}`
	`def mystery(a, b):`
	`  if (b > a):`
	`  temp = a`
	`  a = b`
	`  b = temp`
	
	`  while (b > 0):`
	`  r = a % b`
	`  a = b`
	`  b = r`
	
	`  return a`

i. Using a trace table, trace this algorithm for the values a = 24 and b = 36. [3 marks]

a	b	temp	r	b > 0	return

ii. State the Big O of this algorithm. [1 mark]

10. Construct an algorithm to sort the following data set.

unsorted: {45, 89, 23, 12, 90, 34, 58, 1, 76, 123, 8} [5 marks]

11. A stack has been created using the following instructions.

```
push(8)
push(10)
peek()
push(14)
pop()
pop()
```

Sketch the resulting stack after these operations have been performed. [1 mark]

12. A queue has been created as follows.

```
enqueue(3)
enqueue(5)
enqueue(2)
enqueue(9)
dequeue()
dequeue()
```

State the result, after these operations have been completed, of the code: `peek()`. [1 mark]

13. The following table keeps a track of the volume of rain over the month.

	[0]	[1]	[2]	[3]	[4]	[5]	[6]
[0]	5.2	5.3	4.6	8.6	4.9	3.9	8.6
[1]	4.1	10.2	11.5	3.8	7.5	7.2	10.5
[2]	9.2	3.6	4.7	4.5	6.8	3.76	11.5
[3]	4.1	0.2	0.7	5.6	6.4	12.4	4.12

a. i. Identify the data structure used.

ii. Outline one reason why this is a suitable data structure for this task. [2 marks]

b. Construct a function that will return the lowest rainfall over the course of the month. [4 marks]

c. The developer is facing difficulties with their coding.

Describe two debugging techniques they could use to identify issues. [4 marks]

d. Construct a function that will return the average rainfall over the course of the month. [5 marks]



B3

Object-oriented programming

Is object-oriented programming (OOP) an appropriate paradigm for solving complex problems?

When developing programs, you often want to model the real world, where objects do not exist as many different separate attributes, but as objects that have attributes and behaviours.

Object-oriented programming (OOP) enables you to model objects closer to the way you view them in the real world, with attributes and behaviours neatly wrapped inside an object. This can often make it easier to solve complex problems. In this topic you will discover how programming objects work, and how they enable us to model the world in a more realistic way.

B3.1 Fundamentals of OOP for a single class

Syllabus understandings

B3.1.1 Evaluate the fundamentals of OOP

B3.1.2 Construct a design of classes, their methods and behaviours

B3.1.3 Distinguish between static and non-static variables and methods

B3.1.4 Construct code to define classes and instantiate objects

B3.1.5 Explain and apply the concepts of encapsulation and information hiding in OOP

Key terms

Object An instantiated class. Instance variables have been given values and the object now exists in memory.

Class A plan or a blueprint of an object identifying the instance variables (attributes) and methods (behaviours).

B3.1.1 Evaluate the fundamentals of OOP

Object-oriented programming, as the name suggests, makes use of **objects** and **classes** to represent real world entities. But what is a class? If you have ever played a computer game, then you have interacted with a program produced in object orientation. Every non-player character (NPC) in a game is an instance of the non-playable character class. The class will determine how the NPC moves and interacts with its environment. Each enemy will be an instance of the enemy class. This class will tell the enemy how to behave when interacting with the protagonist.

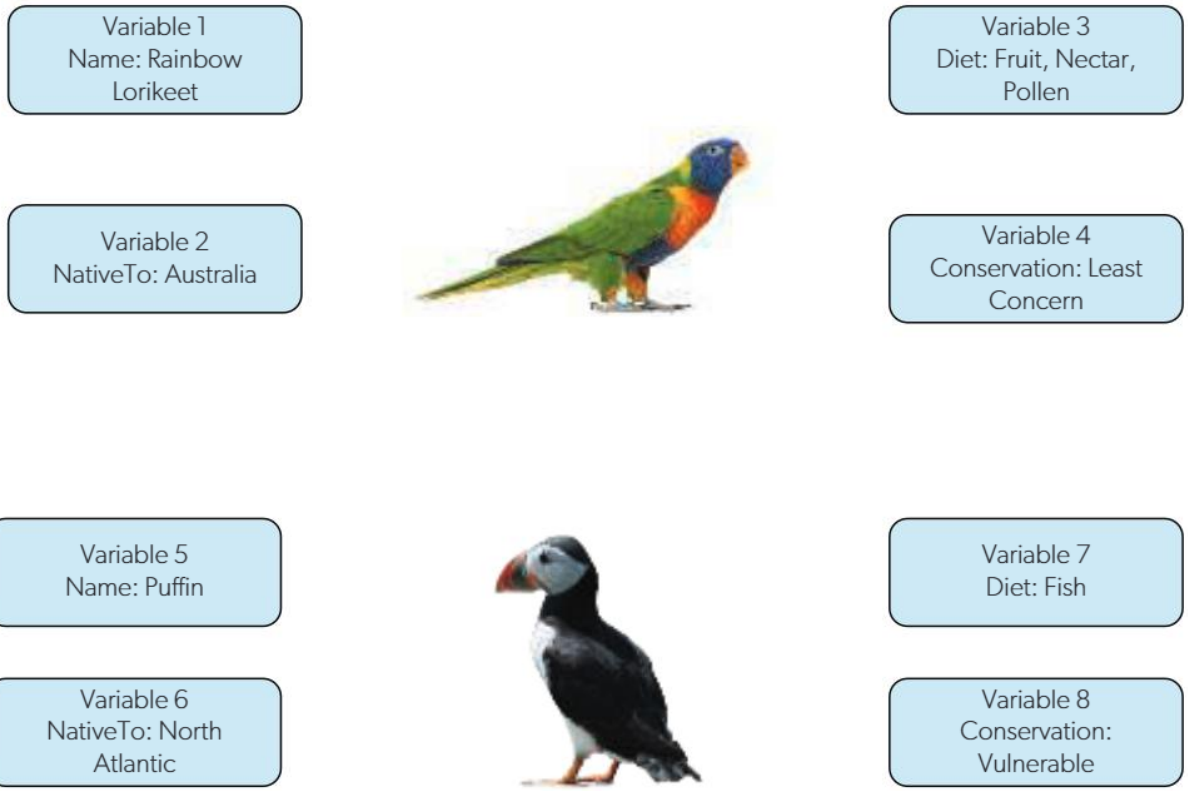


▲ **Figure 1** A non-player character is an instance of the non-playable character class

In B2 Programming, you learned how to develop programs using variables. This enabled you to store singular pieces of data representing an attribute, such as a test score, an age or the name of an item. If you want to store several pieces of data about an item, you need a variable for each attribute you want to store. If you want to store multiple data attributes about multiple items, then you would probably use multiple lists.

Explaining the need for classes

Imagine you are developing a program to store information about the conservation status of birds. You may want to store data similarly to Figure 2.



▲ **Figure 2** Data about the conversation status of birds

The figure shows that you need eight variables to store all the details of two birds. You may have used a separate list for each variable.

Name	
[0]	Rainbow Lorikeet
[1]	Puffin

NativeTo	
[0]	Australia
[1]	North Atlantic

Diet	
[0]	Fruit, Nectar and Pollen
[1]	Fish

Conservation	
[0]	Least Concern
[1]	Vulnerable

▲ **Figure 3** Lists for variables

To solve the problem of many variables, many lists, and confusing code, you can use object orientation. Object orientation enables you to store multiple attributes and behaviours of an item in one class. For example, all the information you need to store about birds would be contained within a bird class. Every time a new bird is added, a new instance of the bird class is created. This means there is one list containing all the information.

In B3.1.2, you will learn how to design and create your own classes.

A class consists of instance variables (the attributes of the item) and behaviours (what the item can do). The class is a blueprint or plan for the item. To create an object, you give the instance variables starting values which produces an object that can be manipulated. The code for the bird class is shown here in both Java and Python.



Java

```
1.  public class Bird {
2.    private String name;
3.    private String nativeTo;
4.    private String diet;
5.    private String conservation;
6.    public Bird (String name, String nativeTo,
      String diet, String conservation) {
7.        this.name = name;
8.        this.nativeTo = nativeTo;
9.        this.diet = diet;
10.       this.conservation = conservation;
11.   }
12.   public String getName() {
13.       return name;
14.   }
15.   public String getNativeTo() {
16.       return nativeTo;
17.   }
18.   public String getDiet() {
19.       return diet;
20.   }
21.   public String getConservation() {
22.       return conservation;
23.   }
24.   public String __str__() {
25.       return "Name: " + name + " Native To: " +
          nativeTo + " Diet: " + diet + " Conservation:
          " + conservation;
26.   }
27. }
```



Python

```
1.  class Bird:
2.      def __init__(self, name, nativeTo, diet,
        conservation):
3.        self.name = name
4.        self.nativeTo = nativeTo
5.        self.diet = diet
6.        self.conservation = conservation
7.     def __str__(self):
8.        return "Name: " +self.name + " Native to: "
          + self.nativeTo + " Diet: " + self.diet + "
          Conservation: " + self.conservation
```

Activity

Choose a book, an item of clothing, or a friend. What attributes (instance variables) could you store about them? What behaviours (methods) would they have? Once you have identified the attributes and behaviours, swap your ideas with a partner. Is there anything you could add to their work?

Tools within object orientation

There are tools in object orientation that you can use to model real-world entities. You can use classes, objects, inheritance, encapsulation and polymorphism.

Classes

As shown in the conservation of birds example, you can use classes to design real-world entities. A class is a blueprint or plan for an object that does not exist until a constructor method (a special method that creates an object of the class) has been called. This plan includes the attributes and behaviours they should have. By designing classes, you can design objects to model their real-world counterparts.

Objects

Once the constructor method has been called, and the instance variables assigned their starting values, you have an object. This object can be manipulated and used to mimic its real-world counterpart. Once an object is **instantiated** (created and given a space in memory), it has an identifier that it can be referenced by.

Inheritance

Inheritance allows you to be efficient with your code. All common methods and variables are stored in a superclass, and the subclasses extend the functionality adding their own variables and methods. If you need to make any changes to the code and the code is in the superclass, you only need to make the change once, which is much more efficient than changing lots of separate classes. A real-world example of inheritance is vehicles. The vehicle superclass stores all the common information such as owner and size. Specific vehicles subclasses have their own specific variables. A car might have fuel type, an aeroplane might have maximum range, or a ship might have number of engines.

Encapsulation

One of the key features of object orientation is encapsulation. Encapsulation enables you to reuse code easily. As all of the variables and methods are wrapped up inside of the class, you can use the class in other programs with little effort. Encapsulation also allows you to keep information safe, as the data cannot be accessed outside of the class. Encapsulation allows you to use classes that are complex. You do not need to understand how the internal methods are working, just the information you need to send, the methods, and the information you would expect to receive. If you have made use of a programming library when coding, you have made use of encapsulation.

Polymorphism

Polymorphism—the word derives from the Greek, meaning "many forms"—enables classes to behave in different ways depending on the situation.

Key term

Instantiation An instantiation of a class is when you call the constructor method and provide starting values for each of the instance variables. The object is allocated a space in memory and can be used throughout the program.

For example, a method may have the same name but different parameters. The version of the method you need to use depends on the number of parameters you send to the method.

To make this easier to understand, think about yourself on your educational journey. If you are asked to "write" in your Language A class, this may mean develop a short story using creative language. In Natural Sciences, "write" may mean producing a formal laboratory report. Outside of school, "write" may mean sending an informal message to a friend. All these instructions use the same word but you act differently depending on the setting.

Advantages and disadvantages of object-oriented programming

Advantages

- As object-oriented development environments have libraries of code prewritten and pretested, you can import useful libraries and save time with development.
- The nature of object orientation enables you to take large, complicated problems and break them into smaller, more manageable tasks.
- Using inheritance allows you to create superclasses to store common items and then develop subclasses to make more specific items. This saves time coding, as it promotes code reuse.
- Breaking a program into objects enables developers to work concurrently on separate items, again saving time on development.
- Encapsulation helps to build secure programs, as the data is hidden within the objects and prevents interference from external systems.
- Using classes to represent objects mimics the real-world.

Disadvantages

- The overhead of creating an object-oriented solution is often excessive for simpler programs.
- Object-oriented programs are large in size, which can slow execution in some instances.
- The modular nature of the code can make programs complex to understand, especially for novice programmers.

In topic A3 Databases, you also tried to represent the real world within the computer. Understanding the attributes and behaviours of objects will help you to develop database models.

Activity

Identify whether OOP would be a suitable programming paradigm for these tasks. Justify your response for each.

Developing a program to:

- store details about passengers
- complete an automated task on a server
- write a script for an automated car
- create a first-person role-playing game.

B3.1.2 Construct a design of classes, their methods and behaviours

When you are designing classes, you need to think about the item you are storing information about. What attributes does it have that you need to know? What behaviours will it be expected to perform? Table 1 shows three examples that give you an indication of attributes and behaviours you may want to store.

▼ **Table 1** Attributes and behaviours of different objects

Object	Image	Attributes	Behaviours
Car		Size of car, number of seats, size of engine, manufacturer, model	Move forward, move backward, turn left, turn right, need energy
Computer game hero		Lives, powers, name, type of character	Defensive action, attack action, lose life, gain life
Person		Name, age, address, education	Complete assigned work, have birthday, learn

The attributes and behaviours are the key parts of the class. The attributes will eventually become the instance variables—the data you want to store about the object. The behaviours will become the methods—what you need the object to be able to do.

ATL Research skills

To design classes, you have to understand the context of the class. To understand the context, you may need to complete some research. This links to real-life problem identification in the computational thinking process.

Develop a class for a hotel room. This Room class will be used to allocate a room to hotel guests.

What attributes does the Room class have? What methods? What research must you do to complete this task successfully?

It is important that all developers can understand your class design. Computer scientists use **universal modelling language (UML) diagrams** to facilitate this.

Complex programs need clear planning, and UML diagrams help with this. UML diagrams can also help to communicate the system to non-technical users. By using UML diagrams that other developers can understand, you avoid wasting time explaining what you mean. There are many different UML diagrams a developer can use when designing a system.

- **Sequence** or **event diagrams** show the order in which objects interact.
- **Use case diagrams** show how the users will interact with the system.
- **State diagrams** show how objects can transition from one stage to the next.
- **Package diagrams** show the dependencies between different packages in the system.

All these diagrams are useful when developing large, complex solutions with multiple different users. In this course, you will use the UML class diagram. This enables software developers to view the classes and the relationships between the classes in a visual form. Class diagrams are an important tool in object-oriented programming design.

UML class diagrams show three boxes. These contain the name of the class, the instance variables of the class, and the methods of the class. The construction of the UML class diagram is shown in Figure 4.

ATL Communication skills

Being able to produce a representation of a class that is universally recognized enables you to become a better communicator.

UML diagrams let you communicate clearly with developers. Using UML diagrams ensures that everyone knows what classes are required in the program and how the classes are linked.

Identify other diagrams that you have developed or studied in this course that also provide clear communication. What are the similarities and differences in the diagrams? What is each kind of diagram used for?

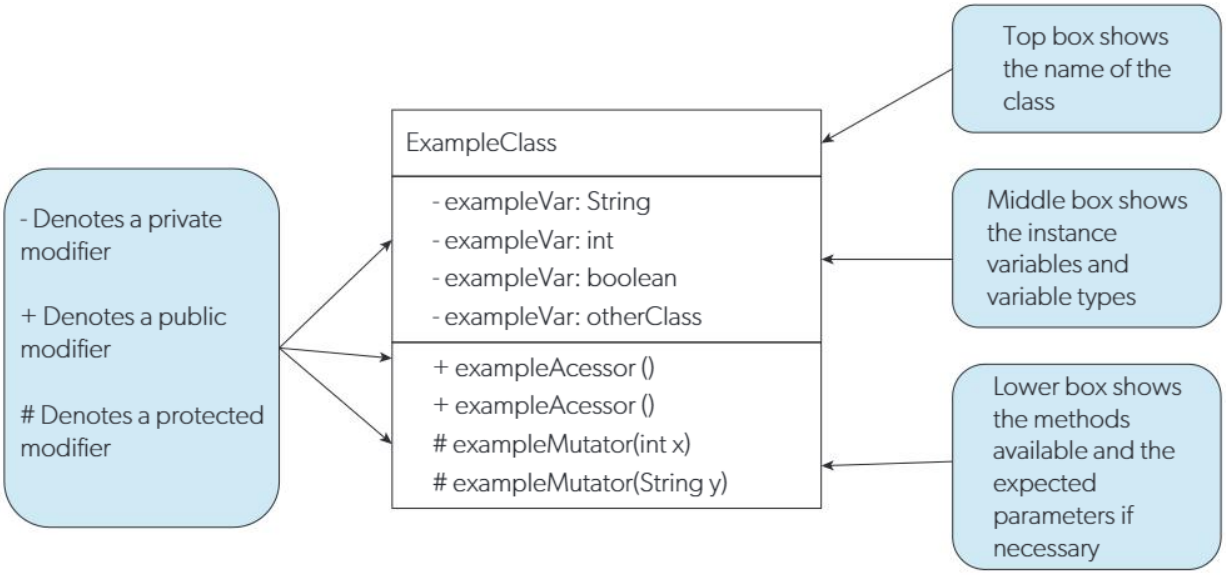
Work in small groups to make a short video to teach other students about these diagrams. If you do not have access to video equipment, present your work to your class or your group.

Key term

Universal modelling language (UML) diagram A universally recognized way to identify the modifiers, identifiers and data types of the variables and methods within a class. These diagrams are then used to develop the class in code.

UML diagrams are unified diagrams that show developers the design of the program. You will also look at standardized models in Databases topic A3.

The different kinds of modifiers—public, private and protected—will be described further in section B3.1.5.



▲ **Figure 4** UML Class example

Activity

Develop a UML class diagram for each of these items.

1. **Student** Attributes: Name, Grade level, Homeroom, Age.
   Behaviours: Access the information, change age, change grade level.
2. **Toy** Attributes: Name, Manufacturer, Minimum age, Batteries required.
   Behaviours: Access the information, set out of batteries.
3. **Song** Attributes: Name, Artist, Song length, Star rating.
   Behaviours: Access the information, set star rating.

TOK

When developing classes, you need to decide which attributes and behaviours are important and which you can ignore. For example, when storing information about a car, you may choose to store the make, the model, and the number of doors but ignore the factory in which it was built. Keep in mind the purpose of the data that you are saving—data which is irrelevant does not need to be saved.

By doing this, to what extent do you affect the production and acquisition of knowledge for the users of your solutions?

AHL

You will learn more about this in subtopic B3.2 Fundamentals of OOP with multiple classes.

Modelling relationships between classes

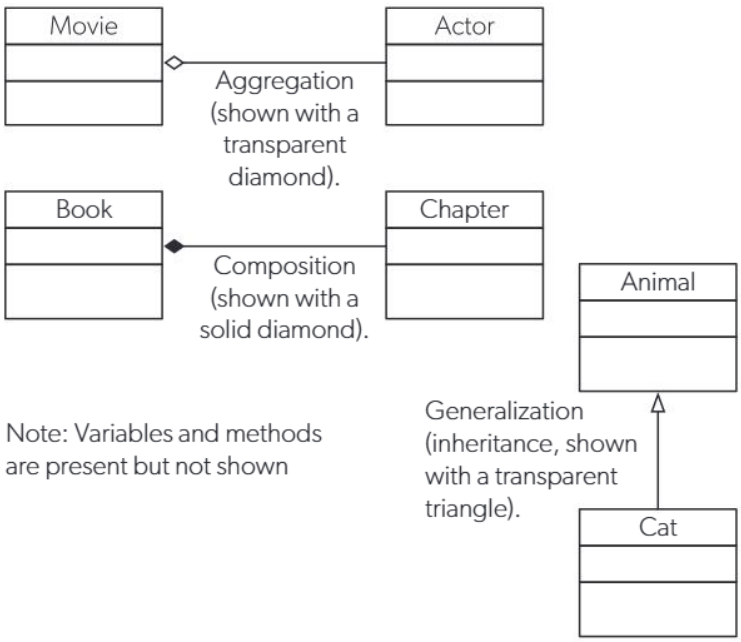
UML class diagrams also show the relationships between classes. There are three types of relationships you need to be aware of in object-oriented programming.

Aggregation: In an aggregated relationship, the classes can exist independent of each other, but in the program being developed they need each other. Example: Actor and Movie. An actor can exist without a movie and a movie without an actor but usually a movie needs actors.

Composition: In a composition relationship, the classes cannot exist without each other. Example: Chapter and Book. You cannot have a chapter without a book or a book without at least one chapter.

Inheritance: In an inherited relationship, one class is the parent class, and the child class contains all the variables and methods of the parent class as well as its own specific variables in methods. Example: Animal and Cat. A cat contains all the general attributes of an animal as well as its own specific cat attributes.

These can be modelled in the following way.



▲ **Figure 5** A diagram showing the different UML relationships

B3.1.3 Distinguish between static and non-static variables and methods

A book in a library has many different attributes that could be stored. These may include the title of the book, the author of the book, the ISBN, and whether it is fiction or non-fiction. A book in a library may also have different behaviours: whether the book is currently borrowed, whether there is a waiting list for the book, and whether the book needs to be replaced. But the book itself cannot tell you how many books have been borrowed from the library. This is why you need **static** variables.

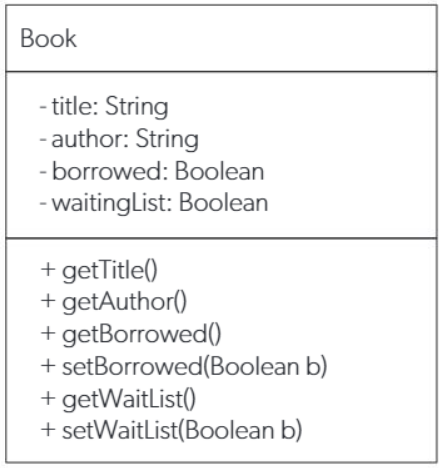
Key term

Static A variable or method belonging to the class rather than the instance of the class.

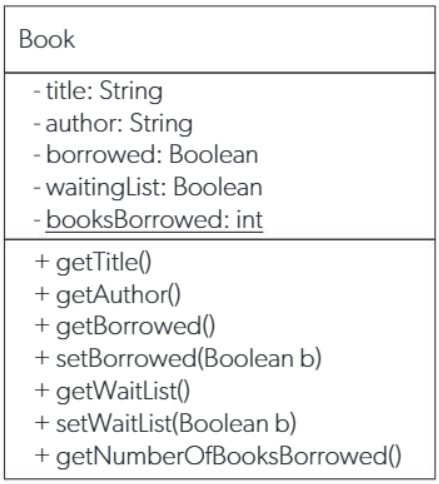


▲ **Figure 6** Books in a library

The UML class diagram for a book might look like the following.



These variables belong to the instance of the class and can only be altered by the instance of the class. When you access these variables, you are accessing the values of the instance. However, if you add a static variable and method, these act differently. You do not need an instance to access a static variable—you can use the class name. The updated UML class diagram, including the underlined static variable—which belongs to the class not the instance—would look similar to the following.



The code below shows the construction of the Library class in Java and Python that makes use of static variables. The instantiation of the class is also shown, demonstrating the difference between referencing the instance of the class (the object) and the class itself.

Java library class code

```
1.  public class Book {
2.
3.      private String title;
4.      private String author;
5.      private boolean borrowed;
6.      private boolean waitList;
7.      private static int booksBorrowed = 0;
```

```
8.
9.      public Book (String title, String author) {
10.
11.         this.title = title;
12.         this.author = author;
13.         this.borrowed = false;
14.         this.waitList = false;
15.
16.     }
17.
18.     public String getTitle() {
19.         return title;
20.     }
21.     public String getAuthor() {
22.         return author;
23.     }
24.     public boolean getBorrowed() {
25.         return borrowed;
26.     }
27.     public void setBorrowed(boolean b) {
28.         borrowed = b;
29.         if (borrowed) {
30.             booksBorrowed = booksBorrowed + 1;
31.         }
32.         else {
33.             booksBorrowed = booksBorrowed - 1;
34.         }
35.     }
36.     public boolean getWaitList() {
37.         return waitList;
38.     }
39.     public void setWaitList (boolean b) {
40.         waitList = b;
41.     }
42.     public static int getNumberOfBooksBorrowed() {
43.         return booksBorrowed;
44.     }
45.     public String toString() {
46.         return "Name of Book: " + title + ",
            Author of Book: " + author + ", Borrowed:
            " + borrowed + ", Wait List: " + waitList;
47.     }
48.
49. }
```

Java code to test the class

```
1.  public class LibraryMain {
2.
3.      public static void main(String[] args) {
4.
5.          Book one = new Book ("Community and
            Support", "D Larkin");
6.          Book two = new Book ("Where is Archibald?",
            "C Rington");
7.          Book three = new Book ("Horticulture for
            the Balcony", "A Abed");
8.
9.          one.setBorrowed(true);
10.         two.setBorrowed(true);
11.         two.setWaitList(true);
12.
13.         System.out.println(one.toString());
14.         System.out.println(two.toString());
15.         System.out.println("The number
            of books borrowed is: " + Book.
            getNumberOfBooksBorrowed());
16.
17.     }
18.
19. }
```

Output:

Name of Book: Community and Support, Author of Book: D Larkin, Borrowed: true, Wait List: false

Name of Book: Where is Archibald?, Author of Book: C Rington, Borrowed: true, Wait List: true

The number of books borrowed is: 2

Python library class code

```
1.  class Book:
2.      __booksBorrowed: int = 0
3.
4.      def __init__(self, title, author):
5.          self.__title = title
6.          self.__author = author
7.          self.__borrowed = False
8.          self.__waitList = False
9.
10.     def getTitle(self):
11.         return self.__title
12.
```

```
13.     def getAuthor(self):
14.         return self.__author
15.
16.     def getBorrowed(self):
17.         return self.__borrowed
18.
19.     def getWaitList(self):
20.         return self.__waitList
21.
22.     def setBorrowed(self, b):
23.         self.__borrowed = b
24.         if (self.__borrowed):
25.             Book.__booksBorrowed = Book.__
                booksBorrowed + 1
26.
27.         else:
28.             Book.__booksBorrowed = Book.__
                booksBorrowed - 1
29.
30.
31.     def setWaitList (self, b):
32.         self.__waitList = b
33.
34.     def getNumberOfBooksBorrowed(Book):
35.         return Book.__booksBorrowed
36.
37.     def __str__(self):
38.         return "Title: " + self.__title + ",
            Author: " + self.__author + ", Borrowed: "
            + str(self.__borrowed) + ", Waitlist: " +
            str(self.__waitList)
```

Python code to test the class

```
1.  from Book import Book
2.
3.  one = Book ("Community and Support", "D Larkin")
4.  two = Book ("Where is Archibald", "C Rington")
5.  three = Book ("Horticulture for the Balcony", "A
    Abed")
6.
7.  one.setBorrowed(True)
8.  two.setBorrowed(True)
9.  two.setWaitList(True)
10.
11. print(one.__str__())
12. print(two.__str__())
13. print("The number of books borrowed is: ", Book.
    getNumberOfBooksBorrowed(Book))
```

Output:

```
Title: Community and Support, Author: D Larkin, Borrowed: True, Waitlist: False
Title: Where is Archibald, Author: C Rington, Borrowed: True, Waitlist: True
The number of books borrowed is: 2
```

The previous code demonstrates the following.

- Static variables belong to the class, not the instance of the class. In Java they are denoted using the static keyword. In Python they are declared prior to the `__init__` function as a class variable.
- In Python you use `import` to include the class file. This is your program rather than a standard library.
- There is only one copy of the static variable and this static variable is shared between all instances of the class.
- The static variable is initialized once (at the start of the execution) and retains their values throughout the program's run.
- The static variable is accessed through the class name and not the instance name.
- Static methods in java include the keyword `static` and in python by passing in the name of the class as a parameter rather than "self".
- Static methods can only access static variables. They cannot access instance variables directly without an object reference.



Activity

For each item, identify one static and three non-static variables that might be in the class.

- Television show on a streaming platform.
- Car in a car salesroom.
- Teacher in a school.

ATL Thinking skills

Being able to identify which variables and methods are static develops your coding skills. Identifying the variables required develops your abstraction skills.

Consider a large supermarket that sells food, household supplies, stationery, clothes and books. What do you need to know in order to successfully identify the variables and methods? How would you know which are static?

Static variables are useful when you wish to share data and behaviours across classes. They need to be used carefully so you do not accidentally change shared data.

B3.1.4 Construct code to define classes and instantiate objects

Implementing classes in code looks very different in Java and Python, as has already been shown. This section will describe each in detail.

There are examples of library code in section B3.1.3.

First, remember that a class is a blueprint or a plan of how to build an object. To successfully construct the class, you should plan it with a UML class diagram showing the plan. Here, you will use a UML class diagram to design the class Plant.



▲ **Figure 7** A blueprint

Plant
+ scientificName: String + scientificFamily: String + distribution: String + bloom: boolean
- getScientificName() - getScientificFamily() - getDistribution() - getBloom()

▲ **Figure 8** UML diagram to design the class Plant

To create this class in your chosen IDE in Java, develop a new class without a main method. This class should be in the same package as your other classes.

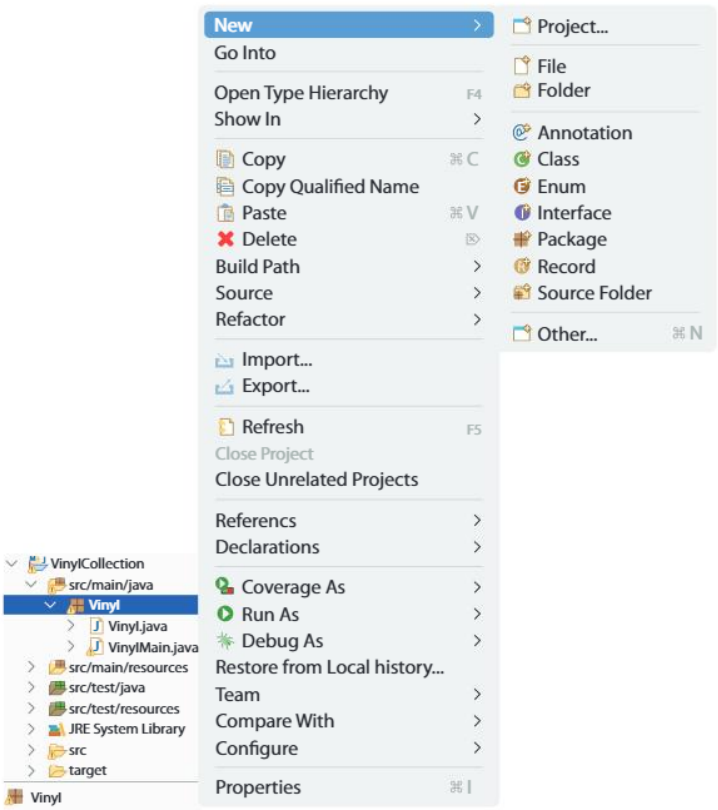
In Java, the filename and class name need to be the same. But in Python, this is optional.

Declare the variables using the correct modifier.

Write the constructor method to allow the class to be allocated a space in memory.

Declare and write the methods using the correct modifier.

A method `toString()` is included in Java to provide a formatted version of the data in the class.



▲ **Figure 9** How to create a new class in Java, using the Eclipse IDE

Note the use of **this** in the constructor method to show the difference between the instance variables and the value passed by the parameters.

Name of file: Plant.java

Name of class

Instance variables private as shown in UML

Expected Parameters

Constructor Method used to create an object of the class

Methods used to access the information from outside the class / complete tasks inside the class

To string method provides a formatted version of the class

```
1.  public class Plant {
2.
3.      private String scientificName;
4.      private String scientificFamily;
5.      private String distribution;
6.      private boolean bloom;
7.
8.      public Plant (String scientificName, String
        scientificFamily, String distribution, boolean
        bloom) {
9.
10.         this.scientificName = scientificName;
11.         this.scientificFamily = scientificFamily;
12.         this.distribution = distribution;
13.         this.bloom = bloom;
14.
15.     }
16.
17.     public String getScientificName() {
18.         return scientificName;
19.     }
20.     public String getScientificFamily() {
21.         return scientificFamily;
22.     }
23.     public String getDistribution() {
24.         return distribution;
25.     }
26.     public boolean getBloom() {
27.         return bloom;
28.     }
29.     public String toString() {
30.         return "Plant Name: " + scientificName +
            ", Plant Family: " + scientificFamily + ",
            Areas Found: " + distribution + ", Flowers:
            " + bloom;
31.     }
32.
33. }
```

▲ **Figure 10** Explaining the Plant class code in Java

To create this class in your chosen IDE in Python, create a new Python file. It is good practice to name the file the same as your class name (but not essential in Python).

Create the file and class with a suitable name.

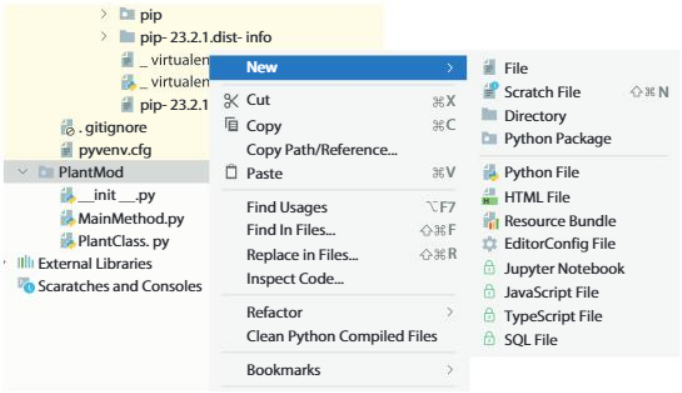
Declare the variables.

Write the constructor method to allow the class to be allocated a space in memory.

Write the methods.

A method `__str__(self)` is included in Python to provide a formatted version of the data in the class.

Note the use of `self` in the constructor method to show the difference between the instance variables and the value passed by the parameters.



▲ **Figure 11** How to create a new class in Python, using the PyCharm IDE

Name of file: Plant.py

Name of class

Expected parameters

Instance variables as shown in UML (modified to private)

Constructor method used to create an object of the class

Methods used to access the information from outside the class or complete tasks inside the class

The toString method provides a formatted version of the class

```
class Plant:
    def __init__ (self, scientificName, scientificFamily,
    distribution, bloom):
        self.__scientificName = scientificName
        self.__scientificFamily = scientificFamily
        self.__distribution
        self.__bloom = bloom

    def getScientificName(self):
        return self.__scientificName

    def getScientificFamily(self):
        return self.__scientificFamily

    def getDistribution(self):
        return self.__distribution

    def getBloom(self):
        return self.__bloom

    def __str__(self):
        return ("Scientific Name: {self._scientificName},
        Scientific Family: {self._scientificFamily}")
```

▲ **Figure 12** Explaining the Plant class code in Python

Once you have created the class in your chosen programming language, you need to create an instantiation of the class, also known as an object. Once you have instantiated the class (called the constructor and given the starting values in the parameters) you have an object. This means a space in memory is assigned to the object, it is given an identifier, and its instance variables are given initial values. Instantiation of an object usually happens in a runner class (a class that controls the flow of the program). In Java, this is usually the class that has the **`main`** method. In Python, this can be the same file or a different file. Figure 13 gives the example code for creating an instance of an object in Java, and Figure 14 gives the example code for creating an instance in Python.

```
1.  public class main {
2.
3.      public static void main(String args[]){
4.
5.          Plant myFavourite = new Plant("Narcissus",
            "Amaeyllidceae", "Worldwide", true);
6.
7.
8.          System.out.println(myFavourite.toString());
9.
10.     }
11.
12. }
```

Instance identifier (variable name)

Actual values given as parameters. These are the initial values of the instance variables.

toString() called for testing purposes

▲ **Figure 13** Creating an instance of an object in Java

```
1.  from Plant import Plant
2.
3.
4.  myFavourite = Plant('Narcissus',
    'Amaryllidaceae','Worldwide', True)
5.
6.  print(myFavourite.toString())
```

Class imported for use

Instance identifier (variable name)

Actual values given as parameters. These are the initial values of the instance variables.

toString() called for testing purposes

▲ **Figure 14** Creating an instance of an object in Python

TOK

The Svalbard Global Seed Vault holds seeds of food crops from all over the world. It has the capacity to store 4.5 million varieties of seeds. As of 2024, the seed vault is preserving from extinction unique varieties of food staples including varieties of maize, rice, wheat, cowpea, sorghum, eggplant, lettuce and potato. The purpose of the seed vault is to safeguard as much genetic material of the world's crops as possible. The information in the seed vault needs to be carefully recorded and managed.



▲ **Figure 15** Svalbard

How can storing data in an organized manner enable us to preserve knowledge for the future? Use the Svalbard Global Seed Vault and at least one other example of large data storage in your answer.

Now that you understand how to make objects through calling the constructor method and creating an instantiation, use the skills you learned in topic B2 to develop more interesting programs. There is also a practical skills section in this topic to support you.

Activity

Develop code to instantiate objects for each of your classes from the activities in sections B3.1.2 and B3.1.3. You can use either Python or Java.

B3.1.5 Explain and apply the concepts of encapsulation and information hiding in OOP

One way to think about **encapsulation** is to think about a capsule. All the variables and methods are hidden from view, but you know they are in there. You can use the capsule without fully understanding how it works. You also know the contents of the capsule are safe, as nobody has been able to access it.

You probably interact with encapsulated code daily. For example, when you log into email or a social media site, you type in your username and password, which are the variables. Several methods will occur—most of them behind the scenes or without your active participation—before you gain access. You do not directly encounter the variables and methods, but you know they will do the job you ask of them.



▲ **Figure 16** A medicine capsule

Key term

Encapsulation When data is hidden within a class and accessed only using publicly accessible methods, promoting reuse and improved code maintenance.

ATL Thinking skills

Developing encapsulated code means you are thinking carefully about how to keep data safe and ensure any data provided by the program accurately represents the object in question. It also helps you to strengthen your abstraction skills.

- What features of code make it encapsulated?
- Why is it important to keep the code safe using encapsulation?
- How does this link to abstraction?

Share your thoughts with a partner. What can you learn from each other?

The main principle of encapsulation is to hide information. Hiding information keeps it protected and ensures that data cannot be changed by external sources, keeping it secure.

The advantages of encapsulation are as follows.

- Encapsulated code works and functions within its own class—no external interaction is needed for it to function. This means code is flexible and easy to adapt.
- Code modification in an encapsulated environment does not cause problems for other code.
- Maintenance is improved because you only have to change the code in one place and this is reflected in all programs that use the encapsulated class.
- Private fields are not accessible so data is secure.
- Data and code are safe from external influences and, therefore, more secure.

To encapsulate your code, first set the modifier for your instance variables. Variables can have one of these three modifiers.

Private: This is the modifier you are most likely to use when developing your own code. The private modifier means the instance variable is only accessible inside the class it is declared in.

Protected: Variables declared with the protected modifier can be accessed by the class and from all classes within the package. This includes the subclasses that inherit variables from their superclasses.

Public: Variables declared with the public modifier can be accessed by the class it is declared in as well as all other classes. Public variables are the least secure. If your variables are public then you do not have encapsulated code.

Inheritance has an impact on access to parent class members.

- Private variables are not accessible to subclasses. To access a private variable outside the class, your class needs to include a public accessor/getter method.
- To access a protected variable outside of the package, you would need to include a public accessor/getter method.

Table 2 gives an overview of the modifiers in Java and Python.

AHL

▼ **Table 2** Variable modifiers in Java and Python

	Modifier		
	Private	**Protected**	**Public**
Example in Java	`private int cost;`	`protected int cost;`	`public int cost;`
Example in Python	`self._cost`	`self.__cost`	`self.cost`
Class	Yes	Yes	Yes
Subclass	No	Yes	Yes
Package	No	Yes	Yes
Other classes	No	No	Yes

Worked example 1

You are programming a computer system for a restaurant. You have a menu that contains many dishes. The dishes each have a name, a type (starter, main, or dessert), allergen information, a cost price (what the dish costs to make), and retail price (the cost to the customer, with a profit margin added). The restaurant does not want the customers to know how much extra they are paying, so you want to keep your calculations private.

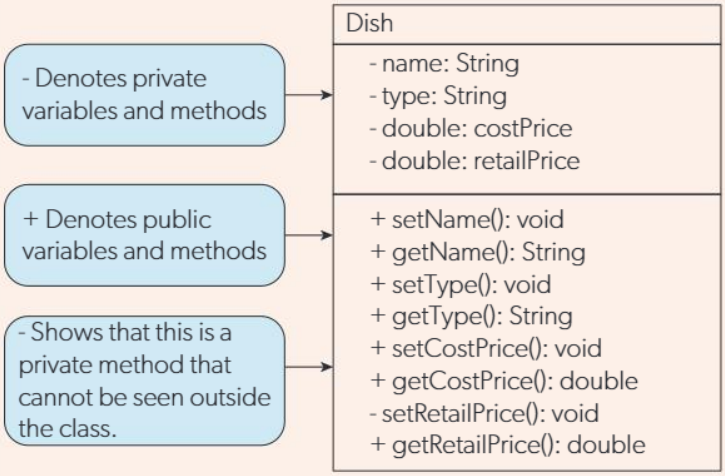
Create a program to help the restaurant store its menu.



▲ **Figure 17** A restaurant could use a program to store its menu

Solution

Use a UML class diagram to represent the class you will use, like this.



▲ **Figure 18** UML class diagram

TOK

When using computer programs for everyday tasks, an element of trust is assumed between the people who create and manage the program (the programmers) and the people who use it (the consumers).

Consumers assume that any data they put in the program will be safe, that it will not be misused, that any calculations made using the data will be correct, and that their data cannot be changed incorrectly.

What ethical responsibilities do programmers have to ensure security of data?

The UML diagram shows that the class is encapsulated. Its private variables and methods limit the access to the class members. The public methods enable access to the private variable values from other classes.

The retail price is calculated within a private modifier so it cannot be seen or manipulated outside of this class. Access to this method and the variables within the class is limited.

Encapsulated class in Java

```
1.  public class Dish {
2.
3.      private String name;
4.      private String type;
5.      private double costPrice;
6.      private double retailPrice;
7.
8.      public Dish (String name, String type, double
        costPrice) {
9.
10.         this.name = name;
11.         this.type = type;
12.         this.costPrice = costPrice;
13.         this.retailPrice = setRetailPrice();
14.     }
15.
16.     public String getName() {
17.         return name;
18.     }
19.     public void setName (String n) {
20.         name = n;
21.     }
22.     public String getType() {
23.         return type;
24.     }
25.     public void setType(String t) {
26.         type = t;
27.     }
28.     public double getCostPrice() {
29.         return costPrice;
30.     }
31.     public void setCostPrice(double c) {
32.         costPrice = c;
33.         setRetailPrice();
34.     }
```

```
35.     private double setRetailPrice() {
36.         retailPrice = costPrice + (costPrice *
            0.6);
37.         return retailPrice;
38.     }
39.     public String toString() {
40.         return "Name of Dish: " + name + ", Type
            of Dish: " + type + ", Price of Dish: " +
            retailPrice;
41.     }
42. }
```

In the runner class, you would create an instance of the encapsulated class and use the encapsulated class just as you would any other. In this example an instance of the class has been created and several methods run.

```
1.  Dish myDish = new Dish ("Spaghetti Bolognaise",
    "Main Course", 8.50);
2.  myDish.setName("Spaghetti Carbonara");
3.  myDish.setCostPrice(10);
4.  System.out.println(myDish.toString());
```

Output:

Name of Dish: Spaghetti Carbonara, Type of Dish: Main Course, Price of Dish: 16.0

Encapsulated class in Python

```
1.  class Dish:
2.    def __init__(self, name, theType, costPrice):
3.      self.__name = name
4.      self.__theType = theType
5.      self.__costPrice = costPrice
6.      self.setRetailPrice(costPrice)
7.
8.    def getName(self):
9.      return self.__name
10.
11.   def setName(self, n):
12.     self.__name = n
13.
14.   def getType(self):
15.     return self.__theType
16.
17.   def setType(self, t):
18.     self.__theType = t
19.
20.   def getCostPrice(self):
21.     return self.__costPrice
```

```
22.
23.   def setCostPrice(self, c):
24.     self.__costPrice = c
25.     self.setRetailPrice(c)
26.
27.   def setRetailPrice(self,__costPrice):
28.     self.__retailPrice = self.__costPrice +
        (self.__costPrice * 0.4)
29.
30.   def getRetailPrice(self):
31.     return self.__retailPrice
32.
33.   def show(self):
34.     return "Name: ", self.__name + " Type: ",
        self.__theType, " Retail Price ", self.__
        retailPrice
```

In the runner class, you would create an instance of the encapsulated class and use the encapsulated class just as you would any other. In the runner class, an instance of the class has been created and several methods run.

```
1.  from Dish import Dish
2.
3.  myDish = Dish("Spaghetti", "Main", 8.50)
4.  myDish.setName("Spaghetti Carbonara")
5.  myDish.setCostPrice(10)
6.  print(myDish.show())
```

Output:

Name: Spaghetti Carbonara Type: Main Retail Price: 16.0

Activity

Develop your own encapsulated class, from UML diagram design to creating and testing. Choose a topic of your own, or use one of these for inspiration.

- A class that stores information about an item in a shop and works out the salesperson commission.
- A class that stores information about hotels that calculates the commission they owe to third-party apps.

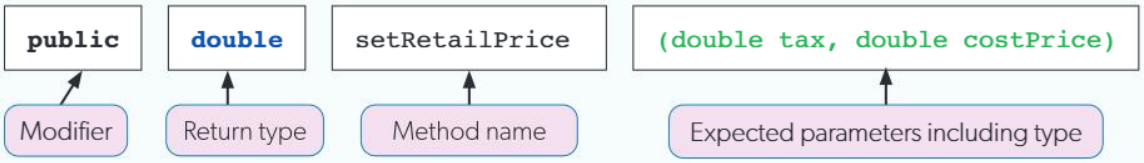
When you have developed your class, share your ideas with a partner. Do they agree with your choices? What advice can they offer you to improve your class?

Practical skills

Method headers in Java

The method header in Java consists of the access modifier, return time, and method signature of the method. Every method in Java needs to have a method header.

In an exam, it is a good idea to check how the method header has been presented as it could provide you a clue to the expected return type. Look at Figure 19.



▲ **Figure 19** A labelled example of a method header in Java

Modifier: Can either be: private (only seen by the class), protected (only seen by the package), or public (seen by all classes).

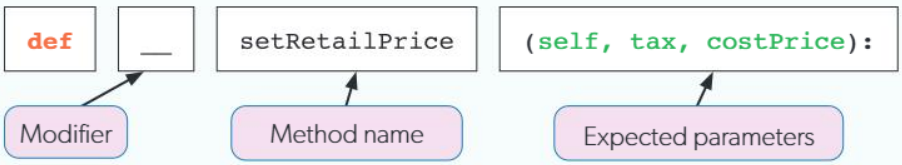
Return type: The type of value that will be returned from the method: int, double, boolean, String, ObjectType, int [], double [], String [], ObjectType [] or void. Void means that no information will come back from the method; for example, in set/mutator methods.

Method name (identifier): How the method will be referred to when being called. For example, in `x.getPrice()`, `the method name is getPrice`.

Parameter list: The values the methods expects to receive when the method is called. When calling the method, you replace the expected parameters with actual parameters (values) of that type. In the example in Figure 19, when calling the method you could say `y.setRetailPrice(0.25, 10)`.

Method headers in Python

Method headers in Python consist of the modifier, function name, and parameters. As Python dynamically **casts** the variable type, a Python function heading does not need to include the return type. In the exam, it is a good idea to check how the method heading has been presented. This will provide you with information such as parameter types.



▲ **Figure 20** A labelled example of a method header in Python

Modifier: Can either be private (only seen by the class), which is denoted by the double underscore __, or public with no preceding underscores (seen by all classes).

Method name (identifier): How the method will be referred to when being called. For example, in `x.getPrice(self)`, the name of the method is `getPrice`.

Parameter list: The values the method expects when the method is called. When calling the method, you replace the expected parameters with actual parameters (values) of that type. In the example in Figure 20, when calling the method you could say `y.__setRetailPrice(self, 0.25, 10)`.

Key term

Casting Forcing the variable to act as a specific variable type, e.g. forcing input from a user to act as a String. Useful when collecting data from a user in Python:

age = `int(input("How old is the person the game is for?"))`

Key term

Dot notation Using a dot (.) to indicate when you access the methods within a class using the following structure:

```
nameOfInstance.
methodName(parameters)
```

This section of the book introduces you to creating data structures, selection statements, loops, searching and sorting using objects. You may find these skills useful in your internal assessment.

TOK

When storing information in a list, what counts as knowledge? Is the knowledge the data within the structure or does the knowledge only occur when you perform an action upon it?

Practical skills

Constructing code with single classes

As classes allow you to develop more sophisticated programs, it is useful for you to understand how the different programming structures can be used together with classes. The structures are the same in all cases, but how you access the data to make comparisons, count occurrences, and search and sort the data is different. Throughout this section you should pay particular attention to the use of **dot notation**. Dot notation allows you to access the data held inside variables within objects.

Creating one-dimensional (1D) structures

Look back at topic B2 to section B2.2.2. You coded an array and list to store individual data values within a list structure. Using classes, you can store a lot of data within the list structure. Here are examples in Java and Python.

Creating a static 1D list in Java (Array)

The code for creating a static array follows this structure.

```
<Classname> [ ] <identifier> = new <Classname> [size of array];
```

Example: `Game [] myBoardGame = new Game [10];`

Creating a dynamic 1D list in Java (ArrayList)

The code for creating a dynamic `ArrayList` will follow this structure.

```
ArrayList <Classname> <identifier> = new ArrayList <Classname> ();
```

Example: `ArrayList <Game> myBoardGame = new ArrayList <Game>();`

Adding to a static 1D list in Java

The code for adding to a static array will follow this structure.

```
<identifier> <index> = new <Classname> <Parameters>;
```

Example: `myBoardGame[2] = new Game("Chicken vs the Egg", "Snoops", 12, 120);`

Adding to a dynamic 1D list in Java

The code for adding to a dynamic `ArrayList` will follow this structure.

```
<identifier>.add(new <Classname> <Parameters>);
```

Example: `myBoardGame.add(new Game("Chicken vs the Egg", "Snoops", 12, 120));`

Remove from a dynamic 1D list in Java (by index)

The code to remove an object from an `ArrayList` will follow this structure.

```
<identifier>.remove(index);
```

Example: `myBoardGame.remove(2);`

Worked example 2

A group of friends regularly play board games together. They would like to have a library of the games they have played so they can recommend them when other people ask. Using Java, develop a system for them to use.

Solution

The code below shows the creation of an `ArrayList` with the identifier `myBoardGame` in Java using objects of the `Game` class. The `Game` class is instantiated four times and each object added to the `myBoardGame ArrayList`. Once the objects have been added to `myBoardGame` one item is removed. The result of this is then printed out.

Game class in Java

```
1.  public class Game {
2.
3.      private String title;
4.      private String manufacturer;
5.      private int recommendedAge;
6.      private int approxGameTime;
7.
8.      public Game (String title, String manufacturer,
        int recommendedAge, int approxGameTime) {
9.
10.         this.title = title;
11.         this.manufacturer = manufacturer;
12.         this.recommendedAge = recommendedAge;
13.         this.approxGameTime = approxGameTime;
14.     }
15.     public String getTitle() {
16.         return title;
17.     }
18.     public String getManufacturer() {
19.         return manufacturer;
20.     }
21.     public int getRecommendedAge() {
22.         return recommendedAge;
23.     }
24.     public int getApproxGameTime() {
25.         return approxGameTime;
26.     }
27.
```

```
28.     public String toString() {
29.         return "Title: " + title + ", Manufacturer:
            " + manufacturer + ", Recommended Age: " +
            recommendedAge + ", Approximate Game Time:
            " + approxGameTime;
30.     }
31. }
```

Runner class in Java

```
1.  import java.util.ArrayList;
2.
3.  public class GameMain {
4.
5.      public static void main(String[] args) {
6.
7.          ArrayList <Game> myBoardGame = new
            ArrayList <Game>();
8.
9.          myBoardGame.add(new Game("Chicken vs the
            Egg", "Snoops", 12, 120));
10.         myBoardGame.add(new Game("Foxes and
            Hedgehogs", "Snoops", 14, 180));
11.         myBoardGame.add(new Game("Cards against
            History", "PopMoon", 18, 60 ));
12.         myBoardGame.add(new Game("Better or Worse",
            "RadioWaves", 5, 20));
13.
14.         myBoardGame.remove(2);
15.
16.         for (int i =0; i < myBoardGame.size();
            i++) {
17.
18.             System.out.println(myBoardGame.get(i).
                toString());
19.         }
20.     }
21. }
```

Output:

Title: Chicken vs the Egg, Manufacturer: Snoops, Recommended Age: 12, Approximate Game Time: 120

Title: Foxes and Hedgehogs, Manufacturer: Snoops, Recommended Age: 14, Approximate Game Time: 180

Title: Better or Worse, Manufacturer: RadioWaves, Recommended Age: 5, Approximate Game Time: 20

Creating a dynamic 1D list in Python (List)

The code for creating a List will follow this structure.

```
<identifier> = []
```

Example: `myBoardGame = []`

Adding to a dynamic 1D list in Python

The code for adding to a list will follow this structure.

```
<identifier>.append(<Classname> <Parameters>);
```

Example:
```
myBoardGame.append(Game("Chicken vs the Egg",
"Snoops", 12, 120))
```

Remove from dynamic 1D list in Python (by index)

The code to remove an object from a list will follow this structure.

```
del = <identifier> [index]
```

Example: `del myBoardGame[2]`

Worked example 3

A group of friends regularly play board games together. They would like to have a library of the games they have played so they can recommend them when other people ask. Using Python, develop a system for them to use.

Solution

The code below shows the creation of a **List** with the identifier **myBoardGame** in Python using objects of the **Game** class. The **Game** class is instantiated four times and each object added to the **myBoardGame List**. Once the objects have been added to **myBoardGame** one item is removed. The result of this is then printed out.

Game class in Python

```
1.  class Game:
2.
3.      def __init__(self,title,manufacturer,
        recommendedAge,approxGameTime):
4.          self.__title = title
5.          self.__manufacturer = manufacturer
6.          self.__recommendedAge = recommendedAge
7.          self.__approxGameTime = approxGameTime
8.
9.      def getTitle(self):
10.         return self.__title
11.
12.     def getManufacturer(self):
13.         return self.__manufacturer
```

```
14.
15.     def getRecommendedAge(self):
16.         return self.__recommendedAge
17.
18.     def getApproxGameTime(self):
19.         return self.__approxGameTime
20.
21.     def __str__(self):
22.         return "Title: " + self.__title + ",
            Manufacturer: " + self.__manufacturer
            + ", Recommended Age: " + str(self.__
            recommendedAge) + ", Approximate Game Time:
            " + str(self.__approxGameTime)
```

Runner class in Python

```
1.  from Game import Game
2.
3.  myBoardGame = []
4.
5.  myBoardGame.append(Game("Chicken vs the Egg",
    "Snoops", 12, 120))
6.  myBoardGame.append(Game("Foxes and Hedgehogs",
    "Snoops", 14, 180))
7.  myBoardGame.append(Game("Cards against History",
    "PopMoon", 18, 60))
8.  myBoardGame.append(Game("Better or Worse",
    "RadioWaves", 5, 20))
9.
10. del myBoardGame[2]
11.
12. for x in myBoardGame:
13.     print(x.__str__())
```

Output:

Title: Chicken vs the Egg, Manufacturer: Snoops, Recommended Age: 12, Approximate Game Time: 120

Title: Foxes and Hedgehogs, Manufacturer: Snoops, Recommended Age: 14, Approximate Game Time: 180

Title: Better or Worse, Manufacturer: RadioWaves, Recommended Age: 5, Approximate Game Time: 20

Activity

Develop a data repository to store information about something you love. You should have a class to represent the information and then several instances of the class in a list. Try having different options: creating a new instance, deleting an instance, searching for an instance, sorting the instances.

Creating two-dimensional (2D) data structures

Developing dynamic 2D data structures can become complex. Accessing the elements of the class within the two-dimensional lists requires a lot of knowledge regarding how the list is traversed and how to access different data within the classes.

Creating a dynamic 2D list in Java (ArrayList)

The code for creating a dynamic `ArrayList` follows this structure.

```
ArrayList <ArrayList<Classname>> <identifier> = new
ArrayList <ArrayList<Classname>> ();
```

Repeat for each row required.

```
<identifier>.add(new ArrayList<className>);
```

Example:

```
ArrayList <Arraylist <Plane>> parking = new ArrayList
<ArrayList<Plane>>();
parking.add(new ArrayList<Plane>());
```

Adding to a dynamic 2D list in Java

The code for adding to a dynamic `ArrayList` will follow this structure.

```
ArrayList <classname> <inner_list>= new ArrayList
<Classname>();
<identifier>.add(<inner_list>);
```

Example:

```
ArrayList <Plane> rowOne = new ArrayList<Plane>();
    rowOne.add(new Plane("BCX 292", "Embrarer 195"));
    rowOne.add(new Plane("BCZ 182", "Embrarer 190"));
    rowOne.add(new Plane("BHT 894", "Embrarer 195"));
```

Remove from dynamic 2D list in Java (by index)

The code to remove an object from an `ArrayList` will follow this structure.

```
<identifier>.get(rowIndex).remove(columnindex);
```

Example: `parking.get(0).remove(1);`

Worked example 4

Airports keep careful track of where planes are parked when they are on the ground.

The planes at an airport are stored in a table-like structure. Develop a program to help the airport achieve this.

Solution

The worked example shows a dynamic 2D `ArrayList` holding objects in practice. The program enables the user to add planes to the table to show where they are parked in the airport.

Class in Java

```
1.  public class Plane {
2.
3.      private String planeNumber;
4.      private String planeType;
5.
6.      public Plane (String planeNumber, String
        planeType) {
7.
8.          this.planeNumber = planeNumber;
9.          this.planeType = planeType;
10.     }
11.     public String getNumber() {
12.         return planeNumber;
13.     }
14.     public String getType() {
15.         return planeType;
16.     }
17.     public String toString() {
18.         return "Plane Number: " + planeNumber + ",
            Plane Type: " + planeType;
19.     }
```

Main method in Java

```
1.  import java.util.ArrayList;
2.
3.  public class MainMethod {
4.
5.      public static void main(String[] args) {
6.
7.          ArrayList <ArrayList <Plane>> parking =
            new ArrayList<ArrayList<Plane>>();
8.
9.          parking.add(new ArrayList<Plane>());
10.         parking.add(new ArrayList<Plane>());
11.         parking.add(new ArrayList<Plane>());
12.
13.         ArrayList <Plane> rowOne = new
            ArrayList<Plane>();
14.         rowOne.add(new Plane("BCX 292", "Embrarer
            195"));
15.         rowOne.add(new Plane("BCZ 182", "Embrarer
            190"));
```

```
16.         rowOne.add(new Plane("BHT 894", "Embrarer
            195"));
17.
18.         ArrayList <Plane> rowTwo = new
            ArrayList<Plane>();
19.         rowTwo.add(new Plane("XSK 922", "Airbus
            330"));
20.         rowTwo.add(new Plane("IUY 293", "Airbus
            350"));
21.         rowTwo.add(new Plane("NFF 912", "Boeing
            777"));
22.         rowTwo.add(new Plane("XSJ 883", "Airbus
            350"));
23.
24.         ArrayList <Plane> rowThree = new
            ArrayList<Plane>();
25.         rowThree.add(new Plane("KSJ 948", "Airbus
            380"));
26.         rowThree.add(new Plane("PSJ 829", "Boeing
            747"));
27.
28.         parking.add(0, rowOne);
29.         parking.add(1, rowTwo);
30.         parking.add(2,rowThree);
31.
32.         parking.get(0).remove(1);
33.
34.         for (int i = 0; i <parking.size(); i++) {
35.             for (int j = 0; j <parking.get(i).
            size(); j++) {
36.                 if (parking.get(i).get(j).
                    getType().contains("Airbus")){
37.                 System.out.println(parking.get(i).
                    get(j).toString() + ", At Row " + i
                    + ", Space: "+j);
38.                 }
39.             }
40.         }
41.     }
42. }
```

Output:

Plane Number: XSK 922, Plane Type: Airbus 330, At Row 1, Space: 0

Plane Number: IUY 293, Plane Type: Airbus 350, At Row 1, Space: 1

Plane Number: XSJ 883, Plane Type: Airbus 350, At Row 1, Space: 3

Plane Number: KSJ 948, Plane Type: Airbus 380, At Row 2, Space: 0

ATL Communication skills

It can be a challenge to think about 2D lists and how they operate logically. Using clear communication with comments and correct syntax can help to make this easier

Comments look different in different languages. In Java you use a double forward slash // and in Python you use a hash #.

Choose one of the activities that you have coded earlier in this unit and practice writing comments. Examples in both languages are shown here to give you some ideas.

Java

```
// loop through the coffee shop list
for (int i = 0; i <coffeeShop.size(); i++) {
   // check if the coffee shop is in the given town
   and has
   // a star rating over the max rating
  if (coffeeShop.get(i).getTown().equals(searchTerm) &&
  coffeeShop.get(i).getOverallRating() > maxRating ) {
     // if over the max rating set new max
        maxRating = coffeeShop.get(i).
        getOverallRating();
     // get index of new max
        max = i;
   }
}
```

Python

```
#Code to loop through the board game list
for x in myBoardGame:
    # if statement to check if the recommended age is
    less than
    # the age provided by the user
    if x.getRecommendedAge()<= age:
        #print the game information out to screen
        print(x.toString())
```

Selection statements with objects

Previously, when using selection statements, you compared a variable with another variable. Being able to compare objects rather than variables allows you to develop more sophisticated programs. You can compare objects and make decisions using the information within the objects by utilizing the accessor methods. To do this, you need to know the identifier of the object and the accessor methods available to you.

Activity

Revisit the program you made to create a map showing a seating plan for a play in B2.2.1. Replace the seat number (the primitive data type) with one of these options:

- a class to represent the ticket that sold the seat
- a class to represent the seat (cost, number, availability)
- a class to represent the person that bought the seat.

Identify the attributes and behaviours required to sell the seat. For example, the user will need to choose their preferred seat. The program needs to check that the seat exists and if the seat is available. If the seat is been sold, the program needs to change the availability to unavailable.

Construct a program that will enable a seat map to be represented by seat objects.

In Java, the code for creating a selection statement using if statements follows this structure.

```
if (<identifier>.<getMethodName()>.equals(<search term>) {
 // code here
}
else if (<identifier>.<getMethodName()>.equals(<search term>){
 //code here
}
else {
 // code here
}
```

Note: `.equals()` has been used if the search term is a String or object data type. == should be used for Java primitive data types. The else statements are optional.

Creating a selection statement using if statements in Java

```
1.  if (temp.getManufacturerName().equals("Snoops"){
2.      System.out.println("Game has long delivery
        time");
3.  }
4.  else if (temp.getManufacturerName().
    equals("PopMoon"){
5.      System.out.println("Supports next day
        delivery");
6.  }
7.  else {
8.      System.out.println("Average delivery time 3 –
        5 working days"):
9.  }
```

Notice the use of the identifier to get the instance of the object and the use of the dot notation to access the method.

Worked example 5

Using the Game class from Worked example 2, write Java code that allows the user to enter an age and then print games that are suitable for that age group.

Solution

You need to create a for loop, which will allow you to search through the objects. You need to ask the user their age and use this to produce a recommendation.

```
1.  import java.util.ArrayList;
2.  import java.util.Scanner;
3.
4.  public class GameMain {
5.
6.      public static void main(String[] args) {
7.
8.          ArrayList <Game> myBoardGame = new
            ArrayList <Game>();
9.          Scanner input = new Scanner(System.in);
10.
11.         myBoardGame.add(new Game("Chicken vs the
            Egg", "Snoops", 12, 120));
12.         myBoardGame.add(new Game("Foxes and
            Hedgehogs", "Snoops", 14, 180));
13.         myBoardGame.add(new Game("Cards against
            History", "PopMoon", 18, 60 ));
14.         myBoardGame.add(new Game("Better or Worse",
            "RadioWaves", 5, 20));
15.         myBoardGame.add(new Game("March of the
            Flamingos", "RadioWaves", 10, 15));
16.         myBoardGame.add(new Game("Travel Spelling",
            "PopMoon", 9, 20 ));
17.
18.         System.out.println("How old is the person
            this game is for?");
19.         int age = input.nextInt();
20.
21.         for (int i = 0; i < myBoardGame.size();
            i++) {
22.
23.             if (myBoardGame.get(i).
                getRecommendedAge()<= age) {
24.                 System.out.println(myBoardGame.
                    get(i).toString());
```

```
            }
        }
    }
}
```

In this example, the most important part to note is the if statement, which uses the dot notation (.) to gain access to the instance. Also note that the dot notation uses the accessor method to access the value of the variable.

In Python, the code for creating a selection statement using if statements follows this structure (remember elif and else are optional):

```
if (<identifier>.<getMethodName()>= =<search term>):
  // code here
elif (<identifier>.<getMethodName()>= =<search term>):
  //code here
else:
  // code here
```

Creating a selection statement using if statements in Python

```
if (myBoardGame[0].getManufacturer()=="Snoop"):
    print("Game has a long delivery time")

elif (myBoardGame[0].getManufacturer()=="PopMoon"):
    print("Next day delivery supported")

else:
    print("Average delivery time 3 - 5 working
    days")
```

Notice the use of the identifier to get the instance of the object and the use of the dot notation to access the method.

Worked example 6

Using the Game class from Worked example 3, write Python code that allows the user to enter an age and then print games that are suitable for that age group.

Solution

You need to create a for loop, which will allow you to search through the objects. You need to ask the user their age and use this to produce a recommendation.

This code also uses a for loop to search through the game objects.

```
1.  from Game import Game
2.
3.  myBoardGame = []
4.
5.  myBoardGame.append(Game("Chicken vs the Egg",
    "Snoops", 12, 120))
6.  myBoardGame.append(Game("Foxes and Hedgehogs",
    "Snoops", 14, 180))
7.  myBoardGame.append(Game("Cards against History",
    "PopMoon", 18, 60))
8.  myBoardGame.append(Game("Better or Worse",
    "RadioWaves", 5, 20))
9.  myBoardGame.append(Game("March of the Flamingoes",
    "RadioWaves", 10, 15))
10. myBoardGame.append(Game("Travel Spelling",
    "PopMoon", 9, 20))
11.
12. age = int(input ("How old is the person the game
    is for?"))
13.
14. for x in myBoardGame:
15.     if x.getRecommendedAge()<= age:
16.         print(x.toStr())
```

Notice the use of casting to force the input to be an integer and the use of the dot notation (.) to access the method.

Repetition statements with objects

Loops using objects are like loops without objects. However, it is important to note that in many cases when you use loops to access an object in a list, you need to use the index to get the current instance of the object in order to access the methods.

In Java, you can loop through a list using a for loop. The index of this list is used to access the current instance of the object in the list. The code follows this structure:

```
for (int i = 0, i < <List_identifier>.size(); i ++){
    <list_identifier>.get(i) //gets current instance of object
}
```

Looping a list using a for loop in Java

```
for (int i = 0; i < myBoardGame.size(); i++){
      myBoardGame.get(i);
}
```

You can also loop through the list using an iterator. The code must be told what information it is looking for by casting it to an object type. The `.next()` method can be used to access the current instance in the list. The code follows this structure:

Iterator <classType> <iterator_identifier> = <List_identifier>.iterator();

while (<iterator_identifier>.hasNext()) {

    <classType> temp = <iterator_identifier>.next() // store current instance in a variable called temp

}



Looping through a list using an iterator in Java

```
1.  Iterator <Game> it = myBoardGame.iterator();
2.
3.   while (it.hasNext()) {
4.      Game temp = it.next();
5.   }
```

Searching with objects

Searching algorithms operate in the same way, whether you use single values or objects. How you access the data you are searching for will be different.

For a **linear search** in Java, the code follows this structure:

```
int index = -1;  // set at -1 in case it is not found
for (int i = 0; i < <list_identifier>.size(); i++){
  if (<list_identifier>.get(i).equals(searchTerm)){
    index = i;
  }
}
if (index > 0){
    System.out.println("Item found at: " + index);
}
else {
    System.out.println("Item not found");
}
```



Linear search in Java

```
1.  Scanner input = new Scanner(System.in);
2.  System.out.println("What shop are you looking for");
3.  String searchTerm = input.nextLine();
4.
5.  int index = -1;
6.
7.  for (int i = 0; i < coffeeShop.size(); i++) {
8.    if (coffeeShop.get(i).getShopName().
      equals(searchTerm)) {
```



```
9.       index = i;
10.    }
11. }
12.
13. if (index > 0) {
14.    System.out.println("Item found at " + index);
15. }
16. else {
17.    System.out.println("Item not found");
18. }
```

For a **binary search** in Java, the code follows this structure:

```
Index = binarySearch (<list_identifier>,<searchTerm>); //
call method

private static int binarySearch (ArrayList <classtypes>
<list_identfier>, <variable type> <search term>) {

  int left = 0, right = <list_Identifier>.size()-1;

  while(left <= right) {
    int mid = left + (right - left) /2;

    if (<list_identifier>.get(mid).getMethodName().
    equals(<searchTerm>)) {
      return mid;
    }

    if((<list_identifier>.get(mid).getMethodName().
    compareTo(<searchTerm>)>0) {
      left = mid + 1;
    }
    else {
      right = mid - 1;
    }
  }
  return -1;
}
```

Binary search in Java

```
1.  private static int binarySearch (ArrayList <Review>
    coffeeShop, String searchTerm) {
2.
3.      int left = 0, right = coffeeShop.size()-1;
4.
5.      while(left <= right) {
6.        int mid = left + (right - left) /2;
7.
8.        if (coffeeShop.get(mid).getShopName().
          equals(searchTerm)) {
9.          return mid;
```

```
10.      }
11.      if (coffeeShop.get(mid).getShopName().
         compareTo(searchTerm)>0) {
12.        left = mid + 1;
13.      }
14.      else {
15.        right = mid - 1;
16.      }
17.    }
18.    return -1;
19. }
```

Worked example 7

Using Java, develop a recommendation system that allows users to add coffee shops to a list and then search for recommendations based on different criteria.

Solution

The program shows details of coffee shops in different towns. The program allows the user to rate the food and the coffee in the coffee shop. These ratings are then used to create an overall rating. The program will then make use of a linear search to find the highest-rated coffee shop in a given town.

Class for the review in Java

```
1.  public class Review {
2.
3.      private String shopName;
4.      private String town;
5.      private int coffeeRating;
6.      private int foodRating;
7.      private double overallRating;
8.
9.      public Review (String shopName, String town,
        int coffeeRating, int foodRating) {
10.
11.         this.shopName = shopName;
12.         this.town = town;
13.         this.coffeeRating = coffeeRating;
14.         this.foodRating = foodRating;
15.         this.setOverallRating();
16.
17.     }
18.     public String getShopName() {
19.         return shopName;
20.     }
```

```
21.     public String getTown() {
22.         return town;
23.     }
24.     public int getCoffeeRating() {
25.         return coffeeRating;
26.     }
27.     public int getFoodRating() {
28.         return foodRating;
29.     }
30.     public double getOverallRating() {
31.         return overallRating;
32.     }
33.     public void setCoffeeRating (int x) {
34.         if (x < 5) {
35.             coffeeRating = x;
36.             this.setOverallRating();
37.         }
38.         else {
39.             System.out.println("not a valid
                rating");
40.         }
41.     }
42.     public void setFoodRating (int x) {
43.         if (x < 5) {
44.             foodRating = x;
45.             this.setOverallRating();
46.         }
47.         else {
48.             System.out.println("not a valid
                rating");
49.         }
50.     }
51.     public void setOverallRating () {
52.         overallRating = (coffeeRating * 0.75)
            +  (foodRating * 0.25);
53.     }
54.     public String toString () {
55.         return "Name of Shop: " + shopName + " in "
            + town + ". Coffee Rating: " + coffeeRating
            + ", Food Rating: " + foodRating + ",
            Overall Rating: " + overallRating;
56.     }
57. }
```

Main method for this class in Java

```
1.  public class CoffeeShopMain {
2.
3.      public static void main(String[] args) {
4.
5.          ArrayList <Review> coffeeShop = new
            ArrayList <Review>();
6.
7.          coffeeShop.add(new Review("Belle Cafe",
            "Valtorcia", 4,4));
8.          coffeeShop.add(new Review("Brewed
            Awakening", "Nord Lozarn", 2,2));
9.          coffeeShop.add(new Review("Cattucino",
            "Valtorcia", 2,2));
10.         coffeeShop.add(new Review("Coffee Haus",
            "Nord Lozarn", 3,4));
11.         coffeeShop.add(new Review("Kickstart",
            "Montebello", 5,3));
12.         coffeeShop.add(new Review("Latte Da",
            "Montebello", 4,4));
13.         coffeeShop.add(new Review("Metro Mocha",
            "Alpenfeld", 5,3));
14.         coffeeShop.add(new Review("The Apothek",
            "Nord Lozarn", 3,4));
15.         coffeeShop.add(new Review("The Grind", "Nord
            Lozarn", 5,4));
16.         coffeeShop.add(new Review("The Java Guys",
            "Alpenfeld", 2,1));
17.
18.         Scanner input = new Scanner(System.in);
19.         System.out.println("What town are you
            looking for");
20.         String searchTerm = input.nextLine();
21.
22.         int max = -1;
23.         double maxRating = 0;
24.
25.         for (int i = 0; i <coffeeShop.size();
            i++) {
26.
27.             if (coffeeShop.get(i).getTown().
                equals(searchTerm) && coffeeShop.get(i).
                getOverallRating() > maxRating ) {
28.
29.                 maxRating = coffeeShop.get(i).
                    getOverallRating();
30.                 max = i;
```

```
31.             }
32.         }
33.         if (max > 0) {
34.             System.out.println("The best coffee
                shop in " + searchTerm + ", is: " +
                coffeeShop.get(max));
35.         }
36.         else {
37.             System.out.println("No shop found");
38.         }
39.     }
40. }
```

For a **linear search** in Python, the code follows this structure:

```
foundIndex = -1
searchTerm = input("<question>")

for i, <variable> in enumerate(<list_identifier>):
  if (<variable> == searchTerm):
    foundIndex = i

if (foundIndex != -1):
  print("<statement" + str(foundIndex))
else:
  print("<statement>")
```

Linear search in Python

```
1.  searchTerm = input("What shop are you looking
for?")
2.  index = -1
3.
4.  for i, shop in enumerate(coffeeShop):
5.      if shop.getShopName() == searchTerm:
6.          index = i
7.
8.  if (index > 0):
9.      print ("Item found at " + str(index))
10. else:
11.     print("Item not found")
```

For a **binary search** in Python, the code follows this structure:

```
def binarySearch (searchArray, left, right, searchTerm):
  while left < right:

    midPoint = left + (right - left) // 2

    #check if searchTerm is present at the midPoint
    if searchArray[midPoint] == searchTerm:
      return midPoint
```

```
    #if searchTerm is greater than midPoint, ignore the
    left half
    elif searchArray[midPoint] < searchTerm:
      left = midPoint + 1
    #if searchTerm is smaller than midPoint, ignore the
    right half
    else:
      right = midPoint -1
# if the searchTerm is not found return -1
  return -1
```

Binary search in Python

```
1.  def binarySearch(searchList, left, right,
    searchTerm):
2.      while left < right:
3.          midPoint = left + (right - left) // 2
4.
5.          if searchList[midPoint].getShopName() ==
            searchTerm:
6.              return midPoint
7.          elif searchList [midPoint].getShopName() <
            searchTerm:
8.              left = midPoint + 1
9.          else:
10.             right = midPoint -1
11.
12.     return -1
```

Worked example 8

Using Python, develop a recommendation system that allows users to add coffee shops to a list and then search for recommendations based on different criteria.

Solution

The program shows details of coffee shops in different towns. The program allows the user to rate the food the coffee in the coffee shop. These ratings are then used to create an overall rating. The program will then make use of a linear search to find the highest-rated coffee shop in a given town.

Class for the review in Python

```
1.  class Review:
2.
3.      def __init__(self,shopName, town, coffeeRating,
        foodRating):
4.          self.__shopName = shopName
5.          self.__town = town
6.          self.__coffeeRating = coffeeRating
```

```
7.          self.__foodRating = foodRating
8.          self.__overallRating = 0
9.          self.setOverallRating()
10.
11.     def getShopName(self):
12.         return self.__shopName
13.
14.     def getTown(self):
15.         return self.__town
16.
17.     def getCoffeeRating(self):
18.         return self.__coffeeRating
19.
20.     def getFoodRating(self):
21.         return self.__foodRating
22.
23.     def getOverallRating(self):
24.         return self.__overallRating
25.
26.     def setCoffeeRating(self, x):
27.         if (x < 5):
28.             self.__coffeeRating = 5
29.             self.setOverallRating()
30.         else:
31.             print("This is not a valid rating")
32.
33.     def setFoodRating(self, x):
34.         if (x < 5):
35.             self.__foodRating = 5
36.             self.setOverallRating()
37.         else:
38.             print("This is not a valid rating")
39.
40.     def setOverallRating(self):
41.         self.__overallRating = (self.__coffeeRating
            * 0.75) + (self.__foodRating * 0.25)
42.
43.     def __str__(self):
44.         return "Name of Shop: " + self.__shopName
            + " in " + self.__town + ", coffee rating:
            " + self.__coffeeRating + ", food rating: "
            + self.__foodRating + ", overall rating: "
            + self.__overallRating
```

Main method for this class in Python

```
from Review import Review

coffeeShop = []

coffeeShop.append(Review("Belle Cafe", "Valtorcia", 4,4))
coffeeShop.append(Review("Brewed Awakening", "Nord Lozarn", 2,2))
coffeeShop.append(Review("Cattucino", "Valtorcia", 2,2))
coffeeShop.append(Review("Coffee Haus", "Nord Lozarn", 3,4))
coffeeShop.append(Review("Kickstart", "Montebello", 5,3))
coffeeShop.append(Review("Latte Da", "Montebello", 4,4))
coffeeShop.append(Review("Metro Mocha", "Alpenfeld", 2,2))
coffeeShop.append(Review("The Apothek", "Nord Lozarn", 3,4))
coffeeShop.append(Review("The Grind", "Nord Lozarn", 5,4))
coffeeShop.append(Review("The Java Guys", "Alpenfeld", 2,1))

searchTerm = input("What town are you looking for?")
index = -1
highest = 0

for i, shop in enumerate(coffeeShop):
    if coffeeShop[i].getTown() == searchTerm and coffeeShop[i].getOverallRating() > highest:
        index = i
        higher = shop.getOverallRating()

if (index > 0):
    print ("The highest rated shop in " + searchTerm + " :" + coffeeShop[index].getShopName())
else:
    print("Item not found")
```

TOK

People often use apps and websites to find the "best" restaurant in town or the "best" tourist attractions. These take many different forms and can be confusing for some users. The coffee shop program developed in Worked examples 7 and 8 is a simple version of programs that are used regularly by consumers.

What challenges are faced by programmers when they are communicating knowledge?

ATL Self-management and social skills

Searching and sorting can be quite challenging in code. It is important to understand what is happening at each stage of the process so you can identify if something has gone wrong. Using debugging techniques such as print statements at key points helps you to communicate the issues in the program, aiding the testing and evaluation cycle.

Review one of the programs you have written for an activity in this unit. Write down how you could use debugging techniques to improve your code. Work with a partner to check through each other's code and implement your ideas.

What were the benefits of working with a partner? Was there anything that made working with a partner challenging? Would you do anything different next time?

Sorting with objects

Sorting objects can be a challenging task depending on what you wish to use to sort the object. In Python and Java, sorting using numbers is relatively easy. The algorithm follows the same steps as non-object sorting. In Python, since a String is treated as a primitive data type, you can use the primitive comparison operators (>, >=, <, <=, ==, !=).

Java is different: a String is an object data type, so you have to use a comparison method called the `compareTo` method. The `compareTo` method allows you to compare two objects to determine if one object is lexicographically (using the order of the alphabet) less than or greater than the other object.

When comparing the two objects using `compareTo` then the method will return values as follows.

- **0** if the two objects are equal.
- **< 0** if the object is lexicographically less than the compared object.
- **> 0** if the object is lexicographically greater than the compared object.

The following two examples show how to sort objects alphabetically using bubble sort: first in Java, then in Python.

In Java, because you are using objects that use pointers, you need a helper method to actually make the swap. This is because you must remove the element from its old position and re-add the element in its new position. The sort algorithm and the helper method will look similar to this:

```
<className>tempOne;
<className>tempTwo;
int fIndex; //first index
int sIndex; //second index

for(int i = 0; i < <arraylist_identifier>.size()-1; i++){
      for(int j = 0; j < arraylist_identifier>.size()-i-1;
      j++){
```

```
            if
(<arraylist_identifier>.get(j).compareTo(<arraylist_
identifier>.get(j+1))>0){

                    tempOne = <arraylist_identifier>.
                    get(j);
                     fIndex = j;
                    tempTwo = <arraylist_identifier>.
                    get(j + 1);
                    sIndex = j + 1;
       swap(<arraylist_identifier>, fIndex, sIndex,
       tempOne, tempTwo);
                    }
    }
  }
}

public static void swap (ArrayList<className> <arraylist_
identifier>, int current, int next, <className>
currentObj, <className> nextObj){

                    <arraylist_identifier>.
                    remove(current);
                    <arraylist_identifier>.add(current,
                    nextObj);
                    <arraylist_identifier>.remove(next);
                    <arraylist_identifier>.remove(next,
                    currentObj);
}
```



Bubble sort with objects in Java

```
1.  student temp;
2.  student temp2;
3.  int fIndex;
4.  int sIndex;
5.
6.  for (int i = 0; i < studentList.size()-1; i++) {
7.      for (int j = 0; j < studentList.size()-i-1;
        j++) {
8.
9.          if (studentList.get(j).
            compareTo(studentList.get(j+1))>0) {
10.
11.             temp = studentList.get(j);
12.             fIndex = j;
13.             temp2 = studentList.get(j + 1);
14.             sIndex = j+1;
15.             swap(studentList, fIndex, sIndex,
                temp, temp2);
16.         }
17.     }
18.  }
```

```
19.  for (int i = 0; i <studentList.size(); i++) {
20.      System.out.println(studentList.get(i));
21.  }
22. }
23. public static void swap(ArrayList <student>
    studentList, int current, int next, student
    currentStudent, student nextStudent) {
24.
25.      studentList.remove(current);
26.      studentList.add(current, nextStudent);
27.      studentList.remove(next);
28.      studentList.add(next, currentStudent);
29.
30.     }
31. }
```

Worked example 9

Using Java, write a program that allows teachers to add students to a student list. They need to be able to include the student's name, age, and any allergies they have. The output must print students in alphabetical order, so that the teachers can find the information as quickly as possible.

Solution

One way to do this would be to develop a class to store the data and then use a list to add an instance of the class for each student. Teachers can then search and sort the data according to their needs.

Student class in Java

```
1.  public class Student {
2.
3.      private String name;
4.      private String surname;
5.      private int age;
6.      private boolean allergy;
7.      private String allergyTo;
8.
9.      public Student (String surname, String name,
        int age, boolean allergy, String allergyTo) {
10.
11.         this.name = name;
12.         this.surname = surname;
13.         this.age = age;
14.         this.allergy = allergy;
```

```
15.         this.allergyTo = allergyTo;
16.
17.     }
18.     public String getName() {
19.         return name;
20.     }
21.     public String getSurname() {
22.         return surname;
23.     }
24.     public int getAge() {
25.         return age;
26.     }
27.     public boolean getAllergy() {
28.         return allergy;
29.     }
30.     public String getAllergyTo() {
31.         return allergyTo;
32.     }
33.     public int compareTo(Student sobj) {
34.         return this.surname.compareTo(sobj.
            getSurname());
35.     }
36.
37.     public String toString() {
38.         return surname + ", " + name + ", Age: "
            + age + ", Allergy: " + allergy + " " +
            allergyTo;
39.     }
40. }
```

Main method in Java

```
1.  public class MainMethod {
2.
3.      public static void main(String[] args) {
4.
5.          ArrayList<Student> studentList = new
            ArrayList <Student>();
6.
7.          studentList.add(new Student("Chai",
            "Yuhan", 15, false, ""));
8.          studentList.add(new Student("Renaudie",
            "Thomas", 16, true, "Shellfish"));
9.          studentList.add(new Student("Shyamsukha",
            "Priya", 15, false, ""));
10.         studentList.add(new Student("Urcum",
            "Kerem", 15, false, ""));
```

```
11.         studentList.add(new Student("Zuaiter",
            "Sebastian", 16, true, "Shellfish"));
12.         studentList.add(new Student("Wallstrom",
            "Tobias", 16, false, ""));
13.         studentList.add(new Student("Cova",
            "Francesca", 16, false, ""));
14.         studentList.add(new Student("Buchli",
            "Maria", 17, true, "Pineapple"));
15.         studentList.add(new Student("Joinson",
            "Ava", 17, false, ""));
16.         studentList.add(new Student("Halldorsson",
            "Kristoffer", 15, false, ""));
17.
18.         Student temp;
19.         Student temp2;
20.         int fIndex;
21.         int sIndex;
22.
23.         for (int i = 0; i < studentList.size()-1;
            i++) {
24.             for (int j = 0; j < studentList.
                size()-i-1; j++) {
25.
26.
27.
28.
29.                     if (studentList.get(j).
                        compareTo(studentList.
                        get(j+1))>0) {
30.                         temp = studentList.get(j);
31.                         fIndex = j;
32.                         temp2 = studentList.get(j
                            + 1);
33.                         sIndex = j+1;
34.                         swap(studentList, fIndex,
                            sIndex, temp, temp2);
35.                     }
36.
37.                 }
38.             }
39.
40.
41.
42.         for (int i = 0; i <studentList.size();
            i++) {
43.             System.out.println(studentList.get(i));
44.         }
```

```
45.
46.
47.     }
48.     public static void swap(ArrayList <Student>
        studentList, int current, int next, Student
        currentStudent, Student nextStudent) {
49.
50.             studentList.remove(current);
51.             studentList.add(current, nextStudent);
52.             studentList.remove(next);
53.             studentList.add(next, currentStudent);
54.
55.     }
56.
57. }
```

Output:

Buchli, Maria, Age: 17, Allergy: true Pineapple

Chai, Yuhan, Age: 15, Allergy: false

Cova, Francesca, Age: 16, Allergy: false

Halldorsson, Kristoffer, Age: 15, Allergy: false

Joinson, Ava, Age: 17, Allergy: false

Renaudie, Thomas, Age: 16, Allergy: true Shellfish

Shyamsukha, Priya, Age: 15, Allergy: false

Urcum, Kerem, Age: 15, Allergy: false

Wallstrom, Tobias, Age: 16, Allergy: false

Zuaiter, Sebastian, Age: 16, Allergy: true Shellfish

In Python, because of the way that lists are stored and String data types are treated, the bubble and selection sort do not require the use of a helper method.

The code is similar to this:

```
for i in range (0, len (<list_identifier>)):

  for j in range (0, len(<list_identifier>)-i-1):

    if (<list_identifier>[j].<method_name> > <list_
    identifier>[j+1].<method_name>):

      temp = <list_identifier>[j]

      <list_identifier> [j] = <list_identifier> [j+1]

      <list_identifier> [j+1] = temp
```

Bubble sort with objects in Python

```
1.  for i in range(0, len(studentList)):
2.      for j in range(0, len(studentList)-i-1):
3.          if (studentList[j].
            getName()>studentList[j+1].
            getName()):
4.              temp = studentList[j]
5.              studentList[j] = studentList[j+1]
6.              studentList[j+1] = temp
```

Worked example 10

Using Python, write a program that allows teachers to add students to a student list. They need to be able to include the student's name, age, and any allergies they have. The output must print students in alphabetical order, so that the teachers can find the information as quickly as possible.

Solution

One way to do this would be to develop a class to store the data and then use a list to add an instance of the class for each student. Teachers can then search and sort the data according to their needs.

Student class in Python

```
1.  class student:
2.
3.      def __init__ (self, name, surname, age,
        allergy, allergyTo):
4.          self.__name  = name
5.          self.__surname = surname
6.          self.__age = age
7.          self.__allergy = allergy
8.          self.__allergyTo = allergyTo
9.
10.     def getName(self):
11.         return self.__name
12.
13.     def getSurname(self):
14.         return self.__surname
15.
16.     def getAge(self):
17.         return str(self.__age)
18.
19.     def getAllergy(self):
20.         return str(self.__allergy)
21.
22.     def getAllergyTo(self):
23.         return self.__allergyTo
24.
```

```
    def __str__ (self):
        return "Surname: " + self.__surname + ",
        Name: " + self.__name + ", Age: "
        + str(self.__age) + ", Allergy: " +
        str(self.__allergy) + ", Allergy To: " +
        self.__allergyTo
```

Main method in Python

```
from student import student

studentList = []

studentList.append(student("Chai", "Yuhan", 15,
False, ""))
studentList.append(student("Renaudie", "Thomas",
16, True, "Shellfish"))
studentList.append(student ("Shyamsukha", "Priya",
15, False, ""))
studentList.append(student("Urcum", "Kerem", 15,
False, ""))
studentList.append(student ("Zuaiter", "Sebastian",
16, True, "Shellfish"))
studentList.append(student ("Wallstrom", "Tobias",
16, False, ""))
studentList.append(student ("Cova", "Francesca",
16, False, ""))
studentList.append(student("Buchli", "Maria", 17,
True, "Pineapple"))
studentList.append(student("Joinson", "Ava", 17,
False, ""))
studentList.append(student("Halldorsson",
"Kristoffer", 15, False, ""))


for i in range(0, len(studentList)-1):
    for j in range(i+1, len(studentList)-i-1):
        if (studentList[j].
        getName()>studentList[j+1].getName()):
            temp = studentList[j]
            studentList[j] = studentList[j+1]
            studentList[j+1] = temp


for x in studentList:
    print(x.__str__())
```

Output:

Surname: Yuhan, Name: Chai, Age: 15, Allergy: False, Allergy To:

Surname: Thomas, Name: Renaudie, Age: 16, Allergy: True, Allergy To: Shellfish

Surname: Priya, Name: Shyamsukha, Age: 15, Allergy: False, Allergy To:

Surname: Francesca, Name: Cova, Age: 16, Allergy: False, Allergy To:

Surname: Maria, Name: Buchli, Age: 17, Allergy: True, Allergy To: Pineapple

Surname: Kristoffer, Name: Halldorsson, Age: 15, Allergy: False, Allergy To:

Surname: Ava, Name: Joinson, Age: 17, Allergy: False, Allergy To:

Surname: Kerem, Name: Urcum, Age: 15, Allergy: False, Allergy To:

Surname: Tobias, Name: Wallstrom, Age: 16, Allergy: False, Allergy To:

Surname: Sebastian, Name: Zuaiter, Age: 16, Allergy: True, Allergy To: Shellfish

Activity

Ms Ava is a teacher who works with primary school children. Every week, she sets a mini-quiz that is a mixture of trivia questions and questions about the topics the students have studied. Students work in table groups to answer the questions. At the end of the month, the table at the top of the leader board gets 15 extra minutes of play time.

Put together everything you know and develop a more complex program to help Ms Ava.

Practice questions

1. Identify two disadvantages of using OOP for programming. [4 marks]
2. An article is a story that appears in a magazine. It has a title, a subtitle, an author and a number of words. Construct a UML diagram for an Article class. [4 marks]
3. Explain the difference between a static and non-static variable. [3 marks]
4. Here is a constructor method heading for an Article class.

```
public Article (String title, String author, int words){
// code missing
}
```

```
def __init__ (self, title, author, words):
    #code missing
```

In *either* Java *or* Python, initialize an instance of the class using the following information.

Title: "Top 10 creepy movies", Author: "Patrick D", Number of words: 350 [3 marks]

5. Identify two advantages of using encapsulation. [4 marks]
6. State three features of encapsulated code. [3 marks]

B3.2 Fundamentals of OOP for multiple classes

AHL

Syllabus understandings

B3.2.1 Explain and apply the concept of inheritance in OOP to promote code reusability

B3.2.2 Construct code to model polymorphism and its various forms, such as method overriding

B3.2.3 Explain the concept of abstraction in OOP

B3.2.4 Explain the role of composition and aggregation in class relationships

B3.2.5 Explain commonly used design patterns in OOP

B3.2.1 Explain and apply the concept of inheritance in OOP to promote code reusability

If you think about a mammal, what do you think of? Do you think of a blue whale? Do you think of a cat? Do you think of a lion? Do you think of a human? They all have attributes and behaviours in common that make them a mammal: warm-blooded, backbones, four limbs, fur, hair. They also have things that make them very different. Another example is when you think of a vehicle, what do you think of? Do you think of a car? Do you think of an aeroplane? Do you think of a boat? All these items have attributes and behaviours in common that make them a vehicle—move forward, brake, transport items—but each have their own attributes that make them different. The ability to identify both commonalities and differences between items helps you when developing code that uses **inheritance**.



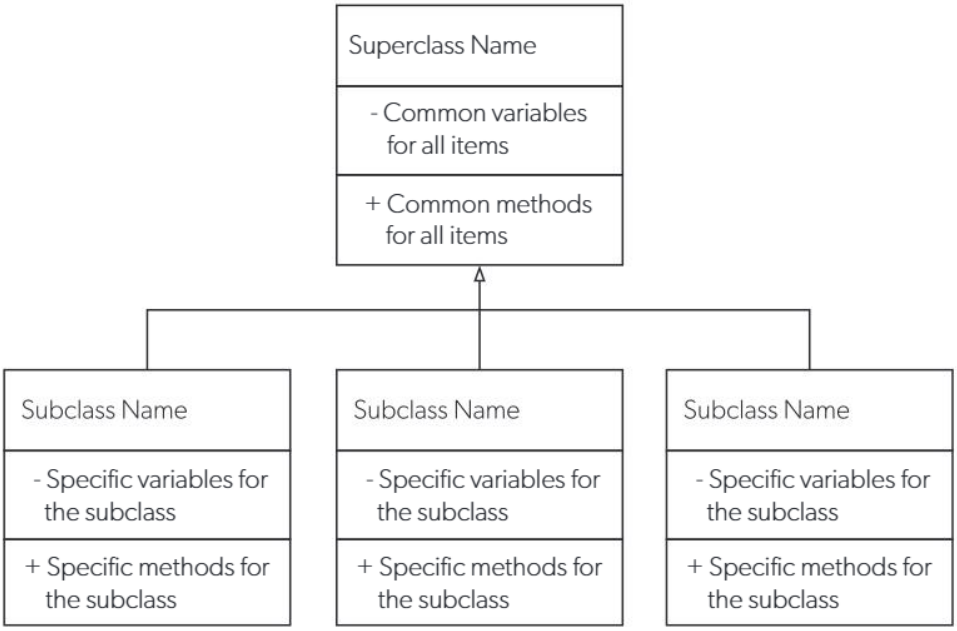
▲ **Figure 21** The platypus has attributes and behaviours in common with other mammals

Inheritance in object-oriented programming makes use of a superclass to store common attributes and behaviours. Subclasses inherit all the common attributes and behaviours from the superclass and add some of their own specific attributes and behaviours. If we continue with the example of vehicles, the superclass may contain information about the fuel type, capacity and maximum range. An aeroplane may contain additional information about whether it is commercial, a car may contain information about whether it is electric, and a ship may contain information about how much cargo it can hold. As all common code is stored in the superclass, you only need to code this once.

You can represent inheritance using a UML diagram (see Figure 23). Note the use of the arrows to denote an `is_a` relationship: Aeroplane is a Vehicle, Car is a Vehicle and Ship is a Vehicle.

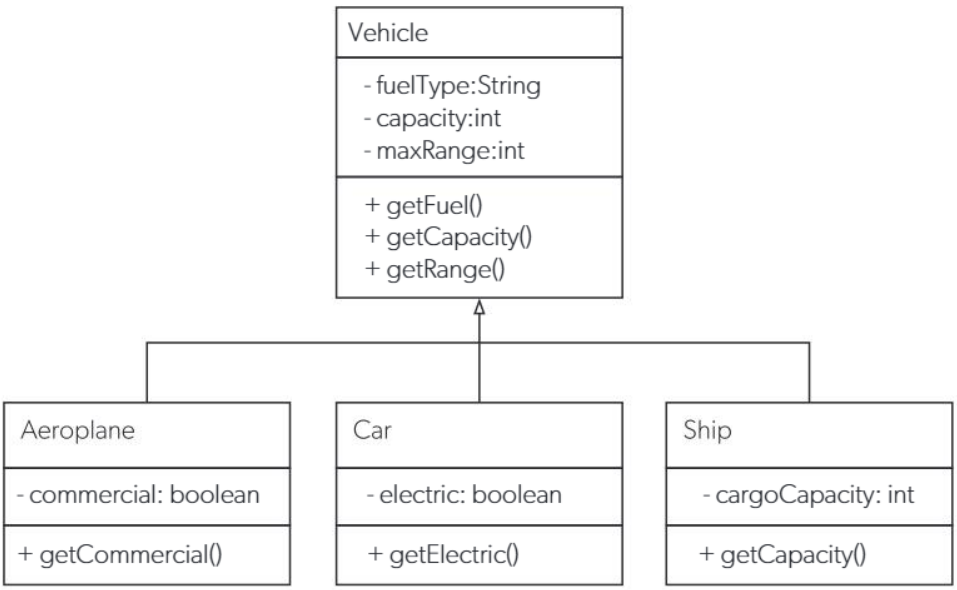
Key term

Inheritance (in coding) Using a superclass to store all common variables and methods then using subclasses to store specific variables and methods, promoting code reuse and easier maintenance.



▲ **Figure 22** Generic UML diagram for a superclass

The vehicle UML diagram will look something like this.



▲ **Figure 23** UML diagram for a Vehicle superclass

The advantages of using inheritance are as follows.

Reusing the code: Multiple subclasses can use the code in the superclass as they inherit the same properties.

More efficient code: It only has to be written once.

Simpler maintenance: Any changes in the superclass only need to be made once and are then reflected across all subclasses.

Modular code: Code that uses inheritance is modular, so it can be used in other applications with relative ease.

When using inheritance, you must be careful with the modifier you use for the variables and methods as this will determine whether the data can be seen by the subclass or not.

- A **private** variable or method can only be seen by the class in which it is declared. Any private variable or method in the superclass cannot be seen in the subclass (unless accessed through a public accessor method).
- **Protected** variables and methods can be seen by the superclass and the subclass. They cannot be seen outside of the inherited class structure. If you declare variables as protected in the superclass, they can be seen in all subclasses.
- Any variable or method declared **public** in the superclass can be seen in the superclass and all subsequent subclasses. Public variables and methods can also be seen outside the inherited class structure.
- Any variable or method declared with the **default** modifier limits the visibility to the package. Any class outside of the package is unable to see the variable and method.

ATL Research skills

When using inheritance, it is good to understand how the objects are expected to work. This is similar to pattern management and identification in computational thinking. Identify the patterns across all classes that can be placed in the superclass. Look for are behaviours or attributes that all items have.

Consider the following animals: cow, dolphin, giraffe, whale, lion, sheep.

- What variables would you need to store the required information?
- What methods would you need to represent the behaviours of the animal?
- What would you put in the superclass?
- What would be in each of the subclasses?

Coding with inheritance in Java

In the superclass you declare all the common variables and methods.
The following example will make use of the Vehicle superclass and Aeroplane, Car and Ship subclasses from the UML diagram in Figure 23. Note that because the variables have been declared private, public accessor methods will need to be used to access this data from the subclass.

Example superclass in Java

```
1.  public class Vehicle {
2.
3.      private String fuelType;
4.      private int capacity;
5.      private int maxRange;
6.
7.      public Vehicle (String fuelType, int capacity,
        int maxRange) {
8.
```

```
9.          this.fuelType = fuelType;
10.         this.capacity = capacity;
11.         this.maxRange = maxRange;
12.
13.     }
14.     public String getFuel() {
15.         return fuelType;
16.     }
17.     public int getCapacity() {
18.         return capacity;
19.     }
20.     public int getMaxRange() {
21.         return maxRange;
22.     }
23.     public String toString() {
24.
25.         return "Fuel: " + fuelType + ", Capacity:
            " + capacity + ", Max Range: " + maxRange;
26.     }
27. }
```

In the subclass you declare variables and methods specific to the subclass. The following example shows the Aeroplane subclass. Note the use of the word **`extends`** to show that the Aeroplane is a Vehicle and therefore inherits the Vehicle variables and methods. The constructor method takes parameters for both the subclass and the superclass. Within the constructor the **`super`** keyword is used to invoke the superclass constructor. The **`toString()`** method will override the **`toString()`** method in the superclass. This will be discussed further in section B3.2.2. The **`toString()`** method also calls upon the superclass **`toString()`** which will print the data from the superclass as well as the specifics from the subclass.

Example subclass in Java

```
1.  public class Aeroplane extends Vehicle {
2.
3.      private boolean commercial;
4.
5.      public Aeroplane (String fuelType, int
        capacity, int maxRange, boolean commercial) {
6.
7.          super (fuelType, capacity, maxRange);
8.          this.commercial = commercial;
9.      }
10.     public boolean getCommercial() {
11.         return commercial;
12.     }
```

```
13.     public String toString() {
14.         return super.toString() + ", Commercial: "
            + commercial;
15.     }
16. }
```

An example of the main method for this class is shown below. The superclass is used as the type for the `ArrayList`. This allows us to add all types of subclass to the array list. This means we do not have to have an array list per class type. When you loop through the list you can check the type of class using the method `.getClass().getSimpleName()`. You can then use this to cast the Vehicle to a type to access the subclass methods.

This is shown in the main method below. An if statement is used to check if the current Vehicle is a Ship. We then use this to create a temp variable of type ship (line 21). We can then use this temp variable to access the specific methods of the ship class.

Example main method in Java

```
1.  import java.util.ArrayList;
2.
3.  public class MainMethod {
4.
5.      public static void main(String[] args) {
6.
7.          ArrayList <Vehicle> theVehicles = new
            ArrayList <Vehicle>();
8.
9.          theVehicles.add(new Car("Petrol", 5, 400,
            false));
10.         theVehicles.add(new Car("Electric", 5,
            250, true));
11.         theVehicles.add(new Ship("Diesel", 5500,
            800, 0));
12.         theVehicles.add(new Ship("Hybrid", 200,
            1100, 10000));
13.         theVehicles.add(new Ship("Diesel", 7000,
            800, 10));
14.         theVehicles.add(new Aeroplane("Diesel",
            200, 600, true));
15.         theVehicles.add(new Aeroplane("Diesel",
            240, 650, true));
16.         theVehicles.add(new Aeroplane("Diesel",
            10, 500, false));
17.
18.         for (int i = 0; i < theVehicles.size();
            i++) {
19.             if(theVehicles.get(i).getClass().
                getSimpleName().equals("Ship")){
20.
21.                 Ship temp = (Ship)theVehicles.
                    get(i);
22.
```

```
23.                 if (temp.getCargoCapacity()>5000)
{
24.                     System.out.println(theVehicles.
                        get(i).toString());
25.                 }
26.             }
27.         }
28.     }
29. }
```

The code above shows that complexity can by minimized by using one `ArrayList` rather than several array lists.

Coding with inheritance in Python

In the superclass you declare all the common variables and methods.
The following example will make use of the Vehicle superclass and Aeroplane, Car and Ship subclasses from the UML diagram in Figure 23. Note that because the variables have been declared private, public accessor methods will need to be used to access this data from the subclass.

Example superclass in Python

```
1.  class Vehicle():
2.      def __init__(self, fuelType, capacity, maxRange):
3.          self.__fuelType = fuelType
4.          self.__capacity = capacity
5.          self.__maxRange = maxRange
6.
7.      def getFuelType(self):
8.          return self.__fuelType
9.
10.     def getCapacity(self):
11.         return self.__capacity
12.
13.     def getMaxRange(self):
14.         return self.__maxRange
15.
16.     def __str__(self):
17.         return "Fuel Type " + self.__fuelType + ",
            Capacity " + str(self.__capacity) + ", Max
            Range " + str(self.__maxRange)
```

In the subclass you declare variables and methods specific to the subclass.
The following example shows the Aeroplane subclass. Note the Vehicle class is passed in as a parameter to the class declaration to show that the Aeroplane is a Vehicle and therefore inherits the Vehicle variables and methods. The constructor method takes parameters for both the subclass and the superclass. Within the constructor, the `super` keyword is used to invoke the superclass constructor. The `__str__()` method will override the `__str__()` method in the superclass. This will be discussed further in section B3.2.2.

The `__str__()` method also calls upon the superclass `__str__()` which will print the data from the superclass as well as the specifics from the subclass. Note that, for the superclass to work, you need to import the superclass ready for use.

Example subclass in Python

```
1.  from Vehicle import Vehicle
2.  class Aeroplane(Vehicle):
3.
4.      def __init__(self, fuelType, capacity,
        maxRange, commercial):
5.          super().__init__(fuelType, capacity,
            maxRange)
6.          self.__commercial = commercial
7.
8.      def getCommercial(self):
9.          return self.__commercial
10.
11.     def __str__(self):
12.
13.         return super().__str__() + ", Commercial:
            " + str(self.__commercial)
```

An example of the main method for this class is shown below. The vehicles (all categories) can be added to the list. This means we do not have to have a list per class type. When you loop through the list you can check the type of class using `isInstance(X, classType)`. You can then use this to access the subclass methods. As Python uses dynamic binding, you do not need to cast the class—it will act as the class type it has been declared as. This is shown in the main method below. An if statement is used to check if the current Vehicle is a Ship. You can then access the specific methods of the ship class.

Example main method in Python

```
1.  from Vehicle import Vehicle
2.  from Ship import Ship
3.  from Car import Car
4.  from Aeroplane import Aeroplane
5.
6.  theVehicles = []
7.
8.  theVehicles.append(Car("Petrol", 5, 400, False))
9.  theVehicles.append(Car("Electric", 5, 250, True))
10. theVehicles.append(Ship("Diesel", 5500, 800, 0))
11. theVehicles.append(Ship("Hybrid", 200, 1100,
    10000))
12. theVehicles.append(Ship("Diesel", 7000, 800, 10))
13. theVehicles.append(Aeroplane("Diesel", 200, 600,
    True))
14. theVehicles.append(Aeroplane("Diesel", 240, 650,
    True))
```

AHL

```
15. theVehicles.append(Aeroplane("Diesel", 10, 500,
    False))
16.
17. for X in theVehicles:
18.     if (isinstance(X, Ship)):
19.
20.         if X.getCargoCapacity()>5000:
21.          print(X.__str__())
```

Activity

The animals in a zoo can be divided into three groups: reptiles, amphibians and mammals. The zoo wants to store information about each animal, including the section of the park where the animal can be found. Develop a program for the zoo to use.

B3.2.2 Construct code to model polymorphism and its various forms, such as method overriding

A smartphone can be used as a metaphor for **polymorphism**.



▲ **Figure 24** A smartphone can take on different forms

In programming, polymorphism is used to mean that a method can take on many different forms. To think about polymorphism in the real world, consider yourself as a person. Sometimes you act as a student, sometimes as a child, sometimes a friend. You have many different forms (roles) depending on the situation you are in. This is the same as your phone. Sometimes you use your phone as a map, sometimes to watch videos, and sometimes to stay in contact with friends and family. The form of your phone changes depending on the situation you are in.

Key term

Polymorphism (in coding) Code that has many forms. The version of the code chosen depends on the type of object or the number of parameters passed into the method. This promotes code reuse and better maintenance.

Polymorphism in code is similar. The form the code takes depends on different factors, including the number of parameters passed into the method and what type of object the method is within. Polymorphism can either be static or dynamic. You will examine these more later in the section.

Advantages of polymorphism

- Code can be reused. You can make a common class and different objects can be treated as a common class.
- You need to write much less code. You can have different versions of methods in the same class.
- Less maintenance, as many of the methods are in the same class.
- Generic code can be used, this enables all objects to be treated as the same generic type, requiring less specific coding.

Disadvantages of polymorphism

- Increased computational power. When using dynamic polymorphism, decisions must be made at runtime. This increases the computational power required.
- Increased code complexity.

ATL Research skills

Researching how an object or method should work is a key skill. Using algorithmic thinking can help you to understand how an object should operate within algorithms. If the object needs to behave in different ways, use polymorphism.

1. Research a character in a computer game. Some characters interact with the main character. Some characters have no interaction. What attributes would a character that interacts with the main character have that the others would not? Think about their characteristics. For example, do they give clues to the main character? Are they there to make the scene more interesting?
2. For the characters who do interact with the main character, would you make a polymorphic class or not? What features would this class have?
3. If you cannot use polymorphism, how would you make the class?

Static polymorphism

Static polymorphism is sometimes called compile-time polymorphism as it happens at compile time. When a program is compiled, it is translated from a high-level language into machine code using the translation process. At this point, the compiler checks which method will be used, based on the number and type of parameters passed into the method.

This is useful as it allows for several methods with the same name that make use of different parameters to carry out a task.

An example is a Salesperson class. Salespeople can either work on a fixed salary or by having a base salary and adding commission. Without polymorphism, you would need to have two different method names, which could be confusing. You would need to remember what type of object you are using or use two different classes (a base salesperson and a commissioned salesperson).

With polymorphism, you can have a single Salesperson class and use two methods with the same name but different parameters.

- `calculateWage()` // when the salesperson does not work on commission.
- `calculateWage(monthlySales)` // when the salesperson works with commission.

You do not need to have many different salesperson classes—one with polymorphism will suffice. This also allows for flexibility if a salesperson changes from basic to commissioned. You do not need to create a new version of the class for this—you just change the method you use. This type of polymorphism is sometimes called method overloading.

Static polymorphism in Java

There are two overloaded methods in the class. The first is the constructor method. There are two constructor methods: one that takes three parameters (Salespeople who do not work on commission) and one that takes four parameters (Salespeople who work on commission). How many parameters there are in the call to the constructor depends on which constructor method is chosen. The second overloaded method is the `calculateWage()` method. One method requires no parameters (for Salespeople who do not work on commission) and one requires one parameter (for Salespeople who work on commission): `calculateWage(double monthlySales)`. Again, whether a parameter is given or not depends on which method is chosen.

Example class that uses static polymorphism in Java

```
1.  public class Salesperson {
2.
3.      private String name;
4.      private String location;
5.      private double annualSalary;
6.      private double commission;
7.
8.      public Salesperson(String name, String
        location, double annualSalary) {
9.
10.         this.name = name;
11.         this.location = location;
12.         this.annualSalary = annualSalary;
13.         this.commission = 0.0;
14.
15.     }
16.     public Salesperson(String name, String
        location, double annualSalary, double
        commission) {
17.
18.         this.name = name;
19.         this.location = location;
20.         this.annualSalary = annualSalary;
```

```
21.         this.commission = commission;
22.
23.     }
24.     public String getName() {
25.         return name;
26.     }
27.     public String getLocation() {
28.         return location;
29.     }
30.     public double getSalary() {
31.         return annualSalary;
32.     }
33.     public double getCommission() {
34.         return commission;
35.     }
36.     public void setCommission(double comm) {
37.         commission = comm;
38.     }
39.     public double calculateWage() {
40.         return annualSalary / 12;
41.     }
42.     public double calculateWage(double monthlySales) {
43.
44.         double monthlyCommission = monthlySales *
            commission;
45.         return (annualSalary/12)
            +  monthlyCommission;
46.     }
47.     public String toString() {
48.         return "Name: " + name + ", Location: " +
            location;
49.     }
50. }
```

The main method instantiates two instances of the Salesperson class: one using the constructor that takes three parameters and one using the constructor that takes four parameters. The main method also calls upon two versions of the overloaded method `calculateWage()`: one that requires no parameter and one that requires the monthly sales.

Example main class in Java

```
1.  public class MainMethod {
2.
3.      public static void main(String[] args) {
4.
5.          Salesperson firstPerson = new
            Salesperson("Cole Martinez", "Porto",
            45000.00);
```

```
6.          Salesperson secondPerson = new Salesperson
            ("Rina Markaj", "Patas", 37500.00, 0.75);
7.
8.          System.out.println(firstPerson.
            calculateWage());
9.          System.out.println(secondPerson.
            calculateWage(3500));
10.     }
11. }
```

This example shows that polymorphism makes code more reusable. You can switch between a non-commissioned and commissioned salesperson by setting the commission of a salesperson and using the overloaded method rather than creating a whole new instance of the Salesperson. Any changes to the code only have to be made in this class, making maintenance easier.

Static polymorphism in Python

Method overloading is not supported in Python. There are ways to pretend it works using if statements, but the object-oriented version of static polymorphism is not supported.

Activity

A campsite has several pitches (spaces) for tents, divided into three price bands: Basic Tent (cost per night), Enhanced Tent (cost per night plus 25 for electricity) and Premium Tent (cost per night plus 25 electricity and 30 per night for access to the luxury shower block). Develop a program using polymorphism that will provide a solution.

Dynamic polymorphism

Dynamic polymorphism is sometimes called runtime polymorphism as it happens at runtime. This type of polymorphism happens in inherited classes where the superclass has a method in it with the same name as the subclass but the subclass method is called when available. The superclass shows the method is available and then, at runtime, the method from the subclass is called and run. This type of polymorphism is sometimes called method overriding as the subclass method overrides the superclass method of the same name.

One example of method overriding could be the `toString()` method. The `toString()` method in the superclass contains a formatted version of the data within the superclass. However, this will be overridden by the `toString()` method in the subclass, which may contain formatted data from both the superclass and the subclass.

This is useful as it means we can have specific classes in the subclasses that act in accordance with the type they have been cast as. There are a few rules we have to know when using method overriding,

- The classes must have an inheritance (`is_a`) relationship.
- The name of the method must be the same in both the superclass and the subclass.

- The number and type of parameters must be the same in the superclass and subclass.
- **Static** methods cannot be overridden.

Dynamic polymorphism in Java

An example of a class that uses dynamic polymorphism (method overriding) is shown below. The superclass contains a `move()` method. Each subclass also contains a `move()` method. The method in the subclass takes precedence over the superclass method and will therefore be the only one that is run. The second method that uses method overriding is the `toString()` method. The `toString()` method in the superclass is never called directly by the main method. However, the subclass uses the superclass `toString()` method to get the data from the superclass and then adds the subclass information.

Code for the superclass in Java

```
1.  public class Vehicle {
2.
3.      private String fuelType;
4.      private int capacity;
5.      private int maxRange;
6.
7.      public Vehicle (String fuelType, int capacity,
        int maxRange) {
8.
9.          this.fuelType = fuelType;
10.         this.capacity = capacity;
11.         this.maxRange = maxRange;
12.
13.     }
14.     public String getFuel() {
15.         return fuelType;
16.     }
17.     public int getCapacity() {
18.         return capacity;
19.     }
20.     public int getMaxRange() {
21.         return maxRange;
22.     }
23.     public String move() {
24.         return "The vehicle is moving";
25.     }
26.     public String toString() {
27.
28.         return "Fuel: " + fuelType + ", Capacity:
            " + capacity + ", Max Range: " + maxRange;
29.     }
30. }
```

AHL

Code for the subclass Car in Java

```
1.  public class Car extends Vehicle {
2.
3.      private boolean electric;
4.
5.      public Car (String fuelType, int capacity, int
        maxRange, boolean electric) {
6.
7.          super (fuelType, capacity, maxRange);
8.          this.electric = electric;
9.      }
10.     public boolean getElectric() {
11.         return electric;
12.     }
13.     public String move() {
14.         return "The Car is moving";
15.     }
16.     public String toString() {
17.
18.         return super.toString() + ", Electric: " +
            electric;
19.     }
20. }
```

Code for the subclass Aeroplane in Java

```
1.  public class Aeroplane extends Vehicle {
2.
3.      private boolean commercial;
4.
5.      public Aeroplane (String fuelType, int
        capacity, int maxRange, boolean commercial) {
6.
7.          super (fuelType, capacity, maxRange);
8.          this.commercial = commercial;
9.      }
10.     public boolean getCommercial() {
11.         return commercial;
12.     }
13.     public String move() {
14.         return "The Aeroplane is moving";
15.     }
16.     public String toString() {
17.         return super.toString() + ", Commercial: "
            + commercial;
18.     }
19. }
```

AHL

Code for the subclass Ship in Java

```
1.  public class Ship extends Vehicle {
2.
3.      private int cargoCapacity;
4.
5.      public Ship (String fuelType, int capacity,
        int maxRange, int cargoCapacity) {
6.
7.          super (fuelType, capacity, maxRange);
8.          this.cargoCapacity = cargoCapacity;
9.      }
10.
11.     public int getCargoCapacity() {
12.         return cargoCapacity;
13.     }
14.     public String move() {
15.         return "The Ship is moving";
16.     }
17.     public String toString() {
18.
19.         return super.toString() + ", Capacity: " +
            cargoCapacity;
20.     }
21. }
22.
```

An example of the main method for this class is shown below. The main method instantiates several vehicles, some of type Car, some of type Aeroplane, and some of type Ship. The main method also calls the overridden method **move()**. Printing the **move()** method shows that only the overridden method in the subclass is called.

Main method in Java

```
1.  import java.util.ArrayList;
2.
3.  public class MainMethod {
4.
5.      public static void main(String[] args) {
6.
7.          ArrayList <Vehicle> theVehicles = new
            ArrayList <Vehicle>();
8.
9.          theVehicles.add(new Car("Petrol", 5, 400,
            false));
10.         theVehicles.add(new Car("Electric", 5,
            250, true));
```

AHL

```
11.         theVehicles.add(new Ship("Diesel", 5500,
            800, 0));
12.         theVehicles.add(new Ship("Hybrid", 200,
            1100, 10000));
13.         theVehicles.add(new Ship("Diesel", 7000,
            800, 10));
14.         theVehicles.add(new Aeroplane("Diesel",
            200, 600, true));
15.         theVehicles.add(new Aeroplane("Diesel",
            240, 650, true));
16.         theVehicles.add(new Aeroplane("Diesel",
            10, 500, false));
17.
18.         System.out.println(theVehicles.get(1).
            move());
19.         System.out.println(theVehicles.get(6).
            move());
20.     }
21. }
```

Output:

The Car is moving

The Aeroplane is moving

Dynamic polymorphism in Python

An example of a class that uses dynamic polymorphism (method overriding) is shown below. The superclass contains a **`move()`** method. Each subclass also contains a **`move()`** method. The method in the subclass takes precedence over the superclass method and therefore will be the only one that is run. The second method that uses method overriding is the __str__() method. The **`__str__()`** method in the superclass is never called directly by the main method. However, the subclass uses the superclass **`__str__()`** method to get the data from the superclass and then adds the subclass information.

Code for the superclass in Python

```
1.  class Vehicle():
2.      def __init__(self, fuelType, capacity,
        maxRange):
3.          self.__fuelType = fuelType
4.          self.__capacity = capacity
5.          self.__maxRange = maxRange
6.
7.      def getFuelType(self):
8.          return self.__fuelType
9.
10.     def getCapacity(self):
11.         return self.__capacity
12.
13.     def getMaxRange(self):
14.         return self.__maxRange
```

```
15.
16.     def move(self):
17.         return "the Vehicle is moving"
18.
19.     def __str__(self):
20.         return "Fuel Type " + self.__fuelType + ",
            Capacity " + str(self.__capacity) + ", Max
            Range " + str(self.__maxRange)
21.
```

Code for the subclass Car in Python

```
1.  from Vehicle import Vehicle
2.  class Car(Vehicle):
3.
4.      def __init__(self, fuelType, capacity,
        maxRange, electric):
5.          super().__init__(fuelType, capacity,
            maxRange)
6.          self.__electric = electric
7.
8.      def getElectric(self):
9.          return self.__electric
10.
11.     def move(self):
12.         return "The Car is moving"
13.
14.     def __str__(self):
15.         return super().__str__() + ", Electric: "
            + str(self.__electric)
```

Code for the subclass Aeroplane in Python

```
1.  from Vehicle import Vehicle
2.  class Aeroplane(Vehicle):
3.
4.      def __init__(self, fuelType, capacity,
        maxRange, commercial):
5.          super().__init__(fuelType, capacity,
            maxRange)
6.          self.__commercial = commercial
7.
8.      def getCommercial(self):
9.          return self.__commercial
10.
11.     def move(self):
12.         return "The Aeroplane is moving"
13.
14.     def __str__(self):
15.         return super().__str__() + ", Commercial:
            " + str(self.__commercial)
```

AHL

Code for the subclass Ship in Python

```
from Vehicle import Vehicle
class Ship(Vehicle):

    def __init__(self, fuelType, capacity, maxRange, cargoCapacity):
        super().__init__(fuelType, capacity, maxRange)
        self.__cargoCapacity = cargoCapacity

    def getCargoCapacity(self):

        return self.__cargoCapacity

    def move(self):
        return "The Ship is moving"

    def __str__(self):

        return super().__str__() + " Capacity: " + str(self.__cargoCapacity)
```

An example of the main method for this class is shown below. The main method instantiates several vehicles, some of type Car, some of type Aeroplane, and some of type Ship. The main method also calls the overridden method **`move()`**. Printing the **`move()`** method shows that only the overridden method in the subclass is called.

Main method in Python

```
from Vehicle import Vehicle
from Ship import Ship
from Car import Car
from Aeroplane import Aeroplane

theVehicles = []

theVehicles.append(Car("Petrol", 5, 400, False))
theVehicles.append(Car("Electric", 5, 250, True))
theVehicles.append(Ship("Diesel", 5500, 800, 0))
theVehicles.append(Ship("Hybrid", 200, 1100, 10000))
theVehicles.append(Ship("Diesel", 7000, 800, 10))
theVehicles.append(Aeroplane("Diesel", 200, 600, True))
theVehicles.append(Aeroplane("Diesel", 240, 650, True))
```

```
15. theVehicles.append(Aeroplane("Diesel", 10, 500, False))
16.
17. print(theVehicles[1].move())
18. print(theVehicles[6].move())
```

Output:

The Car is moving

The Aeroplane is moving

Duck typing and Python

One of the major differences between Java and Python is the use of data types. When you create a variable in Java, you have to declare its type. For example:

```
int count = 0;
```

When you declare a variable in Java, you are telling it how to behave and limiting its functionality. While this can be useful, because you do not need to cast items later on, sometimes it can limit the reuse and functionality of a method.

Python does not have this concept. Instead, Python uses **duck typing**. Whether a variable or object is suitable for a task or not is determined by the presence of the correct methods and attributes—it is decided at runtime. This process is called duck typing because it works on the principle that "if it looks like a duck, swims like a duck, and quacks like a duck, then it probably is a duck."

By allowing objects of different types to be used interchangeably, as long as they have the correct methods available, code becomes more flexible and reusable, making loose typed or duck typed languages preferable in certain situations.

In the example program below, three classes have been declared: duck, goose and magpie. Duck and goose have a function called `waddle()` but the magpie has a function called `fly()`.

Duck typing in Python

```
1.  class duck:
2.      def waddle(self):
3.          print("I am a duck for sure")
4.
5.  class goose:
6.      def waddle(self):
7.          print ("I am a goose but I am still valid")
8.
9.  class magpie:
10.     def fly(self):
11.         print("I am a magpie and I fly")
```

In the following test code, the list `wildlife` contains only classes that contain a `waddle()` method, so no error is thrown. However, trying to perform the same operation on `wildlifeTwo` throws an error because a magpie can not act like a duck, thereby breaking the code.

Key term

Duck typing A form of using objects with Python. The method or function calling the object does not care what type of object it is dealing with, only that the methods or attributes it wishes to use are available within the object it is using.

AHL

Test code in Python

```
1.  def testMe(item):
2.      item.waddle()
3.
4.  wildlife = [duck(), duck(), goose()]
5.  wildlifeTwo = [duck(), duck(), goose(), magpie()]
6.
7.  for X in wildlife:
8.      print(testMe(X))
9.
10. for Y in wildlifeTwo:
11.     print(testMe(Y))
```

Output wildlife:

I am a duck for sure

I am a duck for sure

I am a goose but I am still valid

Output wildlifeTwo:

item.waddle()

^^^^^^^^^^^

AttributeError: 'magpie' object has no attribute 'waddle'

TOK

Flexibility is fundamental to making programs work. Understanding how objects are represented in coding and how they can take on many forms can help you understand how knowledge is represented within code.

- How does polymorphism enable flexibility in the code?
- How does the perspective of the programmer affect the flexibility of the code?

Key term

Abstraction Hiding all essential details of implementation from the user.

B3.2.3 Explain the concept of abstraction in OOP

Abstraction is one of the key features of object-oriented code, along with encapsulation and inheritance. Abstraction in object-oriented programming refers to the idea that, in object orientation, all non-essential details are hidden from the user. You could consider a car to be an abstraction. When you place your foot on the accelerator, the car moves forward, and when you brake, the car slows down. You do not know the detail of how the car works as the implementation details have been hidden from you, but you know how they function. This is similar to how classes function in object-oriented programming. Classes hide the complex implementation from the end user.



▲ **Figure 25** A car can be considered as an abstraction

Building code with abstraction

To achieve abstraction in your code, abstract all the essential data about an item and place it in an abstract class, filtering out all the irrelevant data. Specific data can then be placed into concrete classes. Abstract classes allow you to have abstract methods showing the methods that are present in the concrete classes, guaranteeing they are there but allowing you to mix classes within a data structure. For example, consider animals. An animal can be extracted (put into) into a type, a habitat, a diet, and whether it is endangered. This information can then be placed into an abstract class.

The abstract class guarantees that the subclasses contain all the methods. So, if you have several different concrete classes within one data structure, an error will not occur because there is a common abstract class.

Lion extends Animal
- park: String
+ getPark()
+ toString()

Panda extends Animal
- province: String
+ getProvince()
+ toString()

Elephant extends Animal
- park: String
+ getPark()
+ toString()

▲ **Figure 27** UML diagrams showing subclass use with an abstract class

This example shows that an abstract class is similar to a superclass in inheritance but none of the implementation is provided. The abstract class is a generalized form of the class that is shared by subclasses. The subclasses are necessary to provide the implementation information.

ATL Thinking skills

When using abstract classes, you will probably have to use data to identify items that can be abstracted into the abstract class. Abstraction in computational thinking can help with this. It enables you to focus only on the relevant information.

- In what way is an abstract class different to using inheritance?
- How do abstract classes enhance the stability of the code?

Abstract class in Java

Animal class. The use of `abstract` shows that the class cannot be instantiated.

```
1.  public abstract class Animal {
2.
3.      public Animal () {
4.
5.      }
6.      public abstract String getType();
7.      public abstract String getHabitat();
8.      public abstract String getDiet();
9.      public abstract boolean getEndangered();
10.     public abstract String toString();
11. }
```

Abstract Class Animal
- type: String - habitat: String - diet: String - endangered: Boolean
+ Abstract getType() + Abstract getHabitat() + Abstract getDiet() + Abstract getEndangered() + Abstract toString()

▲ **Figure 26** UML diagram for the Animal abstract class

Using an Abstract class is using a technique known as dynamic binding. This is a form of polymorphism (covered in section B3.2.2). Dynamic binding is a runtime polymorphism technique. During the compile phase, the compiler looks to the abstract class to check that the methods being called exist; the abstract class acts as a guarantee that they are there and no problem will occur. At runtime, the abstract class is ignored and the actual class of the type provided it runs. Choosing the specific subclass to run at runtime based on guarantees from the abstract class is known as dynamic binding.

Subclasses provide the concrete implementations of the method.
Use **extends** to show they are related to Animal.



Code for the subclass Elephant in Java

```
1.  public class Elephant extends Animal {
2.
3.      private String type;
4.      private String habitat;
5.      private String diet;
6.      private boolean endangered;
7.      private String park;
8.
9.      public Elephant (String type, String habitat,
        String diet, boolean endangered, String park) {
10.
11.         this.type = type;
12.         this.habitat = habitat;
13.         this.diet= diet;
14.         this.endangered = endangered;
15.         this.park = park;
16.
17.     }
18.
19.     public String getType() {
20.         return type;
21.     }
22.     public String getHabitat() {
23.         return habitat;
24.     }
25.     public String getDiet() {
26.         return diet;
27.     }
28.     public boolean getEndangered() {
29.         return endangered;
30.     }
31.     public String getPark() {
32.         return park;
33.     }
34.     public String toString() {
35.
36.         return "Type: " + type + ", habitat
            " + habitat + ", Diet: " + diet + ",
            Endangered: " + endangered + ", Park: " +
            park;
37.     }
38. }
```

Code for the subclass Lion in Java

```
1.  public class Lion extends Animal {
2.
3.      private String type;
4.      private String habitat;
5.      private String diet;
6.      private boolean endangered;
7.      private String park;
8.
9.      public Lion (String type, String habitat,
        String diet,boolean endangered, String park) {
10.
11.         this.type = type;
12.         this.habitat = habitat;
13.         this.diet= diet;
14.         this.endangered = endangered;
15.         this.park = park;
16.
17.     }
18.     public String getType() {
19.         return type;
20.     }
21.     public String getHabitat() {
22.         return habitat;
23.     }
24.     public String getDiet() {
25.         return diet;
26.     }
27.
28.     public boolean getEndangered() {
29.         return endangered;
30.     }
31.     public String getPark() {
32.         return park;
33.     }
34.     public String toString() {
35.
36.         return "Type: " + type + ", habitat
            " + habitat + ", Diet: " + diet + ",
            Endangered: " + endangered + ", Park: " +
            park;
37.     }
38. }
```

AHL

Code for the subclass Panda in Java

```
1.  public class Panda extends Animal {
2.
3.      private String type;
4.      private String habitat;
5.      private String diet;
6.      private boolean endangered;
7.      private String province;
8.
9.      public Panda (String type, String habitat,
        String diet, boolean endangered, String
        province) {
10.
11.         this.type = type;
12.         this.habitat = habitat;
13.         this.diet= diet;
14.         this.endangered = endangered;
15.         this.province = province;
16.
17.     }
18.     public String getType() {
19.         return type;
20.     }
21.     public String getHabitat() {
22.         return habitat;
23.     }
24.     public String getDiet() {
25.         return diet;
26.     }
27.
28.     public boolean getEndangered() {
29.         return endangered;
30.     }
31.     public String getProvince() {
32.         return province;
33.     }
34.     public String toString() {
35.
36.         return "Type: " + type + ", habitat
            " + habitat + ", Diet: " + diet + ",
            Endangered: " + endangered + ", Park: " +
            park;
37.     }
38. }
```

The main method shows that all instances that inherit the abstract class can be instantiated within one linked list. The subclasses have concrete methods rather than abstract methods. The compiler decides which version of the subclass to run at runtime.

Main method in Java

```
1.  import java.util.LinkedList;
2.
3.      public class Main {
4.
5.      public static void main(String[] args) {
6.
7.
8.          LinkedList <Animal> myZoo = new
            LinkedList<Animal>();
9.
10.         myZoo.add(new Panda("Giant Panda",
            "Jungle", "Bamboo", true, "Schezan"));
11.         myZoo.add(new Lion("Lion", "Plains",
            "Meat", false, "Krugar"));
12.         myZoo.add(new Lion("Lion Lion", "Plains",
            "Meat", false, "tanzania"));
13.
14.         for(int I = 0; i < myZoo.size(); i++) {
15.         System.out.println(myZoo.get(i).toString());
16.         }
17.
18.     }
19.
20. }
```

Abstract class in Python

To develop an abstract class in Python you need to import ABC (abstract base class) otherwise this will not work. The class then inherits ABC through the class parameter brackets. You need to tell Python this is an abstract method with the @abstractmethod decorator.

```
1.  from abc import ABC, abstractmethod
2.  class Animal(ABC):
3.
4.      #def __init__Animal(self):
5.      @abstractmethod
6.      def getType(self):
7.          pass
8.      @abstractmethod
9.      def getHabitat(self):
10.         pass
11.     @abstractmethod
12.     def getDiet(self):
13.         pass
14.     @abstractmethod
15.     def getEndangered(self):
16.         pass
17.     @abstractmethod
18.     def __str__(self):
19.         pass
```

Subclasses are used to provide the concrete implementations of the method. To make this a subclass that inherits the superclass methods, you need to import the Animal class and tell the subclass it is part of the inherited, Animal structure using the parameter brackets.

Code for Elephant subclass in Python

```
1.  from Animal import Animal
2.
3.  class Elephant(Animal):
2.
5.      def __init__ (self, type, habitat, diet,
        endangered, park):
6.          self.__type = type
7.          self.__habitat = habitat
8.          self.__diet = diet
9.          self.__endangered = endangered
10.         self.__park = park
11.
12.     def getType(self):
13.         return self.__type
14.     def getHabitat(self):
15.         return self.__habitat
16.
17.     def getDiet(self):
18.         return self.__diet
19.
20.     def getEndangered(self):
21.         return self.__endangered
22.
23.     def getPark(self):
24.         return self.__park
25.
26.
27.     def __str__(self):
28.         return "Type: " + self.__type + ", Habitat:
            " + self.__habitat + ", Diet" + self.__diet
            + ", Endangered: " + str(self.__endangered)
            + ", Park: " + self.__park
```

Code for Lion subclass in Python

```
1.  from Animal import Animal
2.
3.  class Lion(Animal):
4.
5.      def __init__ (self, type, habitat, diet,
        endangered, park):
6.          self.__type = type
7.          self.__habitat = habitat
8.          self.__diet = diet
9.          self.__endangered = endangered
10.         self.__park = park
11.
12.     def getType(self):
13.         return self.__type
14.
15.     def getHabitat(self):
16.         return self.__habitat
17.
18.     def getDiet(self):
19.         return self.__diet
20.
21.     def getEndangered(self):
22.         return self.__endangered
23.
24.     def getPark(self):
25.         return self.__park
26.
```

```
    def __str__(self):
        return "Type: " + self.__type + ", Habitat:
        " + self.__habitat + ", Diet" + self.__diet
        + ", Endangered: " + str(self.__endangered)
        + ", Park: " + self.__park
```

Code for Panda subclass in Python

```
from Animal import Animal

class Panda(Animal):

    def __init__ (self, type, habitat, diet,
    endangered, province):
        self.__type = type
        self.__habitat = habitat
        self.__diet = diet
        self.__endangered = endangered
        self.__province = province

    def getType(self):
        return self.__type

    def getHabitat(self):
        return self.__habitat

    def getDiet(self):
        return self.__diet

    def getEndangered(self):
        return self.__endangered

    def getProvince(self):
        return self.__province

    def __str__(self):
        return "Type: " + self.__type + ", Habitat:
        " + self.__habitat + ", Diet" + self.__diet
        + ", Endangered: " + str(self.__endangered)
        + ", Park: " + self.__park
```

The main method shows that all instances that inherit the abstract class can be instantiated within one list. The subclasses have concrete methods rather than abstract methods. The compiler decides which version of the subclass to run at runtime.

Main method in Python

```
from Animal import Animal
from Panda import Panda
from Lion import Lion
from Elephant import Elephant

myZoo = []
myZoo.append(Panda("Giant Panda", "Jungle",
"Bamboo", True, "Schezan"))
myZoo.append(Lion("Lion", "Plains", "Meat", False,
"Krugar"))
myZoo.append(Lion("Lion Lion", "Plains", "Meat",
False, "tanzania"))

for x in myZoo:
    print(x.__str__())
```

AHL

Activity

A restaurant wants to use an OOP program to manage its menu. The menu is separated into three sections: Small Plates (a starter or appetizer), Large Plates (a main course or entrée), Sweet Plates (a dessert). Customers can order individual plates or a shared family meal. For the shared meal, each person can pick a combination of any three Small Plates, Large Plates and/or Sweet Plates.

The restaurant wants to be able to add and remove items on the menu list depending on the season. Use abstract classes to develop a program to help the restaurant.

Key things to know about abstract classes

- You need to use the abstract keyword so the compiler knows that the class cannot be instantiated.
- Abstract methods are declared with no implementation. They serve as a guarantee that the subclass will provide implementation details.
- The methods must be overridden in the subclass.

Understanding the concept of abstraction and how you can abstract important information to represent real-world concepts is particularly useful in the A3 Databases and A4 Machine learning topics.

TOK

When developing code, decisions are often made regarding what data should be stored within a system and what data is irrelevant. For example, if you are storing data from a retail business, you may decide to store information about the products your customers buy and where they buy them. This allows data warehouses to make connections between the objects being bought, their geographic location and related products. You may decide not to store data about the consumer themselves—this kind of data could be used to make judgements or assumptions about the consumers.

What are the ethical implications of leaving out the details when representing data?

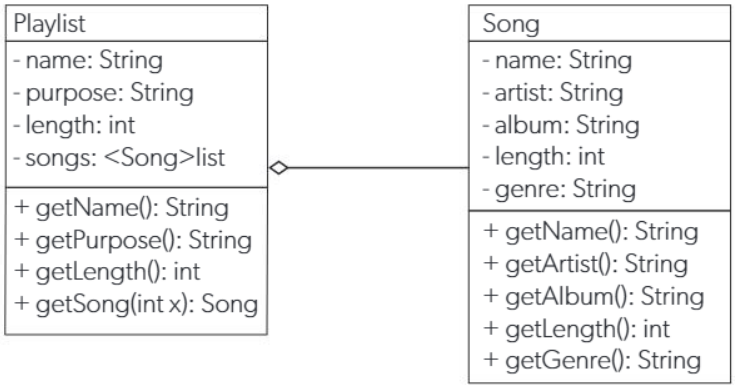
B3.2.4 Explain the role of composition and aggregation in class relationships

There are three types of relationship you need to be aware of between classes in object-oriented programming. The first one, `is_a`, has been covered in sections B3.2.1 and B3.2.3. This is when there is a superclass or an abstract class with common methods that are inherited or implemented by the subclasses. The other two are aggregation and composition.

One of the main principles of object orientation is breaking larger problems into smaller, more manageable, encapsulated objects of code. This promotes ease of maintenance and code reuse, but it does mean that there are often many relationships between objects.

Aggregation

An **aggregation** relationship is a relationship between two objects where both objects can exist separately from each other. This is sometimes known as a `has_a` relationship. For example, this could be a song and a playlist. A playlist can exist with no songs in it, and a song can exist without being in a playlist. This could be represented using this UML diagram.



▲ **Figure 28** UML diagram of an aggregated relationship

Key term

Aggregation When two classes rely on each other but can exist separately from each other.

This problem in practice

If you use a streaming service to listen to music, it is unlikely you will listen to songs album by album. You are more likely to have songs you like from different artists together in one place, in a playlist. Aggregated relationships are used to make this possible. When you create a playlist and spend time thinking of the best songs to go on there, the songs exist elsewhere. The collection of songs that make up your playlist, however, does not exist without those songs. This is investigated further in the following example.

Aggregated relationships are usually one-directional. Each can exist separately. For example, a playlist can be empty or it can have songs. Songs exist independently and they can be included on playlists. They do not rely on each other, as you could delete the playlist and the songs would still exist. Similarly, if you deleted a song, the playlist would still exist. This is aggregation. Aggregated relationships usually have a variable of one object type within another class.

Aggregation in Java

In this example, the Song class is used within the Playlist class to store details about each song on the playlist. This is an example of an aggregated relationship. The Song class is also shown for reference.

Playlist class in Java

```
1.  import java.util.ArrayList;
2.
3.  public class Playlist {
4.
5.      private String name;
6.      private String purpose;
```

AHL

```
7.      private int length;
8.      private ArrayList <Song> playlistSongs;
9.
10.     public Playlist(String name, String purpose) {
11.         this.name = name;
12.         this.purpose = purpose;
13.         playlistSongs = new ArrayList<Song>();
14.     }
15.
16.     public String getName() {
17.         return name;
18.     }
19.     public String getPurpose() {
20.         return purpose;
21.     }
22.     public int getLength() {
23.         return length;
24.     }
25.     public Song getSong(int x) {
26.         return playlistSongs.get(x);
27.     }
28.     public void addSong(Song x) {
29.         playlistSongs.add(x);
30.     }
31.     public String toString() {
32.         return "Name: " + name + ", Purpose: " +
            purpose + ", Length" + length;
33.     }
34. }
```

Song class in Java

```
1.  public class Song {
2.
3.      private String name;
4.      private String artist;
5.      private String album;
6.      private int length;
7.      private String genre;
8.
9.      public Song (String name, String artist,
        String album, int length, String genre){
10.
11.         this.name = name;
12.         this.artist = artist;
13.         this.album = album;
14.         this.length = length;
15.         this.genre = genre;
```

```
    }

    public String getName() {
        return name;
    }
    public String getArtist() {
        return artist;
    }
    public String getAlbum() {
        return album;
    }
    public int getLength() {
        return length;
    }
    public String getGenre() {
        return genre;
    }
    public String toString() {
        return "Name: " + name + ", Artist: " +
        artist + ", Album: " + album;
    }
}
```

The main method uses the Song class to make a list of songs. This list of songs is then used to populate the playlist. Although the playlist makes use of the Song class, the Song class can exist alone.

Main method in Java

```
import java.util.ArrayList;

public class MainMethod {

    public static void main(String[] args) {

        ArrayList <Song> allSongs = new
        ArrayList<Song>();
        allSongs.add(new Song("Low Lights", "The
        Judges", "Bridge over Gray", 340, "Pop"));
        allSongs.add(new Song("Shiny Sparkles",
        "Sprites", "Bubblegum", 280, "Pop"));
        allSongs.add(new Song("Elephant Light",
        "Jarvis Happy Hippy", "The Zoo and You",
        410, "Alternative"));
        allSongs.add(new Song("Tiles and Smiles",
        "The Leons", "Album ZX", 350, "Rock"));
        allSongs.add(new Song("For my Friends",
        "Catherine Torres", "Short Stories for
        All", 440, "Spoken"));
        allSongs.add(new Song("At the Beach", "The
        Judges", "Bubblegum Two", 350, "Pop"));
```

```
14.         allSongs.add(new Song("Fireplace
            Fairytales", "Jarvis Happy Hippy", "The
            Patterns", 410, "Alternative"));
15.         allSongs.add(new Song("Static Noise", "The
            Leons", "Fibo and Recur", 320, "Rock"));
16.
17.         Playlist party = new Playlist("PartyTime",
            "Party");
18.
19.         for (int i = 0; i <allSongs.size(); i++) {
20.             if (allSongs.get(i).getGenre().
                equals("Pop")) {
21.                 party.addSong(allSongs.get(i));
22.             }
23.         }
24.         System.out.println(party.getSong(1).
            toString());
25.     }
26. }
```

Output:

Name: Shiny Sparkles, Artist: Sprites, Album: Bubblegum

Aggregation in Python

In this example, the Song class is used within the Playlist class to store details about each song on the playlist. This is an example of an aggregated relationship. The Song class is also shown for reference.

Playlist class in Python

```
1.  from Song import Song
2.
3.  class Playlist:
4.
5.      def __init__(self, name, purpose):
6.
7.          self.__name = name
8.          self.__purpose = purpose
9.          self.__length = 0
10.         self.__playlistSongs = []
11.
12.     def getName(self):
13.         return self.__name
14.
15.     def getPurpose(self):
16.         return self.__purpose
17.
18.     def getLength(self):
19.         return self.__length
20.
21.     def getSong(self, x):
22.         return self.__playlistSongs[x]
23.
24.     def setSong(self, x):
25.         self.__playlistSongs.append(x)
26.
27.     def __str__(self):
28.         return "Name: " + self.__name + ", Purpose:
            " + self.__purpose +", Length: " + self.__
            length
29.
```

AHL

Song class in Python

```
class Song:

    def __init__(self, name, artist, album, length,
    genre):

        self.__name = name
        self.__artist = artist
        self.__album = album
        self.__length = length
        self.__genre = genre

    def getName(self):
        return self.__name
    def getArtist(self):
        return self.__artist
    def getAlbum(self):
        return self.__album
    def getLength(self):
        return self.__length
    def getGenre(self):
        return self.__genre
    def __str__(self):
        return "Name: " + self.__name + ", Artist: "
        + self.__name + ", Album: " + self.__album
```

The main method uses the Song class to make a list of songs. This list of songs is then used to populate the playlist. Although the playlist makes use of the Song class, the Song class can exist alone.

Main method in Python

```
from Song import Song
from Playlist import Playlist

allSongs = []

allSongs.append(Song("Low Lights", "The Judges",
"Bridge over Gray", 340, "Pop"))
allSongs.append(Song("Shiny Sparkles", "Sprites",
"Bubblegum", 280, "Pop"))
allSongs.append(Song("Elephant Light",
"Jarvis Happy Hippy", "The Zoo and You", 410,
"Alternative"))
allSongs.append(Song("Tiles and Smiles", "The
Leons", "Album ZX", 350, "Rock"))
allSongs.append(Song("For my Friends", "Catherine
Torres", "Short Stories for All", 440, "Spoken"))
allSongs.append(Song("At the Beach", "The Judges",
"Bubblegum Two", 350, "Pop"))
allSongs.append(Song("Fireplace Fairytales",
"Jarvis Happy Hippy", "The Patterns", 410,
"Alternative"))
allSongs.append(Song("Static Noise", "The Leons",
"Fibo and Recur", 320, "Rock"))

party = Playlist("PartyTime", "Party")

for x in allSongs:
    if x.getGenre() == "Pop":
        party.setSong(x)

print(party.getSong(1).__str__())
```

Output:

Name: Shiny Sparkles, Artist: Sprites, Album: Bubblegum

AHL

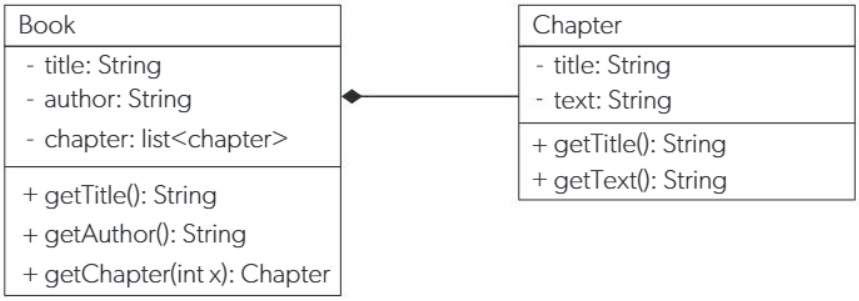
Key term

Composition When two classes rely on each other but neither can exist without the other.

Composition

A **composition** relationship is a relationship between two objects where the objects cannot exist separately from each other. It is a more dependent form of aggregation. This is sometimes called a `part_of` relationship.

For example, consider a book and a chapter. The book cannot exist without chapters, and a chapter cannot exist without a book. This could be represented using this UML diagram.



▲ **Figure 29** UML diagram of a composition relationship

This is a composition relationship. Composition relationships are bi-directional. For example, a book has to have chapters and chapters have to have books. They cannot exist separately. If we deleted the book, all of the chapters would also disappear. As with other aggregated relationships, one object exists inside the other.

Understanding how objects relate to one another is useful when developing relational databases. You will find this in section A3.2.2.

Composition in Java

In this class. the Course class is used within the School class to store details about each course in the school. If you delete the instance of the School class then you lose all the information about the courses within the School. The two are tightly coupled—the school cannot exist without courses and the courses cannot exist without the school. This makes sense in this situation, as all schools represent their courses in their own way. This is an example of a composition relationship. The Course class is also shown for reference.

School class in Java

```
1.  import java.util.ArrayList;
2.
3.  public class School {
4.
5.      private String name;
6.      private String address;
7.      private String age;
8.      private ArrayList <Course> courses;
9.
10.     public School(String name, String address,
        String age) {
11.         this.name = name;
12.         this.address = address;
13.         this.age = age;
14.         courses = new ArrayList <Course>();
```

```
15.     }
16.
17.     public String getName() {
18.         return name;
19.     }
20.     public String getAddress() {
21.         return address;
22.     }
23.     public String getAge() {
24.         return age;
25.     }
26.     public Course getCourse(int x) {
27.         return courses.get(x);
28.     }
29.     public void addCourse(Course x) {
30.         courses.add(x);
31.     }
32.     public int getSize() {
33.         return courses.size();
34.     }
35.     public void removeCourse(String x) {
36.
37.         int index = -1;
38.         for (int i = 0 ; i < courses.size(); i++) {
39.             if (x.equals(courses.get(i).getID())){
40.                index = i;
41.             }
42.         }
43.         if (index > -1) {
44.             courses.remove(index);
45.         }
46.     }
47.
48. }
```

Course class in Java

```
1.  public class Course {
2.
3.      private String identification;
4.      private String name;
5.      private String level;
6.
7.      public Course(String identification, String
        name, String level) {
8.          this.identification = identification;
```

```
9.          this.name = name;
10.         this.level = level;
11.     }
12.
13.     public String getID() {
14.         return identification;
15.     }
16.     public String getName() {
17.         return name;
18.     }
19.     public String getLevel() {
20.         return level;
21.     }
22.     public String toString() {
23.         return "ID: " + identification + ", Name: "
            + name + ", Level: " + level;
24.     }
25.
26. }
```

The main method uses the Course class to make a list of courses in the school. These are added and stored within the instance of the School class. The courses do not exist outside of the school.

Main method in Java

```
1.  public class MainMethod {
2.
3.      public static void main(String[] args) {
4.
5.          School theSchool = new
            School("Blossomhill", "Riverside Drive, HL,
            6332", "11 - 18");
6.
7.          theSchool.addCourse(new Course("CS001",
            "Computer Science Basics", "LS"));
8.          theSchool.addCourse(new Course("HS002",
            "Human Geography", "LS"));
9.          theSchool.addCourse(new Course("CS002",
            "Computer Science Advanced", "HS"));
10.         theSchool.addCourse(new Course("DT001",
            "Human Centered Design", "HS"));
11.         theSchool.addCourse(new Course("DT002",
            "Designing Solutions", "HS"));
12.         theSchool.addCourse(new Course("MT001",
            "Algebra 101", "HS"));
13.         theSchool.addCourse(new Course("MT002",
            "Algebra 102", "HS"));
14.
15.         theSchool.removeCourse("DT002");
16.
```

AHL

```
17.         for(int i = 0; i < theSchool.getSize();
            i++) {
18.             System.out.println(theSchool.
                getCourse(i).toString());
19.         }
20.     }
21. }
```

Output:

ID: CS001, Name: Computer Science Basics, Level: LS

ID: HS002, Name: Human Geography, Level: LS

ID: CS002, Name: Computer Science Advanced, Level: HS

ID: DT001, Name: Human Centered Design, Level: HS

ID: MT001, Name: Algebra 101, Level: HS

ID: MT002, Name: Algebra 102, Level: HS

Composition in Python

In this class, the Course class is used within the School class to store details about each course in the school. If you delete the instance of the School class then you lose all the information about the courses within the school. The two are tightly coupled—the school cannot exist without courses and the courses cannot exist without the school. This makes sense in this situation, as all schools represent their courses in their own way. This is an example of a composition relationship. The Course class is also shown for reference.

School class in Python

```
1.  class School:
2.
3.      def __init__ (self, name, address, age):
4.          self.__name = name
5.          self.__address = address
6.          self.__age = age
7.          self.__courses = []
8.
9.      def getName(self):
10.         return self.__name
11.     def getAddress(self):
12.         return self.__address
13.     def getAge(self):
14.         return self.__age
15.     def getCourse(self, x):
16.         return self.__courses[x]
17.     def addCourse(self, x):
18.         self.__courses.append(x)
19.     def getSize(self):
20.         return len(self.__courses)
21.     def removeCourse(self, x):
22.         index = -1
23.         for i, course in enumerate(self.__courses):
24.             if(self.__courses[i].getID() == x):
25.                 index = i
26.
27.             if (index > -1):
28.                 del(self.__courses[index])
```

AHL

Course class in Python

```
class Course:

    def __init__ (self, identification, name, level):
        self.__identifcation = identification
        self.__name = name
        self.__level = level

    def getID(self):
        return self.__identifcation
    def getName(self):
        return self.__name
    def getLevel(self):
        return self.__level
    def __str__(self):
        return "ID: " + self.__identifcation + ", Name:
        " + self.__name + ", Level: " + self.__level
```

The main method uses the Course class to make a list of courses in the school. These are added and stored within the instance of the School class. The courses do not exist outside of the school.

Main method in Python

```
from Course import Course
from School import School

theSchool = School("Blossomhill", "Riverside
Drive, HL, 6332", "11 - 18")

theSchool.addCourse(Course("CS001", "Computer
Science Basics", "LS"))
theSchool.addCourse(Course("HS002", "Human
Geography", "LS"))
theSchool.addCourse(Course("CS002", "Computer
Science Advanced", "HS"))
theSchool.addCourse(Course("DT001", "Human Centered
Design", "HS"))
theSchool.addCourse(Course("DT002", "Designing
Solutions", "HS"))
theSchool.addCourse(Course("MT001", "Algebra 101",
"HS"))
theSchool.addCourse(Course("MT002", "Algebra 102",
"HS"))

theSchool.removeCourse("DT002")

for x in range (0, theSchool.getSize()):
    print(theSchool.getCourse(x).__str__())
```

Output:

ID: CS001, Name: Computer Science Basics, Level: LS

ID: HS002, Name: Human Geography, Level: LS

ID: CS002, Name: Computer Science Advanced, Level: HS

ID: DT001, Name: Human Centered Design, Level: HS

ID: MT001, Name: Algebra 101, Level: HS

ID: MT002, Name: Algebra 102, Level: HS

ATL Thinking skills

When identifying relationships, you are looking to examine how the classes are interrelated and at which points. Decomposition can help you with this. You can break down the problems and identify the different overlap between classes, which helps you to identify the relationship.

For each of the following relationships, identify whether you would use aggregation or composition, and explain why.

- Television show and actor
- Car and engine
- Mobile phone and apps
- Hotel and hotel room
- Town and tourist attractions

Activity

You have created many programs that have relationships within them. Spend some time reviewing your programs. Where have you used an aggregated relationship? Where have you used a composition relationship? How do you know?

Which type of relationship do you prefer to work with when coding? Explain why.

Constructing code with multiple classes

Practical skills

It is important to understand how to use programming structures with multiple classes. Being able to access the data correctly in the classes is essential when creating programs, in order to store data correctly and enable users to view the correct data. You need to understand how the code fits together in order to understand how to access the data.

If you have a list instance variable that is of a class variable type contained within a class, it is advisable to have the following: a method that returns the size of the list within the class, a method that returns a specific instance of that class, and a method that allows adding an instance of that class to the list. Having these within the class hosting the list will enable you to access specific instances from within the list more easily.

The following extracts of code show how to create a list of a `classType` within another class, how to add to this list from the driver class, and how to iterate through the class from the driver class.

Multiple classes in Java

An example of this is shown in the revisited School class from the composition example. Shown below, the School class has a list containing the courses available in the school. This list is contained within the School class. In order to make this list easy to use in the driver class, a few helper methods have been added.

- `public int getSize()`
  This returns the size of the `ArrayList` so we can iterate through it easily in the driver class.
- `public Course getCourse(int x)`
  This returns a class from a specific space in the list, again helping to manipulate it from the driver class.
- `public void addCourse (Course x)`
  This class allows us to add new instances of the course from the driver class. When developing code with multiple classes it is worth considering adding these methods habitually.



School class in Java

```
1.  import java.util.ArrayList;
2.
3.  public class School {
4.
5.      private String name;
6.      private String address;
7.      private String age;
8.      private ArrayList <Course> courses;
9.
10.     public School(String name, String address,
        String age) {
11.         this.name = name;
12.         this.address = address;
13.         this.age = age;
14.         courses = new ArrayList <Course>();
15.     }
16.
17.     public String getName() {
18.         return name;
19.     }
20.     public String getAddress() {
21.         return address;
22.     }
23.     public String getAge() {
24.         return age;
25.     }
26.     public Course getCourse(int x) {
27.         return courses.get(x);
28.     }
29.     public void addCourse(Course x) {
30.         courses.add(x);
31.     }
32.     public int getSize() {
33.         return courses.size();
```

AHL

```
34.     }
35.     public void removeCourse(String x) {
36.
37.         int index = -1;
38.         for (int i = 0 ; i < courses.size(); i++) {
39.             if (x.equals(courses.get(i).getID())){
40.                index = i;
41.             }
42.         }
43.         if (index > -1) {
44.             courses.remove(index);
45.         }
46.     }
47.
48. }
```

The main method may use the methods you have added in the a similar manner to those shown below. The code below creates an instance of the School class and stores it within the variable **`theSchool`**. The School instance contains a list of courses. The code below shows how the methods from the School instance are used to access the list within the class by the driver class. The **`public int getSize()`** method is used to end the loop searching through each of the courses. This prevents an error occurring by trying to view a value beyond those available. The **`public void addCourse(Course x)`** method has been used to add an instance of the course variable to the list within the instance of the School class. Finally the **`public Course getCourse(int x)`** method has been used to access the different instances of the course with the loop.

Main method in Java

```
1.  public class MainMethod {
2.
3.      public static void main(String[] args) {
4.
5.          School theSchool = new
            School("Blossomhill", "Riverside Drive, HL,
            6332", "11 - 18");
6.
7.          theSchool.addCourse(new Course("CS001",
            "Computer Science Basics", "LS"));
8.          theSchool.addCourse(new Course("HS002",
            "Human Geography", "LS"));
9.          theSchool.addCourse(new Course("CS002",
            "Computer Science Advanced", "HS"));
10.         theSchool.addCourse(new Course("DT001",
            "Human Centered Design", "HS"));
11.         theSchool.addCourse(new Course("DT002",
            "Designing Solutions", "HS"));
12.         theSchool.addCourse(new Course("MT001",
            "Algebra 101", "HS"));
13.         theSchool.addCourse(new Course("MT002",
            "Algebra 102", "HS"));
```

```
14.
15.         theSchool.removeCourse("DT002");
16.
17.         for(int i = 0; i < theSchool.getSize(); i++) {
18.             if(theSchool.getCourse(i).getLevel().
                equals("HS")) {
19.                 System.out.println(theSchool.
                    getCourse(i).toString());
20.                 }
21.         }
22.     }
23. }
```

Output:

ID: CS002, Name: Computer Science Advanced, Level: HS

ID: DT001, Name: Human Centered Design, Level: HS

ID: MT001, Name: Algebra 101, Level: HS

ID: MT002, Name: Algebra 102, Level: HS

Multiple classes in Python

An example of this is shown in the revisited School class from the composition example. Shown below, the School class has a list containing the courses available in the school. This list is contained within the School class. In order to make this list easy to use in the driver class, a few helper methods have been added.

- `def getSize(self)`
  This returns the size of the list so we can iterate through it easily in the driver class.
- `def getCourse(self, x)`
  This returns a class from a specific space in the list, again helping to manipulate it from the driver class.
- `def addCourse (self, x)`
  This class allows us to add new instances of the course from the driver class. When developing code with multiple classes it is worth considering adding these methods habitually.

School classes in Python

```
1.  class School:
2.
3.      def __init__ (self, name, address, age):
4.          self.__name = name
5.          self.__address = address
6.          self.__age = age
7.          self.__courses = []
8.
9.      def getName(self):
10.         return self.__name
11.     def getAddress(self):
12.         return self.__address
13.     def getAge(self):
14.         return self.__age
15.     def getCourse(self, x):
16.         return self.__courses[x]
```

```
    def addCourse(self, x):
        self.__courses.append(x)
    def getSize(self):
        return len(self.__courses)
    def removeCourse(self, x):
        index = -1
        for i, course in enumerate(self.__courses):
            if(self.__courses[i].getID() == x):
                index = i

        if (index > -1):
            del(self.__courses[index])
```

The main method may use the methods you have added in the a similar manner to those shown below. The code below creates an instance of the School class and stores it within the variable **`theSchool`**. The School instance contains a list of courses. The code below shows how the methods from the School instance are used to access the list within the class by the driver class. The **`getSize(self)`** method is used to end the loop searching through each of the courses. This prevents an error occurring by trying to view a value beyond those available. The **`addCourse(self, x)`** method has been used to add an instance of the course variable to the list within the instance of the School class. Finally, the **`getCourse(self x)`** method has been used to access the different instances of the course with the loop.

Main method in Python

```
from Course import Course
from School import School

theSchool = School("Blossomhill", "Riverside Drive, HL, 6332", "11 - 18")

theSchool.addCourse(Course("CS001", "Computer Science Basics", "LS"))
theSchool.addCourse(Course("HS002", "Human Geography", "LS"))
theSchool.addCourse(Course("CS002", "Computer Science Advanced", "HS"))
theSchool.addCourse(Course("DT001", "Human Centered Design", "HS"))
theSchool.addCourse(Course("DT002", "Designing Solutions", "HS"))
theSchool.addCourse(Course("MT001", "Algebra 101", "HS"))
theSchool.addCourse(Course("MT002", "Algebra 102", "HS"))

theSchool.removeCourse("DT002")

for x in range (0, theSchool.getSize()):
    if (theSchool.getCourse(x).getLevel() == "HS"):
        print(theSchool.getCourse(x).__str__())
```

Output:

ID: CS002, Name: Computer Science Advanced, Level: HS

ID: DT001, Name: Human Centered Design, Level: HS

ID: MT001, Name: Algebra 101, Level: HS

ID: MT002, Name: Algebra 102, Level: HS

Removing instances from a list

If you can add and search the list within a class in the driver class then you also need to be able to remove it. Being able to remove from the list is essential. The class containing the list should have a method to help with this. The method will usually take a variable as a parameter that allows you to search for the correct item to remove within the list. When developing code within multiple classes it is advised to develop methods that enable users to remove instances from the list.

Removing instances in Java

In the School class from the composition example there is a **`public void removeCourse (String x)`** method which allows a course identification to be added to the code. This code is then used to search the list and remove the course if found. The code does not execute anything.

Example in Java

```
1.  public void removeCourse(String x) {
2.
3.          int index = -1;
4.          for (int i = 0 ; i < courses.size(); i++) {
5.              if (x.equals(courses.get(i).getID())){
6.                  index = i;
7.              }
8.          }
9.          if (index > -1) {
10.             courses.remove(index);
11.         }
12.     }
```

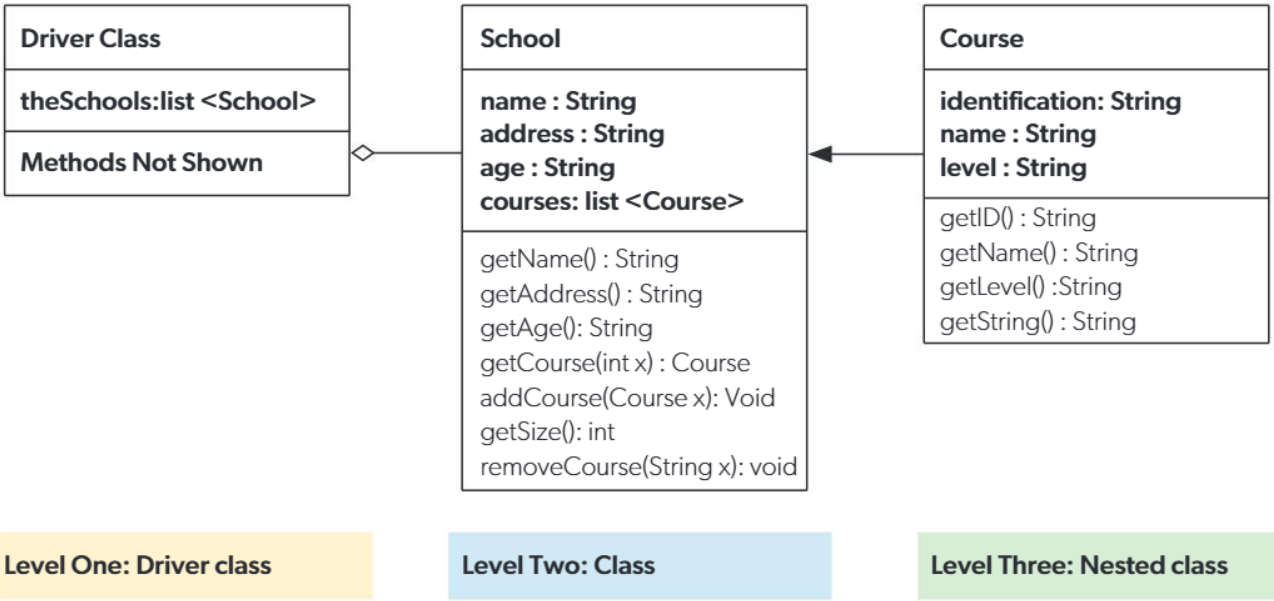
Removing instances in Python

In the School class from the composition example there is a **`def removeCourse (self, x)`** method which allows a course identification to be added to the code. This code is then used to search the list and remove the course if found. The code does not execute anything.

Example in Python

```
1.  def removeCourse(self, x):
2.          index = -1
3.          for i, course in enumerate(self.__courses):
4.              if(self.__courses[i].getID() == x):
5.                  index = i
6.
7.          if (index > -1):
8.              del(self.__courses[index])
```

You need to understand how to reference different methods when using multiple classes. This enables you to check the different values within the instances in the list, to search and sort effectively, and to access the information within the encapsulated classes. Consider the following example.



▲ **Figure 30** UML diagram of an if statement

In the UML diagram, you have the driver class which contains a list of Schools. This example will be used to explain how dot notation works with multiple classes.

Dot notation in Java

In the code below, there are different levels of dot notation being used. The first example is accessing each instance of the School class using the `.get()` method.

The first use of level one dot notation is in the line of code below. The `.get()` is used to access each instance of the school within `theSchools`. If a school is for ages 11–18 then HS level courses are added. If a school is for ages 5–11 then LS course are added. The `.get()` method in this case gets the instance of the school.

```
theSchools.get(i).addCourse(new Course("MT001", "Algebra
101", "HS"));
```

`theSchools.get(i)` accesses the instance of the school and then adds the course into the courses list within that instance.

To get information from this class, you need to access the instance of the class and then use the accessor methods of that class to discover what the instances of the class contain.

If you think about the levels, this could be thought of as second-level accessing, as you are accessing the instance and then accessing the information within the instance. For example:

```
theSchools.get(i).getAge().contains("11–18")
```

- `theSchools.get(i)`
  Gets the instance of the school.
- `.getAge()`
  Gets the age range of that instance of school.

Key term

Accessing In programming, accessing means getting information from the class.

To get information from an instance within the current object—for example, in this program you want to access a specific school object and then the specific instance of the course which will allow you to find course information—you need to use another level of dot notation.

This can be seen in the following code.

```
if(theSchools.get(i).getCourse(j).getName().
contains("Algebra"))
```

- `theSchools.get(i)`
  Gets the instance of the school (level one).
- `.getCourse()`
  Gets the instance of the course within the instance of the school (level two).
- `.getName()`
  Gets the name of the course within the current instance of the course (level three).



Dot notation example in Java

```
1.    public static void main(String[] args) {
2.
3.          ArrayList <School> theSchools = new
            ArrayList <School>();
4.
5.          theSchools.add(new School("Blossomhill",
            "Riverside Drive, HL, 6332", "11 - 18"));
6.          theSchools.add(new School("IS Porta Susa",
            "Corso San Patrizio, IT, 38292", "11 - 18"));
7.          theSchools.add(new School("MS Fort
            Clifford", "Boulevard Michel, FR, 88844",
            "5 - 11"));
8.
9.          for(int i = 0; i < theSchools.size(); i++)
{
10.
11.             if(theSchools.get(i).getAge().
                contains("11 - 18")) {
12.                 theSchools.get(i).addCourse(new
                    Course("CS002", "Computer Science
                    Advanced", "HS"));
13.                 theSchools.get(i).addCourse(new
                    Course("DT001", "Human Centered
                    Design", "HS"));
14.                 theSchools.get(i).addCourse(new
                    Course("DT002", "Designing
                    Solutions", "HS"));
15.                 theSchools.get(i).addCourse(new
                    Course("MT001", "Algebra 101",
                    "HS"));
16.                 theSchools.get(i).addCourse(new
                    Course("MT002", "Algebra 102", "HS"));
17.             }
18.             else {
19.                 theSchools.get(i).addCourse(new
                    Course("CS001", "Computer Science
                    Basics", "LS"));
```

```
20.                 theSchools.get(i).addCourse(new
                    Course("HS002", "Human Geography",
                    "LS"));
21.             }
22.         }
23.
24.         for(int i = 0; i < theSchools.size(); i++)
{
25.             for (int j = 0; j < theSchools.get(i).
                getSize(); j++) {
26.
27.             System.out.println(theSchools.get(i).
                getCourse(j).toString());
28.
29.             }
30.         }
31.     }
32. }
33.
```

The whole code for the main method above shows the different levels of dot notation in use.

Another key thing to note is that if you want to loop through all instances of a class and then all instances of the class within that class contained within a list (such as the courses within schools) then you need a double loop such as the one demonstrated in lines 24–30.

Dot notation in Python

In the code below, the different levels of dot notation are in use. The first example is accessing each instance of the school class using a for loop.

The first use of level one dot notation can be viewed in line of code below. The X is used to access each instance of the school within **theSchools** list. If a school is for ages 11–18 then HS level course are added. If a school is for ages 5–11 then LS course are added. The X in this case gets the instance of the school.

```
for X in theSchools:
    if(X.getAge()=="11 - 18"):
        X.addCourse(Course ("CS002", "Computer Science
Advanced", "HS"))
```

X accesses the instance of the school and then adds the course into the courses list within that instance.

To get information from this class, we need to access the instance of the class and then use the accessor methods of that class to discover what the instances of the class contains.

If you think about the levels, this could be thought of as second-level accessing, as you are accessing the instance and then accessing the information within the instance. For example:

`if(X.getAge()=="11 - 18"):`

- **X** Gets the instance of the school.
- **.getAge()** Gets the age range of that instance of school.

To get information from an instance within the current object—for example, in this program you want to access a specific school object and then the specific instance of the course which will allow you to find course information—you need to use another level of dot notation.

This can be seen in the following code.

```
for i in range(0, len(theSchools)):
    for j in range (0, theSchools[i].getSize()):
        if ("Algebra" in theSchools[i].getCourse(j).
        getName()):
            print(theSchools[i].getCourse(j).toString())
```

- `theSchools[i]`
  Gets the instance of the school (level one).
- `.getCourse(j)`
  Gets the instance of the course within the instance of the school (level two).
- `.getName()`
  Gets the name of the course within the current instance of the course (level three).



Dot notation example in Python

```
1.  from Course import Course
2.  from School import School
3.
4.  theSchools = []
5.  theSchools.append(School("Blossomhill", "Riverside
    Drive, HL, 6332", "11 - 18"))
6.  theSchools.append(School("IS Porta Susa", "Corso
    San Patrizio, IT, 38292", "11 - 18"))
7.  theSchools.append(School("MS Fort Clifford",
    "Boulevard Michel, FR, 88844", "5 - 11"))
8.
9.  for X in theSchools:
10.     if(X.getAge()=="11 - 18"):
11.         X.addCourse(Course ("CS002", "Computer
            Science Advanced", "HS"))
12.         X.addCourse(Course("DT001", "Human Centered
            Design", "HS"))
13.         X.addCourse(Course("DT002", "Designing
            Solutions", "HS"))
14.         X.addCourse(Course("MT001", "Algebra 101",
            "HS"))
15.         X.addCourse(Course("MT002", "Algebra 102",
            "HS"))
16.
17.     else:
18.         X.addCourse(Course("CS001", "Computer
            Science Basics", "LS"))
19.         X.addCourse(Course("HS002", "Human
            Geography", "LS"))
20.
21.
```

```
22. for i in range(0, len(theSchools)):
23.     for j in range (0, theSchools[i].getSize()):
24.         if ("Algebra" in theSchools[i].
            getCourse(j).getName()):
25.             print(theSchools[i].getCourse(j).__
                str__())
```

The whole code for the main method above shows the different levels of dot notation in use.

Another key thing to note is that if you want to loop through all instances of a class and then all instances of the class within that class contained within a list (such as the courses within schools) then you need a double loop such as the one demonstrated in lines 22–25.

Activity

You have developed several programs using multiple classes. Now, choose one of these problems (or come up with your own) and use computational thinking patterns and the other skills you have learned to solve it.

- The theatre group wants to organize props, costumes, microphones, and other equipment for the school play. Develop a program that stores this information, scene by scene.
- The school rugby club wants to develop off-season training routines for the team. Develop a program that stores a personalized training routine for each player.
- A local business is developing an inventory to store all the information about their shops. This includes the stock, customers and sales. Develop a program that allows the shop to store this information, as well as what each customer bought.

B3.2.5 Explain commonly used design patterns in OOP

Design patterns are like a toolkit for software engineers—they are a blueprint of known solutions to recurring problems. Design patterns are not code. Instead, they can be thought of as instructions telling you how to tackle different problems when trying to design a solution. It is sensible to use design patterns when designing solutions because they are proven to be successful. Patterns are not specific to one particular coding software but should be thought of as a guideline for developing the required code. If you follow the pattern to implement the suggested code, then you can develop robust solutions to your problems. It is important to remember that the design patterns will not give you the code.

Several design patterns are described below and contain example code in both Java and Python. This code has been provided to show you how the code could be implemented and the fundamentals of how the code could work. The code is not intended to be the definitive version of the design pattern code.

One reason developers choose to utilize design patterns is because they can speed up the development process because they are tried and tested. When developing large software solutions, there are many different factors to take into consideration. If you do not factor in the different variables that could affect the software, then issues may arise when the software is in use. Using design patterns helps to negate these problems early in the implementation process as they allow developers to make use of well-known and well-understood conventions for software interactions.

When using design patterns, developers are usually provided with the following information.

Intent: The intent section identifies an overview of the problem and different factors the developer may be facing when deciding whether to use the pattern or not. This may include real-life examples of usage.

Motivation: The motivation section describes the type of patterns the data may display and the types of actions the programmer may want to achieve with the solution. The motivation section then explains how the pattern can be used to implement the actions required.

Structure: Usually in the form of a UML diagram, the structure shows the classes required, the interaction of the classes within the pattern and its related components.

Code example: A programming language provides a code example. In this book you have examples in Python and Java to show the production of classes and the interaction between the classes. They also show how data can be passed between the classes.

Developers can use many different design patterns. These patterns have been broadly split into three different types.

- **Creational design patterns** focus on objects and how they are created. Creational design patterns look to maximize the reuse and flexibility of code, to develop robust reliable programs. Creational design patterns will often look to make use of inheritance and encapsulation.
- **Structural design patterns** enable software engineers to easily identify the relationships between the different entities in a system. They show how to take smaller parts of a system and put them into a larger working system while focusing on keeping the system flexible to adapt to future changes.
- **Behavioural design patterns** focus on the behaviour of objects, looking at their communication and how they interact with each other. When patterns exist within communication, this is leveraged to make flexible, reusable code.

You may find other design patterns useful when developing a solution for your Internal Assessment.

There are more than twenty identified design patterns for software engineering, each with different functions. For this course, you need to be able to explain the functions of the singleton, factory, and observer design patterns.

ATL Research skills and thinking skills

To use a design pattern effectively, you need to research the different patterns available and how these can link to the problem you are solving. Spending time identifying the problem and developing a specification for it can help you to understand the outcome(s) you want to achieve and identify useful design patterns.

When you have a problem specification, you can then break the problem down into its component parts. Decomposition can help with this.

Imagine you have been asked to develop a program to organize the rental of hire cars. The company has a selection of cars in the following categories: budget, mini, classic and van. Cars are rented out to customers. The cost of the rental is the cost of the car plus 5% per day for insurance. Each rental has a start date and end date.

1. Research the available design patterns: singleton, factory, and observer.
2. Decompose this problem and develop a problem specification.
3. What patterns are emerging? Which design pattern would help you solve this problem?

Singleton design pattern

A singleton object allows us to have one instance of an item with many other items able to access it. You use singleton when you want to ensure there can only be one instance of an item, such as a database.



▲ **Figure 31** A stadium is one instance with many people able to access it

You can contextualize this by thinking of a football stadium where your favourite team plays. There can only be one copy of that football stadium but you want many different people to be able to access it otherwise the games would be very quiet.

Similarly, if you are buying tickets for a concert, you only want there to be one set of seats available, with many different people able to buy them. If there were multiple different versions of the seats, your seats may be sold to more than one person.

Uses in programming include storing data that everyone needs access to, developing a log file for error messages, and creating config files.

Intent

The intent of the singleton design pattern is to allow global access to one resource but ensure that there is only one instance of that resource that everyone is accessing.

Motivation

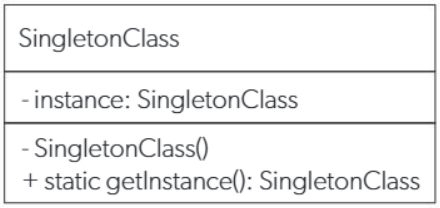
Ensuring a class has one single instance is usually a file or a database type structure. In the past, global variables have been used to create a global resource but this can be complicated and can lead to errors such as accidental deletion of data or multiple entries of the same type, leading to unreliable data. Using global variables to store essential data is also not a good idea because it is hard to track where the variable is being referenced from, it is challenging to track where changes are made, and it leads to further possibilities of data being overwritten. The singleton method allows us to create a single instance of the structure with multiple access in a robust manner.

Solution

The singleton pattern makes use of the following features.

- The constructor is private so the class cannot be instantiated.
- A static instance of the object is created.
- Static accessor/setter methods are created to access and manipulate the members of the class.

Structure



▲ **Figure 32** UML diagram for Singleton class

The UML diagram in Figure 32 shows the structure of a Singleton class. A private instance of the class is declared. We then use a static accessor method `static getInstance()` that is used to access the instance of the class. The constructor is private and only used once when the instance is created.

Singleton design in Java

The following code shows a class that uses the singleton design pattern. Following the example about tickets for a concert, the class has the number of seats available and the option to purchase seats or find out how many seats are available. The constructor method is called once and sets the number of seats. A private static instance of the class is created and this instance is used to make global references to the same resource.

Singleton class in Java

```
1.  public class Singleton {
2.
3.      private static Singleton singInstance = new
        Singleton();
4.      private int seatsAvailable;
5.
6.      private Singleton() {
7.
8.          seatsAvailable = 100;
9.      }
10.
11.     public static Singleton getInstance() {
12.
13.         return singInstance;
14.     }
15.     public void sellSeats(int x) {
16.
17.         seatsAvailable = seatsAvailable - x;
18.     }
19.     public int getSeatsAvailable() {
20.
21.         return seatsAvailable;
22.     }
23. }
```

The main method, as shown below, helps to develop a better understanding of the singleton design method. An object that makes reference to the static instance is created and used to sell tickets. A second object that makes reference to the static instance is created and used to sell tickets as well as enquiring into how many tickets remain on sale. The tickets left are the number of tickets minus the first and second object reference, showing that both objects are making reference to the same global resource.

AHL

Main method in Java

```
1.  public class SingletonMain {
2.
3.      public static void main(String[] args) {
4.
5.          Singleton mySingleObject = Singleton.
            getInstance();
6.
7.          mySingleObject.sellSeats(10);
8.          mySingleObject.sellSeats(20);
9.
10.         Singleton myNewSingleObject = Singleton.
            getInstance();
11.
12.         myNewSingleObject.sellSeats(5);
13.
14.         System.out.println(myNewSingleObject.
            getSeatsAvailable());
15.     }
16. }
```

Output:

65

Singleton design in Python

The code below shows a class that uses the singleton design pattern. Following the example about tickets for a concert, the class has the number of seats available and the option to purchase seats or find out how many seats are available. The constructor method is called once and sets the number of seats. A private static instance of the class is created and this instance is used to make global references to the same resource.

Singleton class Python

```
1. class Singleton:
2.       _instance = None
3.
4.       def __new__(cls):
5.           if cls._instance is None:
6.               cls._instance = super(Singleton,
                 cls).__new__(cls)
7.               cls._instance.seats_available = 100
8.           return cls._instance
9.
10.      def sell_seats(self, number):
11.          self.seats_available -= number
12.
13.      def get_seats_available(self):
14.          return self.seats_available
```

The main method shown below helps to develop a better understanding of the singleton design method. An object is created that references the static instance: this is used to sell tickets. A second object is created that references the static instance: this is used to sell tickets as well as checking how many tickets remain on sale. The number of tickets left is the original number of tickets minus the first and second object reference. This shows that both objects are referring to the same global resource.

Singleton implementation Python

```
1.
2.    if __name__ == "__main__":
3.
4.        singleton_object = Singleton()
5.
6.        singleton_object.sell_seats(10)
7.        singleton_object.sell_seats(20)
8.
9.        new_singleton_object = Singleton()
10.
11.       new_singleton_object.sell_seats(5)
12.
13.       print(new_singleton_object.get_seats_
          available())
```

Output:

65

Advantages of the singleton design pattern

- Helps to keep strict control over shared resources.
- Guarantees that only one instance of the class will be available for use.
- There is only ever one instance of the global resource.

Disadvantages of the singleton design pattern

- Difficult to test.
- There are multiple items accessing the same data item.
- Difficult to delete the instance of the class when no longer required.

Factory design pattern

The factory design pattern allows you to think about having a common design object across all instances. However, the instance decides what kind of object it is. It is commonly implemented using interfaces.

Imagine a school that has students at three stages: primary school, middle school and high school. Primary school, middle school and high school students are similar but they do have qualities that make them different. The factory method would allow you to have a common student interface but the subclass then decides whether it is a primary school student, middle school student or high school student.



▲ **Figure 33** A student class

Another example is a program dealing with one type of product that is very specific, but if it becomes successful the manufacturer may wish to expand their offering and have different products for sale. The factory pattern would allow them to have a common interface for products and then, depending on the type of subclass created, the product would act in different ways.

Intent

The intent of the factory design pattern is to allow easy expansion of programs. A common interface is provided for the objects, and specialisms are added through subclasses. Any number of new objects can be added.

Motivation

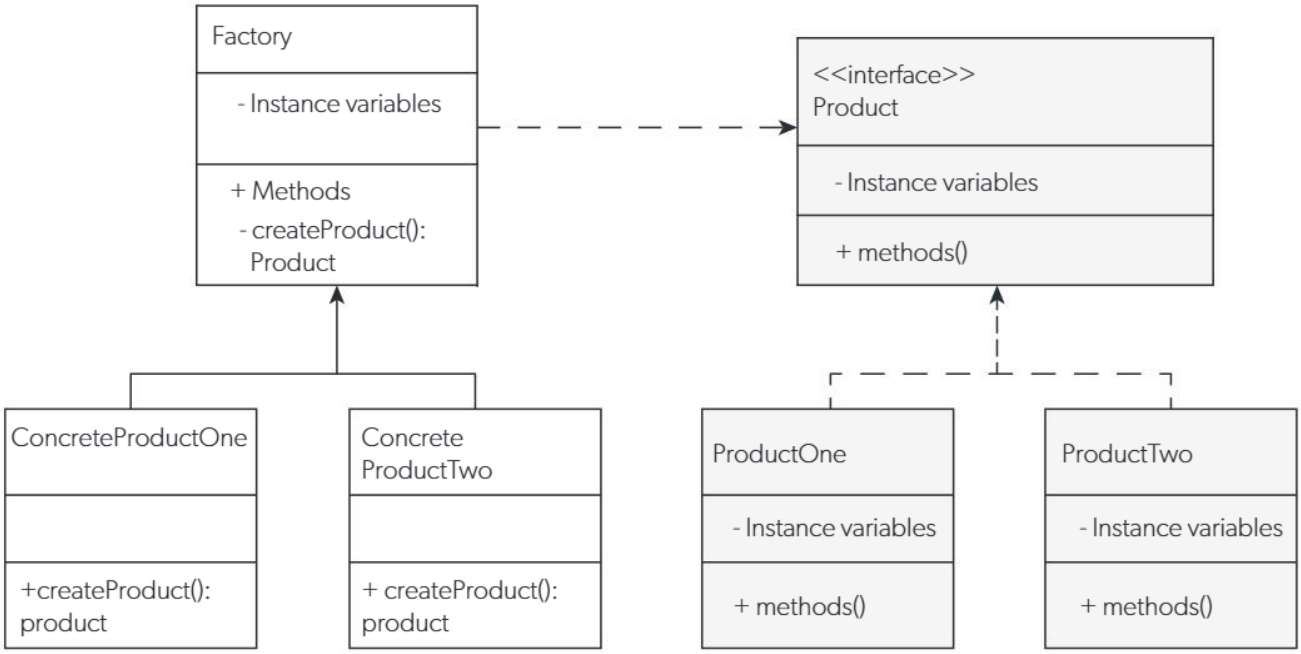
A factory class is created that contains variables and methods associated with being a factory, as well as methods used to create each of the products. Products are usually implemented using an interface. Although it may seem silly to move the constructor to the factory class, this enables the factory to keep a track of the products it has created and all products can be manipulated from directly within the factory class, reducing complexity. For this to work correctly, all products must have the same base class. For this reason an interface is often used. The factory method should be used when you are unsure of the exact products you will be working with. This method can be used to build libraries of products ready for use. It also encourages code reuse.

Solution

The factory pattern uses the following code features.

- An interface for the product class to ensure a common base class between all products.
- Products that are instantiated (concrete) are different implementations of the interface.
- The factory class is used to produce new products. The return type of the create method must match the product interface.
- The factory method can have a central repository of all new instances created which allows centralized management.

Structure



▲ **Figure 34** The structure of the factory design pattern

Factory design in Java

The factory design pattern starts with an interface. The interface contains methods (only methods) common to all concrete products you wish to make. The interface does not contain a constructor. The constructor is implemented in the classes that implement the interface. The example below follows the student example. All methods common to all students are contained within the interface.

Example code in Java

```
1.  public interface Product {
2.     public String getName();
3.     public int getGrade();
4.     public String getHomeroom();
5.     public String getInfo();
6.  }
```

The subclasses allow you to implement different versions of the interface depending on the specifications of the concrete product. Concrete products must contain all of the methods specified in the interface but they can also contain their own. The word "implements" shows that the class implements the interface. The subclasses also need to specify the variables required and must contain a constructor method. The example continues the student example. Concrete product one contains the interface information as well as items specific to the middle school student. Concrete product two contains the interface information as well as items specific to the high school student.

AHL

Concrete product one (a model for a middle school student) in Java

```
1.  public class ConcreteProductOne implements Product {
2.      private String name;
3.      private int grade;
4.      private String homeroom;
5.      private String languageChoice;
6.      public ConcreteProductOne(String name, int
        grade, String homeroom, String languageChoice) {
7.          this.name = name;
8.          this.grade = grade;
9.          this.homeroom = homeroom;
10.         this.languageChoice = languageChoice;
11.     }
12.     public String getName() {
13.         return name;
14.     }
15.     public int getGrade() {
16.         return grade;
17.     }
18.     public String getHomeroom() {
19.         return homeroom;
20.     }
21.     public String languageChoice() {
22.         return languageChoice;
23.     }
24.     public String getInfo() {
25.         return "Name: " + name + ", Grade: " +
            grade + ", Homeroom: " + homeroom + ",
            Language Choice: " + languageChoice;
26.     }
27. }
```

Concrete product two (a model for a high school student) in Java

```
1.  public class ConcreteProductTwo implements Product {
2.      private String name;
3.      private int grade;
4.      private String homeroom;
5.      private int currentPredicted;
6.      public ConcreteProductTwo(String name, int
        grade, String homeroom, int currentPredicted) {
7.          this.name = name;
8.          this.grade = grade;
9.          this.homeroom = homeroom;
```

```
10.         this.currentPredicted = currentPredicted;
11.     }
12.     public String getName() {
13.         return name;
14.     }
15.     public int getGrade() {
16.         return grade;
17.     }
18.     public String getHomeroom() {
19.         return homeroom;
20.     }
21.     public int currentPredicted() {
22.         return currentPredicted;
23.     }
24.     public String getInfo() {
25.         return "Name: " + name + ", Grade: " +
            grade + ", Homeroom: " + homeroom + ",
            Current Predicted: " + currentPredicted;
26.     }
27. }
```

Once you have the interface and the subclasses, you need to have a factory class to control the creation of each product. The factory class contains two methods for creating products: one method for creating concrete product one and one method for creating concrete product two. Each method takes in the specific variables for the subclass they are instantiating.

Factory class in Java

```
1.  public class Factory {
2.      public Factory() {
3.      }
4.      public Product createOne(String name, int
        grade, String homeroom, String language) {
5.          return new ConcreteProductOne(name, grade,
            homeroom, language);
6.      }
7.      public Product createTwo(String name, int
        grade, String homeroom, int predicted) {
8.          return new ConcreteProductTwo(name, grade,
            homeroom, predicted);
9.      }
10. }
```

The main method contains an array list to store all instances of the product that have been created. This allows users to control all instances from the same place. Each item is classed as a Product as it makes use of the product interface. However, each instance of the product acts in accordance with the type of product it has been declared as.

AHL

Main method in Java

```
1.  import java.util.ArrayList;
2.  public class MainMethod {
3.      public static void main(String[] args) {
4.          Factory myFactory = new Factory();
5.          ArrayList <Product> myStudents = new
            ArrayList <Product>();
6.          myStudents.add(myFactory.
            createOne("Hannah", 7, "7LS", "German"));
7.          myStudents.add(myFactory.
            createTwo("Wesley",11, "11KM", 39 ));
8.          for (int i = 0; i < myStudents.size(); i++)
{
9.          System.out.println(myStudents.get(i).
            getInfo());
10.         }
11. }
12. }
```

Factory design in Python

The factory design pattern starts with an interface. The interface contains methods (only methods) common to all concrete products you wish to make. The interface does not contain a constructor. The constructor is implemented in the classes that implement the interface. The example below follows the student example.
All methods common to all students are contained within the interface. In Python, the concept of an interface is typically realized through abstract classes, made possible by the ABC module (ABC for Abstract Base Classes). The @abstractmethod decorator requires that each derived class must implement these methods.

Factory design pattern in Python

```
1.   from abc import ABC, abstractmethod
2.
3.   class Product(ABC):
4.       @abstractmethod
5.       def get_name(self):
6.           pass
7.
8.       @abstractmethod
9.       def get_grade(self):
10.          pass
11.
12.      @abstractmethod
13.      def get_homeroom(self):
14.          pass
15.
16.      @abstractmethod
17.      def get_info(self):
18.          pass
```

The subclasses allow you to implement different versions of the interface depending on the specifications of the concrete product. The concrete product must contain all of the methods specified in the interface but they can also contain their own. The word "implements" shows that the class implements the interface. The subclasses also need to specify the variables required and must contain a constructor method. The example continues the student example. Concrete product one contains the interface information as well as items specific to the middle school student. Concrete product two contains the interface information as well as items specific to the high school student.

Concrete product one in Python

```
class ConcreteProductOne(Product):
    def __init__(self, name, grade, homeroom, language_choice):
        self.name = name
        self.grade = grade
        self.homeroom = homeroom
        self.language_choice = language_choice

    def get_name(self):
        return self.name

    def get_grade(self):
        return self.grade

    def get_homeroom(self):
        return self.homeroom

    def get_info(self):
        return f"Name: {self.name}, Grade: {self.grade}, Homeroom: {self.homeroom}, Language Choice: {self.language_choice}"


```

Concrete product two in Python

```
class ConcreteProductTwo(Product):
    def __init__(self, name, grade, homeroom, current_predicted):
        self.name = name
        self.grade = grade
        self.homeroom = homeroom
        self.current_predicted = current_predicted

    def get_name(self):
        return self.name

    def get_grade(self):
```

```
        return self.grade

    def get_homeroom(self):
        return self.homeroom

    def current_predicted(self):
        return self.current_predicted

    def get_info(self):
        return f"Name: {self.name}, Grade: {self.grade}, Homeroom: {self.homeroom}, Current Predicted: {self.current_predicted}"

```

Once you have the interface and the subclasses, you need to have a factory class to control the creation of each product. The factory class contains two methods for creating products: one method for creating concrete product one, and one method for creating concrete product two. Each method takes in the specific variables for the subclass it is instantiating.

Factory class in Python

```
class Factory:
    def create_one(self, name, grade, homeroom, language):
        return ConcreteProductOne(name, grade, homeroom, language)

    def create_two(self, name, grade, homeroom, predicted):
        return ConcreteProductTwo(name, grade, homeroom, predicted)
```

The main method contains a list to store all instances of the product that have been created. This allows users to control all instances from the same place. Each item is classed as a Product as it makes use of the product interface. However, each instance of the product acts in accordance with the type of product it has been declared as.

Factory main in Python

```
def main():
    my_factory = Factory()

    my_students = []

    my_students.append(my_factory.create_one("Hannah", 7, "7LS", "German"))
    my_students.append(my_factory.create_two("Wesley", 11, "11KM", 39))

    for student in my_students:
        print(student.get_info())
```

```
11.
12.  if __name__ == "__main__":
13.      main()
```

Advantages of the factory design pattern

- You avoid composition relationships between the factory and the concrete products.
- You have all the code in one place, which makes the code easier to maintain.
- You can develop new products without breaking any existing code.

Disadvantages of the factory design pattern

- The code may become over complex because you need to have many subclasses to create the concrete products.

Observer design pattern

In a system, it is important that different objects are kept up to date with changes in other objects. This is because most systems are interdependent. The observer pattern allows us to keep interested objects up to date with other object developments.



▲ **Figure 35** Staying up to date

In the real world, if you like to know what is happening in your local area you have several options. You could check social media and hope to follow the correct accounts, you could buy the local newspaper and hope they are covering the events you are interested in, or you could find out from friends. This can become time consuming and costly. Another thing you could do is subscribe to an event website and they will send you an email when exciting things are happening. This is similar to the observer design pattern.

Another example is when you listen to music using a music app. The app could tell you about every artist that is playing a concert near you, but you would get a lot of updates, many of them about artists you are not interested in. The music apps do not do this. They "subscribe" you to artists they think you are interested in and push these notifications out to you. Again, this is similar to the observer pattern.

The observer pattern has an object that is the publisher, sharing details about events, and subscribers—objects that wish to know about the event.

Intent

The observer design pattern allows objects that wish to be informed of updates to be informed of updates without having to check for themselves and without other objects being subjected to unwanted notifications.

Motivation

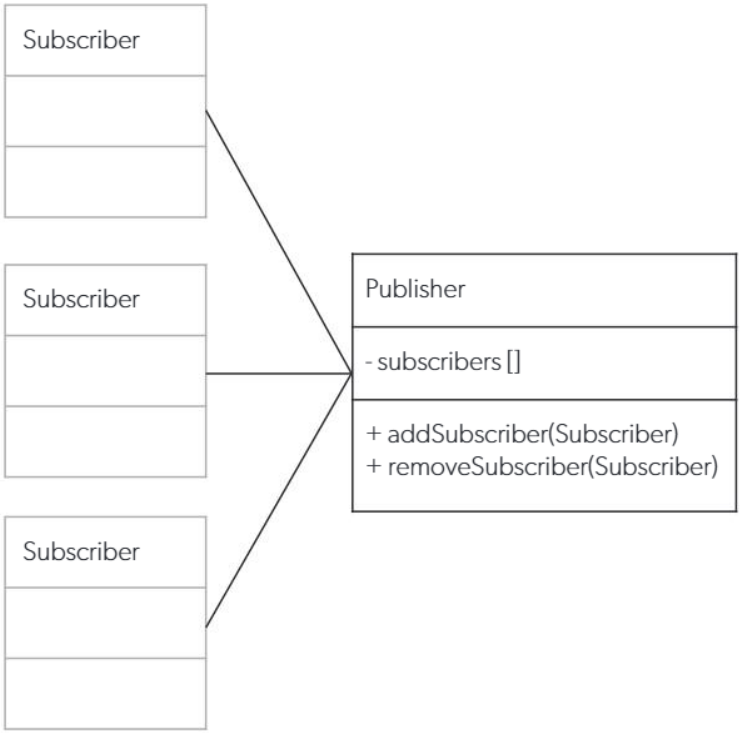
If you get too many notifications about items then it becomes overwhelming and challenging to deal with. You may become disinterested because of the volume of updates, switch off, and miss key information. Conversely, if you have no notifications then you could spend too much time looking for information to make decisions or find something you are looking for. From a commercial point of view, companies want to find the balance between sending too many notifications and not keeping users informed.

Solution

The observer pattern uses the following code features.

- A list, to store a list of subscribers objects.
- Public methods, which allow subscriber objects to add and remove themselves from the list.
- Iteration of the list to notify subscriber objects of updates.

Structure



▲ **Figure 36** Observer pattern

AHL

Observer design in Java

The code below shows a class that uses the observer design pattern in Java. A publisher class is created that allows subscriber objects to subscribe for news. The subscriber objects are added to a list and when a notification is needed an email is sent to the users. Please note the implementation for the EmailSender class has been hidden as it is outside of the scope of this course.

Publisher class in Java

```
1.  public class Publisher {
2.
3.      ArrayList <Subscriber> subscribers = new
        ArrayList <Subscriber>();
4.
5.      public Publisher() {
6.
7.      }
8.
9.      public void subscribe (Subscriber s) {
10.
11.         subscribers.add(s);
12.         System.out.println("You have been
            subscribed");
13.
14.     }
15.     public void unSubscribe(Subscriber s) {
16.
17.         int index = -1;
18.         for (int i = 0; i < subscribers.size();
            i++) {
19.             if (subscribers.get(i).getName().
                equals(s.getName())) {
20.                 index = i;
21.             }
22.         }
23.
24.         if(index > -1) {
25.         subscribers.remove(index);
26.         System.out.println("you have been
            removed");
27.         }
28.     }
29.
30.     public void notifySubscribers(String message) {
31.
```

```
32.         for(int i = 0 ; i < subscribers.size();
            i++) {
33.
34.             String mailer = subscribers.get(i).
                getEmail();
35.             String eMessage = message;
36.             EmailSender email = new
                EmailSender(eMessage, mailer);
37.
38.         }
39.
40.     }
41.
42. }
```

The main method as shown below helps to develop a better understanding of the observer design method. Subscriber classes are created and added to the subscriber list within the instance of the publisher object. If you wish to unsubscribe then you can be removed from the list by passing in the object. If you wish to notify subscribers, you can write a message and send it to the subscribers.

Main method in Java

```
1.  public class MainMethod {
2.
3.      public static void main(String[] args) {
4.
5.          Publisher publish = new Publisher();
6.
7.          Subscriber x = new Subscriber("Fabian
            Schwarz", "F_Scwarz@test.com");
8.          Subscriber y = new Subscriber("Abbie
            Blanc", "A_Blanc@test.com");
9.          Subscriber z = new Subscriber("Lorenzo
            Verde", "L_Verde@test.com");
10.         String message = "Gigs coming up this
            week: karaoke saturday, blues monday";
11.
12.         publish.subscribe(x);
13.         publish.subscribe(y);
14.         publish.subscribe(z);
15.         publish.notifySubscribers(message);
16.         publish.unSubscribe(x);
17.
18.     }
19.
20. }
```

Observer design in Python

The code below shows a class that uses the observer design pattern in Python. A publisher class is created that allows subscriber objects to subscribe for news. The subscriber objects are added to a list and when a notification is needed an email is sent to the users. Please note the implementation for the EmailSender class has been hidden as it is outside of the scope of this course.

Publisher class in Python

```
1.  class Publisher:
2.      def __init__(self):
3.          self.subscribers = []
4.
5.      def subscribe(self, subscriber):
6.          self.subscribers.append(subscriber)
7.          print("You have been subscribed")
8.
9.      def unsubscribe(self, subscriber):
10.
11.         index = -1
12.
13.         for i, s in enumerate(self.subscribers):
14.             if s.get_name() == subscriber.get_
                name():
15.                 index = i
16.                 break
17.
18.         if index != -1:
19.             del self.subscribers[index]
20.             print("You have been removed")
21.         else:
22.             print("Subscriber not found")
23.
24.     def notify_subscribers(self, message):
25.         for subscriber in self.subscribers:
26.             email = subscriber.get_email()
27.
28.             EmailSender.send_email(message, email)
```

The main method as shown below helps to develop a better understanding of the observer design pattern. Subscriber classes are created and added to the subscriber list within the instance of the publisher object. If you wish to unsubscribe then you can be removed from the list by passing in the object. If you wish to notify subscribers, you can write a message and send it to the subscribers.

AHL

Main method in Python

```
1.  def main():
2.      publisher = Publisher()
3.
4.      subscriber_x = Subscriber("Fabian Schwarz",
        "F_Scwarz@test.com")
5.      subscriber_y = Subscriber("Abbie Blanc",
        "A_Blanc@test.com")
6.      subscriber_z = Subscriber("Lorenzo Verde",
        "L_Verde@test.com")
7.
8.      message = "Gigs coming up this week: karaoke
        saturday, blues monday"
9.
10.     publisher.subscribe(subscriber_x)
11.     publisher.subscribe(subscriber_y)
12.     publisher.subscribe(subscriber_z)
13.
14.     publisher.notify_subscribers(message)
15.
16.     publisher.unsubscribe(subscriber_x)
17.
18. if __name__ == "__main__":
19.     main()
```

Advantages of the observer pattern

- You can introduce new subscribers without having to change the publisher code.
- The publisher class has a simple purpose of adding subscribers and notifying subscribers.
- To create a new list you can create a new instance of the subscriber class.

Disadvantages of the observer pattern:

- There is no ordering within the subscribers.
- You cannot have priority subscribers in this situation which can limit some applications.

As you have read through the design patterns, you may recognize that some of the code you have written follows some aspects of the code provided. Most people use elements of design patterns when coding, especially when utilizing encapsulation, inheritance, and abstract classes. Design patterns are formalized versions of logical programming rules divided into useful guides to help you solve problems.

AHL

ATL Thinking skills and social skills

Design patterns feature heavily in most of the software you have used. In this task, work in small groups to create a concept map summarizing the material in this unit and showing how different design patterns relate to real-life situations. You will need sticky notes, large pieces of paper, and coloured pens or pencils.

1. Discuss real-life situations where you have seen a design pattern in action. Agree on two situations you will use.
2. Briefly reflect on how you collaboratively made your choice, and how or if you were able to take everyone's opinions into account.
3. Using one large piece of paper for each situation, complete the following steps.
    - Make a list of the key words in this unit and write them on the sticky notes. These are the nodes of your concept map.
    - Write the title of the design pattern at the centre of the paper.
    - Arrange the nodes (sticky notes) on the paper around the title. Order them from general to more specific terms.
    - Draw lines between the pairs of nodes to represent the connections between them.
    - Write a brief statement along each connection line to describe how the key words are linked.
4. Consider how you will share the work within your group.
5. Complete all the steps to create your concept maps.
6. Share your concept maps with your class. Change or develop your diagrams based on the feedback you receive.
7. Individually, reflect on your role in your group. Identify one IB learner profile attribute that you developed during this task.

This task is inspired by the Harvard Project Zero Visible Thinking Routine, known as Generate-Sort-Connect-Elaborate. Find more information here: pz.harvard.edu > Resources > Thinking Routines Toolbox.

Linking questions

1. In what ways can OOP be applied to database development (A3)?
2. Is OOP necessary for all programming, or only in modelling complex situations (B2)?
3. How can design patterns in OOP facilitate the architecture of scalable and maintainable machine learning models (A4)?
4. How can the principles of encapsulation and information hiding be applied to secure network communication (A3)?

End-of-topic questions

Topic review

1. Using your knowledge from this topic, B3, answer the guiding question as fully as possible:

   Is object-oriented programming (OOP) an appropriate paradigm for solving complex problems? [6 marks]

Exam-style questions

2. Outline the difference between a class and an instantiation of a class. [2 marks]
3. Describe one advantage of using a class to store data. [3 marks]
4. Describe two advantages of object-oriented programming. [4 marks]
5. Identify one scenario when object-oriented programming may be a disadvantage. [2 marks]
6. Construct a UML diagram for the following class.

   Fruit

   `type of fruit //` stores the name of the fruit

   `origin //` stores the country of origin

   `cost //` stores the cost per kg [4 marks]

7. Explain the purpose of the static variable in the following class. [3 marks]

Book
- title: String - author: String - borrowed: Boolean - waitingList: Boolean - booksBorrowed: static int
+ getTitle() + getAuthor() + getBorrowed() + setBorrowed(Boolean b) + getWaitList() + setWaitList(Boolean b) + getNumberOfBooksBorrowed()

8. Identify two features of an encapsulation class. [2 marks]
9. Describe two advantages of using encapsulated code. [4 marks]
10. Evaluate the use of the private and protected modifier when using classes. [4 marks]
11. Define the term dynamic when discussing data structures. [2 marks]

12. Identify two features of inherited code. [2 marks]
13. Sketch a UML diagram to show the relationship between drink, coffee, fizzy drink and water. [3 marks]
14. Outline two benefits of utilizing inheritance. [4 marks]
15. Explain the limitations of using private as a modifier in a superclass. [3 marks]
16. Define, in coding, the term polymorphism. [2 marks]
17. Identify two features of polymorphic code. [4 marks]
18. Describe two differences between static and dynamic polymorphism. [4 marks]
19. Outline two benefits of utilizing polymorphism. [4 marks]
20. Identify two features of abstracted code. [2 marks]
21. Describe how the abstract class and subclasses are interpreted by the compiler. [4 marks]
22. Explain the difference between an aggregated and composition relationship. [4 marks]
23. Outline the purpose of a design pattern. [2 marks]
24. Explain how the singleton design pattern can aid a designer developing a database storing customer information. [5 marks]
25. Explain how the factory design pattern can be used to enhance maintenance and future growth in a program. [5 marks]
26. A developer wishes to create a program to keep people informed of local news events. Explain how the observer design pattern could aid the development. [5 marks]

Example Higher Level Exam Question (Java)

Bike to Work month is a month dedicated to raising awareness for the environment by encouraging people to cycle to work. Prizes are available for the top performing individual (best total number of kilometres) and the top performing team (best average number of kilometres).

1. Each team entering the competition can have up to four team members.

   The attributes for the **`TeamMember`** class are listed below.

   - **`Name`** `//` The name of the participant competing for the team.
   - **`bikeDays`** `//` The number of days the participant cycled to work that month.
   - **`kilometres`** `//` The total number of kilometres the participant travelled that month.

   a. Construct a UML diagram for the **`TeamMember`** class. [4 marks]

   An extract of the **`Team`** class is shown.

```
public class Team {

  private String name;
```

```
  private int totalKilometres;
  private int totalBikeDays;
  private TeamMember [] theTeam;
  private int teamSize;

public Team (String name) {

  this.name = name;
  totalKilometres = 0;
  totalBikeDays = 0;
  theTeam = new TeamMember[4];
}
public String getName() {
  return name;
}
public int getTeamSize() {
  return teamSize;
}
public double getAverage() {
  //code missing
}
public int getTotalKilometres() {
  // code missing
}
public int getTotalBikeDays() {
  // code hidden
}
public TeamMember getTeamMember(int x) {
  return theTeam[x];
}
public void addTeamMember(TeamMember x) {

  if (teamSize < 4) {
    theTeam[teamSize] = x;
    teamSize = teamSize + 1;
  }
  else {
    System.out.println("Sorry the team is full");
  } } }
```

The instance variables in the Team class have been declared as **private**.

b. Outline the implications of using the modifier **private**. [2 marks]

In object-oriented programming there are many types of relationships between classes.

c. Identify the relationship between **Team** and **TeamMember**. [1 mark]

A new team called "Team Maths" would like to enter the competition.

d. Construct the code to initialize this new `Team` object. [3 marks]

2. The competition is encouraging people to "leave the car at home". One of the team prizes will be awarded to the team completing the most kilometres.

a. Construct the `getTotalKilometres()` method which will return the total number of kilometres for the team. [4 marks]

A main method is constructed and an extract shown below.



```
1.  Team [] bikeToWork = new Team [3];
2.
3.  bikeToWork[0] = new Team("Team Design");
4.  bikeToWork[1] = new Team("Team Science");
5.  bikeToWork[2] = new Team("Team Support");
6.
7.  bikeToWork[0].addTeamMember(new
    TeamMember("Catherine"));
8.  bikeToWork[0].addTeamMember(new
    TeamMember("Rosie"));
9.  bikeToWork[0].addTeamMember(new
    TeamMember("Nandor"));
10. bikeToWork[1].addTeamMember(new
    TeamMember("Barbara"));
11. bikeToWork[1].addTeamMember(new TeamMember("Alex"));
12. bikeToWork[2].addTeamMember(new
    TeamMember("William"));
13.
14. TeamMember x = new TeamMember("Scott");
15. bikeToWork[0].addTeamMember(x);
```

b. State the output from the following code.

i. `System.out.println(bikeToWork[1].getName());` [1 mark]

ii. `System.out.println(bikeToWork[0].getTeamMember(3).getName());` [1 mark]

Another team prize is awarded to the team with the best average kilometres over the month.

c. Construct the method `getHighestTeamName(Team [] bikeToWork)` that will return the name of the team with the highest number of kilometres. [4 marks]

A special prize is awarded to the individual on any team with the highest average.

d. Construct the method `getBestIndividual (Team [] bikeToWork)` that will return the name of the individual with the best average kilometres. [5 marks]

AHL

Example Higher Level Exam Question (Python)

Bike to Work month is a month dedicated to raising awareness for the environment by encouraging people to cycle to work. Prizes are available for the top performing individual (best total number of kilometres) and the top performing team (best average number of kilometres).

1. Each team entering the competition can have up to four team members.

   The attributes for the `TeamMember` class are listed below.

   - `Name //` The name of the participant competing for the team.
   - `bikeDays //` The number of days the participant cycled to work that month.
   - `kilometres //` The total number of kilometres the participant travelled that month.

   a. Construct a UML diagram for the `TeamMember` class. [4 marks]

   An extract of the `Team` class is shown.

```
from TeamMember import TeamMember
class Team:
    def __init__(self, name):
        self.name = name
        self.totalKilometres = 0;
        self.totalBikeDays = 0;
        self.theTeam = []
        self.teamSize = 0
    def getName(self):
        return self.name
    def getTeamSize(self):
        return self.teamSize
    def getAverage(self):
        //code missing
    def getTotalKilometres(self):
        //code missing
    def getTotalBikeDays(self):
        //code hidden
    def getTeamMember(self,x):
        return self.theTeam[x]

    def addTeamMember(self, x):
        if self.teamSize < 4:
            self.theTeam.append(x)
            self.teamSize = self.teamSize + 1
        else:
    print ("Sorry the team is full")
```

   The instance variables in the Team class have been declared as `private`.

   b. Outline the implications of using the modifier `private`. [2 marks]

In object-oriented programming there are many types of relationships between classes.

c. Identify the relationship between **`Team`** and **`TeamMember`**. [1 mark]

A new team called "Team Maths" would like to enter the competition.

d. Construct the code to initialize this new **`Team`** object. [3 marks]

2. The competition is encouraging people to "leave the car at home". One of the team prizes will be awarded to the team completing the most kilometres.

a. Construct the **`getTotalKilometres()`** method which will return the total number of kilometres for the team. [4 marks]

A main method is constructed and an extract shown below.

```
from Team import Team
from TeamMember import TeamMember

bikeToWork = []

bikeToWork.append(Team("Team Design"))
bikeToWork.append(Team("Team Science"))
bikeToWork.append(Team("Team Support"))

bikeToWork[0].addTeamMember(TeamMember("Catherine"))
bikeToWork[0].addTeamMember(TeamMember("Rosie"))
bikeToWork[0].addTeamMember(TeamMember("Nandor"))
bikeToWork[1].addTeamMember(TeamMember("Barbara"))
bikeToWork[1].addTeamMember(TeamMember("Alex"))
bikeToWork[2].addTeamMember(TeamMember("William"))

x = TeamMember("Scott")
    bikeToWork[0].addTeamMember(x)
```

b. State the output from the following code.

i. `Print (bikeToWork[1].getName())` [1 mark]

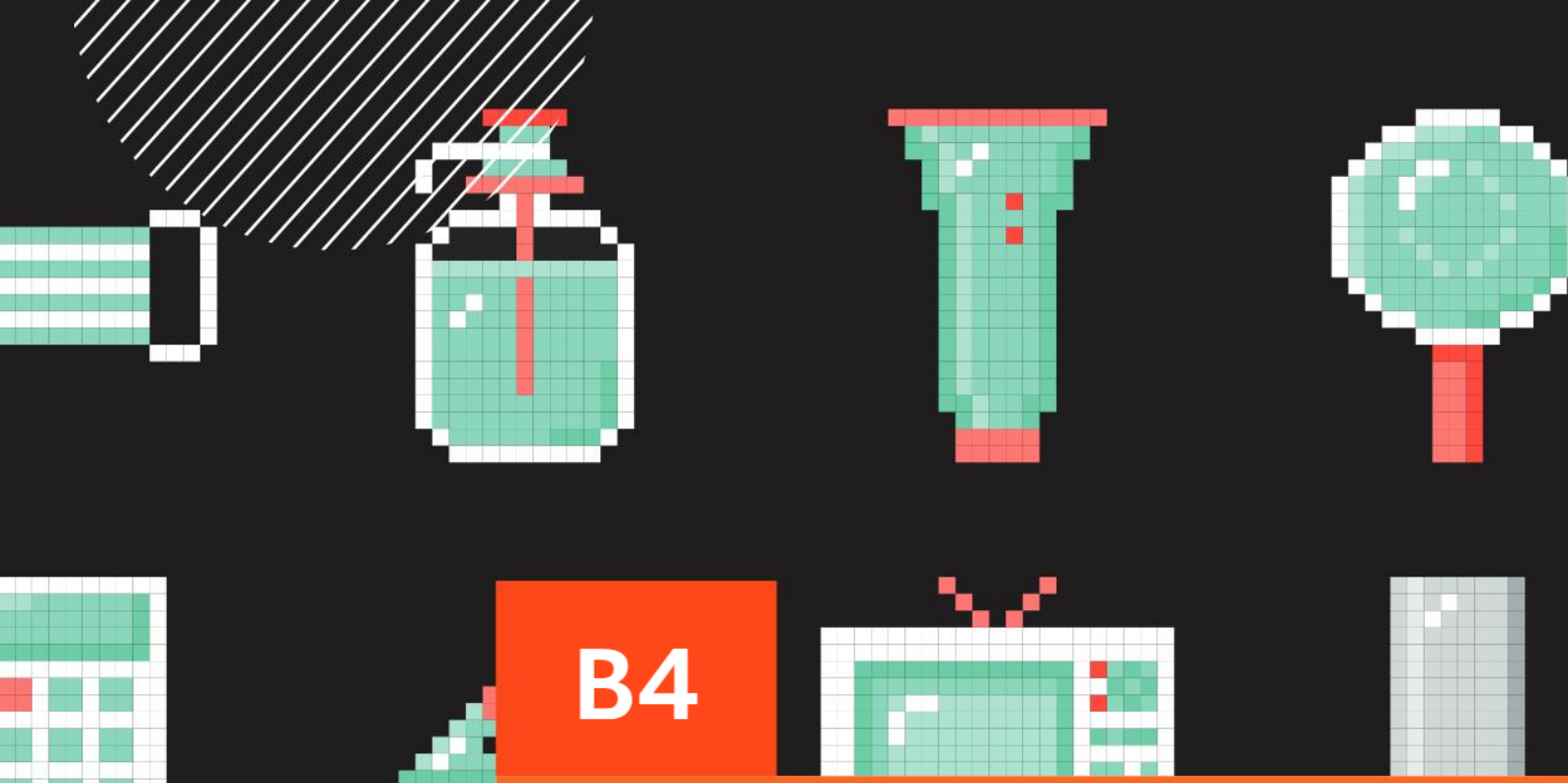
ii. `Print(bikeToWork[0].getTeamMember(3).getName())` [1 mark]

Another team prize is awarded to the team with the best average kilometres over the month.

c. Construct the method **`getHighestTeamName(Team [] bikeToWork)`** that will return the name of the team with the highest number of kilometres. [4 marks]

A special prize is awarded to the individual on any team with the highest average.

d. Construct the method **`getBestIndividual(Team [] bikeToWork)`** that will return the name of the individual with the best average kilometres. [5 marks]



B4

Abstract data types (ADTs)

Which ADTs are most appropriate for different situations?

An abstract data type (ADT) is a concept in computer science that characterizes a data type by its functionality. This approach ignores the specifics of how the data is organized internally and how the operations are executed, focusing instead on the user's perspective of what can be done with the data.

In your favourite video game there might be an inventory where you can store items such as potions, crafting materials and armour. You can do things with this inventory, such as adding new items, removing items when you do not need them anymore, or checking to see if you have a specific item or how many of an item you have.

You, as the player, do not need to know how the game program keeps track of all these items. The items could be stored in a list, an array, or any other way the game developers thought was best. What matters to you is that you can work with your inventory (add, remove, check) without worrying about how it is all managed behind the scenes.

An ADT is like that inventory system. It is a way of organizing data that tells you what you can do with the data (such as adding or removing information) but not how those actions are actually carried out. It is abstract because it does not get into the details of how the data is stored or managed.

So, when computer scientists talk about ADTs, they are talking about setting up rules for what you can do with data without worrying about the specifics of how it is all put together. This helps programmers focus on what they want their program to do (such as manage an inventory) without getting stuck on the details of how to store or organize everything right away.

B4.1 Fundamentals of ADTs

AHL

Syllabus understandings

B4.1.1 Explain the properties and purpose of ADTs in programming

B4.1.2 Evaluate linked lists

B4.1.3 Construct and apply linked lists: singly, doubly and circular

B4.1.4 Explain the structures and properties of binary search trees (BSTs)

B4.1.5 Construct and apply sets as an ADT

B4.1.6 Explain the core principles of ADTs

B4.1.1 Explain the properties and purpose of ADTs in programming

ADTs are important tools in software development, offering a blend of encapsulation, abstraction and modularity that aids in the design, implementation and maintenance of complex software systems. They allow programmers to work at a higher level of abstraction, focusing on what operations are performed rather than on how they are implemented. This improves both the quality and productivity of software development efforts.



▲ **Figure 1** How should a computer efficiently store a player's inventory? Why is it important to consider what operations will be performed on data?

Table 1 Properties of ADTs

Operations and behaviours	An ADT includes a collection of operations that are allowed to be performed on the data. For instance, a **`queue`** ADT would allow operations such as **`enqueue`**, **`dequeue`**, **`isEmpty`** and **`peek`**. ADTs restrict any other operations. For example, you could not sort this data because a queue does not have a sort operation.
Abstraction	ADTs focus on what operations are performed, not how these operations are implemented. This abstraction allows the user to use the data without understanding the underlying complexities. For example, you can use a queue without needing to worry about how it is implemented.
Encapsulation	ADTs encapsulate the data, ensuring that it can only be accessed or manipulated through its defined operations. This prevents external interference and misuse of the underlying data. In the **`queue`** example, you would use the available operations, but no other operations would be available.
Modularity	ADTs contribute to the modularity of code. Since the implementation details are hidden, different implementations of the same ADT can be swapped without affecting the rest of the program. This property is important for developing large and complex software systems, allowing for easier maintenance and updates. For example, an ADT which sorts collections (**`sortCollection`**) might no longer fit in memory. You can change the storage from primary memory to disk storage without changing the operations for the user. Users will still sort collections, with no need to know about the underlying change.
Reusability	The abstract nature of ADTs makes them highly reusable across different programs and projects. Since they define operations at a high level, the same ADT can be applied to various contexts where similar operations are needed, regardless of the specific details of the data. For example, the **`sortCollection`** ADT can be used across different projects.

Table 2 Purposes of ADTs

Simplification of complex systems	By abstracting away the details of data storage and manipulation, ADTs allow developers to focus on higher-level problem-solving and algorithms.
Code reusability and maintenance	The use of ADTs encourages the writing of more generic and reusable code that can be maintained and updated with less effort. This is because changes to the implementation of an ADT do not require changes to the code that uses it, as long as the interface remains the same.
Enhancement of design quality	ADTs play an important role in software design, particularly in object-oriented programming, by helping to define clear interfaces for components of a system. This leads to better-designed systems with clear separation of concerns.
Facilitation of data manipulation	ADTs define a set of operations for data manipulation without exposing the details of how the data is stored or maintained, making it easier to implement complex data structures in a consistent manner.
Improvement of code portability	Since the implementation details are abstracted away, code using ADTs can be more easily ported to different platforms or languages, as the underlying data structures can be reimplemented without changing high-level code.

Topic B3 covers object-oriented programming.

Computer scientists and software engineers generally do not use abstract data structures when working with small amounts of data or making infrequent changes to data. For example, if you have 100 integers in an array, and you are making two or three changes every minute, you do not need to be overly concerned with the data structure you are using. However, if you have 10,000,000 integers in an array, and you are making 1,000 changes a second (or searching through the array 100 times per second), then you should carefully think about which data structures you are using. Another use case for ADTs is in a memory-constrained environment such as an embedded device.

ADTs are defined by the specific set of operations that can be performed on this data. These operations form the ADT's interface, clearly specifying which operations are supported and how they should be executed. This interface abstraction is a key aspect of ADTs, as it hides the details of the implementation from the user, emphasizing the principles of data encapsulation and operation abstraction.

Operations typically associated with ADTs are designed to manage and manipulate the encapsulated data in ways that are consistent with the ADT's intended purpose. The primary categories of operations associated with ADTs are detailed in Table 3.

Table 3 Associated operations of ADTs

Operation	Description	Example
Access	Enable retrieval of data elements from the ADT.	Queue: **peek**
Insertion	Facilitate the addition of new data elements to the ADT.	Queue: **enqueue**
Deletion	Allow for the removal of existing data elements from the ADT.	Queue: **dequeue**

Searching, sorting and iterating are common operations but the **efficiency** of those operations depends on the ADT. For example, a queue is an ADT in which the data elements are kept in order and the only operations on the ADT are the addition of data elements to the rear terminal position, known as **enqueue**, and removal of data elements from the front terminal position, known as **dequeue**. Therefore, there is no need to search, sort or iterate through a queue. However, a binary search tree ADT does require iterating, sorting and searching.

Each ADT is characterized by a specific set of operations, designed to facilitate its interaction with data elements in a manner which aligns with its intended purpose.

This problem in practice

In business computing, queues serve as a useful ADT for modelling customer service systems—such as call centres or ticketing systems—where service requests must be handled in an orderly manner. A queue ADT allows businesses to manage service requests on a first-come, first-served basis, ensuring fairness and efficiency in customer service operations. Implementing a queue helps in optimizing response times and managing workload peaks, thus improving customer satisfaction and operational efficiency.

TOK

Abstract data types hide the complexity of implementation. People using ADTs do not need to understand how ADTs function. If a programmer uses a stack data structure to create an "undo" system, they do not need to understand the underlying allocation of memory, the specific algorithms for pushing or popping elements, or how the stack manages its elements internally. Instead, they can rely on the well-defined interface provided by the stack ADT, which allows them to perform operations like adding or removing elements without worrying about the low-level details. This abstraction enables programmers to focus on higher-level design and logic, improving code modularity and maintainability, and reducing the risk of errors.

To what extent does the abstraction provided by ADTs influence our understanding and manipulation of complex data structures in software development?

Queues and binary search trees are discussed in section B4.1.6.

AHL

There is more information about constant time operations and efficiency in section B2.4.1.

B4.1.2 Evaluate linked lists

A linked list is a data structure consisting of a sequence of elements, each contained in a node. The defining feature of a linked list is that each node contains a pointer to the next node in the sequence, forming a chain-like structure.

About linked lists

There are different types of linked lists, each with specific characteristics. A linked list is a data structure that can be used to implement various ADTs, such as lists, stacks, or queues, depending on how it is used and the operations provided.

Table 4 Advantages of linked lists

Dynamic size	Unlike arrays, linked lists are dynamic in size. This means they can grow or shrink during runtime, which is advantageous for applications where the data size is not known beforehand or can change.
Efficient insertions and deletions	Adding or removing elements in a linked list is efficient because these operations do not require shifting elements, as is the case with arrays. You only need to update the pointers, which is a constant time operation O(1), assuming you have a direct reference to the node you are inserting or deleting.
Memory efficiency	Linked lists allocate memory as needed, which can lead to more efficient memory usage compared with arrays, which must allocate memory upfront. This is especially true for large data sets where not all allocated memory may be used.
Flexibility	Linked lists can easily be modified to become doubly linked lists (where each node points both forwards and backwards) or circular linked lists (where the last node points back to the first), allowing for more complex and flexible data structures.

Table 5 Disadvantages of linked lists

Access time	Accessing an element in a linked list is O(n) because, in the worst case, you may need to traverse the entire list to find the element. This is less efficient than arrays, which offer constant time access O(1) through indexing.
Memory overhead	Each element in a linked list must also store a pointer to the next (and possibly previous) element, which is additional memory overhead (the amount of available memory required to perform the function) compared with arrays.
No random access	Because of their sequential access nature, linked lists are not suitable for applications that require frequent, random access to elements.
Complexity	Operations on linked lists can be more complex to implement and understand, especially for beginners, due to the need to manage pointers correctly and prevent issues like memory leaks or dangling pointers.

When to use linked lists

Linked lists are suited for applications where:

- the size of the data set changes dynamically
- insertions and deletions at arbitrary positions are frequent
- data needs to be added and deleted in a maintained order
- memory utilization efficiency is critical, and the overhead of pointers is acceptable compared with the cost of reallocating and copying large arrays.

Linked lists offer an alternative to arrays for certain types of problems, particularly those involving dynamic data sets and a need for efficient insertions and deletions.

However, their disadvantages, especially concerning access times and memory overhead, make them less suitable for applications requiring fast, random access to elements.

Scheduling algorithms are covered in section A1.3.3.

This problem in practice

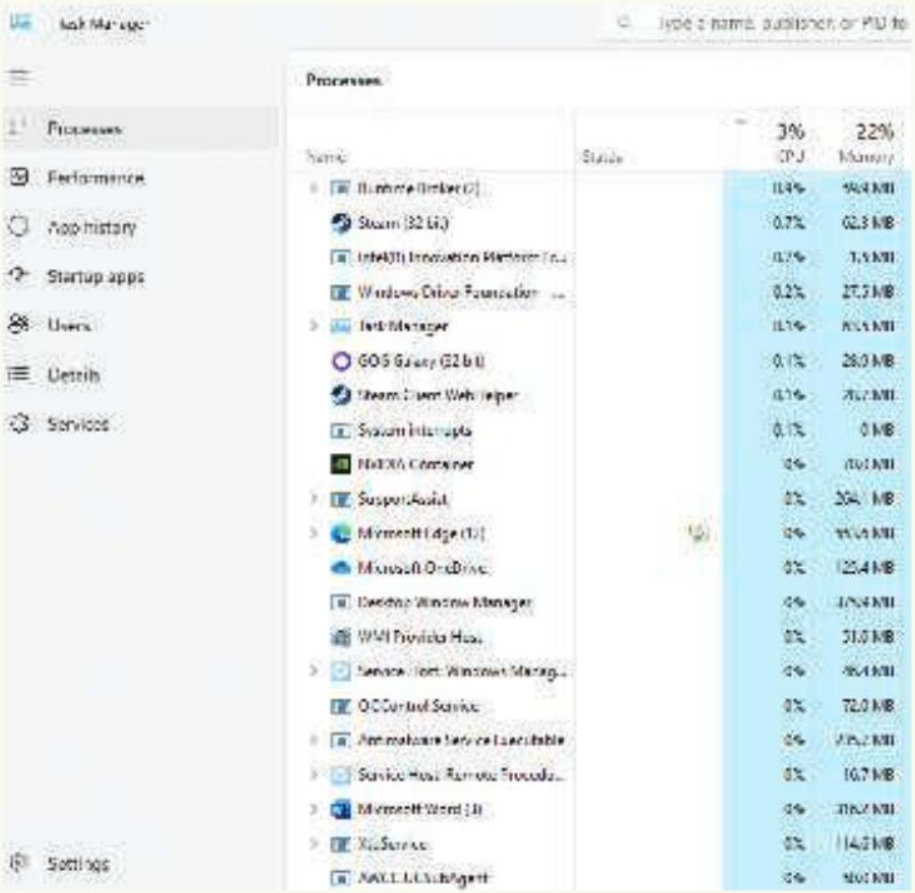
In a task manager or operating system process scheduler, each running application or process can be represented as a node in a singly or doubly linked list, with each node containing information about the process, such as its process ID, state (running, waiting, or suspended), and other relevant metadata. This linked list facilitates the efficient scheduling and management of processes by the operating system.

How it works

Process scheduling: The operating system can traverse the linked list to select the next process to run, based on scheduling algorithms (such as round robin and priority scheduling). The linked list structure allows for easy addition and removal of processes as they start and terminate, respectively.

Dynamic process management: When a new application is launched, the operating system creates a new process node and inserts it into the list at the appropriate position, based on its scheduling priority or other criteria. If a process needs to be terminated, it can be removed from the list, and the pointers of the adjacent nodes are updated accordingly to maintain the list's integrity.

This implementation allows the operating system to efficiently manage multitasking, ensuring that system resources are allocated fairly among all running processes and that user and system tasks are executed smoothly. The use of a linked list for process management exemplifies how dynamic data structures can optimize core functions of complex systems such as operating systems.



▲ **Figure 2** Task Manager in the Windows operating system. With hundreds of changing tasks demanding resources many times a second, a linked list is a good data structure to manage rapidly changing tasks

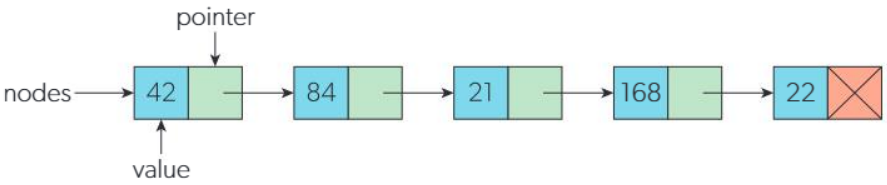
AHL

B4.1.3 Construct and apply linked lists: singly, doubly and circular

Constructing and applying linked lists—including singly, doubly and circular linked lists—is an important skill in computer science.

Singly linked lists

A singly linked list is a collection of nodes where each node contains data and a pointer to the next node in the sequence. This structure allows for efficient insertion and deletion of elements.



▲ **Figure 3** A singly linked list

Characteristics of a singly linked list

Each node in a singly linked list contains data and a reference to the next node. The first node is known as the head. The last node, commonly referred to as the tail node, points to **None** (or **null**), indicating the end of the list.

Operations on a singly linked list

Insertion: Can be O(1) if performed at the beginning or O(n) at the end. If a tail pointer is maintained, this also allows for O(1) insertion at the end.

Deletion: Time complexity varies. It is O(1) for deleting the head node and O(n) for nodes elsewhere, as this requires finding the preceding node.

Traversal: Visiting all nodes in the list has a time complexity of O(n), as it involves sequentially accessing each node from the head to the end of the list.

Search: Searching for a specific node by its value has a worst-case time complexity of O(n), as it may require traversing the list from either end until the desired node is found.

Example of singly linked list in Java

```
1.  public class Node{
2.
3.    // instance variable to store the data in this
      case an integer
4.    // instance variable to store the pointer
5.    private int data;
6.    private Node pointer;
7.
8.
```

Key terms

Insertion The operation of adding a new element to an abstract data type (ADT). The specifics of how insertion is performed depends on the type of ADT.

Deletion The operation of removing an element from an ADT. The process of deletion can vary significantly depending on the type of ADT.

Traversal The operation of visiting each element of an ADT in a specific order. Traversal allows the examination or modification of each element.

Search The operation of finding an element within an ADT that matches a given criterion.

```
9.    public Node() {
10.
11.   //default constructor
12.   }
13.   public Node (int d) {
14.
15.     data = d;
16.     pointer = null;
17.   }
18.
19.   public Node head = null;
20.   public Node tail = null;
21.
22.   public void addNode(int data) {
23.     //Create a new node
24.     Node newNode = new Node(data);
25.
26.     //Checks if the list is empty
27.     if(head == null) {
28.       //If list is empty the new node becomes the
          head and the tail
29.       head = newNode;
30.       tail = newNode;
31.     }
32.     else {
33.       //newNode will be added after the tail
34.       tail.pointer = newNode;
35.       //newNode will become new tail of the list
36.       tail = newNode;
37.     }
38.   }
39.
40.   //display all items in the linked list
41.   public void displayList() {
42.
43.     //Node current node set to the head
44.     Node current = head;
45.
46.     if(head == null) {
```

```
47.       System.out.println("List is empty");
48.       return;
49.     }
50.
51.     System.out.println("Items within the list ");
52.     while(current != null) {
53.       //Prints each node and then gets the next
          node using the pointer
54.       System.out.print(current.data + " ");
55.       current = current.pointer;
56.     }
57.   }
58. }
```

Example of adding nodes and displaying singly linked list in Java

```
1.  public class LinkedListMain {
2.
3.    public static void main(String[] args) {
4.
5.      Node singleLinkedList = new Node();
6.
7.      //add items to list
8.      singleLinkedList.addNode(1);
9.      singleLinkedList.addNode(2);
10.     singleLinkedList.addNode(3);
11.     singleLinkedList.addNode(4);
12.     singleLinkedList.addNode(5);
13.     singleLinkedList.addNode(6);
14.     singleLinkedList.addNode(7);
15.     singleLinkedList.addNode(8);
16.     singleLinkedList.addNode(9);
17.
18.     // display items in the list
19.     singleLinkedList.displayList();
20.
21.   }
22. }
```

Example of singly linked list in Python

```
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None

class SinglyLinkedList:
    def __init__(self):
        self.head = None

    def append(self, data):
        if not self.head:
            self.head = Node(data)
        else:
            current = self.head
            while current.next:
                current = current.next
            current.next = Node(data)

    def display(self):
        elements = []
        current = self.head
        while current:
            elements.append(current.data)
            current = current.next

        return elements

# Create a new instance of SinglyLinkedList
my_linked_list = SinglyLinkedList()

# Append data to the linked list
my_linked_list.append(10)
my_linked_list.append(20)
my_linked_list.append(30)
my_linked_list.append(40)

# Display the contents of the linked list
print(my_linked_list.display())
```

Doubly linked lists

Doubly linked lists extend singly linked lists by allowing each node to have a reference to both the next and the previous node. This bidirectional linkage facilitates more versatile traversal and manipulation.



▲ **Figure 4** A doubly linked list

Characteristics of a doubly linked list

Each node in a doubly linked list contains data, a pointer to the next node (similar to singly linked lists), and an additional reference to the previous node. This dual referencing facilitates backward traversal, enhancing the list's flexibility. The first node is called the head, and the last node is called the tail. Unlike singly linked lists, the tail in a doubly linked list provides a direct reference to traverse the list backward, starting from the last element. The reference to the next node of the tail and the reference to the previous node of the head point to None (or null), indicating the boundaries of the list.

Operations on a doubly linked list

Insertion: Inserting at the beginning (before the head) or at the end (after the tail) can both be done in O(1) time, as direct references to both the head and tail nodes are typically maintained. Inserting a node between two nodes is also O(1), provided you have a reference to the node after or before where the insertion is to take place. However, if you need to find a specific position starting from the head or tail, it could take up to O(n) due to traversal.

Deletion: Deleting the head or tail node can be achieved in O(1) time, thanks to direct access to both ends. Deleting a node from the middle of the list is O(1) if you already have a reference to the node to be deleted. If not, finding the node to delete will require traversal from either the head or the tail, resulting in O(n) time complexity in the worst case.

Traversal: Traversal can be performed in both directions, forward (from head to tail) and backward (from tail to head), each with a time complexity of O(n). This bidirectional traversal is a significant advantage over singly linked lists, offering greater flexibility in navigating the list.

Search: Searching for a specific node by its value has a time complexity of O(n) in the worst case, as it may require traversing the list from either end until the desired node is found.

Doubly linked lists provide enhanced functionality over singly linked lists, particularly with the ease of insertion and deletion operations at both ends of the list and the ability to traverse the list in both directions. These capabilities make doubly linked lists a powerful data structure for applications requiring frequent and flexible modifications to a collection of elements.

Example of doubly linked list in Java

```
1.  public class Node{
2.
3.    // instance variable to store the data in this
      case an integer
4.    // instance variable to store the previous node
      and next node
5.    private int data;
6.    private Node previous;
7.    private Node next;
8.
9.    public Node() {
10.   //default constructor
11.   }
12.   public Node (int d) {
13.     data = d;
14.   }
15.
16.   public Node head = null;
17.   public Node tail = null;
18.
19.   public void addNode(int data) {
20.     //Create a new node
21.     Node newNode = new Node(data);
22.
23.     //Checks if the list is empty
24.     if(head == null) {
25.       //If list is empty the new node becomes the
          head and the tail
26.       head = newNode;
27.       tail = newNode;
28.     }
29.     else {
30.       //new Node will be added after the tail
31.       tail.next = newNode;
32.       //new nodes previous will become the tail
33.       newNode.previous = tail;
```

```
34.       //newNode will become new tail of the list
35.       tail = newNode;
36.       // set the next tail to null
37.       tail.next = null;
38.     }
39.   }
40.
41.   //display all items in the linked list
42.   public void displayList() {
43.
44.     //Node current node set to the head
45.     Node current = head;
46.
47.     if(head == null) {
48.       System.out.println("List is empty");
49.     }
50.
51.     System.out.println("Items within the list ");
52.     while(current != null) {
53.       //Prints each node and then gets the next
          node using the pointer
54.       System.out.print(current.data + " ");
55.       current = current.next;
56.     }
57.   }
58.
59.   public void displayReverseList() {
60.
61.     Node current = tail;
62.     if (tail == null) {
63.       System.out.println("List is empty");
64.     }
65.     System.out.println("Items within the list ");
66.     while(current != null) {
67. //Prints each node and then gets the next node
    using the pointer
68. System.out.print(current.data + " ");
```

```
69. current = current.previous;
70.     }
71.   }
72. }
```

Example of adding nodes and displaying doubly linked list in Java

```
1.  public static void main(String[] args) {
2.    Node doubleLinkedList = new Node();
3.    //add items to list
4.    doubleLinkedList.addNode(1);
5.    doubleLinkedList.addNode(2);
6.    doubleLinkedList.addNode(3);
7.    doubleLinkedList.addNode(4);
8.    doubleLinkedList.addNode(5);
9.    doubleLinkedList.addNode(6);
10.   doubleLinkedList.addNode(7);
11.   doubleLinkedList.addNode(8);
12.   doubleLinkedList.addNode(9);
13.
14.   // display items in the list
15.   doubleLinkedList.displayList();
16.   doubleLinkedList.displayReverseList();
17. }
```

Example of doubly linked list in Python

```
1.  class Node:
2.      def __init__(self, data):
3.          self.data = data
4.          self.next = None
5.          self.prev = None  # Add a reference to the
            previous node
6.
7.  class DoublyLinkedList:
8.      def __init__(self):
9.          self.head = None
10.         self.tail = None  # Keep track of the tail
            to make appending easier
```

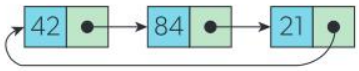
```
11.
12.     def append(self, data):
13.         new_node = Node(data)
14.         if not self.head:  # If the list is empty,
            new node becomes the head and tail
15.             self.head = new_node
16.             self.tail = new_node
17.         else:  # Otherwise, add new node to the
            end and update tail
18.             self.tail.next = new_node
19.             new_node.prev = self.tail
20.             self.tail = new_node
21.
22.     def display(self):
23.         elements = []
24.         current = self.head
25.         while current:
26.             elements.append(current.data)
27.             current = current.next
28.         return elements
29.
30.     def display_reverse(self):  # Additional
        operation to display list in reverse
31.         elements = []
32.         current = self.tail
33.         while current:
34.             elements.append(current.data)
35.             current = current.prev
36.         return elements
37.
38. # Create a new instance of DoublyLinkedList
39. my_doubly_linked_list = DoublyLinkedList()
40.
41. # Append data to the doubly linked list
42. my_doubly_linked_list.append(10)
43. my_doubly_linked_list.append(20)
44. my_doubly_linked_list.append(30)
```

```
my_doubly_linked_list.append(40)

# Display the contents of the doubly linked list
print("Forward:", my_doubly_linked_list.display())
# Display the contents of the doubly linked list
in reverse
print("Reverse:", my_doubly_linked_list.display_
reverse())
```

Circular linked lists

A circular linked list is a variation where the last node points back to the first node, forming a circle. It can be implemented as either a singly linked list or a doubly linked list.



▲ **Figure 5** A circular linked list

Characteristics of a circular linked list

A circular linked list is a variation of the linked list data structure in which the last node's next pointer is not null (or None) but instead points back to the first node (head) of the list. This creates a circular structure that allows for an endless traversal of the list. Circular linked lists can be singly or doubly linked. They are useful for applications requiring cyclic traversals, such as round-robin scheduling.

Operations on a circular linked list

Insertion: The circular nature of the list means the last node's next pointer (in a singly circular list) or both the next and previous pointers (in a doubly circular list) loop back to the head. This circular linkage impacts how insertions are approached. Inserting a new node before the head can still be done in O(1) time. For a singly circular list, the new node becomes the new head, and its next pointer must point to what was the head node. The last node's next pointer (which previously pointed to the old head) must be updated to point to the new head. In a doubly circular list, additional adjustments are made to the previous pointers to maintain the circular linkage. Inserting a node after the tail in a circular linked list is O(1). The new node must be linked such that it follows the original tail and the head follows it, maintaining the circular structure. Inserting between nodes is O(1) provided you have direct references to the nodes between which the new node will be inserted. Without direct references, you must traverse the list from the head to find the correct insertion point, resulting in O(n) time complexity due to traversal.

Deletion: Deleting the head node is an O(1) operation. The head's next pointer (in singly linked lists) or both the next and previous pointers (in doubly linked lists) must be adjusted to the second node, which becomes the new head. Additionally, the tail's pointers must be updated to maintain the circular structure. Deleting the tail node also requires adjusting the circular links, ensuring the new tail correctly points back to the head. This operation is O(1) if you maintain a reference to the node preceding the tail (more straightforward in a doubly linked list). Deleting a node in the middle is O(1) if you have a reference to the node. Otherwise, finding the correct node requires O(n) time due to traversal.

Traversal: The cyclical nature of circular linked lists means traversal can technically continue indefinitely. To traverse the list once, you can start at the head and continue until you circle back to the head. Traversal has a time complexity of O(n).

Search: Searching for a specific node by its value involves traversing the list from the head and comparing each node's value to the target. The time complexity is O(n) in the worst case, as it may require traversing the entire list to find the desired node or to determine that the node is not present.



Example of circular linked list in Java

```
1.  public class Node{
2.
3.    // instance variable to store the data in this
      case an integer
4.    // instance variable to store the previous node
      and next node
5.    private int data;
6.    private Node previous;
7.    private Node next;
8.
9.    public Node() {
10. //default constructor
11.   }
12.   public Node (int d) {
13.     data = d;
14.   }
15.
16.   public Node head = null;
17.   public Node tail = null;
18.
19.   public void addNode(int data) {
20.     //Create a new node
21.     Node newNode = new Node(data);
22.
23.     //Checks if the list is empty
24.     if(head == null) {
25.       //If list is empty the new node becomes the
          head and the tail
26.       head = newNode;
27.       tail = newNode;
```

```
28.       // create reference back to head
29.       newNode.next = head;
30.     }
31.     else {
32.       //new Node will be added after the tail
33.       tail.next = newNode;
34.       //new nodes previous will become the tail
35.       newNode.previous = tail;
36.       //newNode will become new tail of the list
37.       tail = newNode;
38.       // set the next tail back to head to close
          circle
39.       tail.next = head;
40.     }
41.   }
42.
43.   //display all items in the linked list
44.   public void displayList() {
45.
46.     //Node current node set to the head
47.     Node current = head;
48.
49.     if(head == null) {
50.       System.out.println("List is empty");
51.     }
52.
53.     System.out.println(" Items within the list ");
54.     do {
55.       //Prints each node and then gets the next
          node using the pointer
56.       current = current.next;
57.       System.out.print(current.data + " ");
58.     } while(current != head) ;
59.   }
60. }
```

AHL

Example of adding nodes and displaying circular linked list in Java

```
1.  Node circularLinkedList = new Node();
2.  circularLinkedList.addNode(1);
3.  circularLinkedList.addNode(2);
4.  circularLinkedList.addNode(3);
5.  circularLinkedList.addNode(4);
6.  circularLinkedList.addNode(5);
7.  circularLinkedList.addNode(6);
8.  circularLinkedList.addNode(7);
9.  circularLinkedList.addNode(8);
10. circularLinkedList.addNode(9);
11.
12. circularLinkedList.displayList();
```

Example of circular linked list in Python

```
1.  class Node:
2.      def __init__(self, data):
3.          self.data = data
4.          self.next = None  # Only reference to the
            next node
5.
6.  class CircularSinglyLinkedList:
7.      def __init__(self):
8.          self.head = None
9.
10.     def append(self, data):
11.         new_node = Node(data)
12.         if not self.head:  # If the list is empty
13.             self.head = new_node
14.             self.head.next = self.head  # Points
                to itself, making the list circular
15.         else:
16.             current = self.head
```

```
            while current.next != self.head:  # Traverse until you find a node whose next is the head
                current = current.next
            current.next = new_node
            new_node.next = self.head  # New node points back to the head

    def display(self):
        elements = []
        if not self.head:
            return elements  # Return an empty list if the list is empty
        current = self.head
        while True:
            elements.append(current.data)
            current = current.next
            if current == self.head:  # If we've looped back to the head, stop
                break
        return elements

# Create a new instance of CircularSinglyLinkedList
my_circular_singly_linked_list = CircularSinglyLinkedList()

# Append data to the circular singly linked list
my_circular_singly_linked_list.append(10)
my_circular_singly_linked_list.append(20)
my_circular_singly_linked_list.append(30)
my_circular_singly_linked_list.append(40)

# Display the contents of the circular singly linked list
print("Circular Singly Linked List:", my_circular_singly_linked_list.display())
```

TOK

When working with version control systems such as Git, understanding the underlying data structure can provide valuable insights into its efficiency and functionality. Git fundamentally uses a linked list structure to manage the history of commits. Each commit in Git points to the previous commit, forming a chain or linked list of changes. This structure allows for efficient branching, merging and navigating through the commit history.

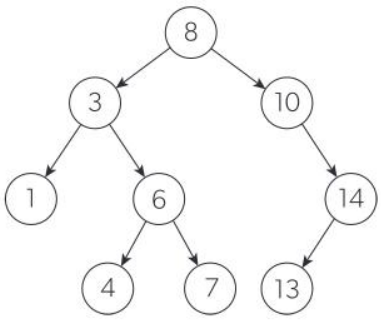
Given this context, we can explore how logic and reasoning are applied in selecting linked lists over other data structures such as arrays. Key factors to consider include dynamic size, memory utilization and access time. Understanding why Git opts for a linked list structure can shed light on the broader decision-making process in choosing appropriate data structures for specific tasks.

AHL

This problem in practice

In computer science, graphs are used extensively as an ADT to model and solve problems related to network routing and connectivity. For example, in internet infrastructure, routers and the connections between them can be represented as vertices and edges of a graph, respectively. This ADT facilitates algorithms for finding the shortest path, optimizing network traffic, and ensuring data packets are efficiently routed from source to destination. Graph ADTs are foundational in understanding and improving the backbone of internet communication.

B4.1.4 Explain the structures and properties of binary search trees (BSTs)



▲ **Figure 6** A binary search tree

A binary search tree (BST) is a node-based binary tree data structure where each node has a unique key and at most two children, with the following properties.

Left child property: The key in any node in the left subtree is less than the key in its parent (and any other node in its right subtree).

Right child property: The key in any node in the right subtree is greater than the key in its parent (and any other node in its left subtree).

BSTs are abstract data structures widely used for organizing and managing **sorted** data efficiently. A BST allows for fast data retrieval, insertion and deletion operations, making it highly suitable for various applications, including database management and memory allocation systems.

Structure of a BST

The structure of a BST has the following properties.

1. Node structure: Each node in a BST contains some key components.

- Key: The value or data that is used to order the nodes within the tree.
- Left child: A pointer or reference to the left child node, which contains a key less than the node's key.
- Right child: A pointer or reference to the right child node, which contains a key greater than the node's key.
- Parent (optional): Some implementations also include a pointer to the parent node for easier traversal and manipulation.

2. **Root node:** The topmost node in a BST, from which every other node is accessible.

3. Leaf nodes: Nodes with no children. These mark the boundaries of the tree.

Properties of a BST

A BST is optimized for binary search operations, making searching, insertion and deletion operations efficient.

Defining properties

Ordered structure: In a BST, for any given node *N*, all elements in *N*'s left subtree are less than *N*, and all elements in *N*'s right subtree are greater than *N*. This ordering holds for every node within the tree, ensuring the entire structure remains sorted.

Dynamic size: BSTs are dynamically sized, allowing for the addition (insertion) and removal (deletion) of nodes. This adaptability makes the BST capable of adjusting to changes in the data set's size over time.

Balanced vs unbalanced trees: For optimal efficiency a BST should be balanced, meaning the tree's depth is minimized, and the O(log n) search time is maintained.

Efficient operations

Search: The BST supports binary search with a time complexity of O(log n) in the best and average cases. However, in the worst case—typically when the tree becomes linearly skewed—the complexity degrades to O(n). Note: A linearly skewed BST occurs when the tree takes the form of a linear chain of nodes, because all elements are inserted in either ascending or descending order. This structure resembles a linked list, resulting in operations degrading to O(n) time complexity, losing the efficiency of a balanced BST.

Insertion: To insert a new node in a BST, the tree is searched to find the correct position for the new node, so that the tree's order—where all nodes to the left of a parent are smaller and all nodes to the right are larger—is maintained.

Deletion: Deleting a node from a BST requires careful adjustments to preserve the tree's structure. This process varies depending on the node's children. Deleting a node with no children is straightforward. Deleting a node with a single child requires linking that child to the rest of the tree, and deleting a node with two children often involves finding a successor, to maintain order.

TOK

In the game "twenty questions", one player thinks of an object, and the other tries to guess what it is by asking yes-or-no questions. The most efficient strategy involves asking questions that systematically narrow down the possibilities. For example, starting with broad categories ("Is it an animal?") and progressively moving to more specific attributes ("Is it a mammal?") mirrors the way humans naturally categorize and retrieve information.

This strategy is similar to the principles and structures of binary search trees. In a BST, each node represents a decision point, where data is categorized based on whether it falls to the left (less than) or right (greater than) of the current node. This hierarchical structure allows for efficient searching, insertion and deletion, reflecting the logical way humans break down information to retrieve it quickly.

How do the principles and structures of BSTs reflect the ways in which humans categorize and retrieve information efficiently?

This problem in practice

Imagine you have a digital music playlist that can automatically sort songs not just by their names but also by other criteria such as genre, release year, or artist popularity. Now, think of organizing these songs using a binary search tree to make finding and playing any song very efficient.

Starting point (root): The root of your BST is a song that splits your playlist into two roughly equal parts based on your sorting criteria, for example the release year. Songs released before this year go to the left, and those after go to the right.

Dividing further: Each "node" or song in this tree does the same thing—it divides the list into older songs to the left, newer to the right. This way, each step halves the number of songs you need to look through.

Finding a song: Want to play a song from 2015? Start at the root and choose left or right based on the year, narrowing down your search quickly. This operation lets you find songs faster than scrolling through a linear list, especially if you have thousands of songs.

AHL

Sketching a BST as a tree diagram

Worked example 1

Constructing a BST with integers

Enter a set of numeric data into a BST and then sketch the resulting tree. Data set: 40, 20, 60, 10, 30, 50, 70

Solution

Step 1: Data is entered sequentially into a BST.

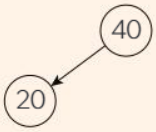
Start with the first number, 40, as the root of your BST.



Step 2: Insert 20 into the BST.

Compare 20 with the root (40).

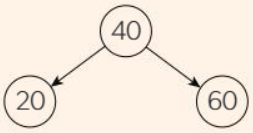
20 is less than 40, so move to the left child of 40. Since there is no left child, insert 20 as the left child of 40.



Step 3: Insert 60 into the BST.

Compare 60 with the root (40).

60 is greater than 40, so move to the right child. Since there is no right child, insert 60 as the right child of 40.

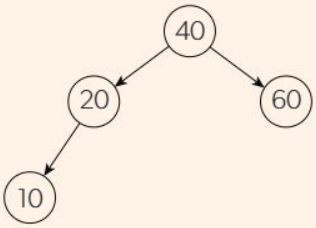


Step 4: Insert 10 into the BST.

Compare 10 with the root (40), move to the left because 10 < 40.

Compare 10 with 20, move to the left because 10 < 20.

Insert 10 as the left child of 20.

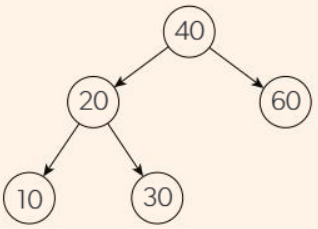


Step 5: Insert 30 into the BST.

Compare 30 with the root (40), move to the left because 30 < 40.

Compare 30 with 20, move to the right because 30 > 20.

Insert 30 as the right child of 20.

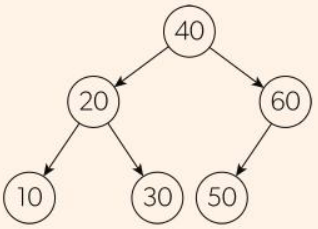


Step 6: Insert 50 into the BST.

Compare 50 with the root (40), move to the right because 50 > 40.

Compare 50 with 60, move to the left because 50 < 60.

Insert 50 as the left child of 60.

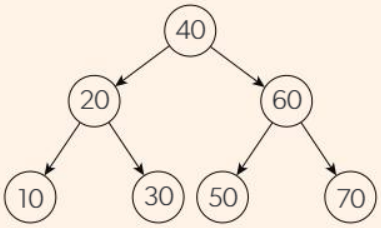


Step 7: Insert 70 into the BST.

Compare 70 with the root (40), move to the right because 70 > 40.

Compare 70 with 60, move to the right because 70 > 60.

Insert 70 as the right child of 60.



Remember, for any given node, all values in the left subtree are less than the node's value, and all values in the right subtree are greater.

Worked example 2

Constructing a BST with names

Enter the list of names in this data set into a BST and sketch the resulting tree.
Data set: Emma, Noah, Olivia, Liam, Ava, Ethan, Sophia

Solution

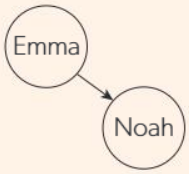
Step 1: Data is entered sequentially into a BST.

Start with the first name, Emma, as the root of your BST.



Step 2: Insert Noah into the BST.

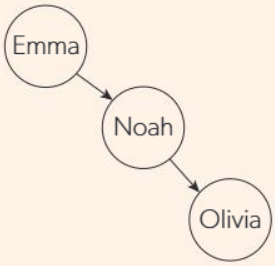
Compare N in Noah to E in Emma. N in Noah comes after E in Emma (it is "greater" in position in the alphabet).
Put N on the right, because N > E.



Step 3: Insert Olivia in the BST.

Compare O in Olivia to E in Emma. O in Olivia comes after E in Emma (is greater in position in the alphabet), O > E. So Olivia moves to the right of Emma.

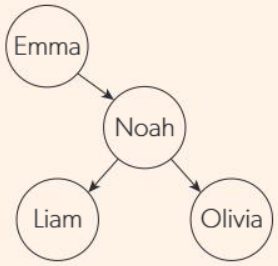
Compare O in Olivia to N in Noah. O in Olivia comes after N in Noah (is in greater position in the alphabet), O > N, so Olivia moves to the right of Noah.



Step 4: Insert Liam into the BST.

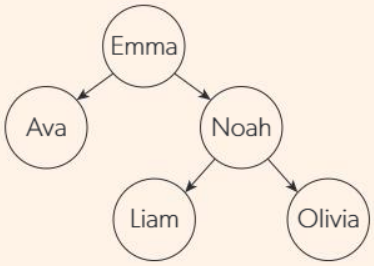
L in Liam comes after E in Emma, L > E, so Liam moves to the right of Emma.

L in Liam comes **before** N in Noah, L < N, so Liam moves to the **left** of Noah.



Step 5: Insert Ava into the BST.

Compare Ava with Emma. A is alphabetically before E, so Ava moves to the left of Emma.

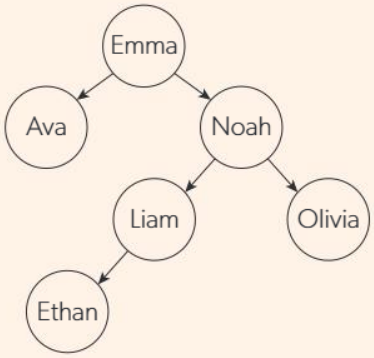


Step 6: Insert Ethan into the BST.

Compare Ethan with Emma. The first letter is the same, so compare the second letters. E**t**han moves to the right of E**m**ma, because T > M.

Compare Ethan with Noah. Move Ethan to the left of Noah because E < N.

Compare Ethan with Liam. Move Ethan to the left of Liam because E < L.



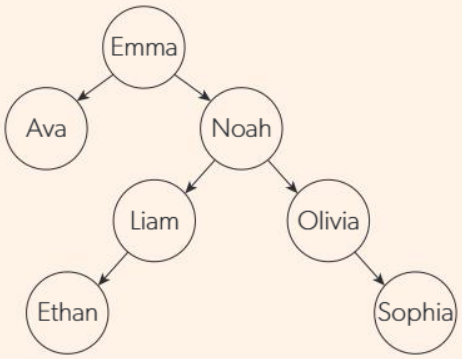
AHL

Step 7: Insert Sophia into the BST.

Compare Sophia to Emma. S > E, so move Sophia to the right of Emma.

Compare Sophia to Noah. S > N, so move Sophia to the right of Noah.

Compare Sohpia to Olivia. S > O, so move Sophia to the right of Olivia.



Activity

Enter each data set into a BST and then sketch the resulting tree.

Data set 1: 35, 26, 28, 92, 41, 53, 99, 10

Data set 2: Carol, Farah, Ben, Claire, Anya, Brian

Data set 3: Managua, Apia, Bishkek, Muscat, Riga, Accra, Paramaribo

Data set 4: 1A91, 152E, 21CF, 10F8, 23BC, 1D1D

Data set 4 contains numerical values written in hexadecimal. What might you need to do to place these hexadecimal numbers into a BST? Discuss with a partner if you wish.

You learned about hexadecimal in section A1.2.1.

This problem in practice

In computational biology, trees—particularly binary trees—are employed as an ADT to model evolutionary trees and to analyse genetic sequences. This approach helps in understanding the evolutionary relationship between different species or strains of viruses. For instance, a binary tree can represent the hierarchical clustering of gene sequences, aiding in the identification of evolutionary ancestors and the prediction of evolutionary paths. This use of trees enables biologists and researchers to infer species evolution and track the mutation of viruses over time.

B4.1.5 Construct and apply sets as an ADT

A set is a collection of **distinct** elements or objects, with no particular order and no duplicate entries. In computer science, sets are used to efficiently store distinct elements and perform operations such as union, intersection, difference and subset checks. In programming, sets are often implemented to quickly test membership or add and remove elements, ensuring that each element appears only once within the collection.

For example, a set could hold the email addresses of all subscribers to a newsletter, ensuring no duplicate addresses are stored.

In Python, a set is commonly referred to as a **set**, while a Java set is commonly referred to as a **HashSet**.

Example of creating a set in Python

```
1.  # In Python, please do not instantiate a set using
    empty curly braces, it will create a dictionary
    instead.
2.
3.  # Using the set() constructor
4.  my_set = set([1, 2, 3, 4, 5])
5.  print(my_set)
6.
7.  # Using curly braces
8.  my_set2 = {1, 2, 3, 4, 5}
9.  print(my_set2)
10.
11. # Creating an empty set
12. empty_set = set()
13. print(type(empty_set))
```

Each language's idioms and practices are different, but the basic operations of a set are similar.

Table 6 Common set operations in Java and Python

Operation	Java	Python
Addition	**add(E e)** Adds the specified element to the set if it is not already present.	**add(element)** Adds an element to the set.
Removal	**remove(Object o)** Removes the specified element from the set if it is present.	**remove(element)** Removes an element from the set. Raises a **KeyError** if the element is not present.
Membership testing	**contains(Object o)** Returns **true** if the set contains the specified element.	**element in set** Evaluates to **true** if the element is in the set.
Set size	**size()** Returns the number of elements in the set.	**len()** Returns the number of elements in the set.

AHL

Example of union, intersection, difference and subset checks in Java

```
1.  // Create a new hash set of integers
2.  HashSet <Integer> myHash = new HashSet <Integer>();
3.  myHash.add(1);
4.  myHash.add(2);
5.  myHash.add(3);
6.  myHash.add(4);
7.  myHash.add(5);
8.
9.  HashSet<Integer> myHash2 = new HashSet<Integer>();
10. myHash2.add(4);
11. myHash2.add(5);
12. myHash2.add(6);
13. myHash2.add(7);
14. myHash2.add(8);
15.
16. // code for union of the hashsets
17. myHash.addAll(myHash2);
18. System.out.println("Union of hash set 1 and 2: " +
    myHash);
19.
20. // code for intersection of hashsets
21. myHash.retainAll(myHash2);
22. System.out.println("Intersection of hash set 1 and
    2: " + myHash);
23.
24. // code to find difference between two hashsets
25. myHash.removeAll(myHash2);
26. System.out.println("Difference of hash set 1 and 2:
    " + myHash);
27.
28. // code to find if myHash2 is a subset of myHash
29. boolean subset = myHash.containsAll(myHash2);
30. System.out.println("myHash2 is a subset of myHash?
    " + subset);
```

Example of union, intersection, difference and subset checks in Python

```
# Example sets
set_a = {1, 2, 3, 4, 5}
set_b = {4, 5, 6, 7, 8}

# Union: Combine the elements of both sets (no duplicates)
union_set = set_a.union(set_b)
print("Union of A and B:", union_set)
# Expected Output: Union of A and B: {1, 2, 3, 4, 5, 6, 7, 8}

# Intersection: Find elements present in both sets
intersection_set = set_a.intersection(set_b)
print("Intersection of A and B:", intersection_set)
# Expected Output: Intersection of A and B: {4, 5}

# Difference: Find elements in set A that are not in set B
difference_set = set_a.difference(set_b)
print("Difference of A and B (elements in A but not in B):", difference_set)
# Expected Output: Difference of A and B (elements in A but not in B): {1, 2, 3}

# Subset: Check if set A is a subset of set B
is_subset = set_a.issubset(set_b)
print("Is A a subset of B?", is_subset)
# Expected Output: Is A a subset of B? False

# Additional example for subset
set_c = {1, 2, 3}
is_subset_c_in_a = set_c.issubset(set_a)
print("Is C a subset of A?", is_subset_c_in_a)
# Expected Output: Is C a subset of A? True
```

AHL

This problem in practice

This example illustrates how sets as an ADT can efficiently manage and manipulate collections of unique items, such as social connections.

In a social media application, each user has a list of friends, which can be represented as a set. This set includes unique identifiers (IDs) for each friend, ensuring no duplicate friendships.

Uniqueness: A user cannot be friends with the same person more than once, mirroring a set's characteristic where each element is unique. When a new friendship is formed, the system checks if the friend's ID already exists in the user's friends set. If not, it is added.

Efficient membership testing: When viewing another user's profile, the system can quickly check if this user is in the current user's set of friends to appropriately display options such as "Send Message" or "Add Friend". This operation is efficient even with a large number of friends, because of the set's fast membership testing.

Adding and removing friends: When a user makes a new friend or unfriends someone, the system performs an addition or removal operation on the set of friends. These operations are straightforward and efficient with sets.

Intersection for mutual friends: To find mutual friends of two users, the system can compute the intersection of their friends' sets. This set operation quickly identifies common elements, highlighting mutual friendships.

Union for suggested friends: To suggest potential new friends, the system might look at the union of the friend sets of a user's friends, excluding those already in the user's set, and potentially filtering further based on other criteria.

B4.1.6 Explain the core principles of ADTs

Hash tables are a fundamental data structure used in computing to store key–value pairs. They offer efficient retrieval, insertion, and deletion operations, typically with an average time complexity of O(1).

The underlying mechanics of hash tables

The efficiency and performance of a hash table depend on three main aspects: the hashing function, collision resolution strategies, and the load factor.

Hashing function

The hashing function is the backbone of a hash table. It takes a key as input and calculates an integer index that determines where the key–value pair should be stored in the table. A good hashing function has a few essential characteristics.

Deterministic: The same key always results in the same index.

Uniform distribution: It spreads keys evenly across the table, minimizing the likelihood of collisions.

Efficient: It computes the hash value quickly to speed up the access time.

The choice of a hashing function can significantly impact the performance of a hash table. It should minimize collisions and evenly distribute keys to utilize the table's storage space effectively.

Collision resolution strategies

A **collision** occurs when two keys hash to the same index. Since each slot in a hash table can hold only one entry, the table must have a strategy for resolving these collisions. Common collision resolution strategies include the following.

Separate chaining: This method stores multiple elements at the same index using a more complex data structure, such as a linked list or a binary search tree. Each cell of the hash table becomes a bucket that holds multiple entries, which are searched sequentially or via the secondary structure's search algorithm to find a specific key.

Open addressing: In open addressing, all elements are stored within the hash table itself, and a collision is resolved by finding another slot for one of the colliding items. Techniques for open addressing include the following.

- Linear probing: Sequentially searches for the next available slot.
- Quadratic probing: Searches for the next slot by adding a quadratic function of the number of attempts to the original hash.
- Double hashing: Uses a second hashing function to determine the interval between probes.

Each collision resolution method has its trade-offs regarding performance, memory usage and implementation complexity.

Load factor

The **load factor** of a hash table is a measure that indicates how full the table is. Represented by the character *lambda*, λ, it is defined as the ratio of the number of stored elements to the total number of slots (buckets) available:

$$\lambda = \frac{\text{number of entries}}{\text{total number of buckets}}$$

The load factor is a critical factor affecting the performance of a hash table. A low load factor means that the table is underutilized, resulting in wasted memory. Conversely, a high load factor increases the likelihood of collisions, which can degrade the average access time to O(n) in the worst case (especially with separate chaining and poorly distributed keys).

To maintain efficient operation, hash tables often resize dynamically. When the load factor crosses a certain threshold (e.g., 0.7), the hash table's capacity is increased, and all existing entries are rehashed according to the new size. This process helps keep the load factor in check but involves additional computational overhead.

Understanding these underlying mechanics is essential for leveraging hash tables effectively in software development. By carefully designing the hashing function, choosing an appropriate collision resolution strategy, and managing the load factor, developers can ensure that their hash tables perform well across a wide range of scenarios.

The underlying mechanics of sets

To understand the underlying mechanics of sets, particularly in how they store and manage data, you need to examine their implementation, operations and typical use cases.

The implementation of sets can vary, but one common underlying mechanism is the use of hash tables. This choice leverages the efficiency of hash tables for quick lookups, insertions and deletions, typically offering average time complexity of O(1) for these operations. In this implementation, the elements themselves are treated as keys with no associated values, or alternatively, the values are simply ignored.

Sets support several key operations, each of which is designed to manage or query the collection of unique elements efficiently. These operations include the following.

Insertion: Adds a new element to the set if it is not already present. The operation first checks if the element exists in the set, which can be done efficiently if the set is implemented using a hash table. If the element is not found, it is added to the collection.

Deletion: Removes an element from the set if it exists. This involves searching for the element and, if found, removing it from the underlying data structure.

Membership testing: Checks whether a specific element exists in the set. This is a fundamental operation for sets and is typically very efficient, especially with an implementation based on hash tables.

Intersection, union, difference: Allow for combining or comparing sets in various ways. The intersection of two sets contains only the elements that are present in both sets, the union contains all elements that are in either set, and the difference contains elements that are in one set but not the other. These operations can be implemented efficiently, especially when the underlying data structure supports quick membership testing.

HashMap in Java and dict in Python



Java implementation: HashMap

Java's `HashMap` class from the `java.util` package offers hash functionality.

`set(key, value)`
In Java, the `HashMap` uses the `put` method to add or update a key–value pair, for example, `hashMap.put(key, value)`.

`get(key)`
The `get` method functions similarly to Python's, returning the value associated with the specified key, or `null` if the map contains no mapping for the key.

`delete(key)`
Java uses the `remove` method to delete a key–value pair, for example, `hashMap.remove(key)`. This method returns the value previously associated with the key, or `null` if the map contained no mapping for the key.

`keys()`
The closest equivalent in Java is `hashMap.keySet()`, which returns a `Set` view of the keys contained in the map.

`values()`
This is similar to the Python method and is achieved with `hashMap.values()`, which returns a `Collection` view of the values contained in the map.

`items()`
In Java, this is accomplished with `hashMap.entrySet()`, returning a `Set` view of the mappings contained in this map. Each element in this set is a key–value pair represented by `Map.Entry`.

`clear()`
The method works the same as in Python, removing all of the mappings from this map.



Python Implementation: dict

In Python, the dictionary (`dict`) is a built-in data structure that implements hash functionality.

`set(key, value)`
In Python, setting a key–value pair is achieved through assignment, for example, `dict[key] = value`. If the key exists, its value is updated; otherwise, a new key–value pair is added.

`get(key)`
This operation is accurately described. Python dictionaries also offer a `get` method that returns the value for a given key. If the key is not found, `None` is returned, or a specified default value if provided, for example, `dict.get(key, default)`.

`delete(key)`
The correct operation in Python to remove a key–value pair is `del dict[key]`. Alternatively, the `pop` method can be used, for example, `dict.pop(key)`, which also returns the value of the deleted key.

`keys(), values(), items()`
These methods return view objects in Python that reflect the current state of the dictionary, meaning if the dictionary changes, the view reflects these changes. They can be iterated over to retrieve keys, values, or key–value pairs, respectively.

`clear()`
This method works as described, removing all items from the dictionary.

TOK

Imagine a system that automates loan approvals or hiring decisions. Such a system needs to process large amounts of data quickly and accurately. Data structures such as BSTs are often employed in these scenarios to ensure that information can be organized and retrieved efficiently. However, the way data is organized in a BST—based on comparative values—can also introduce or amplify biases if the underlying data is not balanced or representative of the population.

This raises important ethical considerations. While BSTs offer significant benefits in terms of efficiency, they can also inadvertently reinforce biases. For instance, if sensitive information is categorized in a way that leads to disproportionate outcomes for certain groups, the system's efficiency comes at the cost of fairness.

Reflecting on these ethical implications is crucial when designing systems that manage sensitive or critical information, ensuring that the pursuit of efficiency does not overshadow the need for equity and justice in data organization.

AHL

This problem in practice

Software engineering frequently utilizes hash tables as an ADT for efficient data retrieval, especially in database indexing and caching mechanisms. A hash table ADT provides a way to store key–value pairs, allowing for rapid data access through unique keys. This is critical in databases where quick lookup, insertion and deletion of records are essential for performance. By using hash tables for indexing, databases can achieve constant time complexity O(1) for these operations under ideal conditions, significantly enhancing the speed and scalability of data-driven applications.

Linking questions

1. What role do stacks and queues play in handling CPU interrupts and polling (A1)?
2. Can ADTs be used to manage data (A2)?
3. How can ADTs be used to optimize file-processing operations like read and write (B2)?
4. Can a BST play a role in the quicksort algorithm (B1)?

End-of-topic questions

Topic review

AHL

1. Using your knowledge from this topic, B4, answer the guiding question as fully as possible:

   Which ADTs are most appropriate for different situations? [6 marks]

Exam-style questions

2. Define an abstract data type (ADT). [2 marks]
3. Explain the purpose of abstract data types (ADTs) ADTs in programming. [4 marks]
4. Describe the importance of encapsulation in ADTs. [3 marks]
5. Distinguish between a queue and a stack in terms of their operations. [4 marks]
6. Explain how a linked list differs from an array. [3 marks]
7. Evaluate the use of linked lists in programming. [4 marks]

AHL

8. a. Construct and apply a singly linked list to store a series of integers. [3 marks]
   b. Construct a method to add a new integer to the list. [2 marks]
9. Explain the structures and properties of binary search trees (BSTs). [4 marks]
10. Evaluate the advantages and disadvantages of using a BST over a linked list. [6 marks]
11. Sketch a BST after inserting the following elements: 50, 30, 70, 20, 40, 60, 80. [4 marks]
12. Discuss the use of ADTs for managing large data sets. [4 marks]

Internal assessment (IA): The computational solution

The internal assessment (IA) is an applied computational thinking process, giving you the opportunity to apply the skills and tools you have learned during the DP computer science course. You will spend about 35 hours in class (and about the same amount of time outside of class) creating a computational solution and detailing your process as well as the solution itself. Pay special attention to the application of computational thinking, which should be evident in your report.

The maximum word count for the report is 2,000 words, but this does not include charts, diagrams, code samples, tables, references, bibliography or headers.

The IA is assessed against five criteria. For standard-level students, the IA is weighted as 30% of your final mark. For higher-level students, it is 20%. The criteria are the same for both standard level and higher level.

	Criterion	Number of marks
A	Problem specification	4
B	Planning	4
C	System overview	6
D	Development	12
E	Evaluation	4
	Total	*30*

▲ **Table 1** Assessment criteria for the IA

Criteria A, B, C and E are process-oriented. For example, describe **how** you arrived at your solution. Criterion D assesses your understanding of the concepts involved in the development of the IA. For example, showcase your programming skills and explain your programming decisions.

Choosing a suitable problem

You have a free choice of topic, but the choice you make could affect your marks.

> In identifying a problem, the student can select to apply to the problem **any topic in computer science** that interests them. **It does not have to be directly related to the specified themes in the syllabus.**
>
> The problem chosen should require a software solution with sufficient complexity to be commensurate with the level of the DP computer science course. It should also require sufficient innovation for the student to demonstrate their organisational skills, algorithmic thinking and ability to code their algorithms.
>
> *Source: Computer science guide, IB. Emphasis is the author's.*

What does "sufficient complexity to be commensurate with the level of the DP computer science course" mean? The best method to select a topic of sufficient complexity for your IA is to become curious (an inquirer) about a topic which **really interests you** within computer science and then explore it **deeply**. It is important to not focus solely on what a user does but rather how the underlying system works.

Possible IA topic	Description
Data visualization with spreadsheets	Collecting a small data set (for example, survey results, school sports statistics) and using Python or Java to create charts and graphs.
Small business inventory system	Creating a CRUD system for a small business to manage inventory (or customers, orders, sales, and so on).
Automated data gathering via API	Gathering data from an API (perhaps stock prices, weather, country information or NASA) and displaying results visually and programmatically.
AI-powered chatbot	Exploring natural language processing and creating a chatbot that can interact with users in a meaningful way.
Cryptography and security	Investigating different encryption algorithms and developing a secure communication application.
Simple game development with procedural generation	Creating a simple game that uses procedural generation for levels or content, exploring algorithms like Perlin noise or cellular automata.
Big data analytics	Analysing large data sets to extract meaningful insights (in biology or business).
Sentiment analysis on social media	Building a tool to analyse the sentiment of social media posts, exploring machine learning models and text analysis.
Internet of Things (IoT) for smart agriculture	Developing an IoT-based system to monitor soil moisture, temperature and other parameters to optimize farming practices.
Digital twins for industrial applications	Exploring the concept of digital twins and creating a digital replica of a process to optimize performance and predict maintenance needs.
Adaptive learning platforms	Developing an educational platform that adapts to the learning style and pace of individual students, using machine learning to personalize content.
Virtual assistants for accessibility	Building a virtual assistant designed to aid people with disabilities, exploring how a computer program can enhance accessibility and independence.
Speech recognition and voice control	Building a system that uses speech recognition to control devices or applications, exploring the challenges of accurate voice processing.
Network packet sniffer	Constructing an application to view incoming and outgoing network packets on a device using libraries such as libcap.
Building a second brain	Building a personal organization system which tracks everything from homework, contacts and appointments to diet and sleep. This will primarily focus on database relations.
Medical diagnosis system	Building a system which uses machine learning to diagnose medical conditions. (There is more to this than simply a collection of "if … then" statements!)
Resource allocation algorithms	Building a system which efficiently uses resources (perhaps a lunch serving area, a limited availability sports court, or a time to meet with a teacher).
Colour analyser	Building a tool which uses a camera to analyse nuanced colours and suggest or overlay complementary colours.

▲ **Table 2** Example IA topics

Problem specification (Criterion A)

The problem specification is the foundation for the development of the solution.

Term	Definition
Problem scenario	The problem scenario is a clear description of the problem, including its measurable solution requirements. The description should relate directly to the problem, whether this is in the world around us, in another field of knowledge, or is a current issue in computing.
Success criteria	These are measurable outcomes derived from the solution requirements that indicate the successful development of the product.
Computation contexts	The computational context is the specific area of computing that you select to use in the solution.

▲ **Table 3** Key terms for problem specification

Case study: Problem specification

A student greatly enjoys playing the video game Minecraft. They are especially curious about how the terrain is procedurally generated. Every time the student starts a new game the terrain is different, making for a different challenge each time the game is played.

The student spends some time talking with their teacher and researching procedural generation. They stumble upon Perlin noise, which is a mathematical technique for creating pseudo-random values used to create textures. The student decides to write an IA about procedural generation using a custom Perlin noise algorithm.

Problem statement

My IA project will involve creating a custom Perlin noise algorithm to procedurally generate terrain similar to that in Minecraft. My project will focus on understanding and implementing Perlin noise to produce varied and realistic terrain features, such as hills, valleys and plains.

Measurable solution requirements

I will implement a custom Perlin noise algorithm in Python, ensuring it can generate a 2D terrain map. The algorithm should be capable of producing distinct terrain features, including hills, valleys and plains, with smooth transitions between them. The solution must accept a random seed input, ensuring that different terrain maps are generated with each run while maintaining reproducibility for the same seed. The generated terrain must be visualized using a suitable graphical representation, such as a heatmap or a 3D plot, to effectively demonstrate the algorithm's output. The algorithm should generate the terrain map within a reasonable time frame (e.g., less than 5 seconds for a 256 × 256 grid).

Case study: Problem specification

Success criteria

- **Accuracy:** *The custom Perlin noise algorithm must correctly generate smooth and continuous pseudo-random terrain that reflects realistic geological features.*
- **Variety:** *Each generated terrain map should be distinct and varied, showcasing the ability of Perlin noise to create diverse landscapes.*
- **Efficiency:** *The algorithm should run efficiently, generating the terrain map within the specified time frame.*
- **Reproducibility:** *The terrain generation process should be reproducible with the same seed input, ensuring consistent results for given seeds.*
- **Visualization quality:** *The visual representation of the terrain must be clear and accurately reflect the underlying terrain data.*

Computational context

I am exploring the area of procedural generation, focusing on terrain generation using Perlin noise. Procedural generation is a method in computer graphics for creating data algorithmically rather than manually, allowing for vast and diverse worlds to be generated efficiently. Perlin noise is a gradient noise function, widely used in procedural texture generation due to its ability to produce natural-looking textures. By understanding and implementing this technique, I aim to gain insights into the broader field of procedural content creation, which has applications in gaming, simulations, and more.

Hints and tips: Problem specification

Carefully read the rubric for section A. Compare the 1–2 mark band with the 3–4 mark band. Your goal should be to exceed every descriptor in the highest band.

Marks	Level descriptor
0	The response does not reach a standard described by the descriptors below.
1–2	The response: • **outlines** a problem scenario • **states** limited success criteria • **outlines** the nature of the solution in a computational context.
3–4	The response: • **describes** the problem scenario in terms of its measurable solution requirements • **states** appropriate success criteria • **explains** the choice of computational context for the solution.

▲ **Table 4** Marks and level descriptors for internal assessment section A

Ensure you confer with your computer science teacher. A key idea here is choosing any topic within computer science and applying computational thinking to explore it deeply. Students occasionally forget to pay close attention to the command terms, which are important guidelines to tell you how deeply you need to explain an idea.

Planning (Criterion B)

The planning of the product must be consistent with the problem specification in criterion A.

- This criterion assesses how the problem scenario has been decomposed into component parts.
- The plan should address the requirements of the solution, in terms of the success criteria, and include a proposed chronology for the steps involved in planning, designing, developing, testing and evaluating the solution.
- A plan can be presented in different forms, but diagrams such as Gantt and Agile charts can effectively support the planning process.
- The plan may include any relevant research, such as the use of existing code libraries.

Term	Definition
Decomposition	Breaking down the problem scenario (identified in criterion A) into smaller, more manageable sub-problems or components. The decomposition can be effectively constructed using diagrams.
Reasonable decomposition	Break the problem down into essential components that support the construction of a plan.

▲ **Table 5** Key terms for planning

Case study: Planning

1. *Understanding Perlin noise and procedural generation*
   - *Research Perlin noise and its mathematical foundations.*
   - *Study examples of procedural terrain generation.*
   - *Explore the relationship between Perlin noise and realistic terrain features.*
2. *Algorithm design*
   - *Design a custom Perlin noise algorithm to generate a 2D array of noise values.*
   - *Implement smooth transitions between different terrain features (hills, valleys, plains).*
   - *Ensure the algorithm accepts a random seed for reproducibility.*
3. *Implementation in Python*
   - *Write the custom Perlin noise algorithm in Python.*
   - *Test the algorithm to ensure it generates smooth and continuous pseudo-random terrain.*

Case study: Planning

4. *Visualization*
    - *Choose appropriate visualisation techniques (e.g., heatmap, 3D plot).*
    - *Implement visualization to display the generated terrain map.*
5. *Performance optimization*
    - *Optimize the algorithm to generate terrain within a reasonable time frame (less than 5 seconds for a 256 × 256 grid).*
6. *Validation and testing*
    - *Validate the accuracy and variety of the generated terrain.*
    - *Test reproducibility by generating terrain with the same seed multiple times.*
    - *Evaluate the visual representation to ensure clarity and accuracy.*

Plan addressing success criteria

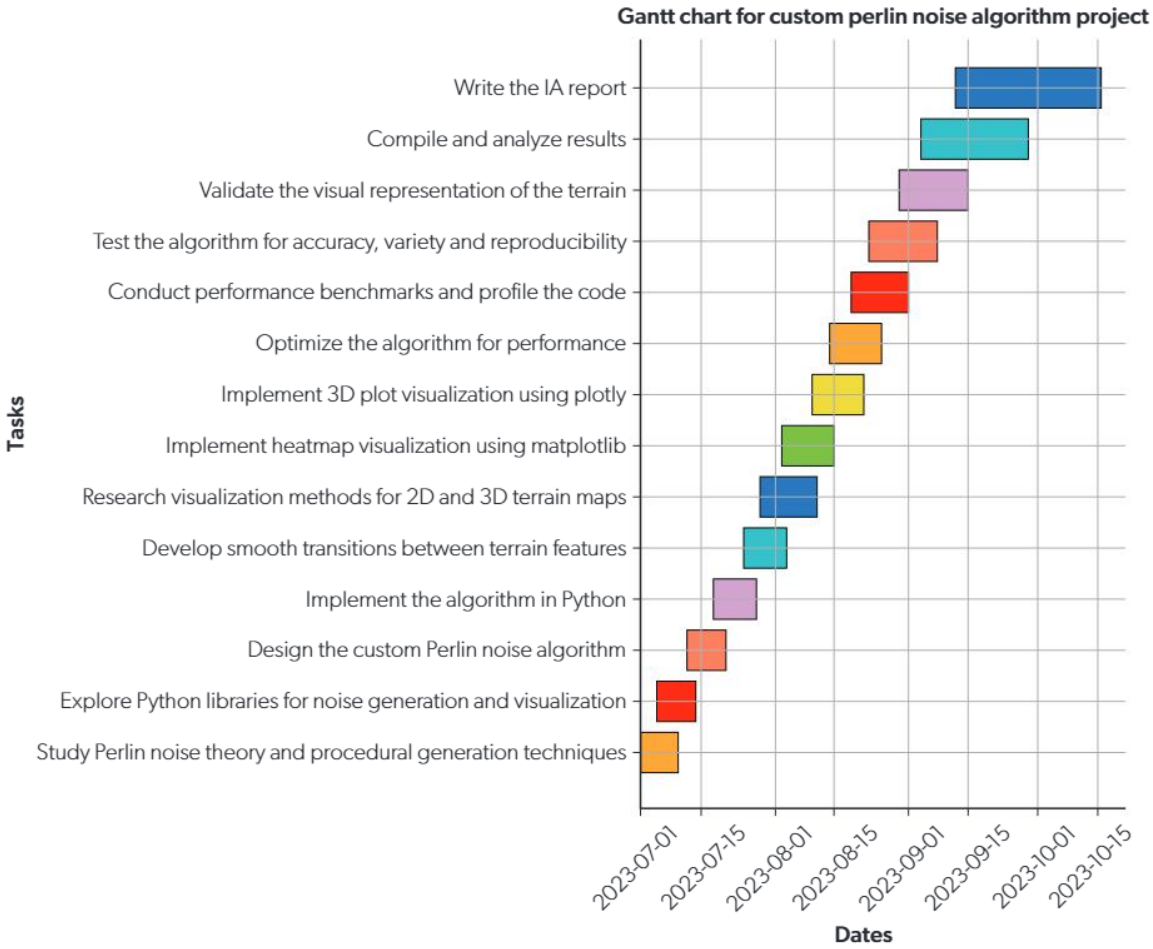
1. *Accuracy*
    - *Develop and test the Perlin noise function to generate smooth and continuous pseudo-random values.*
    - *Implement gradient and interpolation techniques to produce realistic terrain features.*
    - *Validate generated terrain against known geological features for realism.*
2. *Variety*
    - *Ensure the Perlin noise algorithm can generate diverse terrain types.*
    - *Introduce parameters to control terrain features (e.g., frequency, amplitude) to enhance variety.*
    - *Test multiple seeds to verify distinct terrain maps.*
3. *Efficiency*
    - *Optimize the algorithm for performance (for example, using efficient data structures and minimizing computational overhead).*
    - *Benchmark the algorithm's execution time, aiming for less than 5 seconds for a 256 × 256 grid.*
    - *Profile the code to identify and address performance bottlenecks.*
4. *Reproducibility*
    - *Implement seed-based randomness to ensure reproducibility.*
    - *Test the algorithm with fixed seeds to confirm consistent results.*
    - *Document the seed usage and its impact on terrain generation.*

Case study: Planning

5. *Visualization quality*
    - *Select and implement effective visualization techniques (e.g., matplotlib for heatmaps, plotly for 3D plots).*
    - *Ensure the visual representation accurately reflects terrain data (e.g., colour gradients for elevation).*
    - *Validate the visualization with sample data to confirm clarity and accuracy.*

Detailed plan

1. *Research and conceptualization*
    - *Week 1: Study Perlin noise theory and procedural generation techniques.*
    - *Week 2: Explore Python libraries for noise generation and visualization (e.g., numpy, matplotlib, plotly).*
2. *Algorithm design and implementation*
    - *Week 3: Design the custom Perlin noise algorithm.*
    - *Week 4: Implement the algorithm in Python, ensuring it generates a 2D noise map.*
    - *Week 5: Develop smooth transitions between terrain features using gradient and interpolation techniques.*
3. *Visualisation development*
    - *Week 6: Research visualization methods for 2D and 3D terrain maps.*
    - *Week 7: Implement heatmap visualization using matplotlib.*
    - *Week 8: Implement 3D plot visualization using plotly.*
4. *Optimization and testing*
    - *Week 9: Optimize the algorithm for performance, targeting less than 5 seconds for a 256 × 256 grid.*
    - *Week 10: Conduct performance benchmarks and profile the code.*
    - *Week 11: Test the algorithm for accuracy, variety and reproducibility with different seeds.*
5. *Final validation and reporting*
    - *Week 12: Validate the visual representation of the terrain.*
    - *Week 13: Compile and analyse results, ensuring all success criteria are met.*
    - *Week 14: Write the IA report, documenting the design, implementation, testing and findings.*



▲ **Figure 1** Gantt chart for custom Perlin noise algorithm project

Hints and tips: Planning

Here is the rubric for section B. As with section A, you should compare mark bands so you understand what is required.

Marks	Level descriptor
0	The response does not reach a standard described by the descriptors below.
1–2	The response: • **constructs** a partial decomposition of the problem scenario • **constructs** a plan that addresses some of the success criteria of the solution.
3–4	The response: • **constructs** a reasonable decomposition of the problem scenario • **constructs** a plan that addresses the success criteria of the solution.

▲ **Table 6** Marks and level descriptors for internal assessment section B

The success criteria you developed in section A are now driving your IA. Make sure that **each** of the success criteria is reasonably decomposed and included in your plan. Your plan will help you manage your time and successfully complete your project.

System overview (Criterion C)

The system overview of the product must be consistent with the problem specification in criterion A and the planning in criterion B.

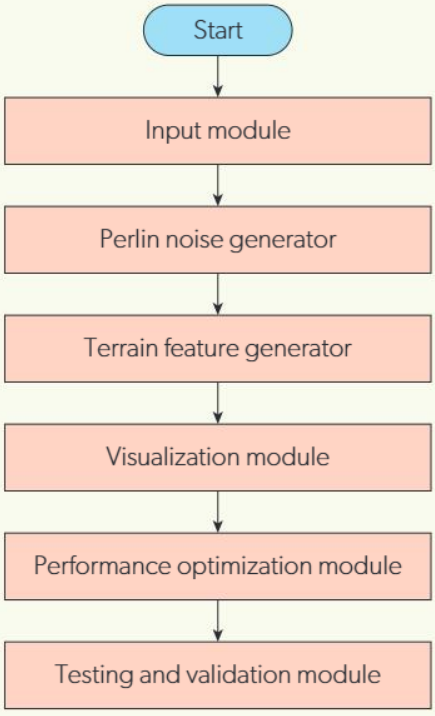
- The system overview should include a system model with the key components, their relationships, the rules governing their interaction, and the algorithms required by these components and the user interface.
- The system overview should be clear enough to enable a third party to recreate the product.
- The system model will provide the information for a viable testing strategy.

The recommended word count for this criterion is 300 words.

Term	Definition
System model	The model consists of diagrams that show the components of the system and how they are connected. The system model will include the design of the user interface. A complete system model does not include the algorithms for each of the components.
Algorithms	These can be presented in different forms, including natural language, flowcharts or pseudocode. They should address the individual components of the system model.
Testing strategy	This is a systematic approach for evaluating whether the computational solution works as intended. The testing strategy should ensure that code functions correctly and handles unexpected or incorrect inputs. This can be represented effectively in a table with proposed test data and expected outcomes.

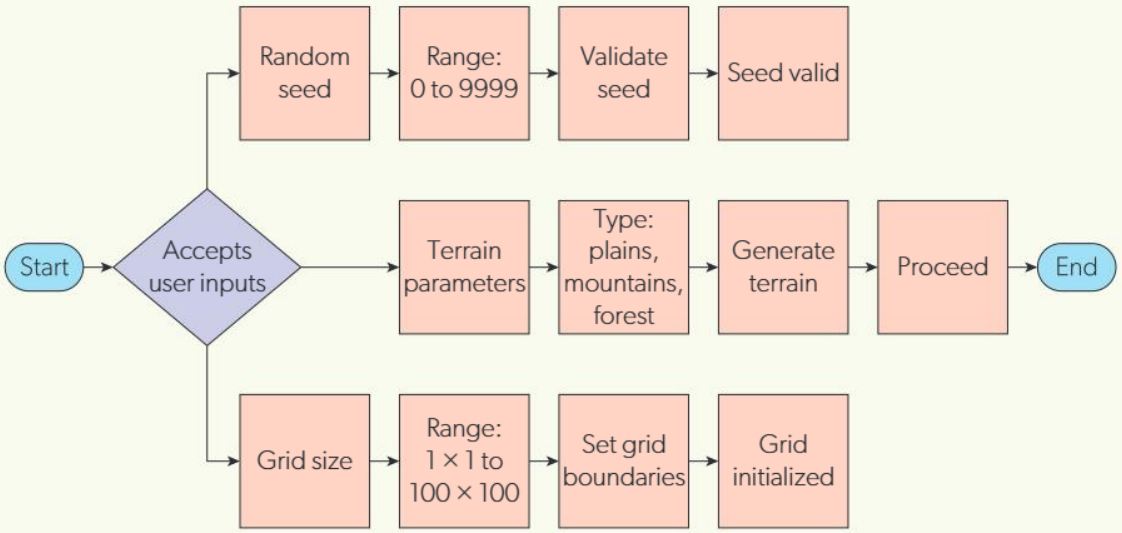
▲ **Table 7** Key terms for system overview

Case study: System overview

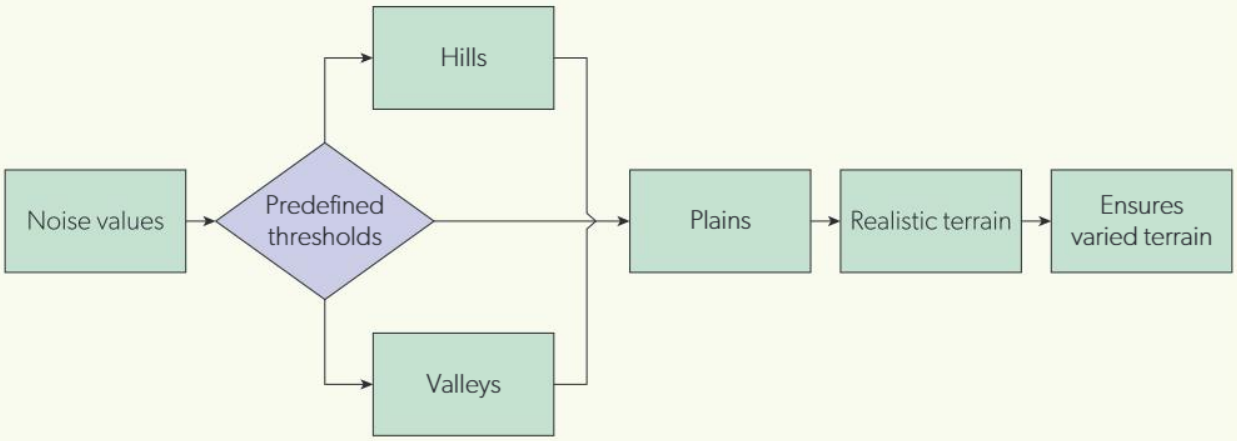


▲ **Figure 2** An overview of major parts of the system

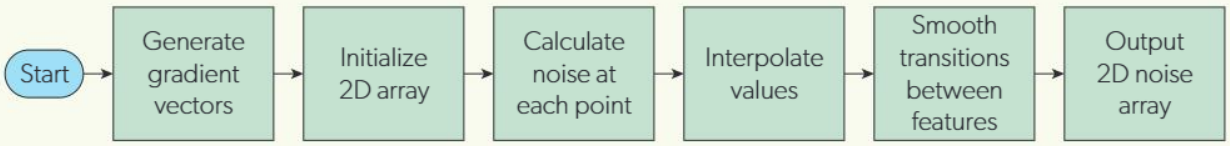
Case study: System overview



▲ **Figure 3** An overview of the input module process



▲ **Figure 4** An overview of Perlin noise generation process



▲ **Figure 5** An overview of the terrain generation process

Natural language of the Perlin noise generation algorithm

1. *Define the "fade" function.*
   - *Input: A value "t".*
   - *Process: Calculate and return "t * t * t * (t * (t * 6 – 15) + 10)".*
2. *Define the "lerp" function.*
   - *Input: A value "t" and two values "a" and "b".*
   - *Process: Calculate and return the linear interpolation of "a" and "b" based on "t", which is "a + t * (b – a)".*

3. *Define the "grad" function.*
   - *Input: A "hash" value and coordinates "x" and "y".*
   - *Process:*
     1. *Calculate "h" as "hash" AND 3.*
     2. *Determine "u" as "x" if the least significant bit of "h" is 0, otherwise "−x".*
     3. *Determine "v" as "y" if the second least significant bit of "h" is 0, otherwise "−y".*
     4. *Return the sum of "u" and "v".*
4. *Define the "perlin_noise" function.*
   - *Input: Coordinates "x" and "y", and an optional "seed" value (default is 0).*
   - *Process:*
     1. *Set the random seed to the given "seed".*
     2. *Create a permutation array containing integers from 0 to 255.*
     3. *Shuffle the permutation array.*
     4. *Create an extended permutation array "p" by concatenating the permutation array with itself.*
     5. *Calculate "xi" as the integer part of "x" AND 255.*
     6. *Calculate "yi" as the integer part of "y" AND 255.*
     7. *Calculate "xf" as the fractional part of "x".*
     8. *Calculate "yf" as the fractional part of "y".*
     9. *Compute "u" by applying the "fade" function to "xf".*
     10. *Compute "v" by applying the "fade" function to "yf".*
     11. *Calculate gradient values.*
         - *"n00" using the "grad" function with hash value "p[p[xi] + yi]" and coordinates "xf" and "yf".*
         - *"n01" using the "grad" function with hash value "p[p[xi] + yi + 1]" and coordinates "xf" and "yf − 1".*
         - *"n10" using the "grad" function with hash value "p[p[xi + 1] + yi]" and coordinates "xf − 1" and "yf".*
         - *"n11" using the "grad" function with hash value "p[p[xi + 1] + yi + 1]" and coordinates "xf − 1" and "yf − 1".*
     12. *Perform linear interpolation.*
         - *"x1" as the result of "lerp" function with inputs "u", "n00", and "n10".*
         - *"x2" as the result of "lerp" function with inputs "u", "n01", and "n11".*
     13. *Return the result of the "lerp" function with inputs "v", "x1", and "x2".*

Natural language for terrain generation algorithm

1. *Define the "generate_terrain" function.*
   - *Input: An optional "seed" value (default is 0), a "size" value (default is 256), and a "scale" value (default is 100.0).*
2. *Generate linear space.*
   - *Create a linearly spaced array "lin" from 0 to 5, with a length of "size" and excluding the endpoint.*
3. *Create a meshgrid.*
   - *Create a meshgrid "x" and "y" using the linear space array "lin", which will provide a grid of coordinates.*
4. *Generate Perlin noise.*
   - *Call the "perlin_noise" function using "x" and "y" coordinates scaled by the "scale" value, and with the given "seed".*
   - *Store the resulting noise values in the variable "noise".*
5. *Normalize the noise values.*
   - *Normalize the "noise" values to a range between 0 and 1.*
   - *Calculate "terrain" as the normalised "noise", where each value is adjusted by subtracting the minimum noise value and dividing by the range of noise values (maximum noise value minus minimum noise value).*
6. *Return the terrain.*
   - *Output the "terrain" array.*

Case study: System overview

Natural language for the visualisation algorithm

1. *Import libraries.*
   - *Import the "pyplot" module from the "matplotlib" library.*
   - *Import "Axes3D" from the "mpl_toolkits.mplot3d" module.*
2. *Define the "plot_heatmap" function.*
   - *Input: A "terrain" array.*
   - *Process:*
     1. *Display the "terrain" array as an image using the "terrain" colour map.*
     2. *Add a colour bar to the side of the image for reference.*
     3. *Set the title of the plot to "Terrain Heatmap".*
     4. *Display the plot.*
3. *Define the "plot_3d" function.*
   - *Input: A "terrain" array.*
   - *Process:*
     1. *Create a new figure.*
     2. *Add a 3D subplot to the figure.*
     3. *Generate a meshgrid "x" and "y" using the range of the "terrain" array dimensions.*
     4. *Plot the 3D surface using the "terrain" array with the "terrain" colour map.*
     5. *Set the title of the plot to "3D Terrain Map".*
     6. *Display the plot.*

Testing strategy

1. *Accuracy testing*
   - *Verify that the generated terrain maps accurately reflect realistic geological features.*
   - *Compare generated terrain with known samples to ensure smooth and continuous terrain.*
2. *Variety testing*
   - *Generate multiple terrain maps using different seeds.*
   - *Ensure each map is distinct and showcases diverse landscapes.*
3. *Performance testing*
   - *Benchmark the algorithm's execution time to ensure it meets the less than 5 seconds requirement for a 256 × 256 grid.*
   - *Profile the code to identify and resolve any performance bottlenecks.*
4. *Reproducibility testing*
   - *Use fixed seeds to generate terrain maps.*
   - *Verify that the maps are identical for the same seeds, ensuring reproducibility.*
5. *Visualisation testing*
   - *Validate the visual representation for clarity and accuracy.*
   - *Ensure that the heatmap and 3D plot accurately reflect the underlying terrain data.*

Hints and tips: System overview

Here is the rubric for section C. As with the previous sections, you should compare mark bands so you understand what is required.

Marks	Level descriptor
0	The response does not reach a standard described by the descriptors below.
1–2	The response: • **outlines** a limited system model • **identifies** algorithms for the components of the system model • **identifies** a testing strategy for at least one success criterion.
3–4	The response: • **constructs** a system model that is not complete • **constructs** algorithms for the components of the system model that lead to partial functionality of the product • **outlines** a testing strategy that aligns with at least three success criteria.
5–6	The response: • **constructs** a complete system model • **constructs** algorithms for the components of the system model that enable the product to perform • **describes** a testing strategy that aligns with the success criteria.

▲ **Table 8** Marks and level descriptors for internal assessment section C

Start by constructing a flowchart of a broad overview of your system. For each major sub-part of your system, you should include a flowchart diagramming how it works. A relational database will need an ERD diagram. If you use OOP, please include UML diagrams. A great way to test your diagrams for clarity is to ask a friend to explain how your program works (only using diagrams). Listen very carefully! If your friend does not understand something, it could mean part of your flowchart should be clarified.

Your algorithms can be pseudocode or natural language. Again, for each **major** subsystem you should have algorithms. Finally, ensure you have a test for each success criteria. Pay careful attention to the command term. There is a difference between "outline a testing strategy" and "describe a testing strategy".

Development (Criterion D)

The development of the product must be consistent with the problem specification in criterion A, the planning in criterion B and the system overview developed in criterion C.

- This section includes a video, which is used to showcase the functionality of your program and of the product. The video should also show examples of the testing of the product. The deployment of the testing strategy and its effectiveness must be described in the documentation, with examples of the testing seen in the video.
- The development of the solution must justify the structure of the product, state why it is appropriate, and demonstrate the techniques used to develop the product based on the algorithms constructed in criterion C. These techniques may include loops, data structures, existing libraries and the integration of software tools.
- The testing strategy must include testing for correctness, reliability and efficiency. The testing must be described and justified in the documentation with supporting examples seen in the video.

Terms	Definition
Implementation and coding of the algorithms	Techniques in the criteria refer to the process of programming algorithms using code. The documentation must highlight key elements of code that are important for the efficient functioning of the algorithms. Any code presented in the solution must include relevant comments, be consistent and be readable. Code excerpts included in the documentation must be referenced to the full source code submitted as an appendix.

▲ **Table 9** Key terms for development

Case study: Development

In this project, I aimed to create a custom Perlin noise algorithm to procedurally generate terrain similar to that in Minecraft. The development process involved several key steps.

1. *Algorithm implementation*
   - **Perlin noise generation:** *The core of the project was the implementation of a custom Perlin noise algorithm. This was achieved by defining functions for fading ("fade"), linear interpolation ("lerp"), and gradient calculation ("grad"). The "perlin_noise" function used these helper functions to generate a smooth and continuous pseudo-random terrain.*
   - **Terrain generation:** *Using the Perlin noise algorithm, I generated a 2D array of noise values. These values were then normalised to produce varied terrain features such as hills, valleys and plains. The "generate_terrain" function accepted parameters for seed, size and scale to customise the terrain generation process.*
2. *Visualization*
   - **Heatmap:** *The "plot_heatmap" function was implemented to visualize the terrain as a heatmap using "matplotlib". This provided an intuitive way to see the variations in terrain elevation.*
   - **3D Plot:** *To offer a more detailed view, a "plot_3d" function was created that rendered the terrain in 3D using "mpl_toolkits.mplot3d". This visualization highlighted the 3D structure of the generated terrain.*

Case study: Development

Evaluation of choices

1. *Algorithm choice*
   - **Perlin noise:** *Perlin noise was chosen due to its ability to create smooth, continuous and natural-looking textures. It is a widely used technique in procedural content generation, particularly in games and simulations. However, one drawback is that Perlin noise can be computationally intensive for large grids, which might affect performance on less powerful systems.*
   - **Python and NumPy:** *Python, along with NumPy, was selected for its simplicity and powerful numerical processing capabilities. These choices allowed for efficient handling of large arrays and mathematical operations. A limitation of this choice is that Python, being an interpreted language, might not be as fast as compiled languages like C++ for extremely large data sets or real-time applications.*
2. *Implementation details*
   - **NumPy operations:** *Utilising NumPy for array operations and random number generation ensured that the algorithm was both efficient and scalable. The use of element-wise operations ("np.where") in the gradient function was critical for handling arrays correctly. A potential drawback is that NumPy operations, while efficient, can sometimes lead to increased memory usage, which might be a concern for very large terrain maps.*
   - **Parameterization:** *Allowing parameters such as seed, size and scale provides flexibility and control over the terrain generation process, enabling the creation of diverse and varied landscapes. However, the complexity of managing multiple parameters can increase, requiring careful tuning to achieve the desired results.*
3. *Visualization Tools*
   - **Matplotlib and MPL Toolkit:** *These libraries were chosen for their robust visualization capabilities. Matplotlib's "imshow" and MPL Toolkit's "plot_surface" functions were ideal for creating clear and informative heatmaps and 3D plots, respectively. A limitation is that while these libraries are powerful, they may not be the most performant for real-time or interactive visualizations compared to other tools like "Bokeh" or "Plotly". Additionally, generating high-resolution 3D plots can be resource-intensive, potentially leading to slower performance on older hardware.*

Testing strategy justification

The testing strategy was designed to align with the success criteria of accuracy, variety, efficiency, reproducibility and visualization quality.

1. *Accuracy testing*
   - **Visual inspection:** *By generating multiple terrain maps and visually inspecting them, I ensured that the terrain features appeared smooth and realistic.*
   - **Comparison with known features:** *The generated terrain was validated against known geological features to confirm its realism.*
2. *Variety testing*
   - **Multiple seeds:** *By testing the algorithm with different seeds, I verified that each terrain map was distinct and varied. This showcased the algorithm's ability to produce diverse landscapes.*
3. *Performance testing*
   - **Execution time:** *The algorithm's execution time was measured to ensure it met the target of generating a 256 × 256 terrain grid in less than 5 seconds. Code profiling helped identify and address performance bottlenecks.*
4. *Reproducibility testing*
   - **Fixed seeds:** *Using fixed seeds, I repeatedly generated terrain maps to confirm that the same seed produced identical results. This guaranteed the reproducibility of the terrain generation process.*
5. *Visualization testing*
   - **Clarity and accuracy:** *The heatmap and 3D plot visualizations were validated to ensure they accurately represented the underlying terrain data. Sample data was used to confirm the visual clarity and effectiveness of the plots.*

The chosen techniques, tools and testing strategies collectively ensured that the custom Perlin noise algorithm produced accurate, varied and visually appealing terrain maps efficiently and reproducibly.

Hints and tips: Development

Here is the rubric for section D. As with the previous sections, you should compare mark bands so you understand what is required.

Marks	Level descriptor
0	The response does not reach a standard described by the descriptors below.
1–3	The response: • **constructs** a product with very limited functionality • **constructs** a product using no appropriate techniques to implement the algorithms • **states** the choices made to implement the algorithm • **states** the testing strategy used.
4–6	The response: • **constructs** a product that has limited functionality • **constructs** a product using at least one appropriate technique to implement the algorithms • **outlines** the choices made to implement the algorithm • **states** the effectiveness of the testing strategy.
7–9	The response: • **constructs** a product that has partial functionality • **constructs** a product that uses some appropriate techniques to implement the algorithms • **explains** the choices made to implement the algorithm • **describes** the effectiveness of the testing strategy.
10–12	The response: • **constructs** a fully functional product • **constructs** a product that uses appropriate techniques to implement the algorithms • **evaluates** the choices made to implement the algorithm • **justifies** the effectiveness of the testing strategy.

▲ **Table 10** Marks and level descriptors for internal assessment section D

In this section, you are demonstrating your product, evaluating the technical decisions you made, and evaluating the testing strategy.

In the video, you should show a success criterion, and then show that success criterion working in your program. Then, go to another success criterion and do the same thing. Your video should also include examples of testing. You do not need to describe the testing in the video.

In this section, make sure you discuss the advantages and disadvantages of each algorithm you used. Considering all the possible advantages and disadvantages, discuss why you made specific technical choices (for example, why you used a relational database instead of a flat file or NoSQL database).

Remember, you are justifying the effectiveness of the testing. Prove that the test is testing what you say it is testing!

Evaluation (Criterion E)

The evaluation of the product must be consistent with the problem specification and success criteria in criterion A.

Case study: Product evaluation

1. *Accuracy*
   - **Perlin noise function:** *The Perlin noise function was successfully developed and tested to generate smooth and continuous pseudo-random values. Gradient and interpolation techniques were effectively implemented to produce realistic terrain features.*
   - **Validation:** *The generated terrain was validated against known geological features, ensuring realism. This criterion was fully met as the terrain features such as hills, valleys and plains appeared natural and continuous.*
2. *Variety*
   - **Diverse terrain types:** *The algorithm successfully generated diverse terrain types. Parameters such as frequency and amplitude were introduced and effectively controlled to enhance variety.*
   - **Multiple seeds:** *Testing with multiple seeds verified that distinct and varied terrain maps were produced each time. This criterion was fully met, demonstrating the algorithm's capability to create diverse landscapes.*
3. *Efficiency*
   - **Performance optimization:** *The algorithm was optimized for performance by using efficient data structures and minimizing computational overhead. However, while the execution time was generally under 5 seconds for a 256 × 256 grid, performance varied depending on the hardware, indicating room for further optimization.*
   - **Benchmarking and profiling:** *Code profiling helped identify and address performance bottlenecks. This criterion was mostly met, but additional fine-tuning could further improve efficiency.*
4. *Reproducibility*
   - **Seed-based randomness:** *Seed-based randomness was implemented to ensure reproducibility. Fixed seeds consistently produced identical results, confirming the reproducibility of the terrain generation process.*
   - **Documentation:** *The usage and impact of seeds on terrain generation were well documented. This criterion was fully met, ensuring that the terrain can be reliably reproduced with the same seed.*
5. *Visualization quality*
   - **Visualization techniques:** *Effective visualization techniques were implemented using "matplotlib" for heatmaps and "plotly" for 3D plots. These tools provided clear and informative visual representations of the terrain.*
   - **Validation:** *The visual representations were validated with sample data to confirm clarity and accuracy. This criterion was fully met, as the visualizations accurately reflected the underlying terrain data with appropriate colour gradients and elevation details.*

Justification for improvements

1. *Efficiency enhancements*
   - **Further optimization:** *Although the algorithm was generally efficient, further optimization could improve performance, particularly on lower-end hardware. Exploring parallel processing or more advanced optimization techniques could reduce execution time.*
   - **Alternative libraries:** *Investigating alternative libraries or tools that offer better performance for large data sets or real-time applications could enhance overall efficiency.*

Case study: Product evaluation

2. *Advanced visualization*
   - **Interactive visualizations:** *While the current visualizations are effective, incorporating more interactive elements using libraries like "Bokeh" or advanced features of "Plotly" could enhance user experience.*
   - **High-resolution support:** *Improving support for high-resolution 3D plots can ensure smoother performance and better visualization quality, especially on higher-end systems.*
3. *User interface improvements*
   - **Enhanced UI:** *Developing a more user-friendly interface for controlling parameters such as seed, frequency and amplitude could make the tool more accessible. This could include sliders, input fields and real-time updates.*
   - **Additional features:** *Adding features such as saving and loading configurations, exporting terrain data and providing detailed documentation or tutorials could improve the usability and appeal of the product.*

Hints and tips: Evaluation

Here is the rubric for section E. As with all the previous sections, you should compare mark bands so you understand what is required.

Marks	Level descriptor
0	The response does not reach a standard described by the descriptors below.
1–2	The response: • **states** the extent to which the success criteria were met • **describes** improvements to the product.
3–4	The response: • **evaluates** the extent to which the success criteria were met • **justifies** improvements to the product.

▲ **Table 11** Marks and level descriptors for internal assessment section E

Ensure you evaluate **every single success criteria.** Remember, "evaluate" refers to weighing the advantages and disadvantages and then arriving at a conclusion. Improvements should be justified—explain why an improvement would make the product better.

External assessment: Paper 1 and Paper 2

Format of the syllabus

The syllabus format (Figure 1) provides a lot of information that can help you when studying.

- The **guiding question** frames the topic and helps you to understand the inquiry you should complete when studying the topic.
- Each topic is divided into **subtopics**. These tell you what you will learn about in each unit.
- Each subtopic is split into **learning statements**. These tell you the specifics of the computer science course. The bullet points below each learning statement identify what you should know and be able to do after studying the work covered by the learning statement.
- **Linking questions** show you the related content across the computer science course, as well as links to theory of knowledge (TOK).

This course guide follows a similar structure as well as providing you with worked examples, the content in the real world and approaches to learning skills.

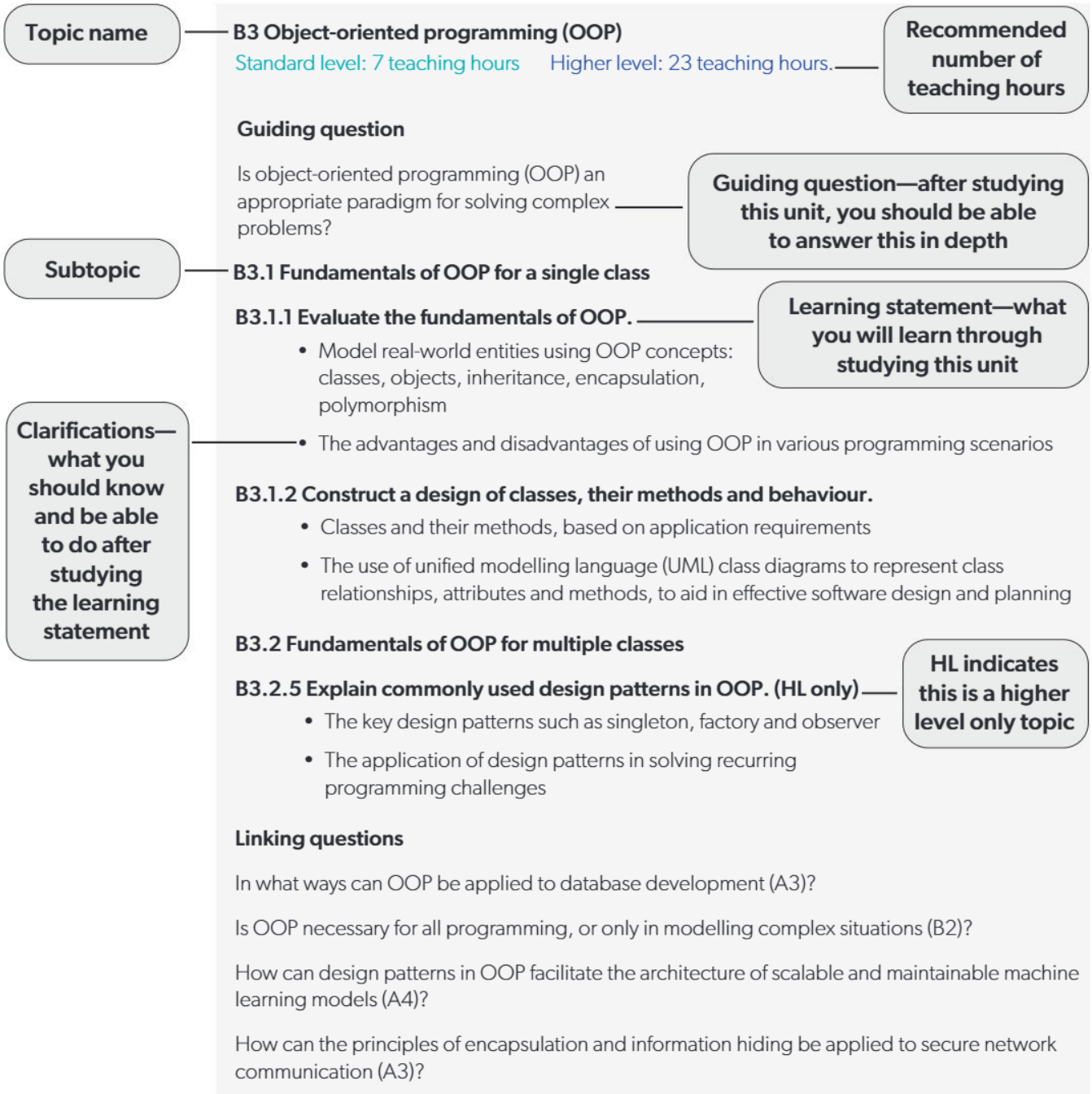
The number of hours identified on the syllabus indicates to your teacher how long they should spend in class teaching the topic. This is not an indication of how long *you* should spend on the topic. You may need to put in significant time and effort to ensure you know and understand the many different topics in the course. You need to be able to understand all aspects of the syllabus included in the level you are taking.

Much of the information you study is linked in some way. For example, knowing how data warehouses use tools to extract information can help support your understanding of machine learning.

External assessment outline

At standard level, 70% of your grade will come from external assessment. At higher level, this increases to 80%. This is a significant percentage of your final grade. Therefore, it is essential for you to be fully prepared.

For both standard level and higher level, there are two papers. These will be completed over two days: paper 1 on day one and paper 2 on the next examination day.



▲ **Figure 1** Part of the unit plan for topic B3 Object-oriented programming (OOP), showing all the features within the guide

The external examinations test three assessment objectives (AOs).

- AO1: Demonstrate knowledge and understanding.
- AO2: Apply and use knowledge.
- AO3: Construct, synthesize, analyse and evaluate.

For each of the assessment objectives, you may be asked to respond to a range of short and extended response questions. You may also be asked to outline a solution to an appropriate problem. Although the questions may look similar, the command term used in the question can help you identify the level of response required.

Term	AO	Definition	Things to include in your response
Calculate	2	Obtain a numerical answer showing the relevant stages in the working.	You need to show all stages of you working out. For example, if you were asked to calculate the hexadecimal representation of a binary number, you need to show the answer and the intermediate stages.
Compare	3	Give an account of the similarities between two (or more) items or situations, referring to both (all) of them throughout.	If you were asked to compare clustering to classification when analysing data in a data warehouse, you might choose three features and compare both techniques against these features for the given scenario.
Construct	3	Display information in a diagrammatic or logical form.	If you see a construct question in computer science, you will either have to draw a diagram you have studied or answer using the coding language you have studied.
Deduce	3	Reach a conclusion from the information given.	You must provide an answer and justify that answer using the information that has been presented to you. This format may be useful here. Point: what you have decided Evidence: what led you to this decision Explain: why you made this decision.
Define	1	Give the precise meaning of a word, phrase, concept or physical quantity.	Give a glossary style definition for the word you have been asked to define. **Example** Recursion: when an algorithm calls itself with updated parameters until the base case is met.
Describe	2	Give a detailed account.	These are "tell me what you know" style questions. For example, if you were asked to describe reinforcement learning, you might give a definition of reinforcement learning with one or two examples of what it could be used for.
Discuss	3	Offer a considered and balanced review that includes a range of arguments, factors or hypotheses. Opinions or conclusions should be presented clearly and supported by appropriate evidence.	These ask you to make a balanced argument. For example, if you were asked to discuss why distributed databases may be an appropriate solution in this scenario, you would offer an advantage of a distributed database while acknowledging a possible disadvantage before stating why in this situation it would still be best.
Distinguish	2	Make clear the differences between two or more concepts of items.	You must make the difference between the ideas very clear. For example, if you were asked to distinguish between static and dynamic data structures, you would identify that static data structures cannot change size at runtime while dynamic data structures can.
Estimate	2	Obtain an approximate value.	Given the presented data, you may be asked to estimate a value and state why you researched that value.
Evaluate	3	Make an appraisal by weighing up the strengths and limitations.	Within an evaluate question you are expected to come to a conclusion. Therefore, you would offer one or two advantages for an idea, one or two disadvantages, and then state which you would choose in the given situation and why.

Term	AO	Definition	Things to include in your response
Explain	3	Give a detailed account including reasons or causes.	When answering an explain question, it is important you emphasize the **why**. For example, if the question asks you to explain why a linked list would be useful in this situation, explain what a linked list is, explain the benefits of using a linked a list, and then explain why you need these benefits in this situation.
Identify	2	Provide an answer from a number of possibilities.	You need to choose an answer from a list of a possible answers and, whenever possible, give a brief outline for your choice.
Justify	3	Give valid reasons or evidence to support an answer or conclusion.	When asked to justify, you must provide reasons for your response. For example, if you were asked to justify which data mining technique you would use in a specific situation, you have to say which technique you would choose and then provide reasons for this linked to the scenario.
Label	1	Add labels to a diagram.	You will be provided with a diagram and you must label the parts of the diagram correctly.
List	1	Give a sequence of brief answers with no explanation.	If you are asked to list, then you only write the stages or items. You do not need to explain your thinking. For example, you may be asked to list the stages in a binary search. You do not need to justify your response, just list the stages in order.
Outline	2	Give a brief account or summary.	When asked to outline, you do not need to provide reasons. An example may be to outline the structure of an artificial neural network (ANN). You will not have to provide reasoning. You would only need to give a brief outline of how the neurons are organized and why they are organized this way.
Sketch	3	Represent by means of a diagram or graph (labelled as appropriate). The sketch should give a general idea of the required shape or relationship, and should include relevant features.	This requires you to draw the item you have been asked to, showing its shape. An example might be sketching the resulting binary tree, in which case you cannot just write the values—you also have to show them in a tree-like structure.
State	1	Give a specific name, value or other brief answer without explanation or calculation.	When responding to a state question you only need to write the word or sentence. For example, state the data type required to store a person's age. A sufficient answer would be: integer.
Suggest	3	Propose a solution, hypothesis or other possible answer.	Propose a solution to a problem with an explanation of the reasons behind your choice.
To what extent	3	Consider the merits or otherwise of an argument or concept. Opinions and conclusions should be presented clearly and supported with appropriate evidence and sound argument.	Consider the positives and negatives of what you have been asked. Describe the positives and describe the negatives before making a conclusion explaining what you think.
Trace	2	Follow and record the actions of an algorithm.	Write down the variables in the algorithm and the decisions being made. Write down the values of the variables and the decisions being made at each stage of the algorithm showing clearly the result or output.

▲ **Table 1** Command terms used in computer science exams

Outline of assessment

Standard level	Higher level
Paper 1: 1 hour 15 minutes Weighting: 35% Marks available: 50	**Paper 1:** 2 hours Weighting: 40% Marks available: 80
Paper 2: 1 hour 15 minutes Weighting: 35% Marks available: 50	**Paper 2:** 2 hours Weighting: 40% Marks available: 80

Examination explanations

It is essential that you check **when** and **where** your examination occurs. It is a good idea to arrive early so you have time to prepare yourself for the exam. Entering the exam room in a calm frame of mind always helps you to do your best.

Paper 1—Theme A: Concepts of computer science

The two sections of paper 1 assess the content you have learned in theme A and the material within the case study. All students complete the standard-level material. There is additional higher-level material for students doing the higher-level course.

Paper 1 Section A

Section A of paper 1 covers topics from theme A, concepts of computer science, from the following units:

- A1 Computer fundamentals
- A2 Networks
- A3 Databases
- A4 Machine learning

Each question in section A will focus on content from one unit (sometimes linking to another unit). The questions will be presented in the following format.

- First, a scenario: a short introduction to a context in which the question paper is being asked.
- There will then follow around two to five questions based on the scenario. For example:
    - Question 1: a short question related to the scenario
    - Question 2: a slightly harder question related to the scenario
    - Question 3: a hard question related to the scenario.

Example question

4 A school stores information about students within a database.

The parent/guardian evening view, MEETING, is an extract showing the different appointments between students, parents/guardians and teachers.

The time represents the time of the meeting.

Student ID	Student Name	Parent/Guardian Name	Teacher ID	Teacher Name	Time	Subject
3948	Colin Stephan	Ray & Barbara	48	Dheepa	10:00	Maths
				Marta	10:15	Spanish
4944	Tessa Allingham	Melanie	34	Dheepa	10:15	Maths
				Michael	10:30	German
3399	Lucy Simons	Rayissa	23	Ole	10:00	Music

(a) Construct a SQL SELECT query to list the name of students and parents/guardians who have appointments to meet with maths teachers. [3]

The database view, MEETING, can be represented using the following notation:

MEETING(Student ID, StudentName, Parent/GuardianName, TeacherID, TeacherName, Time, Subject)

(b) Construct the database in third normal form (3NF) for the three entities in the MEETING view extract. [5]

The school has multiple people accessing the system at the same time. Atomicity needs to be applied to the database to ensure no errors occur.

(c) Describe how atomicity manages transactions within the database. [3]

The school stores data regarding students' grades and medical issues on the database.

(d) Outline one ethical issue regarding this data within the database. [3]

Section A of paper 1 aims to assess your knowledge and understanding as well as your ability to apply the work you have studied in theme A. There are 38 marks available on the standard-level paper and 56 marks available on the higher-level paper.

Remember the following.

- All questions are compulsory.
- There will be questions testing AO1, AO2 and AO3.
- When working through the questions, take your time, read the question carefully, and look for clues in the question to help you identify what you need to do. If you rush, you may misinterpret the question.
- Consider using of a highlighter to identify key terms and phrases. For example:

State **one** reason data cleaning is carried out before training a CNN [1]

- Make sure you answer the question in a manner expected by the command term. A description of these can be found in Table 1.

- If you find a question that you are unsure how to answer, make a note on the page and move on to the next question. You can come back to this question if you have time at the end of the exam.
- Take time before the end of the exam to review your answers. Fill in any gaps. There is no negative marking, so you lose nothing from an educated guess!

Paper 1 Section B—Case study

Section B of paper 1 is based on a previously encountered case study. The case study is a scenario that allows you to study current developments, emergent technologies, and ethical issues in computer science, and you will be given class time to do this. Your teacher may guide the class through the investigation or you may complete the investigation yourself. Your teacher will receive the case study in the June before your exam (for both May and November candidates).

The case study allows you to investigate concepts you have learned throughout the computer science course in a real-world context. Higher-level students will study four challenges in detail and standard-level students will study two challenges.

By the end of the case study investigation, you should be able to:

- show an understanding of how the systems in the case study work
- show an understanding of computational thinking fundamental to the systems in the case study
- apply concepts from the course syllabus in the context of the case study
- explain how scenarios in the case study may be related to other scenarios
- discuss the impacts and ethical issues relevant to the case study
- evaluate, formulate and justify strategies based on the information from the case study itself, your own research, and new stimulus provided in the examination paper.

You will be given a clean copy of the case study in the exam. The case study (section B of the exam) is worth 12 marks at standard level and 24 marks at higher level.

At standard level, you will be expected to answer around three case study questions. At higher level, you will be expected to answer four or five shorter questions and one longer question.

Examples of possible questions about the material in the case study.

Outline one other potential consequence of …
Outline two ways that a X improves the performance of …
To what extent do the benefits of X outweigh the risks of Y …
Describe two advantages of …
Describe two strategies that could be used to …

For the case study section of your exam, you might find it helpful to:

- highlight key terms that will help you to remember what information to include in your answer.
- plan out your answer to longer questions before starting your response.

Paper 2—Theme B: Computational thinking and problem-solving

There are two options for paper 2, allowing you to complete the paper using the Java programming language or the Python programming language. You should not attempt both! The paper will consist of all the content you have learned in theme B of the course. This includes the following units:

- B1 Computational thinking
- B2 Programming
- B3 Object-oriented programming (OOP)
- B4 Abstract data types (HL only)

Each question in paper 2 will be based around computational thinking, programming and problem-solving. Question 1 covers computational thinking, giving you a chance to show you understand OOP concepts and can develop algorithms without coding. The remaining questions will cover coding in either Java or Python.

It is likely that you will be given one or two scenarios and asked to code solutions to problems within the scenario as well as answering questions about OOP concepts.

Example questions

- State one reason why a flowchart could be used to represent a program.
- Complete the trace table where $\boldsymbol{x} = \mathbf{14}$.
- State the Big O notation for the given algorithm.
- Sketch the stack resulting from the following operations:

```
push(1)
push(5)
push(40)
peek()
pop()
pop()
push(2)
```

- Identify two consequences of declaring an attribute as private.
- Construct the method `addStudent()`
- Outline how a `HashSet` can be used to ensure only one email is sent to each address.

Paper 2 aims to assess your knowledge and understanding as well as your ability to apply what you have learned about theme B.

Remember the following.

- Read each question carefully.
- When reading a question, identify what it is asking and what tools you need to answer it. This is particularly important with coding questions. Take a moment to identify what methods you need, what you need to output or return, and what tools you need to do this (for example, if statements, loops, lists).
- Use a highlighter to select the key terms in the question.
- Spend time writing down the steps involved in the solution before translating this into code. You might be surprised at how helpful this step is!
- Make sure your solution returns or outputs what is expected.
- At standard level, you will only look at one class and perhaps reference a driver class. If you have more than this, you may have made an error.
- At higher level, you will deal with multiple classes. Think carefully about what class the method is in and what you need to access to make it work.
- If you find a question that you are unsure how to answer, make a note on the page and move on to the next question. You can come back to this question if you have time at the end of the exam.
- Take time before the end of the exam to review your answers. Fill in any gaps. There is no negative marking, so you lose nothing from an educated guess!

Study skills for the external assessment

ATL Self-management skills

Removing distractions

When revising for your exams, focus on the task at hand. It may help you to set up a study space that will help you maintain your focus. Having a dedicated space will help you to get into the "studying mindset". You may find it useful to minimize the applications open on your computers and remove your phone from your desk. To maximize your time, you may find it useful to set a timer, complete the task you have set yourself and then reward yourself with a little break.

Work out what conditions help you to revise most efficiently, and establish this as your general study routine. Don't wait until the end of the course to work this out!

Tracking your studies

As with all courses, the IB computer science course has prescribed topics and content. You may find it useful to keep track of your learning on a copy of the course guide (digital or print).

- Highlight the sections you have covered in lessons.
- Add your own notes, identifying the topics you need to spend time on at home and which topics you are comfortable with.

The course guide is not sequential, so your teacher may jump from topic to topic. If you are unsure, ask!

ATL Communication skills

Recording information

There are many ways to record information so that you remember it both in the short term and the long term. In the computer science course, you will probably encounter factual and conceptual information that is new to you. Developing a note-taking system that works for you will be essential to helping you to recall the information you have learned closer to the examinations.

- Work with your preferred learning styles. Make notes on paper, on your computer, using diagrams, or using a mixture of different styles.
- Remember that you need to reproduce your knowledge in exam conditions, so try to avoid methods that you cannot use in an exam setting (for example, listening to personal music, using large gestures or dance). Make it practical!

Vocabulary skills

All academic disciplines have specialist vocabulary. Computer science is no different and has an extensive specialist vocabulary. You may find it useful to develop a computer science glossary. Whenever you encounter a new term, you can add this to your glossary and provide a definition. Continued reading of the term within the context and using the term yourself will help you to develop your confidence using the new vocabulary.

- See the ATL feature on page 6 for more ideas on this.
- Remember to update your glossary throughout the course.

ATL Research skills

Information literacy

When completing your research for the case study, you will need to exercise your information and media literacy skills. This includes gathering sources, interpreting information, evaluating the information, and then communicating the knowledge through your exam answers. You need to use media literacy skills to consult online sources and identify the different perspectives within the sources.

Identify the value of each source you use in your research to determine whether it is reliable or not. Ask yourself the following questions.

- Who wrote it?
- When was it written?
- With what purpose?
- What assumptions has the author made?

Planning

Take time to plan your research for the case study.

- Spend time reading through the case study carefully.
- Consider what questions you may be asked about it. Reading through past papers will help you work out the kind of questions you may be asked.
- Once you have an idea of the questions, begin to identify research sources that might be useful. Your teacher should help you with this, but planning your own research will help you gain a deeper understanding of the case study, which in turn will enable you to respond to questions fully.

Case study sources

Case study sources will vary depending on the topic of the case study and your knowledge of the topic. To identify sources of research for the case study, you might consider separating the areas of the case study into different topics then identifying different sources for each topic. To find sources, try the following steps.

- Consider your community. Is there anyone working in the field or who has knowledge of the field you can ask for help?
- What databases does your library have access to?
- Can you find scholarly articles covering the topics?
- What experiments can you complete that will help you understand the topics in detail?

Computational thinking skills

Algorithmic thinking

To practice algorithmic thinking, you can complete the following tasks.

- Identify problems in your community. Then, identify the different objects within the problem and model them using one of the system models you have learned.
- Develop algorithms to complete a task without coding. Writing steps to solve a problem helps you develop algorithmic thinking skills.
- Complete the problems identified in this book, using all the development skills you have learned throughout the course.
- Thinking of yourself as a user of the system, consider what features you would like within the system. Write down the steps of the algorithm to develop those features.

Coding

As with any skills you learn, practising that skill will help you to master it. The best thing you can do when coding is to practice. Here are some ideas to help you.

- Complete the tasks in the book. If you find them overwhelming, start with a simple version and develop it in stages.
- When you complete a task, consider how you could improve the solution. Then develop those improvements.
- If you have developed an algorithm to solve a problem you have identified, try to code the solution to the problem.
- Set yourself little challenges each week and try to complete them. Ask your teacher or a friend for support if you need it.
- Continue coding after the coding units have been completed. It is surprising how natural coding will become if you practise it often for short amounts of time.

AI is both your friend and your foe. AI can be useful if you need inspiration. But if you do not understand the code AI produces, then you will not be able to develop your own code or your own algorithmic thinking skills. Use AI wisely and always cite your source.

Please note: there are academic guidelines regarding the use of AI within IB courses. Please talk to your teacher or IB coordinator for more information.

Index

Page numbers in **bold** refer to tables; page numbers in *italic* refer to figures.

ABC (abstract base class) 497
abstract data types (ADTs) 548–81
  binary search trees 568–72
  collision resolution 577
  hashing function 576
  linked lists 552–67
  load factor 577
  operations 551
  principles 576–9
  properties **550**
  purposes 549, **550**
  sets 572–6, 577–8
abstraction 68
  abstract data types **550**, 551
  computational thinking 323, 326–8, 330, 331
  object-oriented programming 492–500
AC *see* accumulator
access control lists (ACLs) 140
access controls **67**, 70
  see also firewalls
accessing, programming 517
accountability, AI systems 307, 310
accounting 74
accumulator (AC) **4**
ACID (atomicity, consistency, isolation, durability) properties, database transactions 211–14, 233–4
ACLs *see* access control lists
ACP *see* atomic commit protocol
activation functions
  artificial neural networks **294**, **295**
  convolutional neural networks 303
actuators **86**, 87, 88
address bus 5, 20
ADTs *see* abstract data types
advanced persistent threats (ATPs) **155**
agents, machine learning 285–8
  convergence 288
  examples **285**, 287
  exploration versus exploitation trade-off 286–7, 288
  feedback loop 287
  learning process 287
aggregate functions, SQL 206–9
  Average **206–7**
  Count **207**
  Maximum value **208**
  Minimum value **208**
  Sum **209**
aggregation, class relationships 425, 501–5
AI *see* artificial intelligence
algebra *see* Boolean algebra
algorithmic thinking 610
algorithms 376–406
  Apriori algorithm 283–4
  averaging 393–4
  bias in 309, 364
  Big O efficiency 376–80
  binary search 382–5, 398–400, 456–63
  bubble sort 385, 386–8
  call 359
  control algorithms **86**, 87
  count occurrences 394–5
  cubic algorithms **377**
  design 323, 328–9, 330, 331
  factorial algorithms **377**
  fairness of 307
  linear search 381–2, 455–6, 460
  maximum value 395–6
  minimum value 395–6
  quadratic algorithms **377**
  quicksort 400–3
  recursive algorithms 359, 396–406
  routing algorithms 127
  search for data 380–5
  selection sort 385, 388–92
  sorting 385–96
  summing 393
ALU *see* arithmetic logic unit
And command, SQL **202**
AND gate *51*, 52, **53**
ANNs *see* artificial neural networks
anomalies 168, 180, 181, 227–8
anonymity, internet browsing 131
antivirus software 71
APIs (application programming interfaces) 340, 341
appending to file code 408–9
application layer, TCP/IP model **119**, 120, 122–123
application-specific integrated circuits (ASICs) 249
Apriori algorithm 283–4
AR *see* augmented reality
architecture 5
  graphics processing units 6–7
  networks 124–43
arithmetic logic unit (ALU) 4
arithmetic operations 17, 53
Array, Java 442
ArrayList, Java 350, 352, 355, 442–4, 447–9, 477–8
artificial intelligence (AI)
  accountability 307
  algorithmic fairness 307
  bias 307
  decomposition methods **325**
  environmental impact 308
  ethical implications 307–10
  pervasive AI 312
  privacy 308
  risk mitigation 313
  security 308
  societal impact 308, 313
  transparency 308, 313
  see also machine learning
artificial neural networks (ANNs) 292–300
  activation functions **294**, **295**
  classification example 293–4
  components and terminology **292**, **293**
  pattern recognition example 296–8
  regression example 294–5
ASCII (American Standard Code for Information Interchange) 47, 50
ASICs *see* application-specific integrated circuits
assembly language **90**, **91**
assessment
  external assessment 600–10
  internal assessment 582–99
association rule discovery 225–6
association rule learning 282–5
association rule mining 283
asymmetric cryptography 160, 162
atomic commit protocol (ACP) 234
atomicity, databases 182, 211
ATPs *see* advanced persistent threats
attributes, databases 170, 176
audio, data storage 48–9
augmented reality (AR) 312
authentication, data/network security **156**, **158**, 231
automatic doors 88–9
autonomous vehicles 87
Average function, SQL **206–7**
averaging 393–4

back propagation, artificial neural networks **293**, 297, 298
background operations 70
bandwidth, data transmission **148**
bank account transactions 212–13
Barker style, entity relationship diagrams *176*
barriers 12
batch processing 407
behavioural design patterns 522
Between command, SQL **198**
bias
  AI systems 307, 310
  in algorithms 309, 364
  artificial neural networks **292**
  and data mining 227
  gender bias 364
  racial bias 364
  in training data 308–9
big data, and databases 171
Big O efficiency 376–80, **377**
binary
  data storage 47–51
  equivalents and conversions 39–42, **43**, 45–6
  logic gates 51–5
  number system 38–9
  strings and characters 47–8
  subscript 38–9
binary encoding 49–51
binary executable **90**
binary search algorithm 382–5, 398–400, 456–63
binary search trees (BSTs) 568–72, 579
Bitcoin 136
bitmap images 49
bits **39**, **50**
  bit depths 49
BitTorrent 136
blacklists 154, 159
blockchain 136
Boolean algebra 52, **63**, 64–5
  simplifying complex logic diagrams and expressions 63–5
  simplifying output expressions 58–61
Boolean data **338**
Boolean expressions 56–7
Boolean operators **53–4**
boot-up process 14
bots, customer service 321, 322
breakpoint debugging 346
broadcast address 122
BSTs *see* binary search trees
bubble sort 385, 386–8, 464–72
budgeting 223
BufferedReader, Java 409–10
buses 4–5, 20
business computing 551
business environment
  data mining 220, 223–8
  file servers 130
  online analytical processing 220–3
  proxy servers 131
  routers 112
  virtual local area networks 140–141
  wireless access points 113
bytecode **90**
bytecode interpreters **95**
bytes **39**, **50**

cache memory 12–15
calculate function 371
call 359
Canva (graphic design platform) 32
cardinality, entities 170
CAs *see* certificate authorities
cascading, databases 180
casting 441
catch statement, error handling 342–3
central processing unit (CPU) 3–6, 8–12, **67**
  buses 4–5, 20
  cache memory 12–15
  components 3, 4, *5*
  core architecture 9
  design philosophy 8
  fetch–decode–execute cycle 15–19
  graphics processing unit comparison 8–12
  interrupts 78–81
  latency 8
  memory access 10
  memory interactions 13–15, 20–1
  performance monitoring 72
  pipelining 21–4
  polling 78–81
  power efficiency 10–11
  primary memory 12–15
  processing overhead **79**
  processing power 9–10
  processor types 5
  registers 4, *5*, 12, 13, 20–1
  scheduling 70
centroids 279
certificate authorities (CAs) 161
CF *see* collaborative filtering
characters 47–8, **338**
checksums 71
Chen notation style *176*
CIDR *see* classless inter-domain routing notation
circuits, logic gate connection and interaction 61
circular linked lists 563–7
citation of sources vii
classes 417, 418–19, 420, 422–5
  abstract classes 492–500
  aggregated relationship 425, 501–5
  composition relationship 425, 506–10
  creation/design of 431–3
  inherited relationship 425
  instance variables 418
  instantiation of 420, 434
  library class code 426–30
  looping 519, 521
  multiple classes, coding for 511–21
  runner class 440
  see also UML class diagrams
classification 238
  artificial neural networks 293–4
  data mining 224
classification techniques, in supervised learning 268–75
classless inter-domain routing (CIDR) notation 141–3
client–server model 134–5
clock speed 9
cloud-based implementation 215–16
cloud computing 31–6, 104–5
  choice of model **36**
  infrastructure as a service 34, 35–6
  machine learning 249
  platform as a service 33–4, 35, **36**
  software as a service 31–3, 35, **36**
clustering techniques 238, 279–82
  data mining 224–5
  density-based spatial clustering 280
  hierarchical clustering 280
  K-means clustering 279
  mean shift clustering 281
CNNs *see* convolutional neural networks
co-processors 5
code generation **92**, **93**
code sequence 362
coding skills v, 610
collaborative filtering (CF) 271–3
collision resolution strategies, hash tables 577
comments 450
Commit command, SQL 213
commutative law 64
compareTo method, Java 464
comparison operators 464
COMPAS algorithm 309
compilation/compilers 90–7, 499, 500
  definition **90**
  overview of process **93**
compile-time polymorphism 481–4
compiled programming languages 93, **96**
  just-in-time compilation comparison **94**
components 3, 4, *5*, **86**
composite key, databases 170
composition 425, 506–10
compression 28–31
computational thinking 316–35, 607–8
  abstraction 323, 326–8, 330, 331

computational thinking (*Continued*)
algorithmic design 323, 328–9, 330, 331
data analysis 330
database design 331
decomposition 323, 324–5, 330, 331
machine learning 330
network security 331
pattern recognition 323, 325–6, 330, 331
problem solving 330
problem specification 317–23
software development 330
study skills 610
concatenate 339
concatenated key, databases 170
conceptual schemas 173–4
concurrency control, databases 230
confirmation bias 227
congestion, networks 151
consent, AI systems 307
constraints, problem specification 318
content security policies (CSPs) **157**
context switching, scheduling **77**
control algorithms **86**, 87
control bus 5, 20
control systems **86**, 87–9
control unit (CU) 4
controllers **86**
convolutional neural networks (CNNs) 240, 242, 300–3
concepts and terms **301**
image classification 303
input data process 303
process **301**
cookies 116–17
cores (processing units) 5, 6, *7*, 8, 9, 10, 23–4
Count function, SQL **207**
count occurrences 394–5
CPU *see* central processing unit
creational design pattern 522
cross-platform development **96**
cross-site scripting (XSS) **156**
crows foot style, entity relationship diagrams *176*
cryptocurrencies 136
cryptography 159–60, **160–1**, 162
CSPs *see* content security policies
CU *see* control unit
cubic algorithms **377**
curse of dimensionality 259–62
customer feedback systems 320–2
customer service
chatbots 321, 322
data mining 228
cyberattacks
countermeasures against **157–8**
employees' role 158
vulnerabilities to **156**
see also network security; security
cyberbullying 310, 311

data analysis 330
data bus 5, 20
data cleaning 219, 251–6
clean data *252*
data quality impact 252–3
duplicate data 254
initial data *252*
irrelevant data 254
missing data 255
outliers 254, 256
data compression 28–31
data consistency **210**, 211, 213, 230
data control language (DCU) 192
data definition language (DDL) 191–2
data integrity 71, 211–14
data management 217–19
data manipulation language (DML) 192
data mining 220, 223–8
association rule discovery 225–6
bias 227
classification 224
clustering 224–5
fraud detection 226, 228
marketing 224, 228
regression 225
sequential pattern discovery 226–8
data packets 110, 111–12
data partitioning, databases 231
data preprocessing 251–63
data cleaning 251–6
feature selection 256–8
normalization 255–6
standardization 256
data processing
logic gates 52
machine learning **247**, **248**
data representation 38–46
binary and decimal integer conversions 39–42
binary and hexadecimal integer conversions 45–6
binary integers 38–9
hexadecimal and decimal integer conversions 43–5
hexadecimal integers 42–3
terminology **39**
data retrieval 51
data segmentation 149–51
data signal processors (DSPs) 5
data storage 47–51, **67**, 233
data structures 350–61
arrays and lists 351–8
first in, first out 359–60
last in, first out 358–9
one-dimensional structures 442–6
stacks 358–9
static and dynamic structures 350–1
two-dimensional structures 447–50
data/time data types **178**
data transmission 144–53
data segmentation 149–51
IP addressing 144–7
packet switching 149–51
static and dynamic routing across LANs 152–3
types of media for 147–9
data types 337, **338**, 491
see also abstract data types
data units, byte and bit equivalents **50**
data visualization 261
data warehouses 217–23, 500
components of *218*
data cleaning 219
data mining 220, 223–8
historical data 219
online analytical processing 220–3
as subject-oriented 218–19
as time-variant 219
database design 173–90
computational thinking 331
data types, relational databases 177–9
denormalization process 189–90
entity relationship diagrams 173–7
normalization 181–8
schemas 173–5
unnormal form 182, **183**
database languages 191–4
database programming 191–214
aggregate functions in SQL 206–9
calculations on data 206–9
data integrity 211–14
data updates using SQL 204–6
database development using SQL 193–5
database languages 191–4
queries in SQL 195–204
transactions 211–14
databases 166–72
anomalies 168, 180, 181
atomicity 182, 211
calculations on data 206–9
cardinality 170
cloud-based databases 215–16
composite key 170
concatenated key 170
data consistency **210**, 211, 213
data integrity 211–14
data storage 215–17
denormalization 189–90
durability 212
e-commerce database 185–8
entities 167, 169, 175, **210**
flat file databases 167, 168–9
foreign keys 169, 180
in-memory databases 216–17
indexes 204–5
library database 187–8
medical records databases 167
modality 170
multiuser access to 213
normalization 169, 181–8
NoSQL databases 215
performance overhead 214
platform as a service databases 215–16
primary keys 169, *170*, 180
queries **210**
recovery of data 213
relationships 170–1, 175–6
repeating groups 182
rollback 213
rows 205
scalability challenges 214
schemas 171, 173–5
security 231
social media 167
spatial databases 216
tables *170*, 180, **193–4**, **195**
validation rules 231
views 209–10
virtual tables 209–10
see also database…; distributed databases; relational databases; SQL (structured query language)
DBSCAN *see* density-based spatial clustering of applications with noise
DCU *see* data control language
DDL *see* data definition language
DDoS *see* distributed denial of service
deadlocks 83, 362
debugging 344–8, 464
breakpoint debugging 346
print statements 347
step-by-step code execution 348
trace tables 344–5
decapsulation 149, 150
decimal **39**, **338**
equivalents and conversions 39–42, **43**, 44–5
decision making support 52, 214, 222–3
decision trees 269–71, 273–5, 278, **304**
decode phase, fetch–decode–execute cycle 17, 23
decomposition 323, 330, 331
deep learning (DL) 238, 240, 241, 242, 245, **247**, **248**, **304**
deletion
binary search trees 569
linked lists 554, 558, 563
sets 578
denormalization, databases 189–90
density-based spatial clustering of applications with noise (DBSCAN) 280
dependent (response) variables 265
dequeue method 359, *360*, 551
design patterns, object-oriented programming 521–41
device management 73, 82
DHCP *see* dynamic host configuration protocol
dice function 222
dict, Python 579
difference checks 574, 575, 578
digital certificates **157**, 159–60, 161–2
digital circuit symbols **63**
digital infrastructure 104–8
digital signatures 161
dimensionality reduction 258–63
discrete categorical outcomes 268–75
disinformation 309
disk input/output operations, interrupts and polling 80
Distinct command, SQL **196**
distributed databases 228–34
ACID transactions 233–4
advantages/disadvantages 233
atomic commit protocol 234
concurrency control 230
data consistency 230
data partitioning 231
fault tolerance 232
fragmentation 229
global query processing 232
replication 229, 232–3
security 231
server coordinator 234
transparency 231–2
types of 229
distributed denial of service (DDoS) **156**, **157**
distributed systems 105–6
distributed version control system (DVCS) 567
DL *see* deep learning
DML *see* data manipulation language
documents, digital 162
domain name system (DNS) server 127–8
DORA (discovery, offer, request and acknowledgment) process 129
dot notation 442, 517–21
double loop 519, 521
doubly linked lists 558–63
downsampling 29
drill-down function 222
DSPs *see* data signal processors
duck typing 491–2
duplicate data, removal of 254
DVCS *see* distributed version control system
DVD drives *26*
DVFS *see* dynamic voltage and frequency scaling
dynamic data structure 350–1, 442–6
dynamic host configuration protocol (DHCP) 117–18, 128–9
dynamic IP addresses 118, 146
dynamic polymorphism 484–91
dynamic routing 152–3
static routing comparison **153**
dynamic voltage and frequency scaling (DVFS) 11

e-commerce database 185–8
Eclipse IDE, Java *431*
edge computing 106–7, **247**, **248**
edge devices **248**, 249
educational environment
cloud computing 104
machine learning 246
network segmentation 137
proxy servers 131
school databases 209
switches 113
wireless access points 113
efficiency, scheduling **77**
email 130, **157**
embedded methods, feature selection 258
embedded multimedia cards (eMMCs) 26
embedded systems 80
emerging technologies, ethical issues 310–13
employee training, security testing 158–9
encapsulation 149, 150, 420, 435–72
abstract data types **550**
advantages of 436
one-dimensional structures 442–6
repetition statements with objects 454–5
selection statements with objects 450–4
sorting with objects 464–72
two-dimensional data structures 447–50
variable modifiers 436, **437**, 438, 441
encryption 70, 75, 103, 159–60, 163, 231
enqueue method 359, *360*, 551
entities, databases 167, 169, 170–1, 175, *180*, **210**
entity relationship diagrams (ERDs) 173–4
Barker style *176*
Chen notation style *176*
construction of 177
crows foot style *176*
items within 175–6
enums (enumerated data type) 179

epsilon-greedy strategy 287
ERDs *see* entity relationship diagrams
error detection 51, **94**, **95**, 254
error handling, programming 342–4
errors
  deadlocks 83, 362
  exception errors 342
  incorrect inputs 362
  infinite loops 362
  logic errors 342
  program errors 342
Ethernet 110, *112*, 113
ethical considerations 307–13
  binary search trees 579
  emerging technologies 310–13
  machine learning 307–10
  online communication 309–10
ETL *see* extract, transform and load process
events 12, **79**
exception handling 342–4
execute phase, fetch–decode–execute cycle 17, 23
execution, translation process **92**, **93**
exploration versus exploitation trade-off 286–7, 288
exponential complexity **377**
extends keyword, Java 494–6
external assessment (Papers 1 and 2) 600–10
extract, transform and load (ETL) process 219

F1 score, machine learning models 276–7
factorial algorithms **377**
factorial numbers 404
factory design pattern 527–35
fairness, scheduling **77**, 82–3
fast adder *63*
fault tolerance, databases 232
FCFS *see* first-come, first-served
feature selection 256–8
feedback, control systems **84**
fetch–decode–execute cycle 16–19, 23
Fibonacci sequence 396–7
fibre-optic cables 147, **148**
field-programmable gate arrays (FPGAs) 249
FIFO *see* first in, first out data structure
file processing 407–15
  file reading 409–10, 411
  file writing 407–9, 412
file servers 129–30
file systems 70, 71, 73, 82
filter methods, feature selection 257
filtering
  network security **157**
  queries 195
finally block, error handling 343, 344
firewalls 70, 71, 75, 109, **114**, 154–5
first-come, first-served (FCFS) **76**, **77**, **78**
first in, first out (FIFO) data structure 359–60
first normal form (1NF), databases **183**, 184
flash drives 27
flat file databases 167, 168–9
floppy disks 25
flowcharts 328, **331**, *332–3*
for loops 369–71, 452–3, 454
forecasting 223
foreign keys 169, 180, 220
forward propagation, artificial neural networks **293**, 295, 297
FPGAs *see* field-programmable gate arrays
fragmentation, databases 229
fraud detection 226, 228
From command, SQL **197**
front function 359, *360*
full adder *63*
functions, programming 371–5

GAs *see* genetic algorithms
gateways 108, **114**
gender bias 364
genetic algorithms (GAs) 288–91
genetic sequencing 8, 572
Git (distributed version control system) 567
global query processing 232
graphical user interface (GUI) 74
graphics processing unit (GPU) 5, 6–12, **248**, 249
  architecture 6–7
  central processing unit comparison 8–12
  design philosophy 8
  graphics rendering 5, 7
  high throughput **7**, 8
  instruction set architecture 9
  memory access 10
  parallel processing **6**
  power efficiency 10–11
  processing power 9–10
  scientific computing 8
  simulations and modelling 8
  video editing 8
  video random access memory **7**
Group By command, SQL **199**
group data, clustering techniques 279–82
group permissions 70
guessing game 367–9
GUI *see* graphical user interface

half adder *61*
half subtractor *61*
harassment, online 310
hard disk drives (HDDs) 25, 26, *51*
hash functions 163
hash tables 576–7, 578, 580
hashes 71
hashing 576
HashMap, Java 578–9
hate speech 310
Having command, SQL **200**
HDDs *see* hard disk drives
heat management, CPUs 11
heterogeneous databases 229
hexadecimal number system **39**, 42–3
  equivalents and conversions **43**, 44–6
hierarchical clustering 280
high-performance computing centres (HPC) 250
home environment
  hardware firewalls 109
  network segmentation 137
  routers 112
  security systems 88
  wireless access points 113
homogeneous databases 229
host address 138
HPC *see* high-performance computing centres
HR *see* human resources
HTTP *see* hypertext transfer protocol
HTTPS *see* hypertext transfer protocol secure
human resources (HR) 228
hybrid networks 126–7
hyperparameter tuning 275–8
hypertext transfer protocol (HTTP) 115–17, 131–2
  digital certificates 161
  response codes **116**
  state management 117
hypertext transfer protocol secure (HTTPS) 117
hypervisor 75

I/O (input/output) **67**
IA *see* internal assessment
IaaS *see* infrastructure as a service
idempotent law 64
identity certificates 159–60
IDS *see* intrusion detection systems
if statements 363, 451–4, *477*, *517*
images
  data storage in binary form 48
  spatial hierarchies, machine learning 300–3
IMAP *see* internet message access protocol
in-memory databases 216–17
independent (predictor) variables 265
indexes/indexing
  in databases 204–5
  hash tables for 580
inductive reasoning 396
industrial environment
  gateways 108
  modems 110
infinite loops 362
information literacy 609
infrastructure as a service (IaaS) 34, 35–6
inheritance (in coding) 420, 473–80
  in Java 475–8
  in Python 478–80
inherited relationship, classes 425
input–process–output (IPO) model 84–5
input specifications 319–20
insertion
  binary search trees 569
  linked lists 554, 558, 563
  sets 578
instance variables 418
instantiation 420, 434
instruction register (IR) **4**
instruction set architecture (ISA) 9
integers 47, **338**
  binary 38–9
  binary search trees 570–1
  binary to decimal conversion 39–41
  binary to hexadecimal conversion 45–6
  decimal to binary conversion 41–2
  decimal to hexadecimal conversion 44–5
  hexadecimal 42–3
  hexadecimal to binary conversion 46
  hexadecimal to decimal conversion 43–4
intercept, linear regression 265, 266, 267
internal assessment (IA), computational solution 582–99
internet 104, 115, 131, 205
internet layer, TCP/IP model **119**, 121–2
internet message access protocol (IMAP) 130
internet protocol (IP) 144
internet service providers (ISPs) 112
Internet of Things (IoT) **96**
interpreted programming languages 91–2, **95**, **96**
interpreters 90–7
interrupt handling 78–81
intersection, sets 574, 575, 578
intrusion detection systems (IDS) **157**
intrusion prevention systems (IPS) **158**
involution law 64
IoT *see* Internet of Things
IP *see* internet protocol
IP addresses 75, 103, 117, 127–8, 144–7
  concealment of 131
  dynamic IP addresses 118, 146
  internet protocol 144
  IP masquerading 155
  IPv4 and IPv6 addressing 144, 146
  network address translation 146–7
  private/public IP addresses 145–6, 147
  routers 110, 111–12
  smartphones 129
  static IP addresses 118, 146
  subnetting 137–40
IPO *see* input–process–output model
IPS *see* intrusion prevention systems
IR *see* instruction register
irrigation control systems 88
ISA *see* instruction set architecture
isInstance function, Python 479–80
ISPs *see* internet service providers
item-based collaborative filtering 271, 272–3

Java
  abstract classes 493–7
  aggregation 501–4
  Array 442
  ArrayList 350, 352, 355, 442–4, 447–9, 477–8
  averaging 393
  binary search 383–4, 398–9, 456–60
  breakpoint debugging 346
  bubble sort 387–8, 464–9
  BufferedReader 409–10
  classes 419, *431*, *432*
  comments 450
  compareTo method 464
  composition 506–9
  count occurrences 394
  data structure 350
  data types 337, **338**, 491
  difference check 574
  dot notation 517–19
  dynamic 1D lists 442–4
  dynamic 2D lists 447–9
  dynamic polymorphism 485–8
  encapsulation 438–9
  exception handling 342–3
  extends keyword 494–6
  factory design pattern 529–32
  Fibonacci in 397
  file append 408–9
  file overwrite 408
  file reading 409–10
  file writing 407–9
  for loop 370, 452–3, 454
  guessing game 367–8
  HashMap 578–9
  if statements 451–2, *477*
  inheritance (in coding) 475–8
  instantiation of an object *434*
  intersection, sets 574
  library **375**
  library class code 426–8
  linear search 381–2, 455–6
  linked lists 554–6, 559–61, 564–6
  lists 352–6
  method headers 441
  minimum value 395
  move method 485, 486–8
  multiple classes 511–14, 516
  observer design pattern 537–8
  parameters 373
  print statements 347
  quicksort 402
  random library 374
  recursion in 405
  Runner class 444
  Scanner class 410–11
  selection sort 389–90, 391–2
  selection statements 364, 450–3
  sets **573**, 574
  singleton design pattern 524–6
  sorting 464
  static 1D lists 442
  static polymorphism 482–4
  step-by-step code execution 348
  strings **339–40**
  subset checks 574
  summing 393
  toString method 431, 476–7, 484, 485
  trace tables 345
  two-dimensional lists 354–6
  union, sets 574
  variable modifiers **437**
  variables 491
  while loop 366–7
JavaScript **96**
Join command, SQL **200–1**
JPEG 30, **31**, 50
just-in-time (JIT) compilation **94**

K-means clustering 279
K-Nearest Neighbours (K-NN) 268–9, 271–3, **304**
Karnaugh maps (K-maps) 58–61
kernel size and stride, image classification 303
keyboard, interrupts and polling 79
keys, databases 180

LANs *see* local area networks
large data sets
  association rule learning 282–5
  binary searches 385
last in, first out (LIFO) data structure 358–9
latency 8, 13, **15**, **79**, **155**
leased lines, wide area networks 102
left child property 568
lexical analysis **92–93**
libraries 374, **375**
library class code 426–30
library database 187–8
licence plate recognition systems 106
LIFO *see* last in, first out data structure

Like command, SQL **201–2**
linear regression 264–8, 294–5, **304**
  independent/dependent variable relationship 265
  slope and intercept in regression equation 265–7
linear search algorithm 381–2, 455–6, 460
linear time complexity **377**
linked lists 552–3, **552**, 554–67
  circular linked lists 563–7
  doubly linked lists 558–63
  in operating systems 553
  singly linked lists 554–7
Linux, system log *72*
lists 350, 351–8
  one-dimensional lists 351–4, 442–6
  two-dimensional lists **351**, 354–6, 357, 447–50
  for variables *418*
livelock 83
load factor, hash tables *577*
local area networks (LANs) 101, 151, 152–3
log linear **377**
logarithmic time **377**
logging 71–2, *74*, **155**, 407
logic circuits, truth tables 54–61
logic diagrams 57–8, 61–5
  Boolean algebra 63–5
  combining gates to perform complex logical operations 63
  gate symbols 62
  logic gate connection and interaction in a circuit 61
  processing of inputs to produce outputs 62–3
logic errors 342
logic gates 51–5
  AND gate *51*, 52, **53**
  arithmetic operations 53
  data processing 52
  decision making 52
  memory storage 53
  NAND gate **54**
  NOR gate **54**
  NOT gate **53**
  OR gate 52, **53**
  for a security system *52*
  symbols **53–4**
  XNOR gate **54**
  XOR gate **54**
logical expressions, simplification of 57–8
logical schemas *173*, 174
login, remote 161
logistic regression **304**
looping structures 366–71
  double loop 519, 521
  errors 362
  for loops 369–71, 452–3, 454
  guessing game 367–9
  while loops 366–9
  see also repetition statements
loss function 303
lossless compression 28, 29, **31**
lossy compression 28–9, **31**
loyalty cards 217, 239

MAC address table (forwarding table) 112, 151
MAC filtering 159
MAC (media access control) address 110, 129, 159
machine code/instructions **90**, **91**
machine learning *7*, 236–315
  accountability 310
  accuracy of models 275
  AI servers **248**
  algorithm performance 304–5
  algorithm types 237–8
  application-specific integrated circuits 249
  artificial neural networks 292–300
  association rule learning 282–5
  bias 310
  classification techniques 268–75
  cloud-based platforms 249
  clustering techniques 279–82
  computational thinking 330
  continuous outcomes, prediction of 264–8
  convolutional neural networks 300–3
  data preprocessing 251–63
  data processing **247**, **248**
  data quality impact 252–3
  decision trees 269–71, 273–5, 278, **304**
  deep learning 238, 240, 241, 242, 245, **247**, **248**, **304**
  development and testing **247**, **248**
  dimensionality reduction 258–63
  discrete categorical outcomes 268–75
  and disinformation/misinformation 309
  edge computing **247**, **248**
  edge devices **248**, 249
  ethical implications 307–10
  evaluation of classification models 275–7
  F1 score of models 276–7
  feature selection 256–8
  field-programmable gate arrays 249
  genetic algorithms 288–91
  graphics processing unit **248**, 249
  hardware requirements 247–50
  high-performance computing centres 250
  house sale prices model 259–62
  hyperparameter tuning 275–8
  K-Nearest Neighbours 268–9, 271–3, **304**
  large data sets, association rule learning 282–5
  large-scale deployment **247**, **248**
  linear regression 264–8, **304**
  logistic regression **304**
  market basket analysis 239
  medical imaging diagnostics 240
  model selection and comparison 304–5
  model training **247**, **248**
  multi-layer network modelling 292–300
  natural language processing 240–1, 246
  neural networks **304**
  object detection and classification 242–3
  online communication 309–10
  overfitting of models 277–8
  precision of models 276
  privacy concerns 310
  random forest **304**
  recall of models 276
  reinforcement learning 238, 243, 285–8
  robotics navigation 243–4
  security 308
  sentiment analysis 244–5
  societal impact 313
  spatial hierarchies in images 300–3
  supervised learning 238, 240, 241, 242, 244
    hyperparameter tuning 275–8
  support vector machines **304**
  tensor processing units 249
  training data 308–9
  transfer learning 238, 240, 241, 245
  underfitting of models 278
  unsupervised learning 238, 239
    clustering techniques 279–82
  variability in algorithm performance 305
  workflow *251*
  see also artificial intelligence
mail servers 130
malware **155**, **156**, **157**
man-in-the-middle (MitM) attacks **156**
management reporting 222
MAR *see* memory address register
market basket analysis 239
market segmentation 281
marketing 222, 224, 228, 282
maximum value **208**, 395–6
MDR *see* memory data register
mean shift clustering 281
measurement bias 227
media access control (MAC) address 110, 129
medical diagnostics 240, 274–5
medical records databases 167
memory 12–15, 20–1, 24–7, **67**, 70, 82, 261
memory access 10, 17
memory address **16**
memory address register (MAR) **4**
memory cards 27
memory controller 14, 20
memory data register (MDR) **4**
memory management 72–3, 82
memory storage, logic gates 53
memory usage, data structure 351
mesh topologies 125
method headers 441
method overriding, object-oriented programming 480–92
MFA *see* multi-factor authentication
Microsoft Office 365 33
Microsoft Outlook 135
Microsoft Teams 32
minimum value **208**, 395–6
misinformation 309
MitM *see* man-in-the-middle attacks
MLP *see* multi-layer perceptron
mobile networks 107
modality, entities 170
modems 109–10, **114**
modifiers, variables 436, **437**, 438, 441, 475
modularity, abstract data types **550**
modularization 371–5
Moodle (learning management system) 33
mouse *79*, 80–1
move method
  in Java 485, 486–8
  in Python 488–91
multi-core architecture 22, 23
multi-core processors 5, 22–4
multi-factor authentication (MFA) **158**
multi-layer networks 292–300
multi-layer perceptron (MLP) *300*
multicasting 115
multilevel queue scheduling **76**, **77**, **78**
multiple classes, coding for 511–21
multitasking 5, 81–2
MySQL, data types 177–9

NAND flash memory 25
NAND gate **54**
NAS *see* networked attached storage devices
natural language processing (NLP) 240–1, 246
network address translation (NAT) 146–7, 155
network communications, interrupts and polling 80
network devices 108–14, **114**, 121
network interface cards (NICs) 110, **114**, 121, 159
network interface layer **119**, 121, 122
network protocols 108, 115–18, 131–2
network routing 568
network security 154–64, 331
  blacklists 154, 159
  common vulnerabilities **156**
  content security policies **157**
  digital certificates **157**, 159–62
  distributed denial of service **156**, **157**
  employee training 158–9
  encryption **157**, 159–60, 163
  firewalls *70*, 71, 75, 109, **114**, 154–5
  input validation **157**
  intrusion detection systems **157**
  intrusion prevention systems **158**
  malware **155**, **156**, **157**
  multi-factor authentication **158**
  network address translation 155
  password policies **157**
  phishing attacks **156**, 158
  public key certificates 159–60
  security testing 158–9
  software updates **157**
  virtual private networks **158**
  wireless security measures 159
  see also security
network segmentation 136–43
  classless inter-domain routing notation 141–3
  methods of 137–43
  security 137
  subnetting 137–40
  virtual local area networks 140–1
network stack **67**
network topologies 124–7
networked attached storage (NAS) devices 27
networking 75, 360
  client–server model 134–5
  peer-to-peer model 135–6
networks 100–65
  architecture 124–43
  characteristics of 101–3
  cloud computing 104–5
  congestion reduction 151
  data transmission 144–53
  devices 108–14
  digital infrastructure 104–8
  distributed systems 105–6
  edge computing 106–7
  firewalls 109, 154–5
  internet 104
  local area networks 101, 151
  mobile networks 107
  personal area networks 102–3
  protocols 108, 115–18, **156**
  security 154–64
  segmentation 136–43
  servers 127–33
  topologies 124–7
  virtual private networks 103
  wide area networks 102
  see also network...
neural networks 238, 240, 241, **304**
  see also artificial neural networks; convolutional neural networks
nibble (unit of data) **39**
NICs *see* network interface cards
NLP *see* natural language processing
nodes
  artificial neural networks **292**, 293, 294, 295, 299–300
  binary search trees 568
  distributed systems 105
  singly linked lists 554
non-playing character (NPC) **325**, 417
non-static variables 425–30
non-volatile memory 12
NOR gate **54**
normalization
  data preprocessing 255–6
  databases 169, 181–8
NoSQL databases 215
Not command, SQL **204**
NOT gate **53**
NPC *see* non-playing character
numerical data types **178**, 179

object-oriented programming (OOP) 416–547
  abstraction 492–500
  advantages/disadvantages 421
  aggregation 501–5
  classes 417, 418–19, 420, 422–5
    design of 431–3
  code reusability 473–80
  composition 506–10
  design patterns 521–41
  encapsulation 420, 435–72
  factory design pattern 527–35
  inheritance (in coding) 420, 473–80
  library class code 426–30
  method overriding 480–92
  multiple classes, coding for 511–21
  observer design pattern 535–40
  one-dimensional data structures 442–6
  polymorphism 420–1, 480–92
  repetition statements with objects 454–5
  searching with objects 455–63
  selection statements with objects 450–4
  for a single class 417–72
  singleton design pattern 523–7
  sorting with objects 464–72
  static and non-static variables and methods 425–30
  tools 420–1

two-dimensional data structures 447–50
UML class diagrams 423–5, 426, *431*, *437*, 438, *506*
objects 417, 420, **422**
bubble sort 464–72
composition 506–10
data types 337
detection and classification 242–3
duck typing 491–2
instantiation of 434
observer design pattern 535–40, *536*
one-dimensional (1D) lists 351–4, 442–6
online analytical processing (OLAP) 220–3
online communication, machine learning 309–10
OOP *see* object-oriented programming
opcodes (operation codes) 4, 17
open addressing 577
operating systems (OS) 66–83
abstraction 68
access control and permissions 70
accounting 74
antivirus software 71
command-line tools 75
CPU scheduling 70
data integrity checks 71
deadlock 83
device management 73, 82
file systems 71, 73, 82
functions of 68–75
graphical user interface 74
hashes 71
interrupts and polling 78–81
linked lists 553
logging 71–2
memory management 72–3
memory protection 70
multitasking 81–2
networking 75
patch management 71
performance monitoring 72
process isolation 70
process management **67**, 68–9
resource allocation 81, 82
resource contention 83
resource monitoring/limits 82
resources managed by 67–8
rollback features 71
scheduling 73, 76–8
security 70, 71, 75
system integrity 69
system resources **67**
task management 72
task scheduling 82–3
update management 71
user and group permissions 70
virtualization 74–5
optical drives 26–7
Or command, SQL **203**
OR gate 52, **53**
Order By command, SQL **198–9**
OS *see* operating systems
outliers, management of 254, 256
output, control systems **84**
output specifications 320–1
overfitting 227, 260, 277–8
overwriting, file code 408

P2P *see* peer-to-peer model
PaaS *see* platform as a service
packets 70, 75, 110
data segmentation 149–51
routers 111–12
PANs *see* personal area networks
parallel processing 5, **6**, 7, 10, 23
parameters 372–3, 441
parsers **90**
parsing, stacks 359
passwords **157**
patch management, software 71, **157**
pattern recognition
artificial neural networks 296–8
computational thinking 323, 325–6, 330, 331
PC *see* program counter
peek method, stacks 358, 359
peer-to-peer (P2P) model 135
perceptrons **292**, 293, 294, 295, 299–300
performance-critical applications **96**
personal area networks (PANs) 102–3
personalized marketing 282
pervasive AI 312
phishing attacks **156**, 158
physical schemas *173*, 175
pipelining 21–4
pixels 48, 50
plagiarism vii
platform as a service (PaaS) 33–4, 35, **36**
databases 215–16
playlists 360, **381**, 501–5, 569
polling 78–81, **79**
polymorphism, in classes 420–1
polymorphism (in coding) 480–92
dynamic polymorphism 484–91
static polymorphism 481–4
pop method, stacks 358, 359
portability, software **94**, **95**
power efficiency 10–11
pre-emption, scheduling **77**, 83
pricing strategies 282
primary keys 169, *170*, 180
primary memory 12–15
primitive data types 337
print statements 347
printer queues 360
priority scheduling **76**, **77**, **78**
privacy, AI systems 308, 310
private IP addresses 145–6, **145**, 147
private modifiers, variables 436, **437**, 438, 441, 475
private-key cryptography 162
problem specification 317–23
problem statement 317–18
process 68–9, **84**
process isolation 70
process management **67**, 68–9, 222, 553
process scheduling *see* scheduling
processing power 9–10
processing speed **15**
program counter (PC) **4**
programming 336–415
accessing 517
algorithms 376–406
arrays 351–8
averaging 393–4
Big O notation 376–80
binary search 382–5, 398–400
bubble sort 385, 386–8
code sequence 362
count occurrences 394–5
data structures 350–61
data types 337, **338**
deadlocks 362
debugging techniques 344–8
dynamic data structure 350–1
exception handling 342–4
file-processing operations 407–15
first in, first out data structure 359–60
flowcharts 328, **331**, *332–3*
functions 371–5
incorrect inputs 362
infinite loops 362
last in, first out data structure 358–9
libraries 374, **375**
linear search 381–2
lists 350, 351–8
looping structures 366–71
maximum/minimum value 395–6
modules 371–5
notation **63**
parameters 372–3
queues 359–60
recursion 396–406
repetition 366–71
scalability of algorithms 376–80
search for data 380–5
selection sort 385, 388–92
selection statements 363–6
sequence of code 362
sorting 385
static data structure 350–1
strings 337, **338**, **339–40**
summing 393
variables 337–8
see also database programming; object-oriented programming
programming languages 90–7, **90**
applicability **95**
compiled programming languages 93, **94**, **96**
interpreted programming languages 91–2, **95**, **96**
programs 68, 342
protected modifiers, variables 436, **437**, 441, 475
protocols
dynamic routing protocols 153
encrypted protocols **157**
networks 108, 115–18, **156**
proxy servers 131
public IP addresses 145, 146, 147
public key certificates 159–60
public-key cryptography 160, 162
public modifiers, variables 436, **437**, 441, 475
push method, stacks 358, 359
Python
abstract classes 497–9
aggregation 504–5
averaging 394
binary search 384–5, 399–400, 460–3
bubble sort 388, 469–72
classes 419, 433
comments 450
composition 509–10
count occurrences 395
data structure 350
data types 337, **338**, 491
dict 579
difference checks 575
dot notation 519–21
duck typing 491–2
dynamic 1D lists 445–6
dynamic polymorphism 488–91
encapsulation 439–40
exception handling 343, 344
factory design pattern 532–5
Fibonacci in 397
file reading and writing 411–12
for loop 370, 371
guessing game 368–9
if statements 453–4
inheritance (in coding) 478–80
instantiation of an object *434*
intersection, sets 575
isInstance function 479–80
library **375**
library class code 428–30
linear search 382, 460
linked lists 557, 561–3, 566–7
lists 350, 352, 353, 357
method headers 441
minimum value 395
move method 488–91
multiple classes 514–16
observer design pattern 539–40
parameters 373
quicksort 403
random library 374
recursion in 405
selection sort 390, 392
selection statements 365, 453–4
sets 573, 575
singleton design pattern 526–7
sorting 464
static polymorphism 484
strings 337, **340**
subset checks 575
summing 393
trace tables 345
translation steps **92**
tuples 357
two-dimensional lists 357
union, sets 575
variable modifiers **437**
variables 338
while loop 367

quadratic algorithms **377**
Quality of Service (QoS) rules 109
quantization 48–9
quantum computing 311
quantum, scheduling **77**
queries, in SQL *see* SQL (structured query language), queries
queues 359–60, 551
quicksort 400–3

racial bias 364
random-access memory (RAM) 13, 14, **16**, 216
random forests **304**
random library 374
read-only memory (ROM) 13, 14
real-time systems 80
recall (sensitivity), machine learning models 276
recovery of data 213
recurrent neural networks (RNNs) 241
recursion 396–406
binary search 398–400
quicksort 400–3
registers 4, *5*, 12, 13, 20–1
regression, data mining 225
reinforcement learning (RL) 238, 243, 285–8
relational databases 169–72, **210**
benefits and limitations 171
data duplication problem 180
data types used in 177–9
table construction 180
relationships, entities 170–1, 175–6
rendering 5, 7
repeating groups, databases 182
repetition 366–71
for loops 369–71, 454
while loops 366–9
replication, databases 229, 232–3
resource allocation 81, 82
resource contention 83
resource management 70
resource monitoring/limits 82
response codes, HTTP **116**
retail environment
databases 167
wireless access points 113
RGB colour model 48
right child property 568
risk mitigation, AI systems 313
RL *see* reinforcement learning
RLE *see* run-length encoding
RNNs *see* recurrent neural networks
robotic vacuum cleaners **285**
robotics navigation 243–4
roll-up function 222
rollback 71, 213
ROM *see* read-only memory
round robin, scheduling **76**, **77**, **78**
route planning 289–91
routers/routing 110–12, **114**, 151
algorithms 127
dynamic routing 152–3
static routing 152, **153**
subnetting 140
routing table (data file) 151, 152
rows, in databases 205
run-length encoding (RLE) 28, 30, **31**
runner class 440, 444
runtime polymorphism 484–91
Rust, translation steps **93**

SaaS *see* software as a service
sales reporting 222
sampling 48, *49*
sandboxing 68
Savepoint command, SQL 213
Scanner class, Java 410–11
scheduling 70, 73, 76–8, 82–3
first-come, first-served **76**, **77**, **78**
linked lists 553
multilevel queue scheduling **76**, **77**, **78**
priority scheduling **76**, **77**, **78**
round robin **76**, **77**, **78**
schemas, databases 171, 173–5, **210**
scheme 117
scientific computing 8

search/searching
  binary search 382–5, 398–400, 456–63
  binary search trees 569
  linear search 381–2, 455–6, 460
  linked lists 554, 558, 564
second normal form (2NF), databases **183**, 184
secondary memory storage 24–7
secure socket layer (SSL) certificate **157**
security
  AI systems 308
  application-level security **155**
  data security 231
  data transmission **148**
  databases 171
  in distributed systems 106
  edge computing 107
  employee training 158–9
  file systems 70
  internal threats **155**
  interrupts and polling **79**
  machine learning 308
  mail servers 130
  mobile networks 107
  network segmentation 137
  operating systems **67**, 70, 71, 73, 75
  sequential pattern discovery 228
  web interactions 117
  see also network security
security system, logic gates for *52*
security testing 158–9
segmentation 82
  data segmentation 149–51
  network segmentation 136–43
Select command, SQL **196**
selection bias 227
selection sort 385, 388–92
selection statements 363–6
semantic analysis **92**, **93**
semantics **90**
sensors **86**
sentiment analysis 244–5
separate chaining 577
sequential pattern discovery 226–8
servers 127–33
  distributed databases 234
  domain name system server 127–8
  dynamic host configuration protocol server 128–9
  file servers 129–30
  mail servers 130
  proxy servers 131
  web servers 131–2
sets 572–6
  implementation of 578
  mechanics of 577–8
  set operations **573**, 578
signatures, digital 161
signed integers 47
SIMD instructions 9, 10
simulations 8
single-core processors 5, *22*
singleton design pattern 523–7
singly linked lists 554–7
Skype 32, 136
SL *see* supervised learning
slice function 222
slice, scheduling **77**
slope, linear regression 265–6, 267
smartphones 129, 324, *480*
social media 167, 223, 227, 311, 576
society, and AI systems 308, 313
software
  digital signatures 161
  firewalls 109
  network security **156**, **157**
  patch management 71, **157**
  portability **95**
software engineering **96**, 330, 522, 550, 580
software as a service (SaaS) 31–3, 35, **36**
solid state drives (SSDs) 25–6
sort/sorting
  bubble sort 385, 386–8, 464–72
  definition 385
  selection sort 385, 388–92
sound waves *48*, *49*
spam email 130, **157**
spatial databases 216
SQL (structured query language)
  adding and modifying records in a database **205–6**
  aggregate functions 206–9
    Average **206–7**
    Count **207**
    Maximum value **208**
    Minimum value **208**
    Sum **209**
  data language types 191–4
  database development 193–5
    alter a table **194**
    create a database **193**
    create a table **193–4**
    delete a table **194**
    modify data in a table **195**
  malicious SQL statements **156**
  queries 195–204, **210**
    And command **202**
    Between command **198**
    Distinct command **196**
    filtering 195
    From command **197**
    Group By command **199**
    Having command **200**
    Join command **200–1**
    Like command **201–2**
    Not command **204**
    Or command **203**
    Order By command **198–9**
    pattern matching 195
    Select command **196**
    Where command **197**
  transaction control language
    Commit command 213
    Rollback command 213
    Savepoint command 213
    Start Transaction command 213
  update operations 204–6
  virtual tables in databases 209–10
SSDs *see* solid state drives
SSH (Secure Shell) 161
SSL *see* secure socket layer certificate
stacks 358–9
standard library **90**
standardization, data preprocessing 256
star topologies 124–5, 126
Start Transaction command, SQL 213
starvation (indefinite blocking) **77**, 82, 83
static data structure 350–1, 442
static IP addresses 118, 146
static polymorphism 481–4
static routing 152, **153**
static variables 425–30
step-by-step code execution 348
strings 47–8, 337, **338**, **339–40**
  data types **178**, 179
  reversing 359
structured query language *see* SQL
study skills 608–10
subclasses 473
  abstract classes *493*, 494–6, 498–9
  factory design pattern 529, 531, 533
  move method 486–91
  variables and methods 475, 476–9
subnet masks 118, 137–138, 141, 142, 143
subnetting 137–40
subset checks 574, 575
Sum function, SQL **209**
summing 393
superclasses 473, *474*
  move method 488–9
  toString method 485
  variables and methods 475–6, 478
supervised learning (SL) 238, 240, 241, 242, 244
  classification techniques 268–75
  hyperparameter tuning 275–8
support vector machines (SVM) **304**
SurveyMonkey 32
switches 112–13, **114**, 151
symmetric cryptography 159–60, **160–1**
syntax analysis **92**, **93**
synthetic data 266
system bus 20
system integrity 69
system logs 71–2, 74, 407

tables, databases *170*, 180, **193–4**, **195**
task management **67**, *72*, 553
task scheduling *see* scheduling
TCL *see* transaction control language
TCP *see* transmission control protocol
TCP/IP model *75*, 118–23
  network device mapping to layers **114**
  role of layers **119**
telecommunications, modems 110
temporal locality 14
tensor processing units (TPUs) 249
thermostats 87
third normal form (3NF), databases **184**, 185–8
Thompson sampling 287
thrashing 83
threads, process 69
time period data 221
TL *see* transfer learning
topologies *see* network topologies
toString method, Java 431, 476–7, 484, 485
TPUs *see* tensor processing units
trace tables 344–5
  factorial numbers **404**
  sum of all natural numbers **405**
traffic monitoring, firewalls **155**
traffic signal control systems 88, 106
training data 224, 238, 307, 308–9
transaction control language (TCL) 192, 212, 213
transactions, databases 211–14
transducers **86**
transfer learning (TL) 238, 240, 241, 245
transform coding 30, **31**
transformers, deep learning 241
transistors 51
translation, interpreters and compilers **90**, **92**
  bytecode interpreters 95
  compiled programming languages 93, **94**, **96**
  interpreted programming languages 91–2, **95**, **96**
  just-in-time compilation **94**
  uses of **96**
translation time **94**, **95**
transmission control protocol (TCP) 115
transmission control protocol/internet protocol (TCP/IP) model **114**, 118–23
transparency
  AI systems 308, 313
  distributed databases 231–2
transport layer, TCP/IP model **119**, 120, 122–123
travelling salesperson problem (TSP) 289–91
  genetic algorithm for **291**
traversal, linked lists 554, 558, 564
truth tables **53–4**, 54–61
  Boolean expressions with inputs and outputs 56–7
  derived from logic diagrams 57–8
  determining outputs from inputs for a problem description 54–6
  simplifying output expressions with Boolean algebra and K-maps 58–61
try statement, error handling 342–3
TSP *see* travelling salesperson problem
tuples, Python 357
twisted pair cables 147, **148**
two-dimensional (2D) lists **351**, 354–6, 357, 447–50
two's complement, signed integers 47

UCB *see* upper confidence bound
UDP *see* user datagram protocol
UI *see* user interface
UL *see* unsupervised learning
UML class diagrams 423–5, 426
  composition relationships *506*
  design of classes *431*
  encapsulated class *437*, 438
  if statements *517*
  singleton class *524*
  subclass use with an abstract class *493*
  for a superclass *474*
underfitting, machine learning models 278
undo functions 359
Unicode 47, 50
unified memory architecture 10, 11
union, sets 574, 575, 578
universal modelling language (UML) 423
unnormal form (UNF), databases 182, **183**
unsigned integers 47
unsupervised learning (UL) 238, 239
  clustering techniques 279–82
update management/operations 71, 204–6
upper confidence bound (UCB) 287
USB flash drives 27
user-based collaborative filtering 271–2
user datagram protocol (UDP) 115
user interface (UI) **67**
user mode (user space) 70
user permissions 70

validation rules, databases 231
variables 337–8, *418*, 491
  casting 441
  instance variables 418
  modifiers 436, **437**, 438, 441, 475
  static and non-static variables 425–30
  subclasses 475, 476–9
  superclasses 475–6, 478
video, binary encoding of 49–50
video editing 8
video games *7*, *93*, **325**, 548, *549*
video random access memory (VRAM) **7**
views (virtual tables) 209–10
virtual local area networks (VLANs) 140–1
virtual machines (VMs) 34, 74, 75
virtual memory 14, *72*, *73*, 82
virtual private networks (VPNs) 103, **158**, 162
virtual reality (VR) 312
virtual tables, databases 209–10
virtualization 34, *74–5*
VLANs *see* virtual local area networks
VMs *see* virtual machines
vocabulary skills 609
Voice over Internet Protocol (VoIP) 108
volatile memory 12
VPNs *see* virtual private networks
VR *see* virtual reality
VRAM *see* video random access memory

WANs *see* wide area networks
WAPs *see* wireless access points
washing machines 87–8
web servers 131–2
website navigation 227
Where command, SQL **197**
while loops 366–9
whitelists **157**, 159
wide area networks (WANs) 102
Windows, Task Manager *553*
wireless access points (WAPs) 113, **114**
wireless network mesh 113
wireless technology 147, **148**, 159
worldwide web (WWW) 104, 115, 117
wrapper methods, feature selection 257–8
write signal 20
writeback phase 23

XNOR gate **54**
XOR gate **54**
XSS *see* cross-site scripting

zero-day exploits **156**
Zoom 32

Oxford Resources for IB
Diploma Programme

2025 EDITION

COMPUTER SCIENCE

COURSE COMPANION

Written by expert and experienced practitioners, and developed in cooperation with the IB, this DP Computer Science course book provides:

- A comprehensive and accurate match to the latest IB DP Computer Science specification, delivering in-depth coverage of all content for both SL and HL
- Fully accessible code in both Python and Java, and advice for learners new to coding
- Engaging real world examples, activities and questions, to apply computational thinking and programming skills
- Thorough preparation for IB assessment, with dedicated sections for the internal and external assessments, plus exam-style practice questions at the end of each topic
- Complete alignment with the IB philosophy, with ATL skills and Theory of Knowledge support woven throughout.

To enhance your teaching and learning experience, use this course book alongside your Kerboodle course. Kerboodle is a digital learning platform that works alongside your print textbooks to create a supportive learning environment and to enable success in DP and beyond.

Also available in this series:



9781382063975

FOR FIRST ASSESSMENT IN 2027

OXFORD UNIVERSITY PRESS

www.oup.com

How to get in touch:
web www.oxfordsecondary.com/ib
email schools.enquiries.uk@oup.com
tel +44 (0)1536 452620

ISBN 978 1 382 06392 0
9 781382 063920