Endorsed for
Pearson Edexcel
Qualifications

PEARSON EDEXCEL INTERNATIONAL GCSE (9–1)

COMPUTER SCIENCE

Student Book

David Waller, Chris Charles, Pete Dring, Alex Hadwen-Bennett, Jason Welch, Shaun Whorton
Series Editor: Ann Weidmann

eBook
included

PEARSON EDEXCEL INTERNATIONAL GCSE (9–1)

COMPUTER SCIENCE

Student Book

David Waller
Chris Charles
Pete Dring
Alex Hadwen-Bennett
Jason Welch
Shaun Whorton

SERIES EDITOR

Ann Weidmann

Published by Pearson Education Limited, 80 Strand, London, WC2R 0RL.

https://www.pearson.com/international-schools

Copies of official specifications for all Pearson Edexcel qualifications may be found on the website: https://qualifications.pearson.com

Text © Pearson Education Limited 2020
Designed by Pearson Education Limited 2020
Typeset by © SPi Global
Edited by Hillary Coaster and Sarah Wright
Original illustrations © Pearson Education Limited 2020
Picture research by Integra
Cover design © Pearson Education Limited 2020
With thanks to Sam Hartburn

The right of David Waller, Chris Charles, Pete Dring, Alex Hadwen-Bennett, Jason Welch, Shaun Whorton to be identified as the authors of this work has been asserted by him in accordance with the Copyright, Designs and Patents Act 1988.

First published 2020

22 21 20
10 9 8 7 6 5 4 3 2 1

British Library Cataloguing in Publication Data
A catalogue record for this book is available from the British Library

ISBN 978 1 292 31022 0

Printed in Slovakia by Neografia

Acknowledgements

Text Credits
Loon LLC: Extracts from Loon LLC, © Loon LLC, 256; **McKinsey & Company:** Stefan Heck, Sri Kasa, Dickon Pinner, Creating Value in the Semiconductor industry, © 2011, McKinsey & Company, 264.

Image Credits
(key: b-bottom; c-center; l-left; r-right; t-top)

Cover Image: Shutterstock/PopTika

123RF: Anton Starikov/123RF 160bc, Jan Mikš/123RF 54, Kbuntu/123RF 214, Lajo_2/123RF 71, Scyther5/123RF 123, Scanrail/123RF 238, Stockyimages/123RF 69; **Alamy Stock Photo:** Jonathan Ball/Alamy Stock Photo 168r, Martin Moxter/imageBROKER/Alamy Stock Photo 248, Paul Broadbent/Alamy Stock Photo 168l, Richard Levine/Alamy Stock Photo 52t, Stuart Kinlough/Alamy Stock Photo 2, 32, 108, 158, 200, 242; **Python Software Foundation:** Copyright ©2001-2019. Python Software Foundation 51, 94, 96, 101; **Shutterstock:** Adisa/Shutterstock 160tl, Asharkyu/Shutterstock 245, Anton Gvozdikov/Shutterstock 265, Burlingham/Shutterstock 34, Baloncici/Shutterstock 106, vii, Bloomicon/Shutterstock 160br, Brian A Jackson/Shutterstock 175, 226, Brainpencil/Shutterstock 181, Cristi180884/Shutterstock 125, Chrisdorney/Shutterstock 166, Crisp/Shutterstock 185, Cybrain/Shutterstock 267t, Cobalt88/Shutterstock 234, DavidTB/Shutterstock 24, Dja65/Shutterstock 160tc, D-VISIONS/Shutterstock 163, Dny3d/Shutterstock 234, ESB Professional/Shutterstock 85, Evgeny Karandaev/Shutterstock 211, Gaby Kooijman/Shutterstock 52b, Gorodenkoff/Shutterstock 172, GraphicINmotion/Shutterstock 268t, Hxdbzxy/Shutterstock 258, Jerry Zitterman/Shutterstock 37, Joshua Davenport/Shutterstock 7, Kavione/Shutterstock 160tr, Luisa Leal Photography/Shutterstock 160bl, Maksym Kapliuk/Shutterstock 25, Michaeljung/Shutterstock 65, Mmaxer/Shutterstock 219, Moomchak V. Design/Shutterstock 268b, Nito/Shutterstock 17, Nikita G. Bernadsky/Shutterstock 266b, Nobeastsofierce/Shutterstock 267b, OlegDoroshin/Shutterstock 5, vii, Platslee/Shutterstock 129, 142, Pakhnyushchy/Shutterstock 160cl, Phonlamai Photo/Shutterstock 224, Paul Fleet/Shutterstock 266t, Silverkblackstock/Shutterstock 184, Shutterstock 93, Thinglass/Shutterstock 248, Trong Nguyen/Shutterstock 238, TY Lim/Shutterstock 4, vii, Vlad Kochelaevskiy/Shutterstock 177, 187, Volodymyr Krasyuk/Shutterstock 235, Wright Studio/Shutterstock 144, Xfilephotos/Shutterstock 237.

Endorsement statement
In order to ensure that this resource offers high-quality support for the associated Pearson qualification, it has been through a review process by the awarding body. This process confirmed that this resource fully covers the teaching and learning content of the specification at which it is aimed. It also confirms that it demonstrates an appropriate balance between the development of subject skills, knowledge and understanding, in addition to preparation for assessment.

Endorsement does not cover any guidance on assessment activities or processes (e.g. practice questions or advice on how to answer assessment questions) included in the resource, nor does it prescribe any particular approach to the teaching or delivery of a related course.

While the publishers have made every attempt to ensure that advice on the qualification and its assessment is accurate, the official specification and associated assessment guidance materials are the only authoritative source of information and should always be referred to for definitive guidance.

Pearson examiners have not contributed to any sections in this resource relevant to examination papers for which they have responsibility.

Examiners will not use endorsed resources as a source of material for any assessment set by Pearson. Endorsement of a resource does not mean that the resource is required to achieve this Pearson qualification, nor does it mean that it is the only suitable material available to support the qualification, and any resource lists produced by the awarding body shall include this and other appropriate resources.

PROBLEM SOLVING 2

PROGRAMMING 32

DATA 108

COMPUTERS 158

COMMUNICATION AND THE INTERNET 200

THE BIGGER PICTURE 242

EXAM PREPARATION 272

APPENDICES 291

ABOUT THIS BOOK

This book is written for students following the Pearson Edexcel International GCSE (9–1) Computer Science specification and covers both years of the course.

The course has been structured so that teaching and learning can take place in any order, both in the classroom and in any independent learning. The book contains six units that match the six areas of content in the specification: Problem Solving, Programming, Data, Computers, Communication and the Internet, The Bigger Picture.

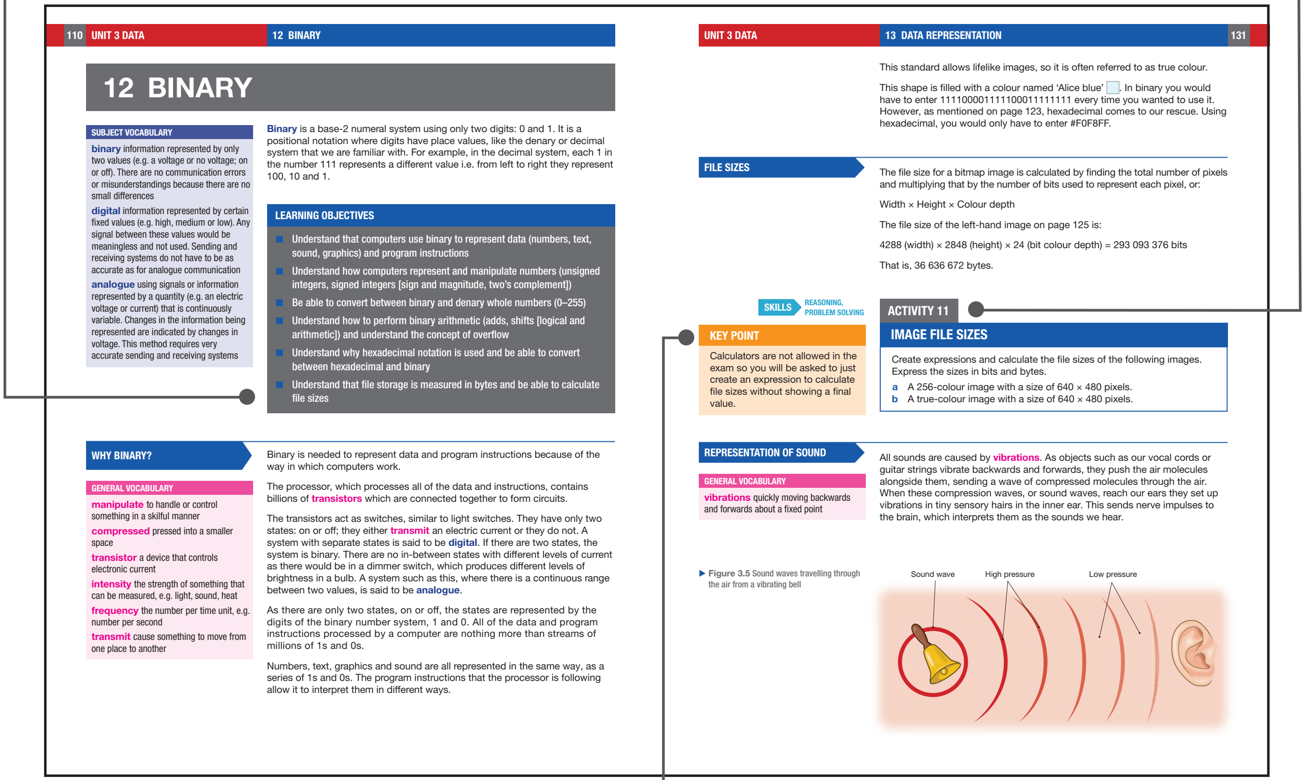
Each unit is split into multiple sections to break down content into manageable chunks and to ensure full coverage of the specification.

Each unit features a mix of learning and activities. Summary questions at the end of each chapter help you to put learning into practice and prepare for the exam.

Paper 1 is Principles of Computer Science and Paper 2 is Application of Computational Thinking. Knowing how to apply your learning to both of these will be critical for your success in the exam. There is a real applied focus to the book. You will be encouraged to put the theory you are learning into context and apply what you have learned to your own practical activities.

Learning objectives
Each section starts with a list of what you will learn from it. They are carefully tailored to address key assessment objectives central to the course.

Activity
Each chapter includes activities to embed understanding through practical tasks and questions.



110 UNIT 3 DATA 12 BINARY

12 BINARY

SUBJECT VOCABULARY

binary information represented by only two values (e.g. a voltage or no voltage; on or off). There are no communication errors or misunderstandings because there are no small differences

digital information represented by certain fixed values (e.g. high, medium or low). Any signal between these values would be meaningless and not used. Sending and receiving systems do not have to be as accurate as for analogue communication

analogue using signals or information represented by a quantity (e.g. an electric voltage or current) that is continuously variable. Changes in the information being represented are indicated by changes in voltage. This method requires very accurate sending and receiving systems

Binary is a base-2 numeral system using only two digits: 0 and 1. It is a positional notation where digits have place values, like the denary or decimal system that we are familiar with. For example, in the decimal system, each 1 in the number 111 represents a different value i.e. from left to right they represent 100, 10 and 1.

LEARNING OBJECTIVES

- Understand that computers use binary to represent data (numbers, text, sound, graphics) and program instructions
- Understand how computers represent and manipulate numbers (unsigned integers, signed integers [sign and magnitude, two's complement])
- Be able to convert between binary and denary whole numbers (0–255)
- Understand how to perform binary arithmetic (adds, shifts [logical and arithmetic]) and understand the concept of overflow
- Understand why hexadecimal notation is used and be able to convert between hexadecimal and binary
- Understand that file storage is measured in bytes and be able to calculate file sizes

WHY BINARY?

GENERAL VOCABULARY

manipulate to handle or control something in a skilful manner

compressed pressed into a smaller space

transistor a device that controls electronic current

intensity the strength of something that can be measured, e.g. light, sound, heat

frequency the number per time unit, e.g. number per second

transmit cause something to move from one place to another

Binary is needed to represent data and program instructions because of the way in which computers work.

The processor, which processes all of the data and instructions, contains billions of **transistors** which are connected together to form circuits.

The transistors act as switches, similar to light switches. They have only two states: on or off; they either **transmit** an electric current or they do not. A system with separate states is said to be **digital**. If there are two states, the system is binary. There are no in-between states with different levels of current as there would be in a dimmer switch, which produces different levels of brightness in a bulb. A system such as this, where there is a continuous range between two values, is said to be **analogue**.

As there are only two states, on or off, the states are represented by the digits of the binary number system, 1 and 0. All of the data and program instructions processed by a computer are nothing more than streams of millions of 1s and 0s.

Numbers, text, graphics and sound are all represented in the same way, as a series of 1s and 0s. The program instructions that the processor is following allow it to interpret them in different ways.

UNIT 3 DATA 13 DATA REPRESENTATION 131

This standard allows lifelike images, so it is often referred to as true colour.

This shape is filled with a colour named 'Alice blue' □. In binary you would have to enter 111100001111100011111111 every time you wanted to use it. However, as mentioned on page 123, hexadecimal comes to our rescue. Using hexadecimal, you would only have to enter #F0F8FF.

FILE SIZES

The file size for a bitmap image is calculated by finding the total number of pixels and multiplying that by the number of bits used to represent each pixel, or:

Width × Height × Colour depth

The file size of the left-hand image on page 125 is:

4288 (width) × 2848 (height) × 24 (bit colour depth) = 293 093 376 bits

That is, 36 636 672 bytes.

SKILLS REASONING, PROBLEM SOLVING

KEY POINT

Calculators are not allowed in the exam so you will be asked to just create an expression to calculate file sizes without showing a final value.

ACTIVITY 11

IMAGE FILE SIZES

Create expressions and calculate the file sizes of the following images. Express the sizes in bits and bytes.

a A 256-colour image with a size of 640 × 480 pixels.
b A true-colour image with a size of 640 × 480 pixels.

REPRESENTATION OF SOUND

GENERAL VOCABULARY

vibrations quickly moving backwards and forwards about a fixed point

All sounds are caused by **vibrations**. As objects such as our vocal cords or guitar strings vibrate backwards and forwards, they push the air molecules alongside them, sending a wave of compressed molecules through the air. When these compression waves, or sound waves, reach our ears they set up vibrations in tiny sensory hairs in the inner ear. This sends nerve impulses to the brain, which interprets them as the sounds we hear.

▶ **Figure 3.5** Sound waves travelling through the air from a vibrating bell

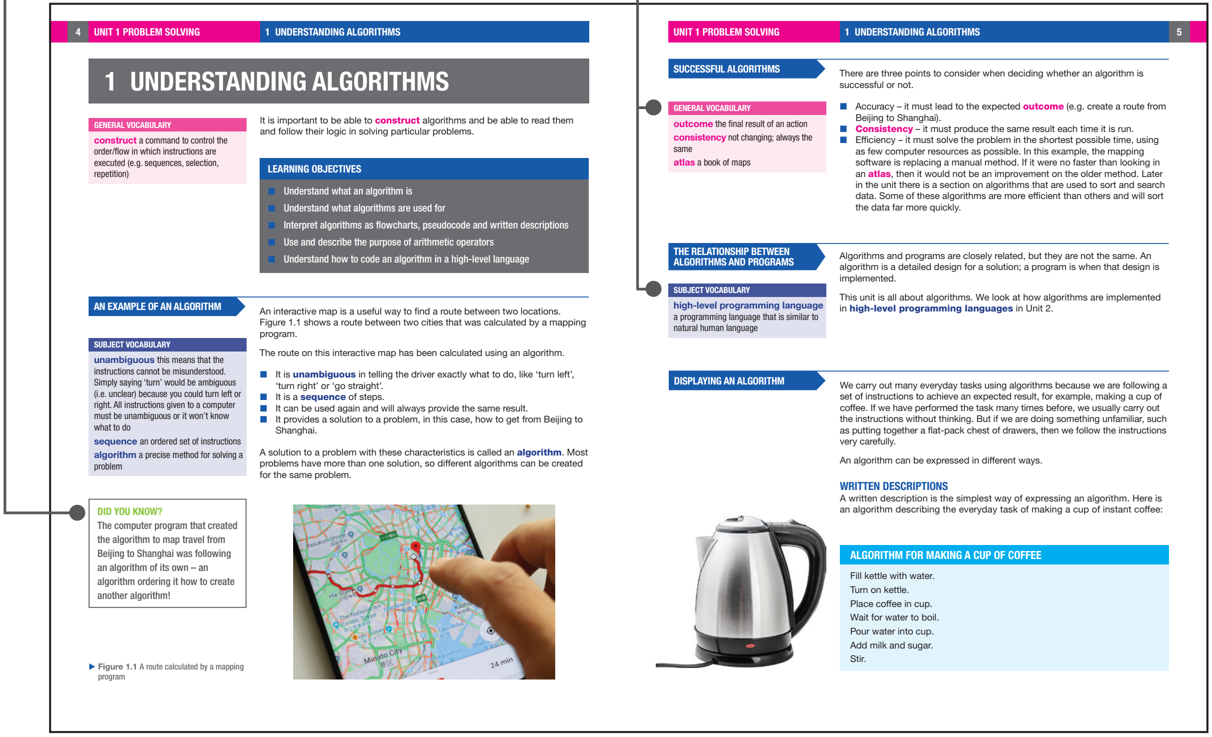
Extend your knowledge
Push yourself further by looking beyond the content of the course.

Key point
Easy to understand, core information to take away from sections.

Worked example
Key concepts can be demonstrated using step-by-step walkthroughs.

Did you know?
Interesting facts to encourage wider thought and stimulate discussion.

Subject vocabulary and General vocabulary
Useful words and phrases are colour coded within the main text and picked out in the margin with concise and simple definitions. These help understanding of key subject terms and support students whose first language is not English.



4 UNIT 1 PROBLEM SOLVING 1 UNDERSTANDING ALGORITHMS

1 UNDERSTANDING ALGORITHMS

GENERAL VOCABULARY

construct a command to control the order/flow in which instructions are executed (e.g. sequences, selection, repetition)

It is important to be able to **construct** algorithms and be able to read them and follow their logic in solving particular problems.

LEARNING OBJECTIVES

- Understand what an algorithm is
- Understand what algorithms are used for
- Interpret algorithms as flowcharts, pseudocode and written descriptions
- Use and describe the purpose of arithmetic operators
- Understand how to code an algorithm in a high-level language

AN EXAMPLE OF AN ALGORITHM

SUBJECT VOCABULARY

unambiguous this means that the instructions cannot be misunderstood. Simply saying 'turn' would be ambiguous (i.e. unclear) because you could turn left or right. All instructions given to a computer must be unambiguous or it won't know what to do

sequence an ordered set of instructions

algorithm a precise method for solving a problem

An interactive map is a useful way to find a route between two locations. Figure 1.1 shows a route between two cities that was calculated by a mapping program.

The route on this interactive map has been calculated using an algorithm.

- It is **unambiguous** in telling the driver exactly what to do, like 'turn left', 'turn right' or 'go straight'.
- It is a **sequence** of steps.
- It can be used again and will always provide the same result.
- It provides a solution to a problem, in this case, how to get from Beijing to Shanghai.

A solution to a problem with these characteristics is called an **algorithm**. Most problems have more than one solution, so different algorithms can be created for the same problem.

DID YOU KNOW?

The computer program that created the algorithm to map travel from Beijing to Shanghai was following an algorithm of its own – an algorithm ordering it how to create another algorithm!

▶ Figure 1.1 A route calculated by a mapping program

UNIT 1 PROBLEM SOLVING 1 UNDERSTANDING ALGORITHMS 5

SUCCESSFUL ALGORITHMS

There are three points to consider when deciding whether an algorithm is successful or not.

GENERAL VOCABULARY

outcome the final result of an action

consistency not changing; always the same

atlas a book of maps

- Accuracy – it must lead to the expected **outcome** (e.g. create a route from Beijing to Shanghai).
- **Consistency** – it must produce the same result each time it is run.
- Efficiency – it must solve the problem in the shortest possible time, using as few computer resources as possible. In this example, the mapping software is replacing a manual method. If it were no faster than looking in an **atlas**, then it would not be an improvement on the older method. Later in the unit there is a section on algorithms that are used to sort and search data. Some of these algorithms are more efficient than others and will sort the data far more quickly.

THE RELATIONSHIP BETWEEN ALGORITHMS AND PROGRAMS

SUBJECT VOCABULARY

high-level programming language a programming language that is similar to natural human language

Algorithms and programs are closely related, but they are not the same. An algorithm is a detailed design for a solution; a program is when that design is implemented.

This unit is all about algorithms. We look at how algorithms are implemented in **high-level programming languages** in Unit 2.

DISPLAYING AN ALGORITHM

We carry out many everyday tasks using algorithms because we are following a set of instructions to achieve an expected result, for example, making a cup of coffee. If we have performed the task many times before, we usually carry out the instructions without thinking. But if we are doing something unfamiliar, such as putting together a flat-pack chest of drawers, then we follow the instructions very carefully.

An algorithm can be expressed in different ways.

WRITTEN DESCRIPTIONS

A written description is the simplest way of expressing an algorithm. Here is an algorithm describing the everyday task of making a cup of instant coffee:

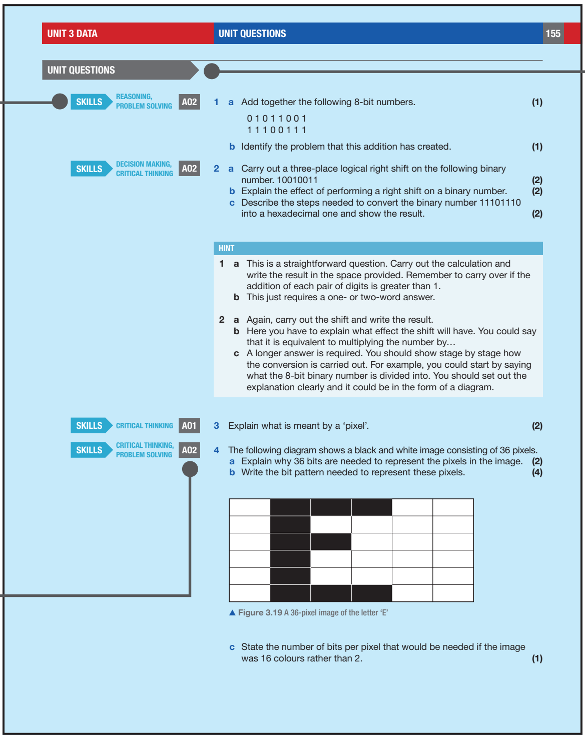
ALGORITHM FOR MAKING A CUP OF COFFEE

```
Fill kettle with water.
Turn on kettle.
Place coffee in cup.
Wait for water to boil.
Pour water into cup.
Add milk and sugar.
Stir.
```

Summary
Quickly recap the core content of each section.

Checkpoint
Checkpoints help you to check and reflect on your learning at the end of each section. Strengthen questions help you to consolidate basic knowledge and understanding. Challenge questions are more demanding and ask you to apply your learning.

Skills
Relevant exam questions have been assigned the key skills that you will gain from undertaking them, allowing for a strong focus on particular academic qualities. These transferable skills are highly valued in further study and the workplace.



UNIT 3 DATA UNIT QUESTIONS 155

UNIT QUESTIONS

SKILLS REASONING, PROBLEM SOLVING A02

1 a Add together the following 8-bit numbers. (1)

0 1 0 1 1 0 0 1
1 1 1 0 0 1 1 1

b Identify the problem that this addition has created. (1)

SKILLS DECISION MAKING, CRITICAL THINKING A02

2 a Carry out a three-place logical right shift on the following binary number. 10010011 (2)
b Explain the effect of performing a right shift on a binary number. (2)
c Describe the steps needed to convert the binary number 11101110 into a hexadecimal one and show the result. (2)

HINT

1 a This is a straightforward question. Carry out the calculation and write the result in the space provided. Remember to carry over if the addition of each pair of digits is greater than 1.
b This just requires a one- or two-word answer.

2 a Again, carry out the shift and write the result.
b Here you have to explain what effect the shift will have. You could say that it is equivalent to multiplying the number by…
c A longer answer is required. You should show stage by stage how the conversion is carried out. For example, you could start by saying what the 8-bit binary number is divided into. You should set out the explanation clearly and it could be in the form of a diagram.

SKILLS CRITICAL THINKING A01

3 Explain what is meant by a 'pixel'. (2)

SKILLS CRITICAL THINKING, PROBLEM SOLVING A02

4 The following diagram shows a black and white image consisting of 36 pixels.
a Explain why 36 bits are needed to represent the pixels in the image. (2)
b Write the bit pattern needed to represent these pixels. (4)

▲ Figure 3.19 A 36-pixel image of the letter 'E'

c State the number of bits per pixel that would be needed if the image was 16 colours rather than 2. (1)

Unit questions
These exam-style questions are found at the end of each unit. They are tailored to the Pearson Edexcel specification to allow for the practice and development of exam writing technique. They also allow for practice responding to the command words used in the exams.

Assessment objectives
Questions are tagged with the relevant assessment objectives that are being examined.

ASSESSMENT OVERVIEW

The following tables give an overview of the assessment for this course. You should study this information closely to help ensure that you are fully prepared and know exactly what to expect in each part of the assessment.

PAPER 1	PERCENTAGE	MARK	TIME	AVAILABILITY
PRINCIPLES OF COMPUTER SCIENCE Written exam paper Paper code 4CP0/01 Externally set and assessed by Pearson Edexcel Single tier of entry	50%	80	2 hours	June exam series First assessment June 2019
PAPER 2	**PERCENTAGE**	**MARK**	**TIME**	**AVAILABILITY**
APPLICATION OF COMPUTATIONAL THINKING Practical and written exam paper Paper code 4CP0/02 Externally set and assessed by Pearson Edexcel Single tier of entry	50%	80	3 hours	June exam series First assessment June 2019

ASSESSMENT OBJECTIVES AND WEIGHTINGS

ASSESSMENT OBJECTIVE	DESCRIPTION	% IN INTERNATIONAL GCSE
A01	Demonstrate knowledge and understanding of the key principles of computer science	27.5
A02	Apply knowledge and understanding of key concepts and principles of computer science	42.5
A03	Analyse problems in computational terms: • to make reasoned judgements • to design, program, test, evaluate and refine solutions	30

RELATIONSHIP OF ASSESSMENT OBJECTIVES TO UNITS

UNIT NUMBER	ASSESSMENT OBJECTIVE		
	A01	A02	A03
PAPER 1	21.5%	21%	7.5%
PAPER 2	6%	21.5%	22.5%
TOTAL FOR INTERNATIONAL GCSE	**27.5%**	**42.5%**	**30%**

ASSESSMENT SUMMARY

PAPER 1	DESCRIPTION	MARKS	ASSESSMENT OBJECTIVES
PRINCIPLES OF COMPUTER SCIENCE PAPER CODE 4CP0/01	**Structure** Paper 1 contributes 50% of the total marks for the Computer Science qualification. Students must answer all questions. The paper consists of multiple-choice, short open-response, open-response and extended open-response answer questions.	The total number of marks available is 80	Questions will test the following Assessment Objectives: AO1 – 21.5% AO2 – 21% AO3 – 7.5%
	Content summary This paper will primarily assess knowledge and understanding of the basic principles of computer science, including some coverage of how these principles are applied when solving problems that relate to a particular situation.		
	Assessment This is a single-tier exam paper and all questions cover the full ranges of grades from 9–1. The assessment duration is 2 hours.		
PAPER 2	**DESCRIPTION**	**MARKS**	**ASSESSMENT OBJECTIVES**
APPLICATION OF COMPUTATIONAL THINKING PAPER CODE 4CP0/02	**Structure** Paper 2 contributes 50% of the total marks for the Computer Science qualification. Students must answer all questions. The paper consists of multiple-choice, short open-response, open-response, extended open-response answer and task-based questions. The task-based questions will be carried out using a computer system under supervision. All other questions requiring a written response will be answered in the paper.	The total number of marks available is 80	Questions will test the following Assessment Objectives: AO1 – 6% AO2 – 21.5% AO3 – 22.5%
	Content summary This paper will primarily assess the practical application of computational thinking, whereby learners will create, use and adapt existing algorithms to solve problems in a particular situation. This paper will also test students' knowledge and understanding of the topics.		
	Assessment This is a single-tier exam paper and all questions cover the full ranges of grades from 9–1. The assessment duration is 3 hours. A choice of three programming languages will be available (Python, C# or Java). A pseudocode reference document will be available for learners to reference during the assessment.		

f(x)
F=ma
Mg
v(t+Δt)

UNIT 1 PROBLEM SOLVING

Assessment Objective 1

Demonstrate knowledge and understanding of the key principles of computer science

Assessment Objective 2

Apply knowledge and understanding of key concepts and principles of computer science

Assessment Objective 3

Analyse problems in computational terms:

- to make reasoned judgements
- to design, program, test, evaluate and refine solutions

In this unit you will learn about algorithms, which are the basis of computer programming, and how they can be presented as flowcharts and pseudocode. You will learn about the basic constructs of an algorithm such as sequence, selection and iteration and how these are used to solve problems using computational thinking. You will also look at completing and correcting algorithms in addition to algorithms to sort and search data. In the next unit you will learn how to code these algorithms using pseudocode and high-level programming languages.

1 UNDERSTANDING ALGORITHMS

GENERAL VOCABULARY

construct a command to control the order/flow in which instructions are executed (e.g. sequences, selection, repetition)

It is important to be able to **construct** algorithms and be able to read them and follow their logic in solving particular problems.

LEARNING OBJECTIVES

- Understand what an algorithm is
- Understand what algorithms are used for
- Interpret algorithms as flowcharts, pseudocode and written descriptions
- Use and describe the purpose of arithmetic operators
- Understand how to code an algorithm in a high-level language

AN EXAMPLE OF AN ALGORITHM

SUBJECT VOCABULARY

unambiguous this means that the instructions cannot be misunderstood. Simply saying 'turn' would be ambiguous (i.e. unclear) because you could turn left or right. All instructions given to a computer must be unambiguous or it won't know what to do

sequence an ordered set of instructions

algorithm a precise method for solving a problem

An interactive map is a useful way to find a route between two locations. Figure 1.1 shows a route between two cities that was calculated by a mapping program.

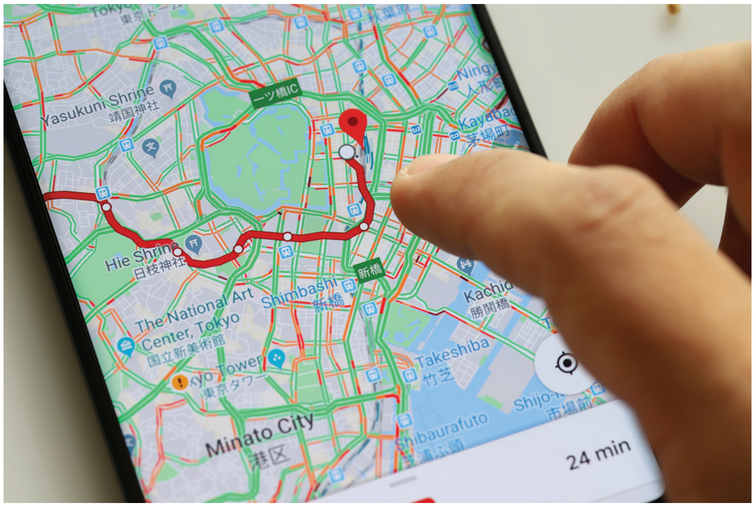
The route on this interactive map has been calculated using an algorithm.

- It is **unambiguous** in telling the driver exactly what to do, like 'turn left', 'turn right' or 'go straight'.
- It is a **sequence** of steps.
- It can be used again and will always provide the same result.
- It provides a solution to a problem, in this case, how to get from Beijing to Shanghai.

A solution to a problem with these characteristics is called an **algorithm**. Most problems have more than one solution, so different algorithms can be created for the same problem.

DID YOU KNOW?

The computer program that created the algorithm to map travel from Beijing to Shanghai was following an algorithm of its own – an algorithm ordering it how to create another algorithm!



▶ **Figure 1.1** A route calculated by a mapping program

SUCCESSFUL ALGORITHMS

There are three points to consider when deciding whether an algorithm is successful or not.

- Accuracy – it must lead to the expected **outcome** (e.g. create a route from Beijing to Shanghai).
- **Consistency** – it must produce the same result each time it is run.
- Efficiency – it must solve the problem in the shortest possible time, using as few computer resources as possible. In this example, the mapping software is replacing a manual method. If it were no faster than looking in an **atlas**, then it would not be an improvement on the older method. Later in the unit there is a section on algorithms that are used to sort and search data. Some of these algorithms are more efficient than others and will sort the data far more quickly.

GENERAL VOCABULARY

outcome the final result of an action

consistency not changing; always the same

atlas a book of maps

THE RELATIONSHIP BETWEEN ALGORITHMS AND PROGRAMS

Algorithms and programs are closely related, but they are not the same. An algorithm is a detailed design for a solution; a program is when that design is implemented.

This unit is all about algorithms. We look at how algorithms are implemented in **high-level programming languages** in Unit 2.

SUBJECT VOCABULARY

high-level programming language a programming language that is similar to natural human language

DISPLAYING AN ALGORITHM

We carry out many everyday tasks using algorithms because we are following a set of instructions to achieve an expected result, for example, making a cup of coffee. If we have performed the task many times before, we usually carry out the instructions without thinking. But if we are doing something unfamiliar, such as putting together a flat-pack chest of drawers, then we follow the instructions very carefully.

An algorithm can be expressed in different ways.

WRITTEN DESCRIPTIONS

A written description is the simplest way of expressing an algorithm. Here is an algorithm describing the everyday task of making a cup of instant coffee:

ALGORITHM FOR MAKING A CUP OF COFFEE

Fill kettle with water.

Turn on kettle.

Place coffee in cup.

Wait for water to boil.

Pour water into cup.

Add milk and sugar.

Stir.

SKILLS REASONING, PROBLEM SOLVING

ACTIVITY 1

GETTING TO SCHOOL

Produce a written description of an algorithm for getting to school. It should start with leaving home and end with arriving at school. For example, the algorithm could start with 'Walk to bus stop'.

Check your algorithm with other members of the group. Would your algorithm work for others? Are there any general statements that are common to all algorithms?

SUBJECT VOCABULARY

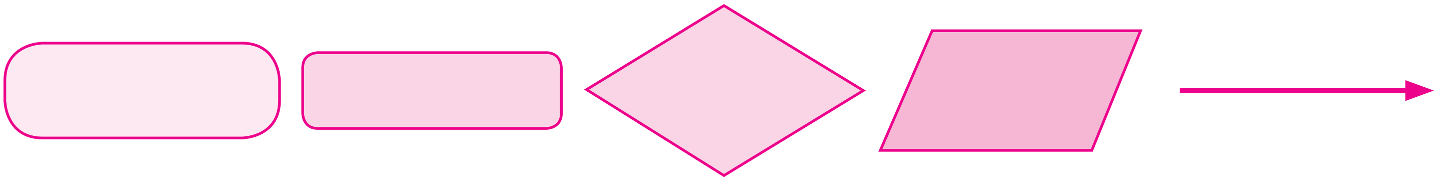
flowchart shows an algorithm as a diagram. Each step in the algorithm is represented by a symbol. Symbols are linked together with arrows showing the order in which steps are completed

FLOWCHARTS

Flowcharts can be used to show an algorithm as a diagram. They provide a more visual display.

There are special symbols that have to be used in a flowchart. You can't just make up your own, because nobody else would be able to follow your algorithm.

Figure 1.2 shows the flowchart symbols that should be used.



Indicates the start or end of an algorithm

Indicates a process to be carried out

Indicates a decision to be made

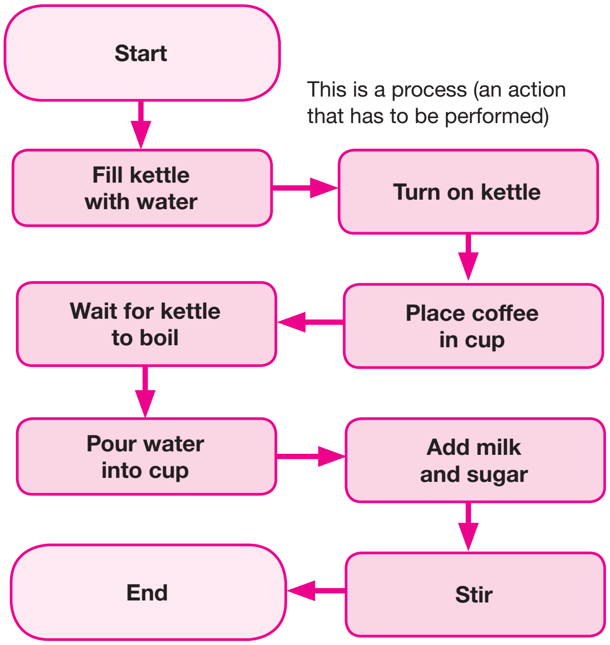
Indicates an input or output

Shows the logical flow of the algorithm

▲ **Figure 1.2** Flowchart symbols

The flowchart in Figure 1.3 is an alternative way of showing the algorithm for making a cup of coffee as a written description.

▶ **Figure 1.3** Flowchart of an algorithm to make a cup of coffee



This is a process (an action that has to be performed)

This instruction is unambiguous – the water must be boiling. Stating 'wait for the water to heat' would be ambiguous. How hot should it be?

SKILLS PROBLEM SOLVING

ACTIVITY 2

SCHOOL JOURNEY FLOWCHART

Display the 'journey to school' algorithm that you created in Activity 1 as a flowchart.

SKILLS PROBLEM SOLVING

ACTIVITY 3

BATH FLOWCHART

A student has created a written algorithm for preparing a bath. Working with a partner, display the following as a flowchart. You may need to change the order or add actions.

- Put in the plug.
- Fill the bath to the correct level.
- Check the temperature is OK.

The algorithms you have looked at so far are designed for humans to follow. Algorithms also form the **basis** of computer programs. Computers are mindless machines that simply do exactly what they are told. They follow a set of instructions, but they can carry out these instructions far more quickly than humans. That is why they are so useful.

GENERAL VOCABULARY

basis an important idea or fact that something is based on

SUBJECT VOCABULARY

pseudocode a structured, code-like language that can be used to describe an algorithm

developer a person whose job it is to create new software

logic the principles and reasoning underlying the constructs and elements to be applied in solving problems

PSEUDOCODE

In addition to flowcharts and written descriptions, algorithms can also be expressed in **pseudocode**. The pseudocode can be used to code the solution in an actual programming language.

It allows the **developer** to concentrate on the **logic** and efficiency of the algorithm without having to bother about the rules of any particular programming language. It is relatively straightforward to translate an algorithm written in pseudocode into any high-level programming language.

SKILLS RESEARCH

ACTIVITY 4

INVESTIGATING PSEUDOCODE

Different organisations or examination boards have their own unique versions of pseudocode. Investigate the Pearson Edexcel pseudocode that you will need for your International GCSE course and which will be used in this book.

EXAMPLE OF A SIMPLE ALGORITHM

To introduce the Pearson Edexcel pseudocode, here is a simple written algorithm that asks the user to input two numbers and then outputs the result of adding them together.

WRITTEN DESCRIPTION

ALGORITHM FOR ADDING TWO NUMBERS

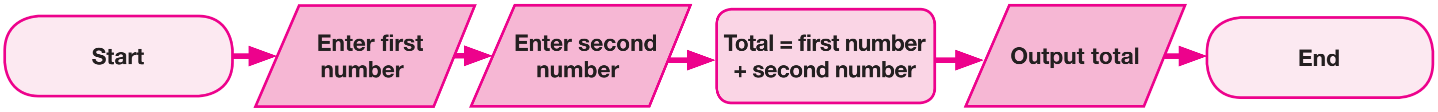
Enter first number.

Enter second number.

Calculate total by adding first and second numbers.

Output total.

FLOWCHART



▲ **Figure 1.4** Flowchart showing the adding of two numbers

PSEUDOCODE

ALGORITHM FOR ADDING TWO NUMBERS

```
SEND 'Please enter the first number' TO DISPLAY
RECEIVE firstNumber FROM KEYBOARD
SEND 'Please enter the second number' TO DISPLAY
RECEIVE secondNumber FROM KEYBOARD
SET total TO firstNumber + secondNumber
SEND total TO DISPLAY
```

The pseudocode gives clear step-by-step instructions that the computer will be expected to carry out. It also introduces some important programming concepts.

- The numbers entered by the user are stored in two **variables** with the **identifiers** firstNumber and secondNumber.
- The result of adding the numbers together is stored in the variable total.
- Text has to be placed in quotation marks (single or double) if it is to be displayed. For example, 'Please enter the first number' (or "Please enter the first number").
- Quotation marks are not used if a variable is to be displayed. If they were, the word 'total' in the last instruction would be displayed instead of the number it represents.
- **Arithmetic operators** are used to perform calculations. Table 1.1 shows the arithmetic operators.

SUBJECT VOCABULARY

variable a 'container' used to store data. The data stored in a variable is referred to as a value. The value stored in a variable is not fixed. The same variable can store different values during the course of a program and each time a program is run

identifier a unique name given to a variable or a constant. Using **descriptive** names for variables makes code much easier to read

arithmetic operator an **operator** that performs a calculation on two numbers

GENERAL VOCABULARY

operator a character that represents an action, e.g. 'x' represents a multiplication and '/' a division

descriptive describing something clearly

ARITHMETIC OPERATORS

OPERATOR	FUNCTION	EXAMPLE
+	Addition: add the values together.	8 + 5 = 13 myScore1 + myScore2
–	Subtraction: subtract the second value from the first.	17 – 4 = 13 myScore1 – myScore2
*	Multiplication: multiply the values together.	6 * 9 = 54 numberBought * price
/	Real division: divide the first value by the second value and return the result including decimal places.	13 / 4 = 3.25 totalMarks/numberTests
DIV	**Quotient**: like division, but it only returns the whole number or *integer*.	13 DIV 4 = 3 totalMarks DIV numberTests
MOD	**Modulus**/modulo: this will return the remainder of a division.	13 / 4 = 3 remainder 1 Therefore 13 MOD4 = 1
^	Exponentiation: this is for 'to the **power** of'.	3 ^ 3 = 27 It is the same as writing 3^3

▲ **Table 1.1** Arithmetic operators

VARIABLES AND CONSTANTS

Variables play an important role in algorithms and programming. The value stored by a variable can change as a program is running. Variables are extremely useful in programming because they make it possible for the same program to process different sets of data.

A **constant** is the opposite of a variable. It is a 'container' that holds a value that always stays the same. Constants are useful for storing fixed information, such as the value of pi, the number of litres in a gallon or the number of months in a year.

Each variable and constant in an algorithm has to have a unique identifier. It is important to choose descriptive names for identifiers. This will make your code much easier to read. For example, a variable to hold a user's first name could be given the identifier `firstName` to show the data it contains. If it were given the identifier `X` instead, it wouldn't clearly show what data it contained.

NAMING CONVENTIONS FOR VARIABLES AND CONSTANTS

It is sensible to write identifiers in the same way throughout an algorithm. A common method is to use 'camel case' for compound words (e.g. `firstName, secondName`) with no space between words and the second word starting with a capital letter. Alternatively, you could capitalise the first letter of both words, e.g. `FirstName, SecondName`, or separate the words with an underscore, e.g. `first_name, second_name`, known as 'snake case'.

KEY POINT

When you chose a variable name, it should be descriptive of the data it will hold, e.g. distance or length. Long variable names are easier to misspell!

SUBJECT VOCABULARY

constant a 'container' that holds a value that never changes; like variables, constants have unique identifiers

GENERAL VOCABULARY

quotient a result found by dividing one quantity by another

modulus the remainder after the division of one number by another

power the small number written to the right and above another number to show how many times it should be multiplied by itself

SKILLS PROBLEM SOLVING, ANALYSIS

KEY POINT

Whichever method you use for naming conventions, you must use it consistently and not keep switching between the different methods.

ACTIVITY 5

WRITING ALGORITHMS IN PSEUDOCODE

Here is a written description of an algorithm:

Enter the first number.
Enter the second number.
The third number is equal to the first number multiplied by the second number.
Display the third number.

Express this algorithm in pseudocode.

SKILLS PROBLEM SOLVING

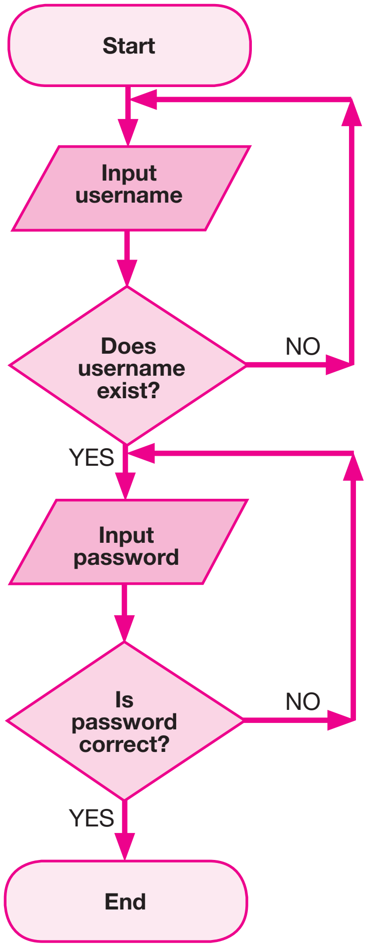
EXTEND YOUR KNOWLEDGE

When you enter a search term into Google®, a list of links to websites is returned. But why are they presented in that particular order? With a partner, research the `PageRank` algorithm that Google uses to rate the importance of websites and write a short report about your findings.

ACTIVITY 6

WRITTEN DESCRIPTIONS OF ALGORITHMS

This algorithm is displayed as a flowchart.



▲ **Figure 1.5** Flowchart of an algorithm

Produce a written description of this algorithm.

CHECKPOINT

Strengthen

SKILLS CRITICAL THINKING

S1 Produce a written description of an algorithm for borrowing a book from the library.

SKILLS DECISION MAKING

S2 What is the function of each of the seven arithmetic operators?

SKILLS REASONING

S3 What is a variable? Why are they useful?

SKILLS CRITICAL THINKING

S4 What is the difference between a variable and a constant?

Challenge

SKILLS CRITICAL THINKING, PROBLEM SOLVING

C1 Produce a flowchart describing an algorithm for making a cheese sandwich.

SKILLS PROBLEM SOLVING

C2 Write an algorithm expressed in pseudocode that receives three numbers from the keyboard, then calculates and displays the average.

How confident do you feel about your answers to these questions? If you're not sure you answered them well, try the following activities again.

- For S1 re-read 'The relationship between algorithms and programs'.
- For S2 study Table 1.1.
- For S3 and S4 look again at 'Variables and constants'.

SUMMARY

- An algorithm is a precise method for solving a problem.
- Algorithms can be displayed as written descriptions, flowcharts and in pseudocode.
- Pseudocode is a **structured**, code-like language.
- Pseudocode is translated into program code.
- Arithmetic operators are used in calculations.
- Variables and constants are 'containers' for storing data. The value stored in a variable can change, whereas the value of a constant never changes.
- Selecting descriptive names for identifiers makes code easier to read.

GENERAL VOCABULARY

structured organised so that the parts work well together

2 CREATING ALGORITHMS

In this section you will learn how to create algorithms to solve particular problems using the constructs of sequence, selection and iteration. You will also practise displaying algorithms in flowcharts and pseudocode.

LEARNING OBJECTIVES

- Understand how to create an algorithm to solve a particular problem
- Make use of programming constructs (sequence, selection and iteration) and use appropriate conventions (flowchart, pseudocode, written description, draft program code)

ALGORITHMS FOR COMPUTERS

SUBJECT VOCABULARY

construct a smaller part from which something is built. Letters and numbers (i.e. a to z and 0 to 9) are the constructs we use to build our language and convey meaning. Bricks and cement are the basic constructs of a building

selection a construct that allows a choice to be made between different alternatives

iteration a construct that means a process is repeated. An action is repeated until a **condition** is met or a particular outcome is reached. It is often referred to as a 'loop'

GENERAL VOCABULARY

ambiguous when a statement or command does not have one obvious meaning but can be interpreted in different ways

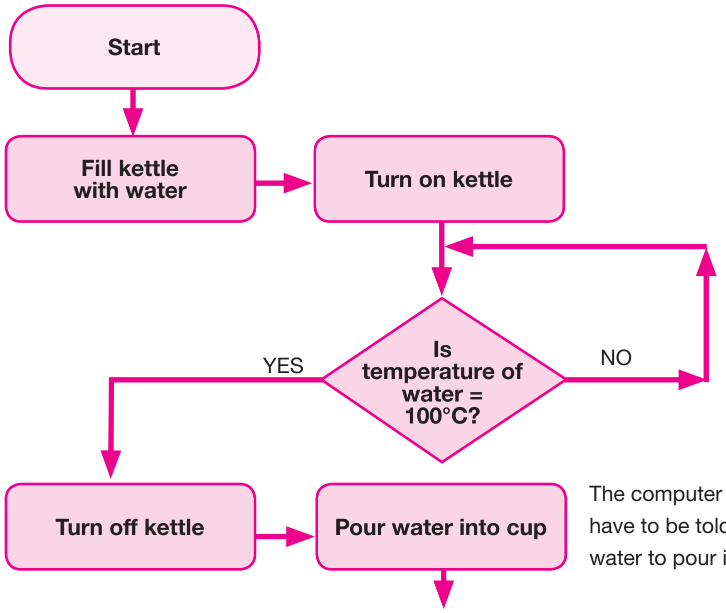
condition something that must happen before something else can happen

There was an **ambiguous** statement in the algorithm for making a cup of coffee. After filling the kettle with water and adding coffee to the cup, the next instruction was 'Wait for water to boil'.

A human can understand that this instruction means they have to keep checking the kettle over and over again until the water is boiling, but a computer is unable to understand an instruction like this in the same way. It would just wait. And wait. Forever.

Even worse, the algorithm didn't state clearly how to tell the water was boiling. Through experience, we humans assume the water is boiling when there is lots of steam, sound and bubbles; or, even better, when the kettle turns itself off. An algorithm for a computer would have to state that it must wait until the water reached 100°C.

A version of this part of the algorithm, suitable for a computer, is shown in Figure 1.6. This example introduces two new **constructs** from which algorithms are created. We have already met the construct sequence – step-by-step instructions in the correct order. To add to this, we now have **selection** and **iteration**.



The computer would also have to be told how much water to pour into the cup!

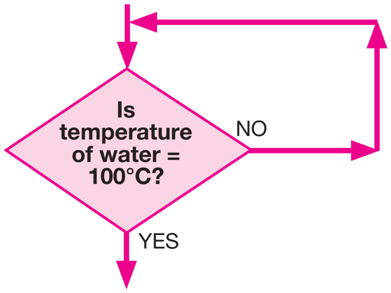
▶ **Figure 1.6** Part of an algorithm suitable for a computer for making coffee

REPRESENTING SELECTION AND ITERATION IN A FLOWCHART

Selection and iteration are represented in a flowchart as shown in Figure 1.7.

▶ **Figure 1.7** Selection and iteration in a flowchart

There is a question with two alternatives. This represents the selection.



These arrows represent the iteration. If the answer is ‘NO’ then the selection question is repeated until the answer is ‘YES’ – the desired outcome.

SKILLS REASONING, PROBLEM SOLVING

DID YOU KNOW?

We use iteration in our daily lives whenever we carry out an action over and over again. For example, at mealtimes we keep on eating until our plate is empty or we have had enough to eat.

When we’re travelling by car and the traffic lights are red, we have to keep waiting until they change to green.

An actor repeats their lines over and over again, learning more each time until they know them all.

ACTIVITY 7

GUESSING GAMES

A student is creating a guessing game. A player has to enter a number no greater than 10. If it is too high, they are informed that they have made an error. But if it is within the range 1 to 10, they are told whether or not they have guessed the correct number. (Assume that the correct number is 3.)

Can you make an algorithm to solve this problem and express it as a written description and a flowchart?

Compare your solution to others in your group.

Check that they are correct and would produce the correct outcome.

Are some of the algorithms more efficient than others? Do they use fewer commands?

SKILLS PROBLEM SOLVING, ANALYSIS

ACTIVITY 8

CALCULATING GRADES

A school uses this algorithm to calculate the grade that students achieve in end-of-topic tests.

```
RECEIVE testScore FROM KEYBOARD
IF testScore >= 80 THEN
  SEND ‘A’ TO DISPLAY
ELSE
  IF testScore >= 70 THEN
    SEND ‘B’ TO DISPLAY
  ELSE
    IF testScore >= 60 THEN
      SEND ‘C’ TO DISPLAY
    ELSE
      IF testScore > 0 THEN
        SEND ‘D’ TO DISPLAY
      ELSE
        SEND ‘FAIL’ TO DISPLAY
```

```
            END IF
        END IF
    END IF
END IF
```

What would be the output of this algorithm for these test scores: 91, 56 and 78?

ITERATION

When writing programs, it is often necessary to repeat the same set of statements several times. Instead of making multiple copies of the statements, you can use iteration to repeat them. The algorithm for making a cup of coffee includes an instruction to keep waiting until the water in the kettle boils.

CHECKPOINT

Strengthen

SKILLS CRITICAL THINKING, DECISION MAKING

S1 How are sequence, selection and iteration used in algorithms? Give examples to justify your answer.

Challenge

SKILLS PROBLEM SOLVING

C1 Develop an algorithm using a flowchart that asks the user to enter their height (in metres) and weight (in kilograms) and displays their body mass index (BMI). The formula for calculating BMI is weight/height^2.

SKILLS PROBLEM SOLVING

C2 Develop an algorithm expressed as a flowchart to control the heating in a house. A thermostat monitors the temperature within the house. During the week the temperature should be 20°C between 06.00 and 08.30 in the morning and between 17.30 and 22.00 at night. At weekends it should be 22°C between 08.00 and 23.00. If the temperature in the house falls below 10°C at any time the boiler is switched on.

How confident do you feel about your answers to these questions? If you're not sure you answered them well, try the following activities again.

- For S1 have a look at the Subject vocabulary sections in 'Understanding algorithms' and 'Creating algorithms'.

SUMMARY

- The constructs sequence, selection and iteration are the basic building blocks of algorithms.

3 SORTING AND SEARCHING ALGORITHMS

Whenever we want to find information, we carry out a search. Just imagine the number of people around the world who are using the same search engine at the same time. Without efficient searching algorithms, we would have to wait a long time to be shown the results.

We are on all sorts of lists (for example, school and college, clubs and groups, voting registers) and all these must be searched to find relevant information. Sorting information is also important to facilitate efficient searches.

LEARNING OBJECTIVES

- Understand how standard algorithms work (bubble sort, merge sort, linear search, binary search)
- Understand how the choice of algorithm is influenced by the data structures and data values that need to be manipulated
- Evaluate the fitness for purpose of algorithms in meeting specified requirements efficiently, using logical reasoning and test data

Two of the most common tasks in computer programs are sorting data into a particular order and searching for particular items of information.

There might be millions of items of stored data and searching for information wouldn't be efficient if the data was not sorted. Imagine the confusion and difficulty of having to find something in a dictionary that wasn't in alphabetical order. Or planning a trip with train timetables that weren't sorted into time order. Even small lists such as football league tables or the Top 20 music charts are much more useful if they are sorted into order.

SORTING ALGORITHMS

As sorting is such a widely used procedure, many algorithms have been created to carry it out. As with all algorithms, some are more efficient than others.

BUBBLE SORT

When data is sorted, different items must be compared with each other and moved so that they are in either **ascending order** or **descending order**.

The bubble sort algorithm starts at one end of the list and compares pairs of data items. If they are in the wrong order, they are swapped. The comparison of pairs continues to the end of the list, each complete **traversal** of the list being called a 'pass'. This process is repeated until there have been no swaps during a pass. This indicates that the items must all be in the correct order.

SUBJECT VOCABULARY

ascending order this is arranging items from smallest to largest (e.g. 1, 2, 3)

descending order this is arranging items from largest to smallest (e.g. 3, 2, 1)

traversal travel across or through something.

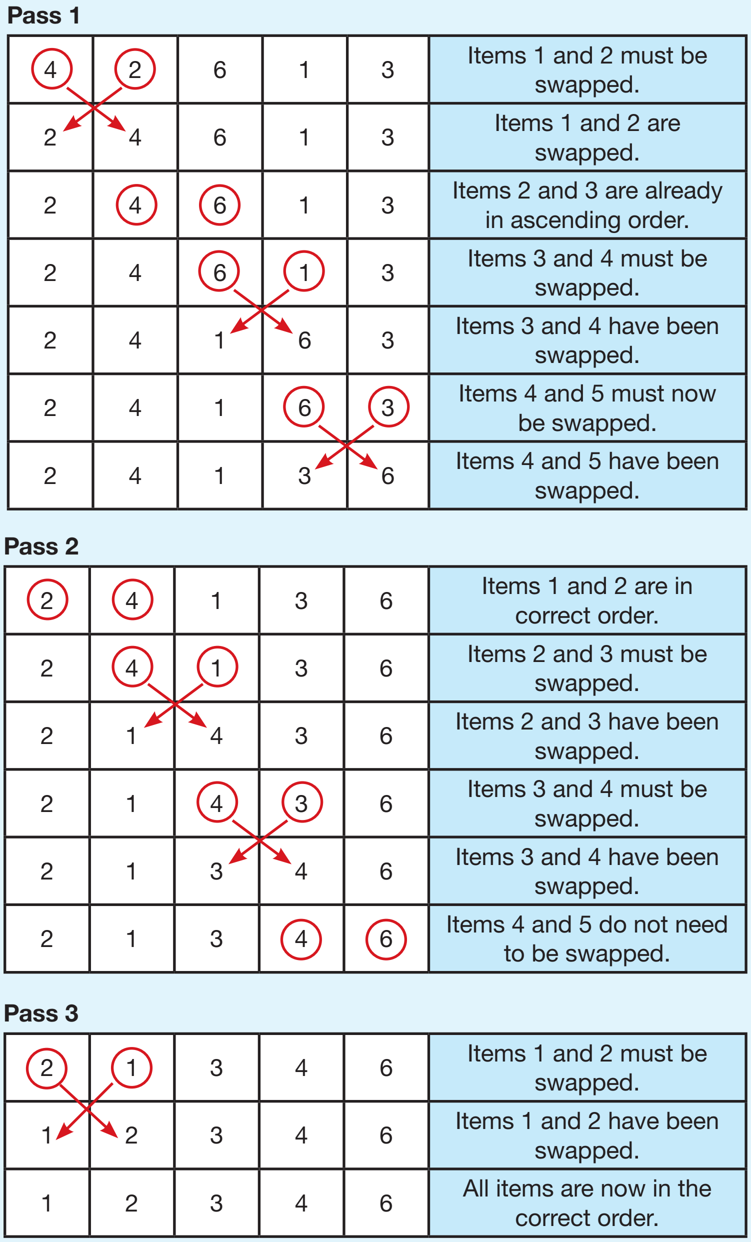
The algorithm can be described as follows.

BUBBLE SORT (ASCENDING ORDER)

1 Start at the beginning of the list.
2 Compare the values in position 1 and position 2 in the list – if they are not in ascending order then swap them.
3 Compare the values in position 2 and position 3 in the list and swap if necessary.
4 Continue to the end of the list.
5 If there have been any swaps, repeat steps 1 to 4.

WORKED EXAMPLE

Here is an example of a bubble sort in action.



Pass 1

4	2	6	1	3	Items 1 and 2 must be swapped.
2	4	6	1	3	Items 1 and 2 are swapped.
2	4	6	1	3	Items 2 and 3 are already in ascending order.
2	4	6	1	3	Items 3 and 4 must be swapped.
2	4	1	6	3	Items 3 and 4 have been swapped.
2	4	1	6	3	Items 4 and 5 must now be swapped.
2	4	1	3	6	Items 4 and 5 have been swapped.

Pass 2

2	4	1	3	6	Items 1 and 2 are in correct order.
2	4	1	3	6	Items 2 and 3 must be swapped.
2	1	4	3	6	Items 2 and 3 have been swapped.
2	1	4	3	6	Items 3 and 4 must be swapped.
2	1	3	4	6	Items 3 and 4 have been swapped.
2	1	3	4	6	Items 4 and 5 do not need to be swapped.

Pass 3

2	1	3	4	6	Items 1 and 2 must be swapped.
1	2	3	4	6	Items 1 and 2 have been swapped.
1	2	3	4	6	All items are now in the correct order.

▲ **Figure 1.8** A bubble sort

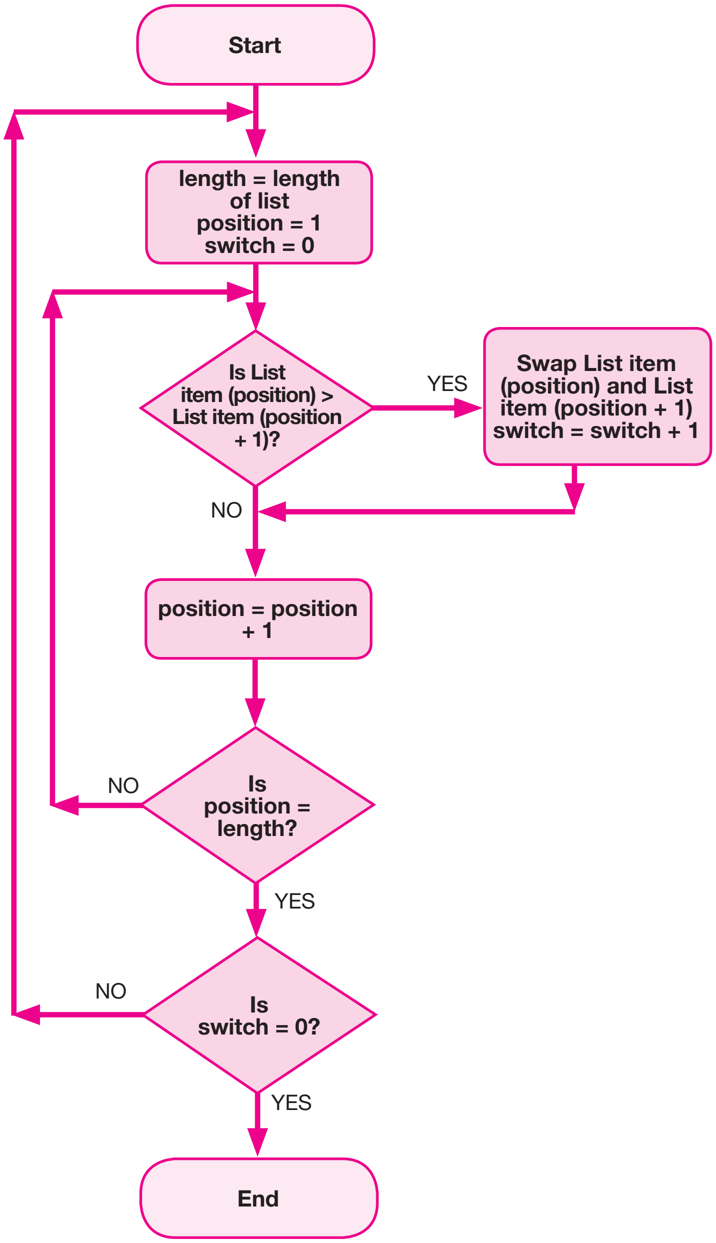
DID YOU KNOW?

Do you know why it is called 'bubble sort'? If you look carefully, you can see that the largest items gradually move to the end, like bubbles rising in water. After the first pass, the largest number is in its correct position. Then after the second pass, the next largest is in its correct position. This happens on each pass and so if the algorithm is to be made more efficient the last set of comparisons can be left out.

It would take a human three passes to carry out this bubble sort. A computer would need four passes because it must continue until there have been no swaps; it cannot just look at all of the numbers at once and see that they are all in order.

The bubble sort algorithm can be represented as a flowchart as shown in Figure 1.9.

▶ **Figure 1.9** Bubble sort algorithm written as a flowchart



SKILLS CRITICAL THINKING

ACTIVITY 9

HOW DOES BUBBLE SORT WORK?

Study the flowchart of the bubble sort algorithm.

Using the variables declared, can you explain the logic behind the algorithm? How does it function to sort a list?

GENERAL VOCABULARY

repeatedly many times

SUBJECT VOCABULARY

recursion a process that is repeated. For example, a document can be checked and edited, checked and edited and so on until it is perfect

KEY POINT

In the exam, you will be expected to show the intermediate stages when the algorithms are applied to data.

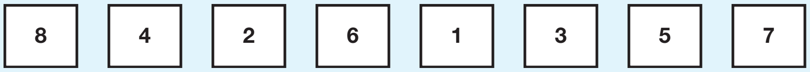
MERGE SORT

Merge sort is a sorting algorithm that divides a list into two smaller lists and then divides these until the size of each list is one. **Repeatedly** applying a method to the results of a previous application of the method is called **recursion**.

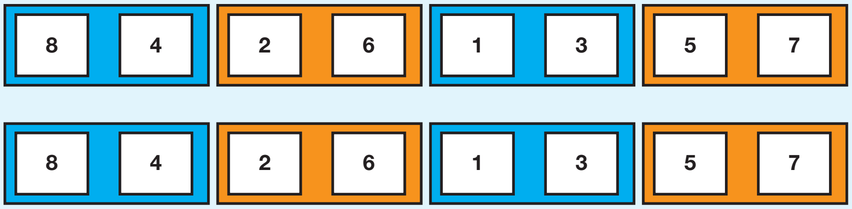
In computing, a problem is solved by repeatedly solving smaller parts of the problem. A part of a program can be run and rerun on the results of the previous run (e.g. repeatedly dividing a number by 2).

WORKED EXAMPLE

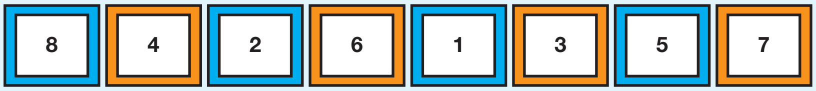
Here is an example of a merge sort which will sort the following list into ascending order.



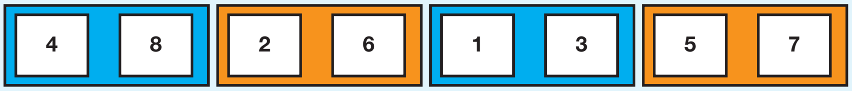
The list is split into half with recursion to produce a left list and a right list each time.



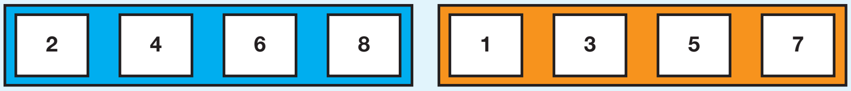
This continues until there is only one item in each list. Therefore, each list is sorted into order.



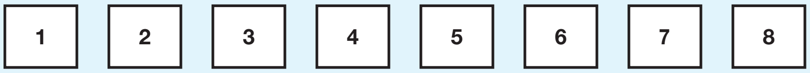
The left and right lists are now merged through recursion with the items in the correct order.



The leftmost items in each list are the lowest items of those lists and the algorithm compares them – in this case 4 with 2. The 2 is inserted in the new list and the 4 is then compared with the second number of the right list – 6. The 4 is inserted and the 6 is compared with the second number of the left list.



The algorithm now merges these two lists in the same way to produce the final sorted list. 1 is compared with 2 and then 2 with 3, 3 with 4, etc.



SKILLS PROBLEM SOLVING

ACTIVITY 10

USING MERGE SORT

Using a table like the one in the worked example on page 18, show how the following list would be sorted into descending order using merge sort.

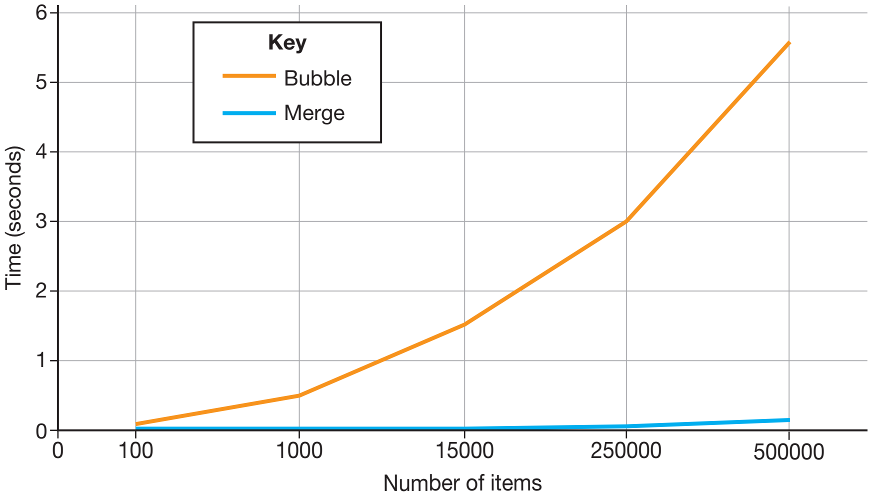
48, 20, 9, 17, 13, 21, 28, 60

EXTEND YOUR KNOWLEDGE

Only two sorting algorithms are required for the specification: bubble sort (the slowest) and merge sort (one of the most efficient). There are far more, and many of them are relatively easy to code. Research the insertion and selection sorts.

EFFICIENCY OF SORTING ALGORITHMS

This graph compares the performance of the bubble and merge sort algorithms.



▶ **Figure 1.10** A graph comparing the performance of bubble and merge sort algorithms

The bubble and merge sort algorithms demonstrate two alternative approaches to algorithm design.

The bubble sort algorithm is said to be using **brute force** because it starts at the beginning and completes the same task over and over again until it has found a solution.

The merge sort uses the **divide and conquer** method because it repeatedly breaks down the problem into smaller sub-problems, solves those and then combines the solutions.

The graph shows that a bubble sort is far slower at sorting lists of more than 1000 items, but for smaller lists the time difference is too small to be of importance.

As the bubble sort algorithm is easier to code, it could be beneficial to use it for smaller lists of less than 1000 items.

SUBJECT VOCABULARY

brute force an algorithm design that does not include any techniques to improve performance, but instead relies on computing power to try all possibilities until the solution to a problem is found

divide and conquer an algorithm design that works by dividing a problem into smaller and smaller sub-problems, until they are easy to solve. The solutions to these are then combined to give a solution to the complete problem

SEARCHING ALGORITHMS

To find a specific item in a list involves carrying out a search. Like sorting, some methods of searching are more efficient than others.

LINEAR SEARCH

A linear search is a simple algorithm and not very sophisticated. It simply starts at the beginning of the list and goes through it, item by item, until it finds the item it is looking for or reaches the end of the list without finding it.

A linear search is **sequential** because it moves through the list item by item.

GENERAL VOCABULARY

sequential following in order, one after the other

LINEAR SEARCH

1 Start at the first item in the list.
2 Compare the item with the search item.
3 If they are the same, then stop.
4 If they are not, then move to the next item.
5 Repeat 2 to 4 until the end of the list is reached.

BINARY SEARCH

Like a merge sort, a binary search uses a ‘divide and conquer’ method.

In a binary search the middle or **median** item in a list is repeatedly selected to reduce the size of the list to be searched – another example of recursion. If the selected item is too high or too low, then the items below or above that selected item can be searched.

SUBJECT VOCABULARY

median the middle number when the numbers are put in ascending or descending order (e.g. if there are 13 numbers, then the 7th number is the median). If there are an even number of items in a list, the median is the mean of the middle two numbers (e.g. if there are 10 numbers, add the 5th and 6th numbers together and divide the result by 2). In a binary search, the higher of the two numbers would be chosen

To use this method, the list must be sorted into ascending or descending order. It will not work on an unsorted list.

DID YOU KNOW?

You have probably used a binary search method when trying to guess a number between two limits. If you are asked to guess a number between 1 and 20 you will probably start at 10, the middle number. If you are told this is too high, you will then guess 5, the middle number between 1 and 10, and then repeat this method until you find the correct one.

BINARY SEARCH (ITEMS IN ASCENDING ORDER)

1 Select the median item of the list.
2 If the median item is equal to the search item, then stop.
3 If the median is too high, then repeat 1 and 2 with the sub-list to the left.
4 If the median is too low, then repeat 1 and 2 with the sub-list to the right.
5 Repeat steps 3 and 4 until the item has been found or all of the items have been checked.

WORKED EXAMPLE

In this list, the search item is the number 13.

3	13	24	27	31	39	45	60	69	SELECT THE MEDIAN NUMBER.

As this is too high, the sub-list to the left of the median must be searched.

3	13	24	27	THE MEDIAN NUMBER OF THIS SUB-LIST IS NOW SELECTED.

This is again too high and so the sub-list to the left must be searched.

3	13	THE MEDIAN NUMBER IS NOW THE SEARCH ITEM.

▲ **Figure 1.11** Binary search including sub-lists

In this example, it took three attempts to find the search item. A linear search would have accomplished this with only two attempts.

SKILLS PROBLEM SOLVING

ACTIVITY 11

USING BINARY SEARCH

Display the stages of a binary search, as in the worked example above, to find the number 13 in this list.

3 9 13 15 21 24 27 30 36 39 42 54 69

Compare your results with those of others in your group. Are all your answers the same?

EFFICIENCY OF SEARCHING ALGORITHMS

In the example on page 20, the linear search was more efficient because it only had to carry out two comparisons instead of the three for a binary search. But is this always the case?

Searching algorithms can be compared by looking at the ‘worst case’ and the ‘best case’ for each one.

WORKED EXAMPLE

If you wanted to find a particular item in a list of 1000 items, these are the best- and worst-case scenarios for the linear search and binary search algorithms.

Linear search

A linear search starts at the first item and then works through sequentially.

The best case would be if the item is first in the list.

The worst case would be if it is last in the list.

Therefore, in this example the average would be 500 comparisons.

Binary search

The best case would be if the item is in the median position in the list. The search would require only one comparison.

For the worst case it would have to choose the following medians until it finally hit the target.

(This assumes that the target is always smaller than the median.)

Attempt	Median
1	500
2	250
3	125
4	63
5	32
6	16
7	8
8	4
9	2
10	1

Therefore, the worst case for the binary search is ten comparisons.

The binary search is therefore far more efficient than the linear search.

So, should a binary search be used every time? That depends on the circumstances. The binary search has one great disadvantage. The list must be already sorted into ascending or descending order. Therefore, a sorting algorithm must be applied before the search.

If the list is to be searched just once then a linear search would be better, but if there is a large list that will be searched many times then sorting the list and using a binary search would be better. Once the list has been sorted, new items can be inserted into the correct places.

CHECKPOINT

Strengthen

SKILLS CRITICAL THINKING

S1 What are the differences between the 'bubble sort' and 'merge sort' algorithms?

SKILLS DECISION MAKING

S2 How does a binary search algorithm find the search item?

Challenge

SKILLS CRITICAL THINKING

C1 When might a linear search be preferable to a binary search, even if the binary search algorithm is more efficient?

How confident do you feel about your answers to these questions? If you're not sure you answered them well, try the following activities again.

- For S1 have a look at the 'Sorting algorithms' section.
- For S2 have a look at the 'Binary search' section.

SUMMARY

- There are many algorithms for sorting and searching data.
- The choice of algorithm depends on the data that is to be processed.
- If only a small amount of data needs to be processed, then a simpler, but less efficient search algorithm may be the best choice. The time difference of the search or sort time will be negligible.

4 DECOMPOSITION AND ABSTRACTION

When computer scientists attempt to solve problems by producing algorithms and coding them into programs, they approach the problems in a particular way. This method has been given the name 'computational thinking'. Before they create a structured solution or algorithm and code it into a program, they must define and analyse the problems. This section will introduce two of the **thought processes** they use – decomposition and abstraction.

GENERAL VOCABULARY

thought process the act of using your mind to consider or think about something

LEARNING OBJECTIVES

- Analyse a problem, investigate requirements (inputs, outputs, processing, initialisation) and design solutions
- Decompose a problem into smaller sub-problems
- Understand how abstraction can be used effectively to model aspects of the real world
- Program abstractions of real-world examples

PROBLEM SOLVING

The tasks of a computer scientist include defining and analysing problems; creating structured solutions – algorithms; and coding the solutions into a form that can be implemented by a computer. These tasks are part of what is known as **computational thinking**.

One of the skills required for computational thinking is algorithm design (which we've covered in detail in this unit). If there is a fault in the algorithm design, then the program will not work, however good a coder you are. Two other skills are **decomposition** and **abstraction**.

SUBJECT VOCABULARY

computational thinking the thought processes involved in formulating problems and their solutions so that the solutions are represented in a form that can be effectively carried out by a computer

decomposition breaking a problem down into smaller, more manageable parts, which are then easier to solve

abstraction the process of removing or hiding unnecessary detail so that only the important points remain

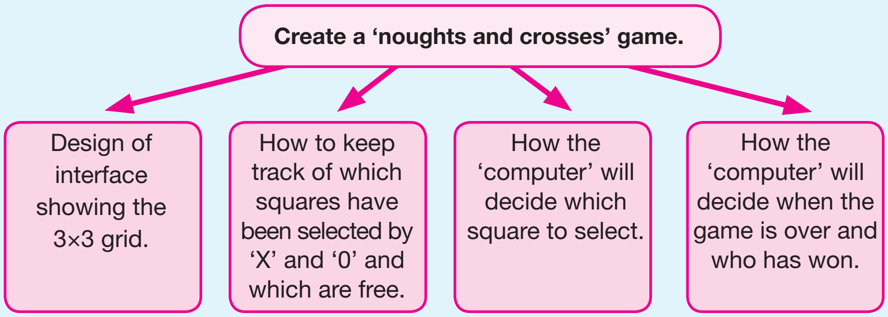
DECOMPOSITION

Decomposition is usually the first step in the problem-solving process. Once a problem has been broken down and the sub-problems have been identified, algorithms can be developed to solve each of them.

Decomposition means that sub-problems can be worked on by different teams at the same time. As smaller algorithms are developed for each sub-problem, it is easier to spot and correct errors. When the algorithm is developed into a program, code can be used again.

WORKED EXAMPLE

A student has been set the task of creating a computer version of the game 'noughts and crosses' (also known as 'tic-tac-toe') where a user plays against the computer.



Create a 'noughts and crosses' game.

Design of interface showing the 3×3 grid.

How to keep track of which squares have been selected by 'X' and '0' and which are free.

How the 'computer' will decide which square to select.

How the 'computer' will decide when the game is over and who has won.

▶ **Figure 1.12** Sub-problems to be solved to create a noughts and crosses computer program

The diagram shows some of the sub-problems that must be solved in order to solve the complete problem and create a version of the game.

ABSTRACTION

We use abstraction all the time in our daily lives. We **abstract** the essential features of something so that we can understand what people are trying to communicate.

GENERAL VOCABULARY

abstract to remove something from somewhere

Somebody might say, 'I was walking down the street when I saw a cat'. You immediately understand what they mean by 'street' – probably a road with a pavement and houses or shops along the side of it. Similarly, you can picture the cat – a small animal with fur, four legs and a tail. An animal that is basically 'cattish'. You have extracted the basic properties of animals called cats so that you can recognise one when you see one, or imagine one when somebody talks about a cat.

▲ Is this the street and cat you imagined?

What you imagine is very unlikely to match the actual street and cat that the person experienced. But, because of our ability to abstract, the person did not have to go into unnecessary detail about exactly where they were and what they saw. They wouldn't get very far with the story if they did.

When we create algorithms, we abstract the basic details of the problem and represent them in a way that a computer is able to process.

WORKED EXAMPLE

Yasmin is designing a computer version of a game in which users have to throw a die to find out their number of moves.

In the computer game, the users can't have an actual die, so she will have to design a 'pretend' or virtual die that behaves in exactly the same way as a real-life die.

Yasmin will have to use her powers of abstraction to work out the essential features of a die and then represent them in computer code.

To represent the die, she will have to create a routine that will select a random number from 1 to 6 because that's what a die does.

Yasmin has used abstraction to **model** a real-life event.

GENERAL VOCABULARY

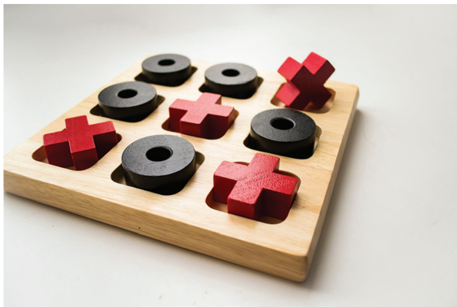
model to make a simple version of something to show how it works

LEVELS OF ABSTRACTION

There are different levels or types of abstraction. The higher the level of abstraction, the less detail is required. We use abstraction all the time in accomplishing everyday tasks.

When programmers write the 'print' command they do not have to bother about all of the details of how this will be accomplished. They are removed from them. They are at a certain level of abstraction.

A driver turning the ignition key to start a car does not have to understand how the engine works or how the spark to ignite the petrol is generated. It just happens and they can simply drive the car. That is abstraction.



▲ A game of noughts and crosses

AN EXAMPLE – NOUGHTS AND CROSSES

Figure 1.12 showed some of the sub-problems that the problem of creating a noughts and crosses game could be divided into. The following could be written at a high level of abstraction.

- The computer goes first. Then the user. This continues until either one wins, or all of the squares have been used.

Immediately a pattern can be recognised – a loop will be needed.

Inputs and outputs

The following inputs from the user will be needed.

- Start the game.
- Entries for the user.
- Select a new game or finish.

The following outputs will be needed.

- A message to inform the user when it is their turn.
- A message to inform the user if they try to select a square that has already been used.

- A message to inform the user if the game is a draw.
- A message to inform the user if they or the computer has won.
- A message to ask the user if they want to play another game or want to finish.

Processing and initialisation

The following processing will be needed.

- Set up the grid with the nine squares.
- Initialise all variables to a start value.
- Decide which square the computer will select.
- Allow the user to select a square.
- Check if the user has selected an already used square.
- Check if the computer or the user has won.
- Check if all squares have been used and the game is a draw.
- Allow the user to select a new game or finish.

The solution is still at a high level of abstraction and more details will need to be added.

For example, the programmer will need to decide how the game will record which player has selected each square; how the computer will decide which square to select; how the game will decide if the computer or the user has won.

The programmer will have to go into more and more detail or move to lower levels of abstraction.

Eventually, the programmer will be able to design an algorithm for the game and code it using a high-level programming language such as Python or Java. Even before they start to implement the game, they will need to plan how they will test the finished program to make sure that it works correctly, what test data they will use and what outcomes it should produce.

CODING AN ALGORITHM

High-level programming languages make it easier for a programmer to write code. Unfortunately, the processor that has to execute the program cannot understand the language it is written in. It therefore needs a translator to translate the code into the only language it does understand – a stream of 1s and 0s.

These high-level languages are therefore at a high level of abstraction – very far removed from the actual language of a computer.

The processing can be split into parts. For example, in the example of the noughts and crosses game there could be separate algorithms for:

- deciding where the computer should make its next selection – it could be called ‘computer entry’
- checking if the computer or the player has won – it could be called ‘check if won’
- checking if there are any empty squares left – it could be called ‘check draw’.

These separate algorithms could be used when they are needed. It is efficient because it means that the same code doesn’t have to be rewritten whenever it is needed.

These items of code are called **subprograms**.

In Unit 2, we'll look in detail at how subprograms are used to reduce the complexity of programs and to make them easier to understand.

In the die example above, the designer could write a subprogram called 'die' that generates a random number from 1 to 6. In the main program the designer could just call the 'die' subprogram without having to think about how to implement it each time.

SUBJECT VOCABULARY

subprogram a self-contained module of code that performs a specific task. It can be 'called' by the main program when it is needed

SKILLS REASONING, PROBLEM SOLVING

GENERAL VOCABULARY

decompose to divide something into smaller parts

ACTIVITY 12

CREATING A QUIZ GAME

In a game, each player spins a wheel that is divided into four colours: red, blue, green and yellow. Each player has to answer a question on a particular topic depending on the colour next to a pointer when the wheel stops. Red is for science, blue for history, green for general knowledge and yellow for geography. A player scores two points if they answer correctly on the first attempt and one point for being correct on the second attempt. The first player to reach 30 points is the winner.

Your task is to design a computer version of the game for up to four players. You must analyse the problem and list all of the requirements; **decompose** the problem, list all the sub-problems and write a brief description of each; list all of the input, output and processing requirements.

One of the requirements that will have to be modelled is the spinning of the wheel. Using a written description and pseudocode shows how this could be done.

EXTEND YOUR KNOWLEDGE

Complete the noughts and crosses game by coding it in the language you are studying. See if you can end up with a working game.

CHECKPOINT

Strengthen

SKILLS CRITICAL THINKING

S1 What is meant by 'decomposition'? What are the benefits it provides for programmers?

SKILLS CRITICAL THINKING

S2 What is meant by 'abstraction'?

SKILLS CRITICAL THINKING, PROBLEM SOLVING

S3 A student is creating a model of the cost of a car journey, a real-world problem. Write down the important items she will have to include in her model and how they interact to calculate the cost of the journey.

Challenge

SKILLS CRITICAL THINKING

C1 Can you think of some examples when 'decomposition' and 'abstraction' are used when solving a problem?

SKILLS PROBLEM SOLVING

C2 In your own words, can you explain what is meant by 'computational thinking'?

SKILLS DECISION MAKING

C3 Explain how we use abstraction in our daily lives when we are communicating with others.

How confident do you feel about your answers to these questions? If you're not sure you answered them well, re-read the sections about 'Decomposition' and 'Abstraction'.

DID YOU KNOW?

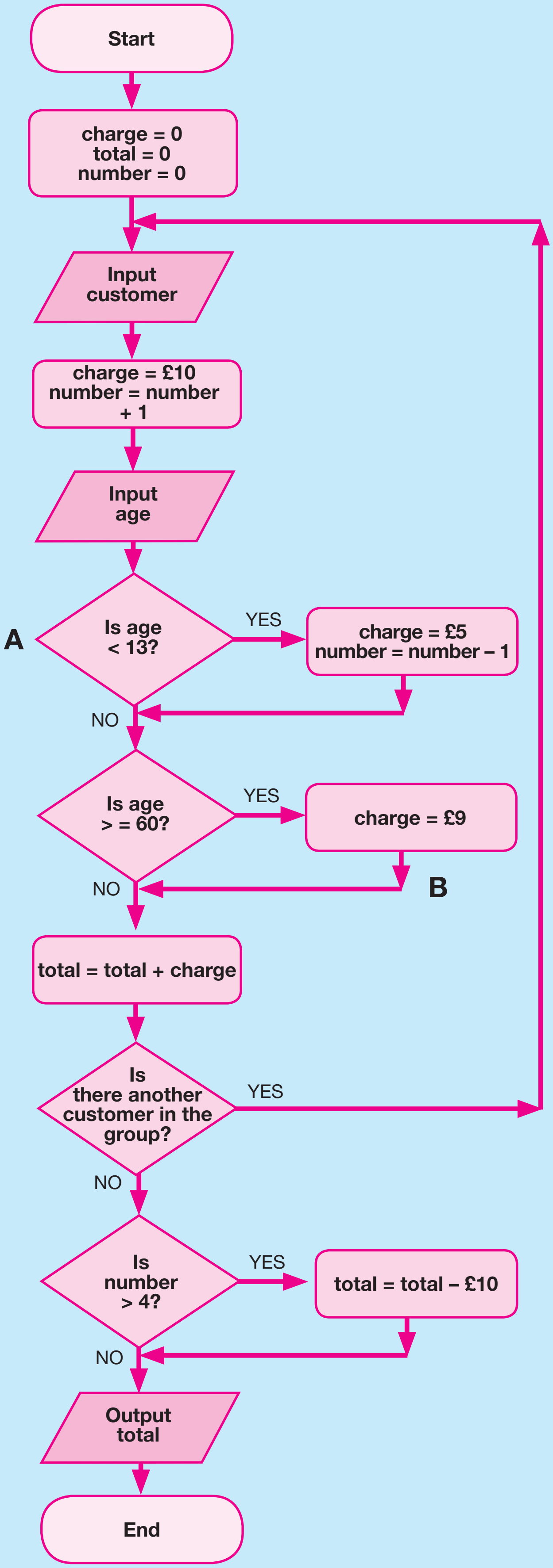
Computer models or 'simulations' of real life are widely used. It is far cheaper and safer to train pilots on flight simulators than on real aircraft. They are also used in weather forecasting, designing and testing new cars and bridges and even teaching people to drive. Computer models are used by all governments around the world to experiment with the short and long-term effects of changing variables such as tax rates on the economy.

SUMMARY

- Computational thinking is an approach to solving problems, such as traffic flow in a city, or how many products a business needs to make and sell to produce a profit. It includes techniques such as decomposition and abstraction.
- Problems are easier to solve if they are decomposed into smaller sub-problems.
- Abstraction is used to remove unnecessary detail to make a problem easier to understand and solve. For example, when modelling traffic flow in a city, unnecessary details could include the colours of the vehicles or the ages of the drivers.
- When designing a solution to a problem the inputs, outputs and processing requirements should be identified at the outset.

UNIT QUESTIONS

▶ **Figure 1.13** Flowchart of an algorithm showing charges in a theme park



The flowchart in Figure 1.13 displays an algorithm used by Holiday Theme Parks Limited.

SKILLS PROBLEM SOLVING A02

1 Explain how the algorithm calculates the total amount that should be paid. **(4)**

SKILLS PROBLEM SOLVING A02

2 Give two variables that are used in the algorithm. **(2)**

SKILLS PROBLEM SOLVING A02

3 In the flowchart, two of the constructs are labelled A and B. State the type of each construct. **(2)**

SKILLS PROBLEM SOLVING A02

4 The Lim family is visiting the park. The family consists of two children, one aged 8 and one aged 10, their two parents and their grandfather, who is aged 65. Use the algorithm to calculate how much the family should have to pay for entry. **(4)**

HINT

Spend some time studying the algorithm to ensure that you fully understand it.

In **1** you are asked to 'explain' how the algorithm works. A longer answer is required, which includes all of the stages of the algorithm. Use the correct terms to explain the constructs.

In **2** and **3** short answers are sufficient.

In **4** calculate the charge for each person using the rules of the algorithm. Then calculate the overall charge and check to see if the family qualifies for a group discount.

SKILLS PROBLEM SOLVING A02

5 A teacher has stored learner surnames as shown below.

Marek	Jackson	Bachchan	Wilson	Abraham	French	Smith

Identify the stages of a bubble sort when applied to this data. **(5)**

SKILLS PROBLEM SOLVING A02

6 The teacher has a sorted list of names from another class as shown below.

Azikiwe	Bloom	Byrne	Davidson	Gateri	Hinton	Jackson	Linton	Smith	Wall

Identify the stages of a binary search to find the name 'Jackson' when applied to this list. **(4)**

HINT

These questions are testing knowledge of the sort and search algorithms.

In question **5**, the answer should be set out to show how the data is progressively sorted using the bubble sort and the result of each pass should be shown.

Question **6** is to check that you know that this is a **recursive method** where the median is repeatedly selected.

SUBJECT VOCABULARY

recursive method a recursive method calls a function over and over until a required goal is met

SKILLS PROBLEM SOLVING AO3

7 Create an algorithm to calculate the cost of sending a parcel.

If the weight of the parcel is 2 kg or under then the standard charge is $3. There is then a charge of $2 for each extra kilogram up to 10 kg. After 10 kg the charge per extra kilogram is $3.

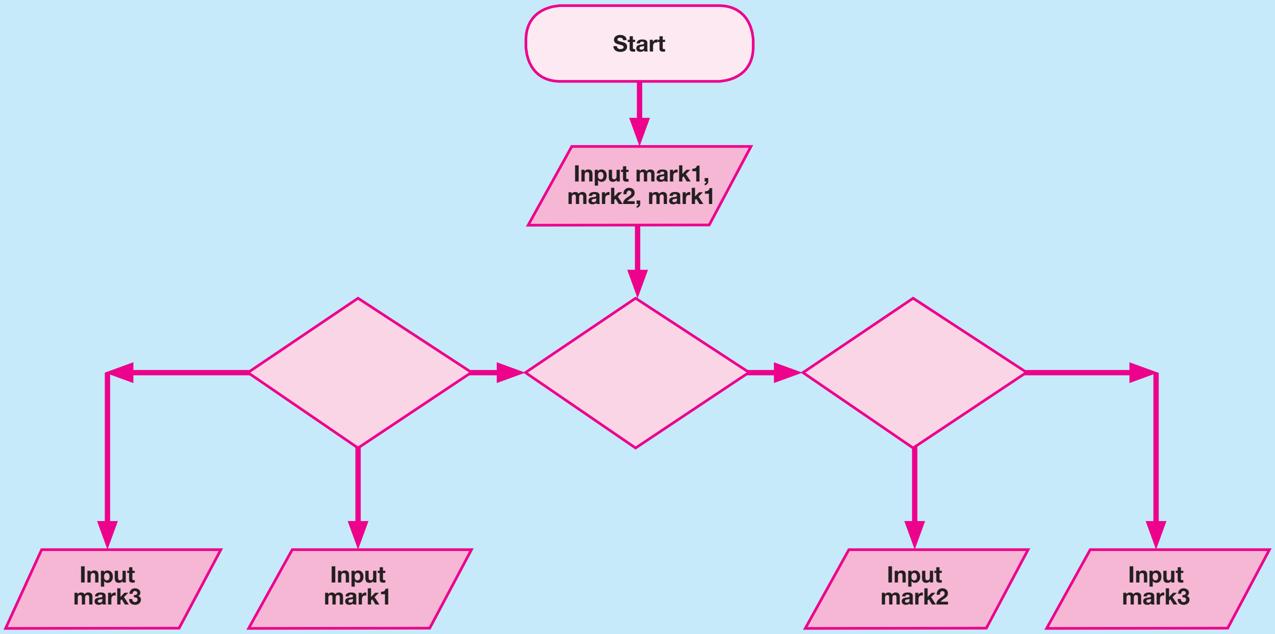
a Display your algorithm as a flowchart. **(5)**

b Construct your algorithm as pseudocode. **(5)**

SKILLS PROBLEM SOLVING AO3

8 A learner hands in three homework assignments, which were each given a mark out of 10. All of the marks were different. The following is part of an algorithm to find the highest mark but some of the decision symbols are empty.

Complete the decision symbols and add ‘YES’ and ‘NO’ labels where required. **(6)**



SKILLS PROBLEM SOLVING AO2

9 A list is made up of the numbers 4, 1, 2, 6, 3, 5.
Identify the steps involved when sorting this list using a bubble sort algorithm. **(2)**

14
15
nOutC
22
23
1
2
22
23
24
25

UNIT 2 PROGRAMMING

Assessment Objective 1

Demonstrate knowledge and understanding of the key principles of computer science

Assessment Objective 2

Apply knowledge and understanding of key concepts and principles of computer science

Assessment Objective 3

Analyse problems in computational terms:

- to make reasoned judgements
- to design, program, test, evaluate and refine solutions

In this unit you will learn about translating algorithms into programs written in a high-level language using constructs, such as variables and arrays. You will also look at how programs can be structured using subprograms and how data input can be validated to ensure that it is reasonable. Humans are not perfect. Most programs have one or two errors and you will look at how programs can be tested, and errors corrected.

5 DEVELOP CODE

PROGRAMMING LANGUAGES

Throughout the book, we have coloured the different programming languages in order for you to easily find the one you are looking for.

The colours are:

`Pearson Edexcel pseudocode`

`Python`

`Java`

`C#`

Once an algorithm has been developed to solve a particular problem, it has to be coded into the programming language that the developer is using. This usually means that the written descriptions, flowcharts and pseudocode have to be converted into actual programming code.

As you will be expected to understand, use and edit Pearson Edexcel `pseudocode` in the examination, examples will be given in the pseudocode as well as in `Python`, `Java` and `C#`.

LEARNING OBJECTIVES

- Explain the difference between algorithms and programs
- Code an algorithm in pseudocode and a high-level programming language
- Describe the characteristics of data types and select appropriate data types for variables
- Use sequencing, selection and iteration constructs in your programs

ALGORITHMS AND PROGRAMS

As you learnt in Unit 1, an algorithm is a precise method of solving a problem. It consists of a sequence of unambiguous, step-by-step instructions. A program is an algorithm that has been converted into program code so that it can be **executed** by a computer. A well-written algorithm should be free of logic errors and easy to code in any high-level language.

SUBJECT VOCABULARY

execution the process by which a computer carries out the instructions of a computer program

As part of this course you will learn to write programs in a high-level programming language. All high-level programming languages are like natural human languages, which makes them easier for humans to read and write but impossible for computers to understand without the help of a translator. You will learn more about how a program written in a high-level language is translated into machine code – the language of computers – in Unit 4. The aim of this unit is to develop your programming skills.

▶ A computer programmer at work writing code

DATA TYPES

GENERAL VOCABULARY

determine to discover facts about something

As you learnt in Unit 1, algorithms use variables (named memory locations) to store values. Variables have a variety of uses. For example, controlling the number of times a loop is executed, **determining** which branch of an IF statement is taken, keeping running totals and holding user input.

When algorithms are converted into programs, the computer needs to be told what type of data is stored in each variable. Every programming language has a number of built-in data types.

DATA TYPE	DESCRIPTION	EXAMPLE	EXAMPLES OF USE

integer	Used to store whole numbers without a fractional part	30	age = 30
			number = 5

real or float	Used to store numbers with a fractional part (decimal place). Real numbers are sometimes referred to as floats (short for floating point)	25.5	weight = 25.5
			price = 12.55

Boolean	Only has two possible values: True or False	False	correct = False
			lightOn = True

character*	A character can be a single letter, a symbol, a number or even a space. It is one of the four basic data types	‘m’	gender = ‘m’
			char = ‘:’

string	A set of characters which can include spaces and numbers and are treated as text rather than numbers	‘the computer’	name = ‘Catherine’
			type = ‘liquid’

*Python does not have a character data type.

▲ **Table 2.1** Common data types

GENERAL VOCABULARY

assign to give somebody a particular task

HINT

Although you don’t have to declare the data types of variables in pseudocode, it is a good idea to do so. You can simply list the variables and their data types at the start of an algorithm, for example:

```
INTEGER age
REAL weight
BOOLEAN correct
CHARACTER gender
```

You can also specify the data type of a variable in the RECEIVE statement, for example:

```
RECEIVE age FROM (INTEGER) KEYBOARD
RECEIVE price FROM (REAL) KEYBOARD
```

When writing pseudocode, you don’t have to specify the data types of variables. However, data types become much more important once you start programming in a high-level language. This is because the data type of a variable determines the operations that can be performed on it.

For example, the result of multiplying a value by 5 differs according to its data type.

integer	8 * 5 = 40
real	8.0 * 5 = 40.0
character	‘8’ * 5 = ‘88888’

The method of declaring variables differs between programming languages. Some languages, such as Python, automatically select the appropriate data type for a variable based on the data **assigned** to it. Others, such as Java and C#, require the data type of variables to be declared before the variables can be used.

VARIABLE INITIALISATION

When a variable is declared, the computer gives it a location in its memory. Initially, this location is empty, so before a variable can be used it has to be given a value.

SUBJECT VOCABULARY

initialise to set variables to their starting values at the beginning of a program or subprogram

initialisation the process of assigning an initial value to a variable

assignment statement the SET…TO command is used to initialise variables in pseudocode, for example:

```
SET anotherGo TO 0
SET correct TO False
```

You can put an initial value into a variable by:

- initialising it when the program is run (e.g. `SET total TO 0`, in pseudocode)
- reading a value from a keyboard or other device (e.g. `RECEIVE admissionCharge FROM (INTEGER) KEYBOARD`).

Once a variable has been **initialised** an **assignment statement** is used to change its value (e.g. `SET total TO total + admissionCharge`).

In Python this would be:

```
total = 0
total = total + admissionCharge
```

In Java this would be:

```
Scanner scan = new Scanner(System.in);
int admissionCharge = scan.nextInt();
int total = 0;
total = total + admissionCharge;
```

In C#, variables must be declared before use. When you declare a variable, you need to state the data type that the variable will store, for example:

```
// declare a variable called total that will be used to
store floating point numbers and assign the value 0.0 to it
float total = 0.0;
// add the value stored in the variable admissionsCharge
to the total
total = total + admissionsCharge;
```

GENERAL VOCABULARY

intended designed for a certain purpose

If a variable, such as a loop counter, is **intended** to hold a running total, then it should always be initialised to a starting value. Some programming languages won't execute if the programmer fails to do this; others will do so but may well produce some unexpected results.

SKILLS CRITICAL THINKING, DECISION MAKING

ACTIVITY 1

DATA TYPES

1 Investigate what data types are available in the high-level language you are studying. Produce a table similar to Table 2.1 to give a summary of your findings.

2 What do you think is an appropriate data type for each of these items?
 a the test score of an individual learner
 b the average score for a group of learners
 c whether or not the pass mark for the test has been achieved.

3 Look back over the algorithms you wrote in Unit 1 and find instances of variable initialisation.

SUBJECT VOCABULARY

type coercion the process of converting the value stored in a variable from one data type to another

TYPE COERCION

Sometimes the data type of a variable gets changed during program execution. This is known as **type coercion**. For example, if an integer value and a real value are used in an arithmetic operation, the result will always be a real.

In Python, type coercion is done automatically, for example, the output from the following program is 3.25.

```
x = 1
y = 2.25
z = x + y
print(z)
```

Type coercion is also automatic in Java:

```
int x = 1;
double y = 2.25;
double z = x + y;
System.out.print(z);
```

In C# this is generally referred to as casting. As with Python this can be automatic for many data type conversions and is known as implicit casting. For more complicated conversions you may need to research explicit casting.

One frequently used type of explicit casting is converting from a string to a numeric value. This is used when getting input from a user as `Console.Readline`. For example, the following C# code will output 3.25 and demonstrates automatic type coercion.

```
int x;              // declares a variable called x that will store integers
double y, z;        // declares two variables called y and z that will store
                    floating point numbers
x = 1;
y = 2.25;
z = x + y;
Console.WriteLine(z);
```

SKILLS CRITICAL THINKING, PROBLEM SOLVING, DECISION MAKING

ACTIVITY 2

MONITORING VISITOR NUMBERS

A theme park uses a program to monitor the number of people entering and exiting the park. The maximum number of visitors at any one time must not exceed 10 000. When the number of people in the park reaches the maximum, a ‘Park Full’ message is displayed at the entrance gate. Children can visit the park free of charge. Adults must pay AED 125 admission. The program records the amount of money collected at the gate.

1 What are the variables needed in the program?

2 Select an appropriate data type for each variable and constant.

COMMAND SEQUENCE, SELECTION AND ITERATION

In Unit 1 you learnt that the three key building blocks of algorithms are command sequence, selection and iteration. In this unit, you will have the opportunity to implement these constructs in the high-level programming language you are studying.

SKILLS CRITICAL THINKING, PROBLEM SOLVING, CREATIVITY

ACTIVITY 3

UNDERSTANDING ALGORITHMS

Read the following algorithm written in pseudocode and then answer the questions below.

```
RECEIVE number1 FROM (INTEGER) KEYBOARD
RECEIVE number2 FROM (INTEGER) KEYBOARD
SET result1 TO number1 / number2
SEND result1 TO DISPLAY
SET result2 TO number1 MOD number2
SEND result2 TO DISPLAY
SET result3 TO number1 DIV number2
SEND result3 TO DISPLAY
```

1 What does this algorithm do?

2 What is the output of the algorithm, given the following inputs:
   a 4, 2
   b 10, 3
   c 20, 6?

Implement this algorithm in the high-level language you are studying.

SELECTION

The selection construct is used to create a branch in a program. The computer selects which branch to follow based on the outcome of a condition, using an `IF…THEN…ELSE` statement. For example, in pseudocode:

```
IF day = 'Saturday' OR day = 'Sunday' THEN
  SET alarm TO 11
ELSE
  SET alarm TO 8
END IF
```

A standard `IF…THEN…ELSE` statement provides two alternatives. If there are more than two, then in Pearson Edexcel pseudocode a nested `IF` must be used. However, many high-level programming languages have an additional built-in selection construct that does away with the need for a **nested IF statement**.

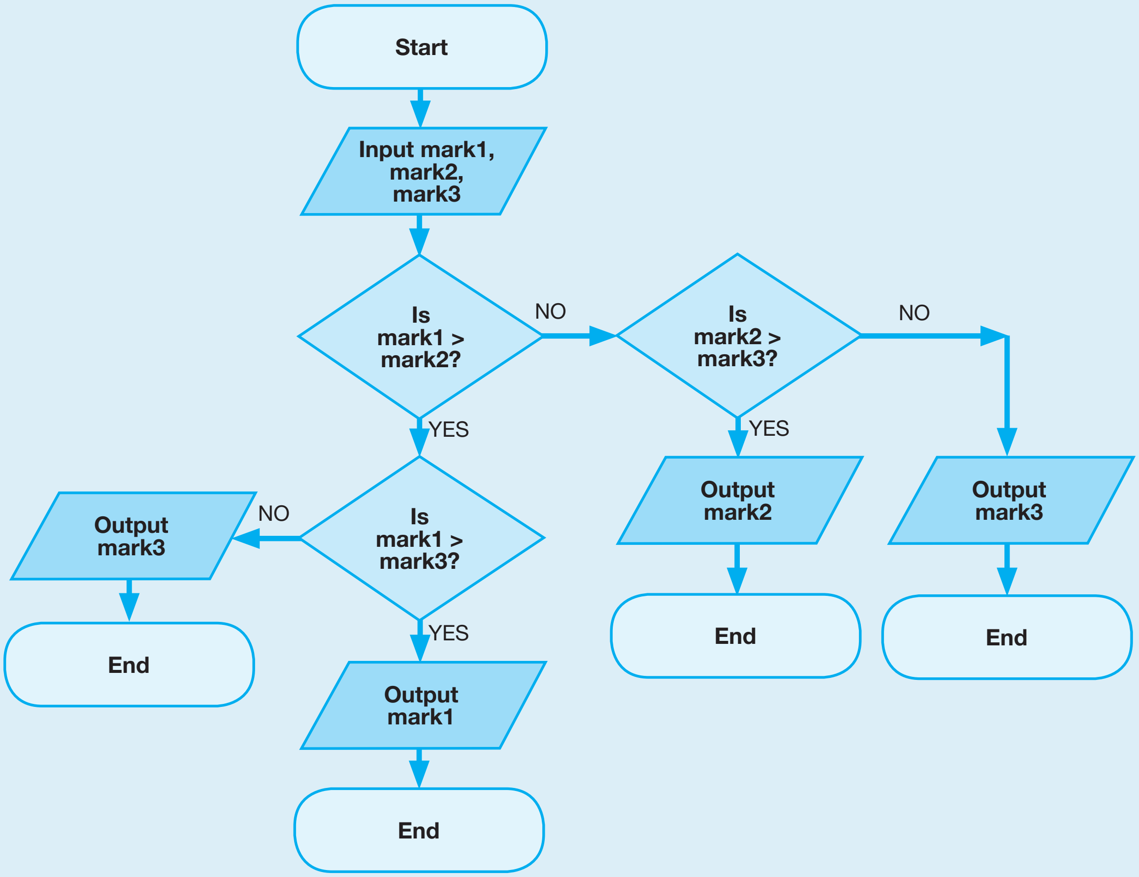
SUBJECT VOCABULARY

nested IF statement a nested IF statement consists of one or more IF statements placed inside each other. A nested IF is used where there are more than two possible courses of action

WORKED EXAMPLE

A learner handed in three homework assignments, which were each given a mark out of 10. All the marks were different. Write an algorithm that would print out the highest mark.

Figure 2.1 shows the algorithm expressed as a flowchart.



▲ **Figure 2.1** Flowchart of an algorithm to print out the highest homework mark

HINT

When you are creating nested `IF` statements, you have to ensure that each one is completed with an `END IF` statement at the correct indentation level. Some programming languages do not need an `END IF` statement and just use the indentation levels to indicate when statements are grouped.

In Pearson Edexcel pseudocode, this algorithm could be expressed as:

```
RECEIVE mark1 FROM KEYBOARD
RECEIVE mark2 FROM KEYBOARD
RECEIVE mark3 FROM KEYBOARD
IF mark1 > mark2 AND mark1 > mark3 THEN
    SEND mark1 TO DISPLAY
ELSE
    IF mark2 > mark1 AND mark2 > mark3 THEN
        SEND mark2 TO DISPLAY
    ELSE
        IF mark3 > mark1 AND mark3 > mark2 THEN
            Send mark3 TO DISPLAY
        END IF
    END IF
END IF
```

In Python, Java and C# this does not have to be done as they have an ‘else if’ statement.

In Python the ‘else if’ statement is `elif` and the algorithm above could be:

```
mark1 = input(‘Please enter the first mark’)
mark2 = input(‘Please enter the second mark’)
mark3 = input(‘Please enter the third mark’)

if mark1 > mark2 and mark1 > mark3:
    print(mark1)
elif mark2 > mark1 and mark2 > mark3:
    print(mark2)
elif mark3 > mark1 and mark3 > mark2:
    print(mark3)
```

Here is the algorithm in Java:

```
import java.util.*;
class Main {
  public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
    System.out.print(“Please enter the first mark: ”);
    int mark1 = scanner.nextInt();
    System.out.print(“Please enter the second mark: ”);
    int mark2 = scanner.nextInt();
    System.out.print(“Please enter the third mark: ”);
    int mark3 = scanner.nextInt();
    scanner.close();

    if (mark1 > mark2 && mark1 > mark3) {
      System.out.print(mark1);
    } else if (mark2 > mark1 && mark2 > mark3) {
      System.out.print(mark2);
    } else if (mark3 > mark1 && mark3 > mark2) {
      System.out.print(mark3);
    }
  }
}
```

And in C#:

```
string mark1String, mark2String, mark3String;
int mark1, mark2, mark3;

Console.WriteLine(“Please enter the first mark”);
mark1String = Console.ReadLine();   // read input into string
mark1 = int.Parse(mark1String);     // convert string to integer

Console.WriteLine(“Please enter the second mark”);
mark2String = Console.ReadLine();   // read input into string
mark2 = int.Parse(mark2String);     // convert string to integer
```

```
Console.WriteLine(“Please enter the third mark”);
mark3String = Console.ReadLine();   // read input into string
mark3 = int.Parse(mark3String);     // convert string to integer

if (markl > mark2 && markl > mark3)
{
    Console.WriteLine(markl);
}
else if (mark2 > markl  && mark2 > mark3)
{
   Console.WriteLine(mark2);
}
else if (mark3 > markl && mark3 > mark2)
{
   Console.WriteLine(mark3);
}
```

These examples use **relational** and **logical operators**.

SUBJECT VOCABULARY

relational operator an operator that tests the relationship between two entities

logical operator a Boolean operator using AND, OR and NOT

RELATIONAL OPERATORS

The relational operators are used to compare two values and in Python, Java and C# are all the same.

RELATIONAL OPERATOR	PYTHON, JAVA, C#
Equal to	==
Greater than	>
Greater than or equal to	>=
Less than	<
Less than or equal to	<=
Not equal to	!=

ACTIVITY 4

Look at the following algorithm and answer the questions.

```
IF score <= highScore THEN
    SEND ‘You haven’t beaten your high score.’ TO DISPLAY
ELSE
    SEND ‘You’ve exceeded your high score!’ TO DISPLAY
END IF
```

What is the output of the algorithm when

- `score` = 5 and `highScore` = 10?
- `score` = 20 and `highScore` = 10?
- `score` = 15 and `highScore` = 15?

LOGICAL OPERATORS

AND

If two conditions are joined by the `AND` operator, then they must both be true for the whole statement to be true.

OR

If two conditions are joined by the `OR` operator, then either one must be true for the whole statement to be true.

NOT

The `NOT` operator reverses the logic of the `AND` and `OR` statements. The statement `IF A = 3 AND B = 6` will be true only if the conditions are met, i.e. A and B are both equal to the values stated.

The statement `IF NOT (A = 3 AND B = 6)` will be true whenever both A and B are NOT equal to the values stated, i.e. either or both are not equal to those values.

Logical operators in high-level languages.

LOGICAL OPERATOR	PYTHON	JAVA	C#
AND	and	&	&&
OR	or	|	||
NOT	not	!	!

ACTIVITY 5

A driving school uses this rule to estimate how many lessons a learner will require.

- Every learner requires at least 20 lessons.
- Learners over the age of 18 require more lessons (two additional lessons for each year over 18).

Create a program in a high-level language that inputs a learner's age and calculates the number of driving lessons they will need.

LOOPS

A loop is another name for an iteration. Loops are used to make a computer repeat a set of instructions more than once. There are two types of loop: definite and indefinite.

A definite loop is used when you know in advance how often the instructions in the body of the loop are to be repeated. For example, if you want the computer to display a character on the screen for a fixed amount of time and then remove it.

An indefinite loop is used when the number of times a loop will need to be repeated is not known in advance. For example, if you want to give a user the option of playing a game as often as they want. Indefinite loops are repeated until a specified condition is reached.

Every programming language has a number of built-in loop constructs. You will need to explore the ones provided in the language you are studying.

DEFINITE ITERATION

This is used when the number of iterations, or turns of the loop, is known in advance.

In the Pearson Edexcel pseudocode there are two ways of doing this using `REPEAT…END REPEAT` and `FOR…END FOR`.

An example of a `REPEAT` loop is shown below.

```
REPEAT 50 TIMES
  SEND ‘*’ TO DISPLAY
END REPEAT
```

Using a `FOR` loop, this would be:

```
FOR times FROM 1 TO 100 DO
    SEND ‘*’ TO DISPLAY
END FOR
```

`FOR` loops can also include a step so that the counting is not consecutive. A step is included in the following pseudocode example.

```
FOR times FROM 0 TO 100 STEP 25 DO
    SEND times TO DISPLAY
END FOR
```

The output would be 0, 25, 50, 75.

USE	EXAMPLE	RESULT

Using the ‘range’ command	`for x in range(6):`
	`    print(x)`
		0
		1
		2
		3
		4
		5

Stipulating a start number so that it does not begin at 0	`for x in range(2,6):`
	`    print(x)`
		2
		3
		4
		5

Using a step by adding a third number into the brackets	`for x in range(0,100,25):`
	`    print(x)`
	To also print 100 you would use:
	`for x in range(0,101,25):`
		0
		25
		50
		75

Printing items from a list	`aList = [‘red’, ‘blue’, ‘green’, ‘yellow’, ‘purple’, ‘orange’]`
	`for colour in aList:`
	`    print (colour)`
	The number of iterations does not have to be given as the length of the list is used.
		red
		blue
		green
		yellow
		purple
		orange

(continued)

(continued)

USE	EXAMPLE	RESULT

Leaving out items from the list	`aList = [‘red’, ‘blue’, ‘green’, ‘yellow’, ‘purple’, ‘orange’]`
	`for colour in aList:`
	`      if colour == ‘yellow’:`
	`            continue`
	
	`      print (colour)`
		red
		blue
		green
		purple
		orange

▲ **Table 2.2** Python

USE	EXAMPLE	RESULT

Using a `for` loop	`for(int x = 0; x < 6; x++) {`
	`  System.out.println(x);`
	`}`
		0
		1
		2
		3
		4
		5

Stipulating a start number so that it does not begin at 0	`for(int x = 2; x < 6; x++) {`
	`  System.out.println(x);`
	`}`
		2
		3
		4
		5

Using a step by adding an amount to increase by	`for(int x = 0; x < 100; x+=25) {`
	`  System.out.println(x);`
	`}`
	
	To also print 100 you would use:
	`for(int x = 0; x <= 100; x+=25) {`
	`  System.out.println(x);`
	`}`
		0
		25
		50
		75

Printing items from a list	`String[] aList = {“red”, “blue”, “green”, “yellow”, “purple”, “orange”};`
	`for(String colour: aList) {`
	`  System.out.println(colour);`
	`}`
		red
		blue
		green
		yellow
		purple
		orange

Leaving out items from the list	`String[] aList = {“red”, “blue”, “green”, “yellow”, “purple”, “orange”};`
	`for(String colour: aList) {`
	`  if(colour == “yellow”) {`
	`    continue;`
	`  }`
	`  System.out.println(colour);`
	`}`
		red
		blue
		green
		purple
		orange

▲ **Table 2.3** Java

USE	EXAMPLE	RESULT

Using a `for` loop	`for (int i = 0; i <= 5; i++)`
	`{`
	`    Console.WriteLine(i);`
	`}`
		0
		1
		2
		3
		4
		5

Stipulating a start number so that it does not begin at 0	`for (int i = 2; i <= 5; i++)`
	`{`
	`    Console.WriteLine(i);`
	`}`
		2
		3
		4
		5

Using a step by altering the increment value of a `for` loop	`for (int i = 0; i < 100; i+= 25)`
	`{`
	`    Console.WriteLine(i);`
	`}`
		0
		25
		50
		75

Printing items from an array*	`string[] colours = {“red”, “blue”, “green”, “yellow”, “purple”, “orange”};`
	
	`foreach (string i in colours)`
	`{`
	`    Console.WriteLine(i);`
	`}`
	
	The number of iterations does not have to be given as the length of the array is used.
		red
		blue
		green
		yellow
		purple
		orange

Leaving out items from the array*	`string[] colours = {“red”, “blue”, “green”, “yellow”, “purple”, “orange”};`
	
	`foreach (string i in colours)`
	`{`
	`    if (i == “yellow”)`
	`          {`
	`            continue;`
	`          }`
	`      else`
	`          {`
	`            Console.WriteLine(i);`
	`          }`
		red
		blue
		green
		purple
		orange

*See pages 62–68 for more information on arrays.

▲ **Table 2.4** C#

SKILLS PROBLEM SOLVING

ACTIVITY 6

Produce a program in a high-level language that asks a user to enter a start number and an end number and then outputs the total of all the numbers in the range. For example, if the start number was 1 and the end number was 10, the total would be 55 (10 + 9 + 8 + 7 + 6 + 5 + 4 + 3 + 2 + 1).

NESTED LOOPS

A **nested loop** is made of a loop within a loop. When one loop is nested within another, each iteration of the outer loop causes the inner loop to be executed until completion.

ACTIVITY 7

1 Python

Look at the following program and then answer the questions below.

```
for student in range(1, 21):
    sum = 0
    for mark in range(1, 6):
        nextMark = int(input(‘Please enter a mark’))
        sum = sum + nextMark
    averageMark = sum/5
    print(averageMark)
```

a What is the purpose of this program?

b Why is `int` used in the line `nextMark = int(input(‘Please enter a mark’))`?

HINT

You should initialise the variable total to zero before the start of the loop.

2 Java

Look at the following program and then answer the questions below.

```
import java.util.*;
class Main {
  public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
    for(int student = 1; student < 21; student++) {
      int sum = 0;
      for(int mark = 1; mark < 6; mark++) {
        System.out.print(“Please enter a mark: ”);
        int nextMark = scanner.nextInt();
        sum = sum + nextMark;
      }
      double averageMark = sum / 5;
      System.out.println(averageMark);
    }
    scanner.close();
  }
}
```

a What is the purpose of this program?

3 C#

```
int nextMark, sum, averageMark;
string markString;
sum = 0;
for (int mark = 1; mark <= 5; mark++)
{
    Console.WriteLine(“Please enter a mark”);
    markString = Console.ReadLine();
    nextMark = int.Parse(markString);
            sum = sum + nextMark;
      }

      averageMark = sum / 5;
      Console.WriteLine(averageMark);
```

a What is the purpose of the program?

b Why is `int.Parse` used in the line `nextMark = int.Parse(markString);`?

SKILLS PROBLEM SOLVING

ACTIVITY 8

Produce an algorithm that will print out the times tables (up to 12 times) for the numbers 2 to 12.

INDEFINITE ITERATION

An indefinite loop is used when the number of times a loop will need to be repeated is not known in advance. For example, if you want to give a user the option of playing a game as often as they want. Indefinite loops are repeated until a specified condition is reached.

Python

For indefinite iteration, Python uses the ‘while’ loop – something is done while a condition is met.

The following program asks a user to enter a number while the number is less than 20.

```
number = 1
while number <= 20:
    number = int(input(‘Please enter a number’))
print(‘You entered a number greater than 20’)
```

As soon as the number is greater than 20, the program breaks out of the loop and prints a message for the user.

Java

For indefinite iteration, Java uses a ‘while’ loop – instructions are repeated while a condition is met.

The following program asks a user to enter a number while the number is less than 20.

```
import java.util.*;
class Main {
  public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
    int number = 1;
    while(number <= 20) {
      System.out.print(“Please enter a number: ”)
      number = scanner.nextInt();
    }
    System.out.println(“You entered a number greater than 20”)
    scanner.close();
  }
}
```

C#

```
string numberString;
int number = 1;
while (number <=20)
{
    Console.WriteLine(“Please enter a number”);
    numberString = Console.ReadLine();
    number = int.Parse(numberString);
}

Console.WriteLine(“You entered a number greater than 20”);
```

As soon as the number is greater than 20, the program breaks out of the loop and prints a message for the user.

ACTIVITY 9

What do these two algorithms do? Implement them in the high-level programming language you are studying.

a Algorithm A

```
FOR index FROM 1 TO 10 DO
   SEND index * index * index TO DISPLAY
END FOR
```

b Algorithm B

```
SET counter TO 10
WHILE counter > 0 DO
```

```
   SEND counter TO DISPLAY
   SET counter TO counter – 1
END WHILE
```

RANDOM NUMBERS

Random numbers are commonly used in games of chance such as flipping a coin or rolling a dice. The aim is to make an event random.

All high-level programming languages have functions to create random numbers.

Python

Python has a random module. The following code will generate a random number between 1 and 10.

```
import random
x = random.randint(1, 11)
print(x)
```

The 'import' command is necessary so that the 'random' module can be used.

Java

Java can generate a random integer using `System.util.Random`. The following code will generate a random number between 1 and 10.

```
import java.util.*;
class Main {
  public static void main(String[] args) {
    Random r = new Random();
    int x = r.nextInt(10) + 1;
    System.out.println(x);
  }
}
```

Alternatively, you can generate a random real value using `Math.random()`.

```
class Main {
  public static void main(String[] args) {
    int x = (int)(Math.random() * 10) + 1;
    System.out.println(x);
  }
}
```

C#

C# can generate random numbers using the Random Class. The following code will generate a random number between 1 and 10 and display on screen.

```
Random rand = new Random();
randomNumber = rand.Next(1, 11);
Console.WriteLine(randomNumber);
```

ACTIVITY 10

Create a guessing-game program with the following specification.

- The computer generates a random number between 1 and 20.
- The user is asked to enter a number until they enter this random number.
- If their guess is too low or too high they are told.
- They are told when their guess is correct.
- They are asked if they want to play another game until their answer is 'NO'.

CHECKPOINT

Strengthen

SKILLS CRITICAL THINKING

S1 Why are variables needed?

SKILLS REASONING

S2 Provide examples of the four data types.

SKILLS CRITICAL THINKING

S3 How are selection and iteration implemented in the high-level language you are studying?

Challenge

SKILLS PROBLEM SOLVING

C1 Outline the following structural components of a program: variable and type declarations, command sequences, selection and iteration constructs.

How confident do you feel about your answers to these questions? If you're not sure you answered them well, try redoing the activities in this section.

SUMMARY

- A program is an algorithm that has been converted into program code.
- Pseudocode is far more forgiving than program code.
- The four basic data types are integer, float/real, Boolean and character.
- The data type of a variable determines the operations that can be performed on it.
- Data types don't have to be declared in pseudocode but it's a good idea to do so.
- Variable and type declarations, command sequences, selection and iteration are four of the structural components of a program.

6 MAKING PROGRAMS EASY TO READ

Developers usually work in teams and it is important that they understand how each other's code and programs work, especially if there are errors that need correcting. Code should be written in standard ways and be explained using comments.

LEARNING OBJECTIVES

- Explain the benefit of producing programs that are easy to read
- Use techniques to improve the readability of code and describe how code works

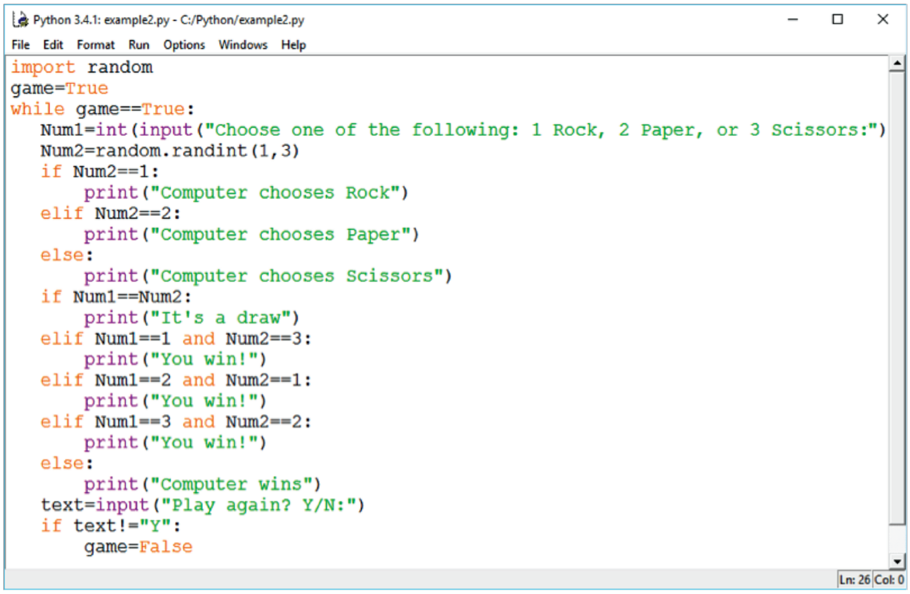
CODE READABILITY

You should always try to ensure that any code you write is easy to read and understand. We refer to this as 'readability'. This benefits you and anyone else who needs to understand how your programs work.

It is surprising how quickly you forget. Try revisiting the programs you have already written in this unit and make sure it is still clear to you what they do and how they work. Imagine how much more difficult it would be to make sense of a complex program with lots of variables, subprograms, nested loops and multiple selection statements.

Programming is not a solo activity. Programmers usually work in teams, with each programmer developing a different part of the program. This only works if they all adopt a standard approach to writing readable code.

▶ **Figure 2.2** An example of Python code



```
import random
game=True
while game==True:
   Num1=int(input("Choose one of the following: 1 Rock, 2 Paper, or 3 Scissors:"))
   Num2=random.randint(1,3)
   if Num2==1:
       print("Computer chooses Rock")
   elif Num2==2:
       print("Computer chooses Paper")
   else:
       print("Computer chooses Scissors")
   if Num1==Num2:
       print("It's a draw")
   elif Num1==1 and Num2==3:
       print("You win!")
   elif Num1==2 and Num2==1:
       print("You win!")
   elif Num1==3 and Num2==2:
       print("You win!")
   else:
       print("Computer wins")
   text=input("Play again? Y/N:")
   if text!="Y":
       game=False
```

GENERAL VOCABULARY

practice the way that you do something

SUBJECT VOCABULARY

block of code a grouping of two or more code statements

The programmer who produced the program in Figure 2.2 did not follow good **practice**. There are a number of ways that the readability of this code could be improved.

- Use descriptive names for variables (e.g. `userChoice` instead of `Num1`).
- Add blank lines between different **blocks of code** to make them stand out.
- Add comments that explain what each part of the code does.

HINT

Here's how to write comments in Python, Java and C#:

- **Python:** start each line of a comment with the # symbol
- **Java:** start single line comment with // or start a multi-line comment with /* and end it with */
- **C#:** Same as for Java.

▲ Programmers often work in teams; it is vital that any coding they share is clear and error free

Table 2.5 lists the techniques you should use to make your programs easy to read and understand.

TECHNIQUE	DESCRIPTION
Comments	Comments should be used to explain what each part of the program does.
Descriptive names	Using descriptive identifiers for variables, constants and subprograms helps to make their purpose clear.
Indentation	Indentation makes it easier to see where each block of code starts and finishes. Getting the indentation wrong in Python will result in the program not running or not producing the expected outcomes.
White space	Adding blank lines between different blocks of code makes them stand out.

▲ **Table 2.5** Techniques for program clarity

DID YOU KNOW?

Microsoft Windows® uses roughly 50 million lines of code – code readability obviously becomes more and more important as programs get larger.

SKILLS CRITICAL THINKING, PROBLEM SOLVING, CREATIVITY

ACTIVITY 11

REWRITING AND IMPLEMENTING ALGORITHMS

1 Program the following algorithm in a high-level language and make it readable by adding comments and indentation.

```
SET x TO 10
WHILE x >= 0 DO
  IF x > 0
    SEND x TO DISPLAY
  ELSE
    SEND 'Blast Off' TO DISPLAY
    END IF
  SET x TO x -1
END WHILE
```

2 Develop an algorithm for a simple calculator and code it in a high-level language that:
 a allows the user to choose from these options: addition, subtraction, division and multiplication
 b prompts the user to input two numbers
 c performs the calculation and displays the result
 d offers the user the option of performing another calculation.

▲ Calculators have a number of useful functions

SKILLS REASONING

SKILLS CRITICAL THINKING

SKILLS PROBLEM SOLVING

CHECKPOINT

Strengthen

S1 Why it is important to make your code easy to read?

S2 Outline the four techniques that a programmer should use to make code easy to read.

Challenge

C1 Revisit the programs you have already written. Do you still understand what they do and how they work? If not, try to improve their readability.

KEY POINT

Readability of code is important. You should always make your code easy to read.

How confident do you feel about your answers to these questions? If you're not sure you answered them well, look again at Table 2.5.

SUMMARY

- Using comments, descriptive names, indentation and white space makes code easier to read.
- Producing readable code makes it easier to understand what a program does and how it does it.

7 STRINGS

A string is a data type that is used to represent a sequence of characters, such as numbers, text, spaces and punctuation. Strings must be enclosed in quotation marks to distinguish them from variable names. A string could be a user name, a whole sentence or paragraph of data.

LEARNING OBJECTIVES

- Describe what a string is and explain what strings are used for
- Use iteration to traverse a string
- Concatenate and split strings

SUBJECT VOCABULARY

string a sequence of characters. They can be letters, numbers, symbols, punctuation marks or spaces

A character is one of the four basic data types. It can be a single letter, a symbol, a number or even a space. A sequence of characters is called a **string**. Although strings can contain different sorts of characters, including numbers, they are all treated as if they were text.

When a computer executes a program, it needs a way of telling the difference between a string and an instruction. In most programming languages, this is achieved by enclosing strings in quotation marks (e.g. 'johnsmith@mail.com', '10/04/15' or '123'). Both single ' ' and double " " quotation marks are acceptable, as long as they are used in the same way each time. In the example, "It's her book", the apostrophe is treated as part of the string because double quotes have been used to enclose the string.

Strings are very useful when communicating with users. For example, asking them to enter some information into a program or displaying the output of a program in a format that humans can read and understand.

STRING INDEXING

GENERAL VOCABULARY

reference to refer to something

Each character in a string has an index number, with the first character at position 0. You can use the index to **reference** individual characters in a string.

Therefore the index position of the letter 'm' is 2, even though it is the third character, as shown in Table 2.6.

INDEX	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
String	C	o	m	p	u	t	e	r		S	c	i	e	n	c	e

▲ **Table 2.6** An example string index

KEY POINT

Computers start counting at 0. Therefore, although the length of the string 'Computer Science' is 16, the indexes of the characters range from 0 to 15.

LENGTH

SUBJECT VOCABULARY

function a subprogram that performs a specific task and returns a value to the main program. High-level programming languages have a number of useful built-in functions. You can also create your own or use functions available in online libraries

The Pearson Edexcel pseudocode has a built-in `LENGTH` **function**, which you can use to find the number of characters in a string. Therefore:

```
SET numChars TO LENGTH(myText)
SEND numbChars TO DISPLAY
```

would print ‘16’. Table 2.7 gives the method for Python, Java and C#.

LANGUAGE	METHOD	EXAMPLE	OUTPUT

Python	The function to find the length of a string is `len()`.	`string = ‘Computer Science’`
		`length = len(string)`
		`print(length)`
			16

Java	Each string object has a `.length()` method.	`String s = “Computer Science”;`
		`int length = s.length();`
		`System.out.println(length);`
			16

C#	The length of a string can be found by accessing the length property.	`string subject = “Computer Science”;`
		`Console.WriteLine(subject.Length);`
			16

▲ **Table 2.7** Finding the length of a string in Python, Java and C#.

SKILLS PROBLEM SOLVING, CREATIVITY, DECISION MAKING

ACTIVITY 12

CHECKING PASSWORD LENGTH

Create and write a program to check the length of a password. If the password entered is less than six characters, the program should output ‘The password you have entered is not long enough’; otherwise it should output ‘Length of password OK’.

STRING TRAVERSAL

SUBJECT VOCABULARY

string traversal using a loop to cycle through each character in a string

You can use a `FOR` loop to cycle through each of the characters in a string. This is known as **string traversal**.

The following algorithm prints out the word ‘monkey’ letter by letter, displaying each letter on a separate line.

```
SET animalName TO ‘monkey’
FOR index = 0 TO LENGTH(animalName) – 1
    SEND animalName[index] TO DISPLAY
END FOR
```

The loop runs until `LENGTH(animalName) – 1` because string indexing starts at 0 and therefore has to print characters with indexes of 0 to 5.

PYTHON

```
animalName = ‘monkey’
for index in range 0, len(animalName):
    print(animalName[index])
```

Using Python you should not use `len(animalName) – 1` as the range command does not include the end number of the loop.

JAVA

```
class Main {
  public static void main(String[] args) {
    String animalName = “monkey”;
    for(int index = 0; index < animalName.length(); index++) {
      System.out.println(animalName.charAt(index));
    }
  }
}
```

C#

```
string animalName = “monkey”;

for (int index = 0; index < animalName.Length; index ++)
{
Console.WriteLine(animalName[index]);
}
```

SKILLS PROBLEM SOLVING, CREATIVITY, DECISION MAKING

ACTIVITY 13

WHICH CAR?

Write a program that will check if a make of car entered by the user is in the string ‘The cars present included Ford, Mercedes, Toyota, BMW, Audi and Renault.’

If the car entered by the user is present, then ‘It is present’ should be returned or ‘It is not present’, if not.

It should not matter which case the car name is entered by the user.

OTHER WAYS TO MANIPULATE STRINGS

FINDING A CHARACTER WITH A PARTICULAR INDEX

The index of a character is the position of that character within a string. Note that indexing starts at 0 not 1. This means the first character will have an index of 0 and the second character will have an index of 1.

PYTHON

Python uses square brackets to access elements in a string:

```
string = ‘Computer Science’
print(string[11])
```

This code would return the character ‘c’.

JAVA

Java allows you to get a character at a particular index using the `charAt` method.

```
class Main {
  public static void main(String[] args) {
    String string = “Computer Science”;
    System.out.println(string.charAt(11));
  }
}
```

This code would display the character ‘i’ (the 12th character at index 11).

C#

```
string text;
text = “Computer Science”;
Console.WriteLine(text[3]);
```

This code would return the character ‘p’.

CHANGING ALL CHARACTERS TO LOWER CASE

PYTHON

```
string = ‘Computer Science’
string = string.lower()
print(string)
```

This code would return ‘computer science’.

JAVA

```
class Main {
  public static void main(String[] args) {
    String string = “Computer Science”;
    string = string.toLowerCase();
    System.out.println(string);
  }
}
```

C#

```
string text;
text = “Computer Science”;
text = text.ToLower();
Console.WriteLine(text);
```

Would output ‘computer science’.

CHANGING ALL CHARACTERS TO UPPER CASE

PYTHON

```
string = ‘Computer Science’
string = string.upper()
print(string)
```

This would return ‘COMPUTER SCIENCE’.

JAVA

```
class Main {
  public static void main(String[] args) {
    String string = “Computer Science”;
    string = string.toUpperCase();
    System.out.println(string);
  }
}
```

C#

```
string text;
text = “Computer Science”;
text = text.ToUpper();
Console.WriteLine(text);
```

This would output ‘COMPUTER SCIENCE’.

HINT

Changing strings to upper or lower case is useful when you are comparing two strings and the case does not matter, only the characters.

EXTRACTING CHARACTERS FROM A STRING

PYTHON

```
string = ‘Computer Science’
substring = string[3:6]
print(substring)
```

This code would return ‘put’ as it selects the characters from index 3 to index 5. Index 6 is not included.

JAVA

```
class Main {
  public static void main(String[] args) {
    String string = “Computer Science”;
    String substring = string.substring(3,6);
    System.out.println(substring);
  }
}
```

C#

```
string text;
text = “Computer Science”;
text = text.Substring(3,4);
Console.WriteLine(text);
```

This would display ‘pute’ as it selects 4 characters starting at index 3 (remember that the first character is at index 0).

CHECKING A PHRASE IN THE STRING

These examples check to see if the string ‘Computer Science’ contains the strings ‘put’ and ‘PUT’: ‘put’ is a substring of ‘Computer Science’ but ‘PUT’ is not, so these examples will display ‘True’ then ‘False’.

PYTHON

```
string = ‘Computer Science’
present = ‘put’ in string
print(present)
present = ‘PUT’ in string
print(present)
```

JAVA

```
class Main {
  public static void main(String[] args) {
    String string = “Computer Science”;
    boolean present = string.contains(“put”);
    System.out.println(present);
    present = string.contains(“PUT”);
    System.out.println(present);
  }
}
```

C#

You can check if a substring is present in a string using the contains method. If the substring is present in the string then ‘True’ is returned, otherwise ‘False’ is returned.

```
string text;
bool found = false;
text = “Computer Science”;
found = text.Contains(“put”);
Console.WriteLine(found);
```

This would display ‘True’, as ‘put’ is in ‘Computer Science’.

CONCATENATION

Concatenation involves joining two or more items of information together. Concatenating two strings produces a new string object. It is very useful when displaying text on screen.

In Pearson Edexcel pseudocode concatenation is done in the following way.

```
RECEIVE userName FROM (STRING) KEYBOARD
SEND ‘Hello’ & userName TO DISPLAY
```

Note that literal text is enclosed in speech marks but the variable name is not. In Python, this would be:

```
userName = input(‘Please enter your username’)
print(‘Hello’ + userName)
```

The ‘+’ character is used for concatenation.

In Python, you cannot concatenate strings with numbers. For example, the following would produce an error message.

```
length = 13
print(‘The length is’ + length)
```

To overcome this, the number must be converted to a string:

```
length = 13
print(‘The length is’ + str(length))
```

In Java, you can concatenate strings with most other data types:

```
Scanner scanner = new Scanner(System.in);
System.out.println(“Please enter your username:”);
String username = scanner.nextLine();
System.out.println(“Hello” + username);

int length = 13;
System.out.print(“The length is” + length);
```

In C# this would be:

```
string userName;
Console.WriteLine(“Please enter your username:”);
userName = Console.ReadLine();
Console.WriteLine(“Hello” + userName);
```

In C#, you can concatenate with numbers.

COMPARING STRINGS

Comparing strings in Python and C# can be done using the == operator but in Java you need to use a string object’s equals method:

```
String food = “pie”;
if(food.equals(“pie”)) {
      System.out.println(“Yum! I like pie”);
}
```

SKILLS

PROBLEM SOLVING,
CREATIVITY,
DECISION MAKING

ACTIVITY 14

CONCATENATING AND SLICING STRINGS

A company wants a program to generate usernames for new employees. Each username consists of the first four letters of the employee’s last name and the first letter of their first name joined together. If the employee’s last name is less than four characters in length a letter ‘X’ is used to fill in for each of the missing characters. Develop a program that asks the user to input their first and last names and outputs their username.

CHECKPOINT

Strengthen

SKILLS CRITICAL THINKING

S1 How are individual characters in a string referenced?

SKILLS PROBLEM SOLVING, CREATIVITY

S2 Develop an algorithm that uses a loop to traverse a string.

SKILLS CRITICAL THINKING

S3 How are a string and a non-string concatenated?

Challenge

SKILLS PROBLEM SOLVING, CREATIVITY

C1 Develop a program that asks the user to input a sentence and then splits it up wherever a space occurs. Each word should then be displayed on a separate line.

How confident do you feel about your answers to these questions? If you're not sure you answered them well, have another go at the activities in this section.

SUMMARY

- A string is a sequence of characters.
- Each character in a string has a unique index value representing its position in the string. The first character in a string has the index value 0.
- High-level programming languages have a built-in length function that finds the length of a string.
- A loop is used to traverse a string, character by character.
- Concatenation is the process of joining two or more strings together.
- Slicing is the process of extracting part of a string.
- String formatting is used to control the way text is displayed on screen.

8 DATA STRUCTURES

Data should be classified and organised into similar types so it can be easily searched and analysed. They are stored together in data structures, such as records and arrays.

LEARNING OBJECTIVES

- Describe the structure of one- and two-dimensional arrays and give examples of their use
- Create and use one- and two-dimensional arrays in programs
- Describe the record data structure and explain what it is used for
- Design record structures

SUBJECT VOCABULARY

data structure an organised collection of related elements. Arrays and records are two common data structures used in programming

A **data structure** is an organised collection of related elements. There are many different data structures that can store multiple data items used in programming. You have already encountered one of them – strings. In this section we will investigate two more: arrays and records.

ARRAYS

SUBJECT VOCABULARY

array a structure that contains many items of data of the same type. The data is indexed so that a particular item of data can be easily found

index a number that identifies each element of an array in Python and Java

An **array** is an organised collection of related values with a single shared identifier. All the elements in an array are the same data type. Each has a unique **index** value denoting its position in the array.

In the Pearson Edexcel pseudocode, an array is initialised using the `SET` command. For example, this statement initialises an array called `firstNames` with four elements, all of string type. The square brackets denote the start and end of the array.

```
SET firstNames TO [‘Ashura’, ‘Bryn’, ‘Eloise’, ‘Mei’]
```

In most programming languages, arrays are static. A static array has a fixed size and when it is declared the number of items it can hold must be stated.

For example, `array friends [5]`.

As with strings, arrays have a length indicating the number of items. Here are the functions to find the length of an array declared as `myArray`.

PYTHON	JAVA	C#
`len(myArray)`	`myArray.length`	`myArray.Length`

Loops can be used to traverse arrays.

ACTIVITY 15

Declare an array with the elements Ford, Mercedes, Toyota, BMW, Audi and Renault.

In a high-level language, create a program to find the length of the array and then traverse it, printing out each of the items.

PYTHON

In Python, arrays are not commonly used. Instead **lists** are used but they are very similar in the way that they operate. Just like an array, a list is created by adding items, separated by commas, inside square brackets.

Lists are easier to use as they are **dynamic** – they do not have a fixed size and can grow as new elements are added. When they are declared they do not have to be given a size, e.g.:

```
cars = []
```

When adding items to an array the `append` command is used, for example:

```
cars = []
cars.append(‘Audi’)

print(cars)
```

would return the following:

```
[‘Audi’]
```

Another advantage of lists is that the items do not have to be of the same data type.

```
cars = []
cars.append(‘Audi’)
cars.append(3)
print(cars)
```

would return:

```
[‘Audi’, 3]
```

There are many functions in Python for manipulating arrays.

Investigate the following:

```
max()
min()
slice()
```

JAVA

Arrays are static data structures, which means that you can’t change the size of an array to make it bigger or smaller in order to add or remove values.

For example, you can have an array of 5 integers:

```
int[] integers = {1,2,3,4,5};
```

The following is not possible with an array:

`integers.add(6);` or `integers.remove(5);`

In order to add or remove values you need to use a list which is a dynamic data structure.

The most common type of list is an `ArrayList`:

```
import java.util.*;
class Main {
  public static void main(String[] args) {
    ArrayList<Integer> integers = new ArrayList<Integer>();
    integers.add(1);
    integers.add(2);
    integers.add(3);
    integers.add(4);
    integers.add(5);
    integers.add(6);
    integers.remove(0); // remove the value at index 0
                        (not the value 0)
    System.out.println(integers);
  }
}
```

This example will display [2,3,4,5,6] because the first value will be removed.

Arrays are supported for any data type in Java. For example, you can have an array of strings:

```
String[] colours = {“red”, “green”, “blue”};
```

Or an array of integers:

```
int[] numbers = {1,2,3,4};
```

However, you can’t have an array that contains both strings and integers (or any other mix of data types).

For this, you need to use a vector:

```
import java.util.*;
class Main {
  public static void main(String[] args) {
    Vector allSorts = new Vector();
    allSorts.add(1);
    allSorts.add(“Two”);
    allSorts.add(true);
    System.out.println(allSorts);
  }
}
```

This example will output [1, Two true] because the vector `allSorts` contains an integer, a string and a Boolean value.

C#

C# supports lists and arrays. The code below uses an array to hold the items.

```
string[] make = { “Ford”, “Mercedes”, “Toyota”, “BMW”,
“Audi”, “Renault” };

foreach (string currentItem in make)
{
    Console.WriteLine(currentItem);
```

SKILLS CRITICAL THINKING , PROBLEM SOLVING, CREATIVITY

ACTIVITY 16

DESCRIBING AND IMPLEMENTING A LINEAR SEARCH ALGORITHM

The following code uses a linear search algorithm.

```
SET firstNames TO [‘Ashura’, ‘Bryn’, ‘Eloise’, ‘Mei’, ‘James’, ‘Irena’]
RECEIVE searchName FROM (STRING) KEYBOARD

SET found TO False
SET index TO 0

WHILE found = False AND index <= (LENGTH(firstNames) – 1) DO
   IF searchName = firstNames[index] THEN
      SET found TO True
   END IF
   SET index to index + 1
END WHILE

IF found = True THEN
   SEND searchName & ‘is at position’ & index & ‘in the list’ TO DISPLAY
ELSE
   SEND searchName & ‘is not in the list’ TO DISPLAY
END IF
```

Implement this algorithm in the high-level programming language you are studying.

MULTIDIMENSIONAL ARRAYS

In the array `cars = [‘Ford’, ‘Mercedes’, ‘Toyota’, ‘BMW’, ‘Audi’, ‘Renault’]` there is only one item at each index position: the name of the manufacturer.

A multidimensional array is an ‘array of arrays’; each item at an index is another array.

We declared the above array using this statement, `array cars[6]`.

If we wanted to create a two-dimensional array we could declare it as `array cars[3, 2]` so that there is another array at each index to store two items of information.

Here is an extract from an array named `examResults`. It has three rows, each of which stores a set of four exam results. The mark of 47 is located at `examResults [1, 2]` – second row, third element along – and the value 80 at `examResults [0, 0]` – first row, first column.

	0	1	2	3
0	80	59	34	89
1	31	11	47	64
2	29	56	13	91

▲ Table 2.8 `examResults` two-dimensional array

Each item of data has two indexes. An array to hold this data would be declared as `array[3, 4]`.

If the array was printed it would be `[[80, 59, 34, 89], [31, 11, 47, 64], [29, 56, 13, 91]]`. There are square brackets around each set of results and around the whole array.

In Python, the array would be declared and initialised as:

```
Scores = [[80, 59, 34, 89], [31, 11, 47, 64], [29, 56, 13, 91]]
```

In Java, it would be:

```
int[][] Scores = new int[][] {
  {80, 59, 34, 89},
  {31, 11, 47, 64},
  {29, 56, 13, 91}};
```

In C# the array would be declared and initialised as:

```
int[,] Scores= new int[,] { { 80, 59, 34, 89 }, { 31, 11,
47, 64 }, { 29, 56, 13, 91 } };
```

WORKED EXAMPLE

A teacher has stored the surnames and test scores of a class of students in a two-dimensional array, e.g. `results[[‘Smith, ‘69’], [‘Jackson’, ‘90’]`, etc. Create a program that would print out the names and test scores of all the students who have scored 50 or over in the test.

Python

```
results = [[‘Smith’, 69], [‘Jackson’, 90], [‘Dubois’, 30]]
for index in range(0, len(results)):
    if results[index][1] >= 50:
        print(results[index][0] + str(results[index][1]))
```

In Python each element is written as `results[0][1]` and not `results[0, 1]`.

Java

```
class Main {
  public static void main(String[] args) {
    String[][] results = {
      {“Smith”, “69”},
      {“Jackson”, “90”},
      {“Dubois”, “30”}};
    for(int index = 0; index < results.length; index++) {
      int score = Integer.parseInt(results[index][1]);
      String name = results[index][0];
      if(score >= 50) {
        System.out.println(name + score);
      }
    }
  }
}
```

In Java, a two-dimensional array has two indexing expressions (e.g., `results[0][1])`. The first index is the row number and the second index is the column number. Both index expressions count from 0. This means `results[0][1]` will get the value in the second column (index 1) of the first row (index 0), which in the above example would be ‘69’.

C#

```
string[,] results = new string[,] { { “Smith”, “69” },
{ “Jackson”, “90” }, { “Dubois”, “30” } };

for (int index = 0; index <= results.GetUpperBound(0);
index++)
{
    if (int.Parse(results[index,1]) >= 50)
    {
        Console.WriteLine(results[index,0] + “ ” +
results[index,1]);
    }
}
```

In C# an array only consists of one data type, hence the student scores are in double quotes. To be able to check if the student scores are above 50, the score in the array must be cast as an integer using `int.Parse`.

SKILLS CRITICAL THINKING, PROBLEM SOLVING, CREATIVITY

HINT

All the data values stored in an array must be of the same data type. In this case they are all strings. You will have to convert the scores to integers to find the highest. But as Python uses lists this is not necessary. Strings and integers can be combined within the same list.

ACTIVITY 17

USING TWO-DIMENSIONAL ARRAYS

1 Develop a program that creates and initialises an array to hold these five sets of marks.
80, 59, 34, 89
31, 11, 47, 64
29, 56, 13, 91
55, 61, 48, 0
75, 78, 81, 91

2 Extend your program so that it calculates and displays the highest mark, the lowest mark and the average mark achieved.

ACTIVITY 18

This two-dimensional array holds the highest score achieved by each player in each of the three levels of an online game.

1 Develop a program that initialises the array and then searches through it to find the player with the highest score in each of the three levels.

PLAYER	LEVEL	SCORE
Alexis	1	19
Seema	1	29
Seema	2	44
Lois	1	10
Alexis	2	17
Alexis	3	36
Dion	1	23
Emma	1	27
Emma	2	48

▲ **Table 2.9** Example of two-dimensional array

RECORDS

We have already said that the elements of an array must all be the same data type. In contrast, the **record** data structure stores a set of related values of different data types.

Each element in a record is known as a **field** and is referenced using a field name.

SUBJECT VOCABULARY

record a data structure that stores a set of related values of different data types

field an individual element in a record

Table 2.10 shows how the record data structure works. Each row of the table holds a set of information about a particular learner (these are the records). Each column stores one item of information about the learner – their learner number, their age, their form, etc. (these are the fields). All the values in a column have the same data type – `learnerNum` and 'age' are integers; `firstName`, `lastName` and 'form' are strings.

Programming languages vary in the way they handle the record data structure.

learnerNum	firstName	lastName	AGE	FORM
1	Isla	Smith	15	10H
2	Shinji	Fujita	14	10B
3	Anita	Khan	15	10A
4	Abdur	Rahman	15	10G

▲ **Table 2.10** Example of record data structure

SKILLS CRITICAL THINKING, PROBLEM SOLVING, CREATIVITY

▲ It is essential to have a way to sort through the vast amounts of recorded music available

ACTIVITY 19

USING RECORDS TO STORE MUSIC DETAILS

1 A record data structure is to be used to store the details of music albums. Provide the appropriate data type for these fields:
   - a the title of the album
   - b the name of the artist
   - c the year of release
   - d the genre.

2 Develop a program that uses a record structure for storing the details of music albums. It must:
   - a have fields for title, artist, year of release and genre
   - b allow the user to input the details of new albums
   - c allow the user to search for an album by name and display its details.

CHECKPOINT

Strengthen

SKILLS CRITICAL THINKING

S1 What is the index of the first element in a one-dimensional array?

SKILLS REASONING

S2 How does a linear search algorithm find an element in a one-dimensional array?

SKILLS REASONING

S3 How is an element stored in a two-dimensional array referenced?

SKILLS REASONING

S4 How is a `nested IF` used to traverse a two-dimensional array?

Challenge

SKILLS CRITICAL THINKING, PROBLEM SOLVING

C1 Develop a program for a simple address book that uses a two-dimensional array to store a set of names and email addresses, and allows the user to search for a person by name and returns their email address.

SKILLS CRITICAL THINKING, PROBLEM SOLVING

C2 Develop a program that uses a two-dimensional array to represent a treasure map consisting of a grid of 4 rows and 4 columns. A random number function should be used to establish the location of the treasure. The user must hunt for the treasure by repeatedly entering the coordinates of squares. The program should tell them when they have found the treasure and help them in their search by indicating how close they are.

How confident do you feel about your answers to these questions? If you're not sure you answered them well, try redoing activities 15–19.

SUMMARY

- A data structure is an organised collection of related elements. Arrays and records are common data structures.
- A one-dimensional array is a list of elements, each of which has a unique index value representing its position in the list.
- A two-dimensional array is a matrix of rows and columns. Each element in the array has a unique pair of indices, one to identify the row and one the column in which it is located.
- All the elements in an array have the same data type (Python uses lists).
- A record consists of a collection of fields. The values stored in a record can be of different data types.

9 INPUT/OUTPUT

All program data has to be entered (input) and information is output. The input could be automatic, e.g. from a sensor, but is often provided by human users. It is essential that this input is checked to ensure that it is what could reasonably be expected and falls within a certain range. Incorrect input could cause problems or even catastrophes.

LEARNING OBJECTIVES

- Explain the need for validation
- Write code that validates user input
- Write code that accepts and responds appropriately to user input
- Write code that reads to and writes from a text file

USER INPUT

Most programs require some form of input either from a user or from a file. You already know how to receive user input from a keyboard.

A program can be made much more ‘user friendly’ by displaying helpful messages informing users of what they are expected to enter and confirming successful input.

VALIDATION

It is important to ensure that data entered by the user is valid, as invalid data can cause a program to behave unexpectedly or even stop altogether. If the data entered into a program is incorrect, the output it produces will also be wrong. This is sometimes called the Garbage In, Garbage Out (GIGO) principle.

Any program that requires data entry should have appropriate forms of **validation** built in. But validation can’t guarantee that the data entered is correct. It can only make sure that it is reasonable.

SUBJECT VOCABULARY

validation to check that the data entered by a user or from a file meets specified requirements

There are a number of different types of validation.

RANGE CHECK

A range check is used to ensure that the data entered is within a specified range. Study this algorithm, written in Pearson Edexcel pseudocode which checks that the number entered is between 1 and 10.

```
BOOLEAN valid
SET validNum TO False
WHILE validNum = False DO
    SEND ‘Please enter a number between 1 and 10:’ TO DISPLAY
    RECEIVE number FROM (INTEGER) KEYBOARD
    IF number >= 1 AND number <= 10 THEN
        SET validNum TO True
    END IF
END WHILE
SEND ‘You have entered:’ & number TO DISPLAY
```

The algorithm uses a Boolean variable named `validNum` as a status flag. It is initially set to False. The `WHILE` loop continues running until `validNum` is equal to True. An `IF` statement determines if the value of `validNum` should be set to True.

SKILLS CRITICAL THINKING, PROBLEM SOLVING

ACTIVITY 20

IMPLEMENTING RANGE CHECKS

Implement the range check algorithm in the high-level programming language you are studying.

PRESENCE CHECK

Another type of validation is a presence check. This simply ensures that a value has been entered, preventing the user from leaving an input blank.

This algorithm asks the user to input their name and uses a presence check to ensure they have entered a value. Any value will cause the loop to finish. It will keep asking the user to input their name until they input a value.

```
SET userName TO ‘ ’
WHILE userName = ‘ ’ DO
    RECEIVE userName FROM (STRING) KEYBOARD
END WHILE
SEND ‘Hello’ & userName TO DISPLAY
```

SKILLS CRITICAL THINKING, PROBLEM SOLVING

ACTIVITY 21

IMPLEMENTING A PRESENCE CHECK

Implement the presence check algorithm in the high-level programming language you are studying.

GENERAL VOCABULARY

predefined already defined before the start of something

LOOK-UP CHECK

A look-up check is used to test that a value is one of a **predefined** set of acceptable values. The list of acceptable values can be stored in a one-dimensional array.

This algorithm stores a list of valid form names in an array. It compares the form name entered by the user with the values in the array.

```
SET arrayForms TO [‘7AXB’, ‘7PDB’, ‘7ARL’, ‘7JEH’]
RECEIVE form FROM (STRING) KEYBOARD
SET valid TO False
SET index TO 0
SET length TO LENGTH(arrayForms)
WHILE valid = False AND index < length DO
    IF form = arrayForms[index] THEN
        SET valid TO True
    END IF
    SET index TO index + 1
END WHILE
```

```
IF valid = True THEN
    SEND 'Valid form' TO DISPLAY
ELSE
    SEND 'The form you entered does not exist.' TO DISPLAY
END IF
```

SKILLS CRITICAL THINKING, PROBLEM SOLVING

ACTIVITY 22

IMPLEMENTING A LOOK-UP CHECK

Implement the look-up check algorithm in the high-level programming language you are studying.

LENGTH CHECK

It is sometimes necessary to check that the length of a value entered falls within a specified range. For example, all UK postcodes are between six and eight characters long, so validation could be used to check that the length of a postcode entered is within this range. Needless to say, that doesn't necessarily mean it is correct.

```
RECEIVE postCode FROM (STRING) KEYBOARD
SET length TO LENGTH(postCode)
IF length >= 6 AND length <= 8 THEN
    SEND 'Valid' TO DISPLAY
ELSE
    SEND 'Invalid' TO DISPLAY
END IF
```

SKILLS CRITICAL THINKING

ACTIVITY 23

IMPLEMENTING A LENGTH CHECK

Implement the length check algorithm in the high-level programming language you are studying.

TESTING VALIDATION RULES

It is important to test your validation rules to ensure they work as expected. You should use:

Normal data – This is data that is within the limits of what should be accepted by the program. For example, a password with seven characters fulfils the validation rule that states that passwords must be between six and eight characters in length.

Boundary data – This is data that is at the outer limits of what should be accepted by the program. For example, if a validation rule specifies that the range of acceptable values is >= 75 AND <= 100, then a value of 100 is at the upper limit and a value of 75 at the lower.

GENERAL VOCABULARY

erroneous not correct; based on incorrect information

Erroneous data – This is data that should not be accepted by the program. For example, a value of 0 should not be accepted by either of the validation rules given above.

WORKING WITH TEXT FILES

SUBJECT VOCABULARY

text file a sequence of lines, each of which consists of a sequence of characters

The programs you have created so far haven't required a huge amount of data entry, but imagine typing a set of test results for everyone in your computer science class. It would take a considerable amount of time to do and – even worse – when the program terminates, all of the data will be lost. Should you need to use it again you'd have to re-enter it. This is where storing data in an external file comes in really useful. If the data you enter is stored in an external **text file** you can access it as often as you like without having to do any further keying in.

Commas are used to separate individual values on a line and a special character is used to denote the end of a line.

Text files provide permanent storage for data. This means that the data can be reused without having to be retyped. Data can be read from, written to and appended to a file.

SUBJECT VOCABULARY

file handle a label that is assigned to a resource needed by the program. It can only access the file through the computer's operating system

overwritten if a file exists on the computer and a new file is created with the same name, the new file is kept and the old file is written over and lost

PYTHON

To access a text file, you must first give it a **file handle**, e.g. `myFile`.

Files can be opened to read from them, to write to them or to append data to them (Table 2.11).

`myFile = open('names.txt', 'r')`	Open a file named `names.txt` so that the data can be read.
`myFile = open('names.txt', 'w')`	Open a file named `names.txt` to be written to and create a new file if it does not exist. Unfortunately, if the file did exist it would **overwrite** all of its data.
`myFile = open('names.txt', 'a')`	Open a file named `names.txt` to add data to it.

▲ **Table 2.11** Using text files

After a file has been opened it must be closed. The data is not written to a file until this command is executed.

The following program would create a text file called 'names' and add two items of data.

```
myFile = open('names.txt', 'w')
myFile.write('First item')
myFile.write('Second item')
myFile.close()
```

The file it creates would be: `‘First itemSecond item’`.

There is nothing to separate the two items of data. Therefore, it is useful to add a character such as a ‘,’ or a ‘;’ between them.

```
myFile = open(‘names.txt’, ‘w’)
myFile.write(‘First item’ + ‘,’)
myFile.write(‘Second item’ + ‘,’)
myFile.close()
```

The file created would be: `‘First item,Second item’`.

It is important to separate the data items so that they can be read back into a data structure.

The following program would write the contents of an array into a text file called `cars.txt` with a comma between them.

```
Alist = [‘BMW’, ‘Toyota’, ‘Audi’, ‘Renault’, ‘Rover’]

myFile = open(‘cars.txt’, ‘w’)

for index in range(len(Alist)):
    myFile.write((Alist[index]) + ‘,’)

myFile.close()
```

The file can be read back into an array in the following way.

```
Blist = []
myFile = open(‘cars.txt’, ‘r’)

Blist = myFile.read().split(‘,’)

MyFile.close()
```

`Blist = myFile.read().split(‘,’)` reads the data and splits it into separate items where there is a ‘,’.

JAVA

Java programs that read and write to files often start with:

```
import system.io.*
```

This allows your Java code to access files such as `File` and `FileReader`, FileWriter from the `system.io` namespace.

```
import java.io.*;
class Main {
  public static void main(String[] args) {
    try {
      FileWriter fw = new FileWriter(“names.txt”);
      fw.write(“First item”);
      fw.write(“Second item”);
      fw.close();
    } catch (IOException e) {
      System.out.println(“Could not write to file”);
    }
  }
}
```

This example will write two strings to the file `names.txt`, which would then contain: `First itemSecond item`.

It is useful to separate each data value in a file with a comma (,) or new line (`\n`) so that they can then be read back into an array or opened as a comma separated value (CSV) file to be edited in a spreadsheet program:

```
import java.io.*;
class Main {
  public static void main(String[] args) {
    try {
      FileWriter fw = new FileWriter(“names.txt”);
      fw.write(“First item,”);
      fw.write(“Second item\n”);
      fw.close();
    } catch (IOException e) {
      System.out.println(“Could not write to file”);
    }
  }
}
```

Note that the two examples above contain `try…catch` blocks of code. These are necessary in Java because file input or output can often cause runtime errors and Java needs to know what to do instead of crashing the program.

The following program will write the contents of an array of car brands into a file called `cars.txt` with each value separated by a comma:

```
import java.io.*;
class Main {
  public static void main(String[] args) {
    String[] cars = {“BMW”, “Toyota”, “Audi”, “Renault”,
“Rover”};
    try {
      FileWriter fw = new FileWriter(“cars.txt”);
      for(int index = 0; index < cars.length; index++) {
        fw.write(cars[index] + “,”);
      }
      fw.close();
    } catch (IOException e) {
      System.out.println(“Could not write to file”);
    }
  }
}
```

The file can be read back into an array in the following way:

```
import java.io.*;
import java.util.*;
class Main {
  public static void main(String[] args) {
    String[] cars;
    try {
      File file = new File(“cars.txt”);
      Scanner scanner = new Scanner(file);
      String line = scanner.nextLine();

      // split each line into an array of strings
      cars = line.split(“,”);

      // display each car brand
      for(int index = 0; index < cars.length; index++) {
        System.out.println(cars[index]);
      }

      scanner.close();
    } catch (IOException e) {
      System.out.println(“Could not read from file”);
    }
  }
}
```

C#

In C#, you’re likely to use the File class for reading and writing files. To use the File class you will need to add the line `Using System.IO;` at the top of your C# program, usually under the existing `Using System;` line. The File class provides over 50 different methods for working with files, so this section only provides an introduction to some of the basic methods.

WRITING TO A FILE

C#

The following program creates a file called `“names.txt”` and adds two items of data.

```
StreamWriter sw = new StreamWriter(“names.txt”);
sw.Write(“First Item”);
sw.Write(“Second Item”);
sw.Close();
```

The file it creates would contain: `First itemSecond item`. There is nothing to separate the two items of data. Therefore, it is useful to add a character such as a comma or semicolon between them. The file created would then contain: `First item,Second item`. It is important to separate the data items so that they can be read back into a data structure.

The following program would write the contents of an array into a text file called `cars.csv` with a comma between each item.

```
string[] makes = {“BMW”, “Toyota”, “Audi”, “Renault”, “Rover”};
StreamWriter sw = new StreamWriter(“cars.csv”);

foreach (string currentItem in makes)
{
    sw.Write(currentItem + “,”);
}

sw.Close();
```

The file can be read back into an array using:

```
string[] makes = File.ReadAllText(“z:cars.txt”).Split(‘,’);
```

ACTIVITY 24

A student has coded a computer game which stores the five highest scores in an array. She now wants to save those scores to a file and load them back in when the game is run again.

Write a program that will save the array of scores to a suitable text file and then load them back in again. Run your program to check that it is working as intended.

PYTHON

A two-dimensional array can be stored in a text file in a similar way.

A teacher stores the names of students and their test scores in a two-dimensional array.

```
Slist = [[‘Faruq’, 60], [‘Jalila’, 90]]
```

The names and scores can be saved in a text file:

```
Slist = [[‘Faruq’, 60], [‘Jalila’, 90]]
myFile = open(‘results.txt’, ‘w’)

for x in range(len(Slist)):
    myFile.write(Slist[x][0] + ‘,’)
    new = str(Slist[x][1])
    myFile.write(new + ‘,’)

myFile.close()
```

It is easier to copy the items from a text file into a two-dimensional array in two stages.

First, copy them into a one-dimensional array:

```
Blist = []
myFile = open(“results.txt”, “r”)
Blist = myFile.read().split(‘,’)
myFile.close()
```

Second, move them into a two-dimensional array:

```
newList = []
for index in range(0, len(Blist) – 1, 2):
    newLlist.append([Blist[index], Blist[index + 1]])
```

A step of 2 is used as each record in the two-dimensional array contains 2 items from the one-dimensional array.

JAVA

The following code writes the student test scores to a file:

```
import java.io.*;
import java.util.*;
class Main {
  public static void main(String[] args) {
    // store student scores in a 2d array
    String[][] studentScores = {
      {“Faruq”, “60”},
      {“Jalila”, “90”}
    };

    // write scores to a file
    try {
      FileWriter fw = new FileWriter(“results.txt”);
      for(int row = 0; row < studentScores.length; row++) {

        // write student’s name
        fw.write(studentScores[row][0] + “,”);

        // write student’s score
        fw.write(studentScores[row][1] + “\n”);
      }
      fw.close();
    } catch (IOException e) {
      System.out.println(“Could not write to file”);
    }
  }
}
```

The following code reads the test scores back into a two-dimensional array:

```
import java.io.*;
import java.util.*;
class Main {
  public static void main(String[] args) {
    // list of arrays of strings to store results
    ArrayList<String[]> results = new ArrayList<String[]>();
    // read scores from a file
    try {
      File file = new File(“results.txt”);
      Scanner scanner = new Scanner(file);

      // read each line from the file
      while(scanner.hasNextLine()) {
        String line = scanner.nextLine();

        // split line into name and score and add it to
           the list
        String[] singleStudent = line.split(“,”);
        results.add(singleStudent);
      }
      scanner.close();
    } catch (IOException e) {
      System.out.println(“Could not read from file”);
    }

    // display scores for each student
    for(int i = 0; i < results.size(); i++) {
      System.out.println(results.get(i)[0] + “: ” +
results.get(i)[1]);
    }
  }
}
```

In the above example, a list is used instead of an array because a list is a dynamic data structure. This allows us to add new items as we read them from the file, which we couldn’t do with an array.

C#

```
string[,] grades = new string[,] { { “Faruq”, “60” }, {
“Jalila”, “90” } };

StreamWriter sw = new StreamWriter(“grades.csv”);

for (int i = 0; i < grades.GetLength(0); i++)
{
   sw.WriteLine(grades[i, 0] + “,” + grades[i,1]);
}

sw.Close();

}
```

Reading back in to two-dimenional array:

```
// read all of file into array
string[] allLines = System.IO.File.ReadAllLines(“grades.csv”);
int length = allLines.Length;

string[,] grades;
grades = new string[length, 2];

// for each line in array containing all lines from the file, split using delimiter and assign to 2d array
for (int i = 0; i < length; i++)
{
   string[] temp = allLines[i].Split(‘,’);
   grades[i, 0] = temp[0];
   grades[i, 1] = temp[1];
}
```

ACTIVITY 25

In Activity 24, only the high scores were saved. Change your program so that high scores and the names of the players who attained them are stored.

CHECKPOINT

Strengthen

SKILLS REASONING

S1 What might happen if a program doesn’t include validation on user input?

SKILLS CRITICAL THINKING

S2 Can you think of a data structure that is suitable for storing a list of values used in a look-up check?

SKILLS PROBLEM SOLVING

S3 Show how your name, date of birth and favourite colour would be stored in a text file.

SKILLS REASONING

S4 Why is it beneficial to write data to a text file?

Challenge

SKILLS PROBLEM SOLVING

C1 Develop a program that:

- writes a set of employee records consisting of employee number, name and department to a text file
- reads in the stored records from the text file
- allows the user to search for an employee’s details.

How confident do you feel about your answers to these questions? If you’re not sure you answered them well, try redoing the activities in this section.

SUMMARY

- Validation techniques should be used to ensure that data entered by a user or from a file is valid. They can’t guarantee that the data is correct, only that it is reasonable.
- A range check is used to ensure that data is within a specified range.
- A length check is used to ensure that data has a length within a specified range.
- A presence check is used to ensure the user has entered some data.
- When users are required to choose from a list of options, their input should be validated to ensure that their choice is valid.
- Large sets of data are normally stored in text files. The advantage of writing data to a file is that the data is not lost when the program is terminated. It can be read in from the file whenever it is needed.

10 SUBPROGRAMS

When programs are being coded, they should be as structured as possible so that people reading them can quickly understand their logic. One way of doing this is to use self-contained modules, which can be reused where possible. This ensures that programs are shorter, as commands do not have to be repeated several times. These modules can then be called when they are needed.

LEARNING OBJECTIVES

- Describe what a subprogram is
- Explain the benefits of using subprograms
- Explain the difference between global and local variables
- Explain the concept of passing data into and out of subprograms
- Create subprograms that use parameters
- Write code that uses user-defined and pre-existing subprograms

A subprogram is a self-contained module of code that performs a specific task. For example, a high-level programming language could provide a predefined subprogram that calculates the average of a set of numbers. A programmer can use this subprogram in any program they write without needing to know how it works. In other words, subprograms support the process of abstraction.

FUNCTIONS

EXAMPLE FUNCTIONS

A function returns a value to the main program that called it.

```
FUNCTION dice ( )                # This is the start of the function.
    simDie = RANDOM(6)           # This statement uses the
                                 predefined function RANDOM
                                 to generate a random number
                                 between 1 and 6 which is stored
                                 in the variable simDie.
RETURN simDie                    # This returns the value of the
                                 variable simDie to the main
                                 program.
END FUNCTION                     # This ends the definition of the
                                 function.
```

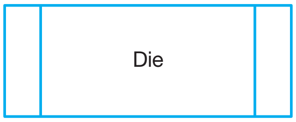
This function is called by the main program like this.

```
dieThrow = dice()
```

The value returned by the function is stored in the variable `dieThrow`.

DID YOU KNOW?

In a flowchart, a subprogram is represented by this symbol.



▲ **Figure 2.3** The symbol for a subprogram in a flowchart

In high-level programming languages, this would be written as:

PYTHON

```
def dice()
      simDie = random.randint(1, 7)
      return simDie
```

It would be called from the main program by:

```
dieThrow = dice()
```

JAVA

```
import java.util.*;
class Main {
  // define the function
  public static int dice() {
    Random r = new Random();
    return 1 + r.nextInt(6);
  }

  public static void main(String[] args) {
    // call the function
    int dieThrow = dice();
    System.out.println(dieThrow);
  }
}
```

C#

As C# is an **OOP language** it has methods rather than functions. There are a number of ways of implementing methods, but it could be written like this:

```
public static int dice()
{
  Random random = new Random();
  return random.Next(1, 7);
}
```

It would be called from the main program by: `int value = dice();.`

SUBJECT VOCABULARY

OOP language an object-oriented programming language. Instead of data structures and separate program structures, both data and program elements are combined into one structure called an object

LOCAL AND GLOBAL VARIABLES

Notice that there are two variables that store the random number generated by the function. In the function itself, the variable `simDie` is used. This variable only exists within the function and is referred to as a **local variable**.

In the main program the value returned by the function is stored in the variable `dieThrow`. It can be used anywhere within the main program and is therefore referred to as a **global variable**.

SUBJECT VOCABULARY

local variable a variable that is accessed only from within the subprogram in which it is created

global variable a variable that can be accessed from anywhere in the program, including inside subprograms

PROCEDURES

Unlike a function, a procedure does not return a value to the main program.

EXAMPLE PROCEDURE

In the dice example, a procedure would be written in pseudocode as:

```
PROCEDURE averageScore(score1, score2, score3)
BEGIN PROCEDURE
    SET total TO score1 + score2 + score3
    SET average TO total / 3
    SEND average TO DISPLAY
END PROCEDURE
```

Python
In Python, it would be exactly the same without the return command.

```
PROCEDURE dice()
    simDie = RANDOM(6)
    SEND simDIE TO DISPLAY
END PROCEDURE
```

Java
Because a procedure doesn't return a value, Java uses the void keyword to indicate that a subprogram doesn't return a value.

```
import java.util.*;
class Main {
  // define the procedure
  public static void dice() {
    Random r = new Random();
    System.out.println(1 + r.nextInt(6));
  }

  public static void main(String[] args) {
    // call the procedure
    dice();
  }
}
```

C#
In C#, the equivalent to a procedure is a method that doesn't return a value. For example:

```
public static void dice()
{
  Random random = new Random();
  int value = random.Next(1, 7);
  Console.WriteLine(value);
}
```

ARGUMENTS AND PARAMETERS

SUBJECT VOCABULARY

parameter the names of the variables that are used in the subroutine to store the data passed from the main program as arguments

Data for the functions and procedures to work on can be passed from the main program as arguments. The function accepts them as **parameters**.

PYTHON

```
# Function rectangle
def rectangle(length, width):
    area = length * width
    return area

#Main program
rectangleLength = int(input(‘Please enter the length of
the rectangle’))
rectangleWidth = int(input(‘Please enter the width of the
rectangle’))
rectangleArea  = rectangle(rectangleLength, rectangleWidth)
print(rectangleArea)
```

JAVA

```
import java.util.*;
class Main {
  // define the rectangle function
  public static int rectangle(int length, int width) {
    int area = length * width;
    return area;
  }

  public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
    System.out.print(“Please enter the length of the
rectangle: ”);
    int rectangleLength = scanner.nextInt();

    System.out.print(“Please enter the width of the rectangle: ”);
    int rectangleWidth = scanner.nextInt();
    scanner.close();

    int rectangleArea = rectangle(rectangleLength,
rectangleWidth);
    System.out.println(rectangleArea);
  }
}
```

C#

```
public static void Main()
{
  string lengthString, widthString;
  int length, width, rectangleArea;
  Console.WriteLine(“Please enter the length of the rectangle”);
  lengthString = Console.ReadLine();
  length = int.Parse(lengthString);

  Console.WriteLine(“Please enter the width of the rectangle”);
  widthString = Console.ReadLine();
  width = int.Parse(widthString);

  rectangleArea = rectangle(length, width);
  Console.WriteLine(rectangleArea);
}

public static int rectangle(int length, int width)
{
  int area;
  area = length * width;
  return area;
}
```

In this example, two data items are passed to the function – `rectangleLength` and `rectangleWidth`. These are called arguments.

The function receives them as parameters called length and width when the function is declared.

In the function, a variable is declared – area. This is called a local variable and its **scope** is within the function. If you tried to use it in the main program you would get an error message.

SUBJECT VOCABULARY

scope the region of code within which a variable is visible

Lots of arguments can be passed to the function and many values can be returned.

In the following example, two values are requested and returned.

PYTHON

```
# Function rectangle
def rectangle(length, width):
    area = length * width
    circumference = (2 * length) + (2 * width)
    return area, circumference

#Main program
rectangleLength = int(input(‘Please enter the length of
the rectangle’))
rectangleWidth = int(input(‘Please enter the width of the
rectangle’))
rectangleArea, rectangleCircumference  = rectangle
(rectangleLength, rectangleWidth)
print(rectangleArea)
print(rectangleCircumference)
```

JAVA

In Java, functions can only return one value so if you need to return more than one value, you need to put them in a list or array.

```
import java.util.*;
class Main {
  // returns an array of the area and perimeter of a rectangle
  public static int[] rectangle(int length, int width) {
    int area = length * width;
    int perimeter = 2 * (length + width);
    return new int[]{area, perimeter};
  }

  public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
    System.out.print(“Please enter the length of the rectangle: ”);
    int rectangleLength = scanner.nextInt();

    System.out.print(“Please enter the width of the rectangle: ”);
    int rectangleWidth = scanner.nextInt();
    scanner.close();

    int[] results =  rectangle(rectangleLength, rectangleWidth);
    int rectangleArea =results[0];
    int rectanglePerimeter = results[1];
    System.out.println(“Area: ” + rectangleArea);
    System.out.println(“Perimeter: ” + rectanglePerimeter);
  }
}
```

C#

Methods in C# can’t return multiple values unless you use `By Reference`, which is beyond the scope of this book.

ACTIVITY 26

From the above example, list:

a the global variables
b the local variables
c the arguments
d the parameters.

SUBPROGRAMS AND MENUS

Functions and procedures are useful when using menus in a program. When a user selects a menu option, they can be sent to a particular function or procedure.

WORKED EXAMPLE

For a system login and password control, the main program could have a menu system like this:

1 Register as a new user
2 Login
3 Change your password
4 Exit.

Now that we are using structured programming using procedures, it is relatively easy to direct a user to the part of the program they need.

The user enters a number between 1 and 4 and is directed to the correct section:

Python

```
print (“1. Register as a new user”)
print (“2. Login.”)
print (“3. Change your password.”)
print (“4. Exit.”)

choice = int(input(‘Please select a menu option.’))
if choice == 1:
    newUser()
elif choice == 2:
    login()
elif choice == 3:
    changePassword()
elif choice == 4:
    exit()
else:
    print(‘Incorrect option. Try again.’)
```

Java

```
import java.util.*;
class Main {
  // procedures can be implemented later
  // just shown here to illustrate the structure
  public static void newUser() {}
  public static void login() {}
  public static void changePassword() {}
  public static void exit() {}
  public static void main(String[] args) {
    // display main menu
    System.out.println(“1. Register as a new user”);
    System.out.println(“2. Login”);
    System.out.println(“3. Change your password”);
    System.out.println(“4. Exit”);
    // get user input
    Scanner scanner = new Scanner(System.in);
    int choice = scanner.nextInt();

    switch(choice) {
      // Register as a new user
      case 1:
        newUser();
      break;

      // Login
      case 2:
        login();
      break;

      // change password
      case 3:
        changePassword();
      break;

      // exit
      case 4:
        exit();
      break;

      // anything else
      default:
        System.out.println(“Incorrect option, try 
again”);
      break;
    }
    scanner.close();
  }
}
```

C#

```
string choiceString;
int choice;

Console.WriteLine(“1. Register as a new user”);
Console.WriteLine(“2. Login”);
Console.WriteLine(“3. Change your password”);
Console.WriteLine(“4. Exit”);
Console.WriteLine(“Please select a menu option”);
choiceString = Console.ReadLine();
choice = int.Parse(choiceString);
if (choice == 1)
{
  newUser();
}
else if (choice == 2)
{
  login();
}
else if (choice == 3)
{
  changePassword();
}
else if (choice == 4)
{
  exit();
}
else
{
  Console.WriteLine(“Incorrect option. Try again”);
}
```

ACTIVITY 27

a Describe the purpose of the program in the worked example above and explain how it functions.

b If the user inputs an incorrect option, they receive an error message and then the program terminates. Edit the program so that the program will run until a suitable option is input.

THE BENEFITS OF USING SUBPROGRAMS

Repeated sections of code need only be written once and called when necessary. This shortens the development time of a program and means that the finished program will occupy less memory space when it is run.

Subprograms also improve the structure of the code, making it easier to read through and follow what is happening. It’s less complicated to check your code and debug your program if you use subprograms because each subprogram can be coded, inspected and tested independently. If changes have to be made at a later date it is easier to change a small module than having to work through the whole program.

In large development teams different members can be working independently on different subprograms. They can use and develop standard libraries of subroutines that can be reused in other programs.

BUILT-IN FUNCTIONS

In addition to user-written subprograms, most high-level programming languages have a set of **built-in functions** for common tasks. These are designed to save the programmer time, such as functions that print, count the number of characters in a string and generate random numbers.

SUBJECT VOCABULARY

built-in functions functions that are provided in most high-level programming languages to perform common tasks

CHECKPOINT

Strengthen

SKILLS CRITICAL THINKING

S1 What are the benefits of using subprograms?

SKILLS REASONING

S2 What is meant by the scope of a variable? Use your own examples.

SKILLS REASONING

S3 What happens when a global variable and a local variable share the same name?

SKILLS CRITICAL THINKING

S4 Provide an example of a common built-in function.

Challenge

SKILLS PROBLEM SOLVING

C1 Create and implement a calculator program that:

- allows the user to enter a set of numbers
- uses separate functions to calculate the mean, mode and median
- allows the user to select which function they want
- uses appropriate validation.

How confident do you feel about your answers to these questions? If you're not sure you answered them well, try redoing the activities in this section.

SUMMARY

- A subprogram is a section of code within a larger piece of code that performs a specific task. It can be used at any point in the program.
- A function is a subprogram that returns a value to the main program.
- A procedure is a subprogram that does not return a value to the main program.
- Parameters are values that are passed to a subprogram when it is called.
- Local variables can only be accessed from within the subprogram in which they are created.
- Global variables can be accessed anywhere in the program, including inside subprograms.
- Built-in functions are functions provided in a high-level programming language to perform common tasks.

11 TESTING AND EVALUATION

However carefully an algorithm has been created and coded into a program, there will always be mistakes and errors. Before a program goes live, it should be thoroughly tested and evaluated to ensure it meets the requirements.

LEARNING OBJECTIVES

- Describe different types of error in programs
- Design and use test plans and test data
- Determine what value a variable will hold at a given point in a program
- Evaluate the strengths and weaknesses of a program and suggest improvements

DID YOU KNOW?

The Ariane 5 space rocket was launched by the European Space Agency in 1996. Its flight lasted 39 seconds before it exploded. Data about velocity was stored in 16-bit variable. This had been fine for the Ariane 4 but the 5 model was much faster and the value was too large. This one software error set of a chain of events leading to self-destruction.

LOGIC ERRORS

Logic errors occur when the thinking behind an algorithm is incorrect so that the output isn't what is expected or intended. Ideally, logic errors should be identified and fixed at the design stage.

The following algorithm is intended to work out whether a learner has passed a test. Learners need a score of 80 or above to pass. However, a logic error in the algorithm means that it produces an incorrect and unexpected result.

```
IF testScore <= 80 THEN
    SEND ‘Pass’ TO DISPLAY
ELSE
    SEND ‘Failed’ TO DISPLAY
END IF
```

EXTEND YOUR KNOWLEDGE

Research other problems and disasters that have been caused by software failures.

DID YOU KNOW?

In computer programming the order of precedence (the order in which you do each calculation) is the same as in mathematics and science – **BIDMAS**.

This is how $3^2 \times 9 + (5 - 2)$ would be evaluated.

Brackets	32 × 9 + (3)
Indices	9 × 9 + (3)
Division	
Multiplication	81 + (3)
Addition	84
Subtraction	

To calculate 24/3 – 2, the division would be calculated before the subtraction.

24/3 = 8
8 – 2 = 6

WORKED EXAMPLE

Here is an algorithm to find the average of two numbers.

```
RECEIVE number1 FROM KEYBOARD
RECEIVE number2 FROM KEYBOARD
SET average TO number1 + number2 / 2
SEND average TO DISPLAY
```

This seems logical. Two numbers are input, they are added together and then they are divided by 2.

However, if this algorithm was given 12 and 6 as the two numbers it would return 15 as the average instead of 9. There is a logic error.

Instead of adding the two numbers and then dividing by 2, as the developer intended, it is dividing the second number by 2 and then adding the result to the first number.

The developer should have written the third line as:

```
SET average TO (number1 + number2) / 2
```

SKILLS PROBLEM SOLVING

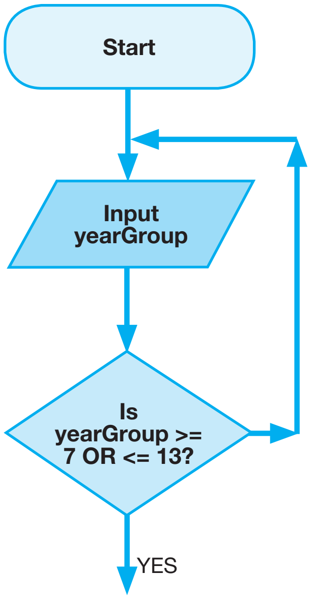
ACTIVITY 28

LOGIC ERRORS

This is part of a larger algorithm designed to ensure that input data falls within a certain range. It is part of a school management system. It checks that the 'year group' entry is acceptable. It has students aged 11 to 18 years with year groups of 7 to 13.

The staff using the system congratulated themselves on never making an error when entering the year group. However, when student lists were printed out, they immediately received complaints.

With a partner, discuss this algorithm and find the logic error in the algorithm?



▲ **Figure 2.4** Flowchart showing an error in an algorithm

WORKED EXAMPLE

Study this algorithm.

```
WHILE index < 10 DO
      SET index TO 1
      SEND index TO DISPLAY
      SET index TO index + 1
END WHILE
```

The expected output is 1, 2, 3, 4, 5, 6, 7, 8, 9.

But there is a logic error. The variable 'index' is initialised in the wrong place. It should be done before the start of the `WHILE` loop.

This algorithm would loop forever because at each turn in the loop the variable index is set to 1. It will never reach 10. This is an example of an '**infinite loop**'. The algorithm should have been written as shown below.

```
SET index TO 1
WHILE index < 10
      SEND index TO DISPLAY
      SET index TO index + 1
END WHILE
```

SUBJECT VOCABULARY

infinite loop a loop that is never-ending since the condition required to **terminate** the loop is never reached

GENERAL VOCABULARY

terminate to cause something to end or stop

SKILLS REASONING, PROBLEM SOLVING

ACTIVITY 29

FINDING AND CORRECTING ERRORS

Find and correct the errors in these algorithms.

Example 1

```
SET index TO 1
WHILE index < 10
    SEND index TO DISPLAY
END WHILE
```

Example 2

```
SET index TO 1
WHILE index < 10
    SEND index TO DISPLAY
    SET index TO index – 1
END WHILE
```

Example 3

```
SET index TO 1
WHILE index < 1
    SEND index TO DISPLAY
    SET index TO index + 1
END WHILE
```

TRACE TABLES

The formal way of checking the logic of an algorithm is to use a **trace table**.

SUBJECT VOCABULARY

logic error an error in an algorithm that results in incorrect or unexpected behaviour

trace table a technique used to identify any logic errors in algorithms. Each column represents a variable or output and each row a value of that variable

WORKED EXAMPLE

Create a trace table for the following algorithm written in Pearson Edexcel pseudocode.

```
SET number TO 3
FOR index FROM 1 TO 5 DO
    SET number1 TO number * index
    SET number2 TO number1 * 2
    IF number2 > 20 THEN
        SEND number2 TO DISPLAY
    END IF
END FOR
```

This algorithm can be traced as in Table 2.12.

NUMBER	INDEX	NUMBER1	NUMBER2	OUTPUT
3				
3	1	3	6	
3	2	6	12	
3	3	9	18	
3	4	12	24	24
3	5	15	30	30

▲ **Table 2.12** Trace table

The value of the variable number remains at 3 throughout, but as the index increases from 1 to 5, then so do the values of `number1` and `number2`.

When the value of `number2` is greater than 20, its value is output.

WORKED EXAMPLE

Here is a program.

Python

```
y = 2
for x in range(1, 7):
      y = y + x
print(y)
```

Java

```
class Main {
  public static void main(String[] args) {
    int y = 2;
    for(int x = 1; x < 7; x++) {
      y += x;
    }
    System.out.println(y);
  }
}
```

C#

```
int number = 3;
int number1, number2;

for (int index = 1; index <= 5; index++)
{
    number1 = number * index;
    number2 = number1 * 2;
    if (number2 > 20)
    {
        Console.WriteLine(number2);
    }
}
```

X	Y	OUTPUT	EXPLANATION
1	2		When the loop starts, X becomes 1 and Y already is equal to 2.
2	3		When X is incremented to 2, Y is equal to 3 (2+1) from the previous loop.
3	5		When X is incremented to 3, Y= 3 + 2 from the previous loop.
4	8		Explanations as above.
5	12		
6	17		
6	23	23	The final value of Y is output.

▲ **Table 2.13** Trace table of the programs above

ACTIVITY 30

Complete a trace table for this program written in a high-level language.

Python

```
number1 = 2
number2 = 3
for index in range(1, 6)
    number1 = number1 * index
    number2 = number2 + number1
```

Java

```
class Main {
  public static void main(String[] args) {
    int number1 = 2;
    int number2 = 3;
    for(int index = 1; index < 6; index++) {
      number1 = number1 * index;
      number2 = number2 + number1;
    }
  }
}
```

C#

```
int number1 = 2;
int number2 = 3;
for (int index = 1; index < 6; index++)
{
      number1 = number1 * index;
      number2 = number2 + number1;
}
```

SKILLS PROBLEM SOLVING

ACTIVITY 31

The following program is intended to count the number of female learners in a class.

Python

```
gender = [‘M’, ‘M’, ‘F’, ‘M’, ‘F’, ‘F’, ‘M’, ‘F’, ‘M’, ‘F’]
length = len(gender)
count = 0
for index in range(length):
      if gender[index] ‘F’:
            count = count + 1
```

Java

```
class Main {
  public static void main(String[] args) {
    String[] gender = {“M”, “M”, “F”, “M”, “F”, “F”,
“M”, “F”, “M”, “F”};
    int length = gender.length;
    int count = 0;
    for(int index = 0; index < length; index++) {
      if(gender[index].equals(“F”)) {
        count++;
      }
    }
  }
}
```

C#

```
int length, count;
char[] gender;
gender = new char[]{ ‘M’, ‘M’, ‘F’, ‘M’, ‘F’, ‘F’, ‘M’,
‘F’, ‘M’, ‘F’ };

count = 0;
length = gender.Length;
for (int index = 0; index < length; index++)
{
  if (gender[index] == ‘F’)
  {
    count += 1;
  }
}
```

1 Create and complete a trace table for this algorithm.

2 Create a trace table for the algorithm below using the following sample data: 3, 12, 21, 28, 0.

Python

```
total = 0
number = int(input(‘Please enter the number’)
while number > 0:
      total = total + number
      number = int(input(‘Please enter the number’)
print total
```

Java

```
import java.util.*;
class Main {
  public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
    int total = 0;
    System.out.print(“Please enter the number: “);
    int number = scanner.nextInt();
    while(number > 0) {
      total += number;
      number = scanner.nextInt();
    }
    System.out.println(total);
  }
}
```

C#

```
int total = 0;
int number;
string numberString;

Console.WriteLine(“Please enter the number”);
numberString = Console.ReadLine();
number = int.Parse(numberString);

while (number > 0)
{
   total += number;
   Console.WriteLine(“Please enter the number”);
   numberString = Console.ReadLine();
   number = int.Parse(numberString);
}
Console.WriteLine(total);
```

SYNTAX ERRORS

Syntax errors occur when the grammar rules of a programming language are not followed.

They prevent the code from being compiled or translated.

Examples of syntax errors are:

- writing ‘prnit’ instead of ‘print’
- missing out a closing bracket
- missing out quotation marks.

RUNTIME ERRORS

SUBJECT VOCABULARY

runtime error an error that occurs while the program is running – the operation the computer is asked to do is impossible to execute

Runtime errors occur during program execution and are the most difficult to predict and spot.

This program is designed to take two numbers, divide the first number by the second number and output the result. It will work as intended at least some of the time. However, if the user entered 5 and 0, a runtime error would occur because it is impossible for the computer to divide 5 by 0.

Python

```
firstNumber = int(input(‘Please enter the first number’)
secondNumber = int(input(‘Please enter the second number’)
result = firstNumber / secondNumber
print(result)
```

Java

```
import java.util.*;
class Main {
  public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
    System.out.print(“Please enter the first number: ”);
    int firstNumber = scanner.nextInt();
    System.out.print(“Please enter the second number: ”);
    int secondNumber = scanner.nextInt();
    scanner.close();

    // do the division (converting all integers to reals)
    double result = (double)firstNumber / (double)
secondNumber;
    System.out.println(result);
  }
}
```

C#

```
int firstNumber, secondNumber, result;
string firstNumberString, secondNumberString;

Console.WriteLine(“Please enter the first number”);
firstNumberString = Console.ReadLine();
firstNumber = int.Parse(firstNumberString);

Console.WriteLine(“Please enter the second number”);
secondNumberString = Console.ReadLine();
secondNumber = int.Parse(secondNumberString);

result = firstNumber / secondNumber;
Console.WriteLine(result);
```

ERROR SUMMARY

This table gives a summary of the three types of error you are likely to encounter.

TYPE OF ERROR	DESCRIPTION
Logic	The program seems to run normally; however, there is an error in the logic of the program, which means it does not produce the result you expect.
Syntax	Syntax refers to the rules of the programming language. A syntax error means that part of the code breaks the rules of the language, which stops it running.
Runtime	An error that occurs when the computer tries to run code that it cannot execute.

▲ **Table 2.14** Summary of the three types of error you are likely to encounter

USING AN INTEGRATED DEVELOPMENT ENVIRONMENT (IDE)

SUBJECT VOCABULARY

Integrated Development Environment (IDE) a package that helps programmers to develop program code. It has a number of useful tools, including a source code editor and a debugger

debugger a computer program that assists in the detection and correction of errors in other computer programs

You probably already have first-hand experience of using an **Integrated Development Environment (IDE)** when writing code. It's definitely worth taking some time to get to know the IDE that comes with the language you are using. Useful features such as syntax highlighting, code auto complete and auto indent will help to make your programming experience far less stressful, especially at the beginning.

One of the most useful features of an IDE is the **debugger**. One of its tasks is to flag up syntax errors in the code and issue helpful error messages. It is really important that you get lots of practice interpreting error messages and fixing errors in your code.

HINT

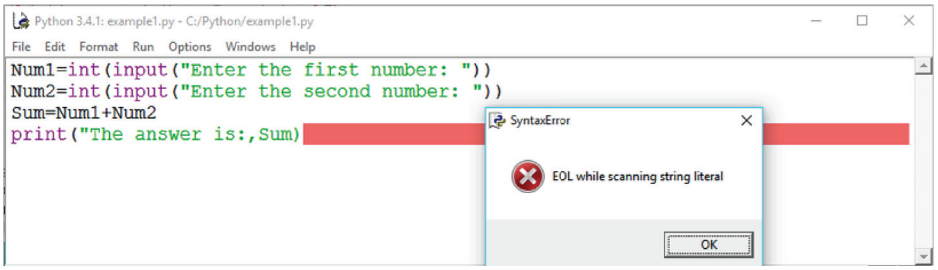
Here are some error messages that you are likely to see in popular IDEs.

IDLE (a popular Python IDE)

- `SyntaxError:` invalid syntax – *part of the code breaks one of the rules of the programming language.*
- `IndentationError:` expected an indented block – *statements after a colon must be indented.*
- `TypeError:` Can't convert 'int' object to str implicitly – *trying to join a string and an integer together.*
- `ZeroDivisionError:` integer division or modulo by zero – *trying to divide a value by 0.*
- `NameError:` name is not defined – *referring to a variable or subprogram that does not exist.*

Eclipse (a popular Java IDE)

- ; expected – *each statement should end with a semicolon.*
- Cannot find symbol – *referring to a variable or subprogram that does not exist.*
- Incompatible types – *trying to mix data of different types, for example a string and an integer.*


Python 3.4.1: example1.py - C:/Python/example1.py
File Edit Format Run Options Windows Help
SyntaxError
OK

```
Num1=int(input("Enter the first number: "))
Num2=int(input("Enter the second number: "))
Sum=Num1+Num2
print("The answer is:,Sum)
```

EOL while scanning string literal

▶ **Figure 2.5** A syntax error flagged up by an IDE in Python code

THE TEST PLAN

At the start of a programming project, it is crucial to make a list of the requirements that the program is expected to meet. Throughout the development phase of the project, you should regularly refer back to this list of requirements to check that you are on track to achieve them.

Deciding how to test the finished program to make sure that it fully meets the requirements can't be left to the last moment either. As part of the design stage, you should create a test plan listing the tests you will carry out, the data that will be used for each test and the expected result.

MARKS TO GRADES

```
RECEIVE examMark FROM (INTEGER) KEYBOARD
IF examMark >= 80 THEN
    SEND ‘A’ TO DISPLAY
ELSE
    IF examMark >= 70 THEN
        SEND ‘B’ TO DISPLAY
    ELSE
        IF examMark >= 60 THEN
            SEND ‘C’ TO DISPLAY
        ELSE
            IF examMark > 0 THEN
                SEND ‘D’ TO DISPLAY
            ELSE
                SEND ‘FAIL’ TO DISPLAY
            END IF
        END IF
    END IF
END IF
```

This algorithm converts an exam mark into a grade. It should only accept marks between 0 and 100.

The test plan extract in Table 2.15 shows some of the tests that have been planned for the finished program to ensure that it meets all the requirements.

TEST NO.	PURPOSE OF THE TEST	TEST DATA	EXPECTED RESULT	ACTUAL RESULT	ACTION NEEDED/ COMMENTS
1	To check correct conversion of valid mark	0 55 65 75 85	‘FAIL’ ‘D’ ‘C’ ‘B’ ‘A’		
2	To check correct conversion of boundary mark	0 1, 59 60, 69 70, 79 80, 100	‘FAIL’ ‘D’ ‘C’ ‘B’ ‘A’		
3	To check response to erroneous mark	–5 105	Error message Error message		

▲ **Table 2.15** Test plan extract

Only the first four columns of the test plan table can be completed at the design stage. The remaining columns are filled in when the program is complete.

At the start, it’s unlikely that you’ll know every test that will be needed to ensure the program works as intended. The test plan is not a fixed document. If additional tests are required, they should be added to the test plan.

If a test produces the expected result, you can simply write 'None' in the 'Action needed/comments' column. If, however, that is not the case then you should instead note what went wrong and what you did to put it right.

It is important to select suitable test data for the tests. Test data falls into three different categories: normal, boundary and erroneous.

Normal data	Data that is well within the limits of what should be accepted by the program.	Test 1 uses normal data to check if marks are converted into grades correctly (e.g. a mark of 65 should be converted to a grade C).
Boundary data	Data that is at the outer limits of what should be accepted by the program.	Test 2 uses boundary data to check that the program works correctly with marks at the upper and lower boundaries (e.g. a mark of 60 and a mark of 69 should both convert to a grade C).
Erroneous data	Data that should not be accepted by the program.	Test 3 uses erroneous data to check that the program does not accept it.

▲ **Table 2.16** Test data categories

Testing is just as important as writing code because it ensures that the finished program works correctly and fully meets the requirements. You should use a 'bottom up' approach to testing (i.e. test each subprogram as you develop it and then test the whole program once it is finished).

Here is the completed test plan for the grade calculator program. As you can see, the expected result was not produced when the program was tested with erroneous data. A range check had to be added to the program to ensure that only marks between 0 and 100 can be entered.

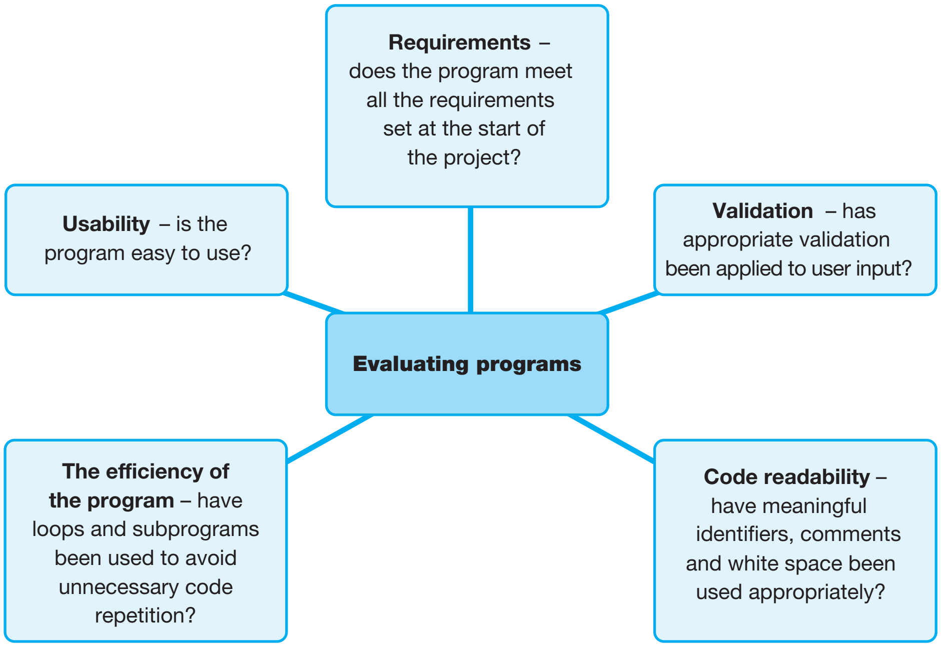
TEST NO.	PURPOSE OF THE TEST	TEST DATA	EXPECTED RESULT	ACTUAL RESULT	ACTION NEEDED/ COMMENTS
1	To check correct conversion of valid mark	0 55 65 75 85	'FAIL' 'D' 'C' 'B' 'A'	'FAIL' 'D' 'C' 'B' 'A'	None
2	To check correct conversion of boundary mark	0 1, 59 60, 69 70, 79 80, 100	'FAIL' 'D' 'C' 'B' 'A'	'FAIL' 'D' 'C' 'B' 'A'	None
3	To check response to erroneous mark	–5 105	Error message Error message	'FAIL' 'A'	A range check has been added to ensure that only valid marks can be added.

▲ **Table 2.17** The completed test plan for the grade calculator program

After you have carried out all of the tests and made all the necessary changes, the program should be retested. This is to ensure that the improvements you have made haven't introduced any new errors into the program.

EVALUATING PROGRAMS

You need to be able to identify the strengths and weaknesses of your own programs as well as those created by other programmers. This will enable you to identify techniques that work well and aspects that could be improved.


Requirements – does the program meet all the requirements set at the start of the project?
Usability – is the program easy to use?
Validation – has appropriate validation been applied to user input?
Evaluating programs
The efficiency of the program – have loops and subprograms been used to avoid unnecessary code repetition?
Code readability – have meaningful identifiers, comments and white space been used appropriately?

▶ **Figure 2.6** Consider the aspects shown when evaluating a program

SKILLS PROBLEM SOLVING

ACTIVITY 32

CREATING A GUESSING GAME ALGORITHM

1 Develop an algorithm for a simple guessing game. The algorithm must:
 a generate a random number between 1 and 6
 b ask the user to input a number between 1 and 6
 c reject any values outside the acceptable range
 d display 'Well Done' if the user guesses correctly or 'Try Again' if their guess is incorrect.

2 Develop a test plan for this guessing game program.

3 Can you write the guessing game program in the high-level programming language you are studying?

4 Test your program by carrying out the tests you planned. Ensure you update your test plan after completing each test.

CHECKPOINT

Strengthen

SKILLS CRITICAL THINKING, REASONING

S1 What are the three types of error associated with program development? Can you identify the stage(s) of development at which they are most likely to occur?

SKILLS CRITICAL THINKING

S2 Outline the function of a trace table.

SKILLS REASONING

S3 What are the features of an IDE that help programmers write error-free code?

SKILLS CRITICAL THINKING

S4 What is the function of a test plan in program development?

SKILLS CRITICAL THINKING

S5 What is meant by normal, boundary and erroneous data?

SKILLS REASONING

S6 Why might it be necessary to retest a program once all the planned tests are completed?

Challenge

SKILLS PROBLEM SOLVING

C1 Think of a problem which could be solved using a computer program. What are the requirements for the program? Create a solution and draw up a test plan for it.

SKILLS PROBLEM SOLVING

C2 Develop and implement the program. Conduct your planned tests and make any necessary changes to your program, ensuring your test plan is kept up to date.

SKILLS REASONING

C3 Reflect on your program and provide an evaluation.

How confident do you feel about your answers to these questions? If you're not sure you answered them well, try completing Activities 28–32.

SUMMARY

- Logic errors occur when there is an error in the logic of the code, causing the program to produce an unexpected result.
- A syntax error occurs when part of the code breaks the rules of the programming language.
- A runtime error occurs while the program is running and it is asked to do something that is impossible to do.
- Trace tables can be used to manually trace the execution of an algorithm, allowing you to track the changes in variable values.
- Before creating a program, it is important to produce a test plan that outlines how the final program will be tested to ensure it meets the requirements.
- Evaluating a program involves considering its strengths and weaknesses and identifying areas for improvement.

UNIT QUESTIONS

SKILLS CRITICAL THINKING, REASONING AO2

1 a Identify the line number(s) that show one example of each of these structural components in the program below: **(6)**

- variable initialisation
- type declaration
- selection
- iteration
- data structure
- subprogram.

```
 1  SET valuesArray TO [3, 9, 12, 16, 4, 98]
 2  REAL max
 3
 4  FUNCTION maxCalc(values)
 5  BEGIN FUNCTION
 6      SET length TO LENGTH(values)
 7      SET max TO 0
 8      FOR index = 0 TO length – 1 DO
 9          IF values[index] > max THEN
10          SET max TO values[index]
11          END IF
12      END FOR
13  RETURN max
14  END FUNCTION
15
16  SET max TO maxCalc(valuesArray)
    SEND max TO DISPLAY
```

b Identify the line number where a subprogram call is made. **(1)**

HINT

Question 1 is testing your ability to identify the seven key structural components of a program. Make sure you have identified the line numbers in which the components can be found.

SKILLS CRITICAL THINKING, REASONING AO3

2 Describe what this algorithm does. **(2)**

```
SET scores TO [45, 67, 34, 98, 52]
SET length TO LENGTH(scores)
SET count TO 0
FOR index FROM 0 TO length – 1 DO
    IF scores[index] >= 50 THEN
        SET count TO count + 1
    END IF
END FOR
```

SKILLS CRITICAL THINKING, PROBLEM SOLVING, REASONING A02

3 Draw and complete a trace table for this algorithm with these column headings: **(5)**

- length
- count
- index
- scores[index].

HINT

Question 3 tests your ability to trace an algorithm using a trace table. Remember each variable change should be recorded and each time a line of code alters the value of a variable or variables a new row of the trace table should be completed.

SKILLS CRITICAL THINKING, REASONING A02

4 Open file Q01a. Answer these questions about the code.

a State the name of a user-defined subprogram. **(1)**
b State the name of one in-built subprogram. **(1)**
c State the names of one input parameter. **(1)**
d State the name of a global variable. **(1)**
e State the name of a local variable. **(1)**
f State the line number of the command that 'calls' the variable. **(1)**

HINT

These questions are asking you to 'state' various elements in the program. You do not have to describe them or explain how they function.

SKILLS CRITICAL THINKING, PROBLEM SOLVING, REASONING A02 A03

5 Open file Q02a. Answer these questions about the code.
A data structure has been used to store the results of a survey to find favourite brands of automobiles.

They have been stored in ascending order.

a Name this type of data structure. **(1)**
b The user is asked to enter the name of a brand and their input is converted into upper case. Explain why this is done. **(2)**
c Complete the program to search for the brand name entered in the data structure. If it is present, then the program should inform the user of its position, e.g. Audi is in position 1.
The user should also be informed if it is not in the list.
Save your completed program as Q02b. **(6)**

HINT

This question is asking you to traverse a list to find a particular item. When you print out the result text and variables have to be concatenated.

SKILLS PROBLEM SOLVING A03

6 Open file Q03a.

It shows a list of users and their (not very strong) passwords stored in a list. Complete the program so that a user can enter their user name. If the name exists, then they should be asked for their password. They should be informed of the following:

- if the username and password are correct
- if the name they entered is not recognised. **(10)**

HINT

The question requires you to use a loop and selection.