

Computer Science for GCSE

Aligned to OCR GCSE (9–1) Computer Science, J277

Adriano G Platania

First published in London, July 2026

© 2026 Adriano G Platania

Terms of use

This book may be read, downloaded, and shared free of charge for personal study and for classroom use. It may not be resold or redistributed for commercial purposes.

This book is intended as a supplement to the textbooks officially adopted by your school. Please use it alongside your prescribed course materials and follow your teacher's guidance.

About this book

I wrote this book for my own classes, which means it was written for you. It follows the OCR GCSE (9–1) Computer Science specification, J277, section by section, so nothing in here is padding and nothing the exam needs is missing.

Each section works the same way. A short purple box at the start asks three questions from earlier sections; answer them from memory before reading on, because retrieving something is what fixes it in place. The section then teaches the content, flags the misconceptions I watch students fall into every year, and gives you the key terms the mark schemes reward. All pseudocode is written in OCR's own Exam Reference Language, the dialect the exam is set in. The green Python boxes turn ideas into running code; type them out rather than reading them, since your fingers learn faster than your eyes. Every section ends with exercises in three tiers, from quick checks to full exam-style questions with marks attached.

Worked answers, with mark allocations broken down the way examiners think, are collected at the back.

Adriano G Platania, London, 2026

Table of Contents

Section 1 – Systems architecture (spec 1.1)	8
1.1 The CPU and the fetch-decode-execute cycle	8
1.2 Inside the CPU	8
1.3 Von Neumann architecture	9
1.4 What makes one CPU faster than another	10
1.5 Embedded systems	10
Key terms	10
Section 1 exercises	11
Quick check	11
Practice	11
Exam-style	11
Section 2 – Memory and storage (spec 1.2)	12
2.1 Primary storage: RAM, ROM and virtual memory	12
2.2 Secondary storage	12
2.3 Units and binary numbers	13
Binary to denary and back	13
Binary addition	13
Binary shifts	13
2.4 Hexadecimal	14
2.5 Characters	14
2.6 Images	14
2.8 Compression	16
Section 2 exercises	17
Quick check	17
Practice	17
Exam-style	17
Section 3 – Computer networks, connections and protocols (spec 1.3)	18
3.1 LANs, WANs and performance	18
3.2 Client-server and peer-to-peer	19
3.3 Network hardware	19
3.4 The internet, DNS and the cloud	19
3.5 Topologies	20
3.6 Wired and wireless	20

3.7 Addresses	20
3.8 Protocols and layers	21
Key terms	21
Section 3 exercises	22
Quick check	22
Practice	22
Exam-style	22
Section 4 – Network security (spec 1.4)	23
4.1 Threats	23
4.2 Defences	23
Key terms	24
Section 4 exercises	25
Quick check	25
Practice	25
Exam-style	25
Section 5 – Systems software (spec 1.5)	26
5.1 The operating system	26
5.2 Utility software	26
Key terms	27
Section 5 exercises	27
Quick check	27
Practice	27
Exam-style	27
Section 6 – Ethical, legal, cultural and environmental impacts (spec 1.6)	28
6.1 The impacts	28
6.2 The legislation	28
6.3 Open source and proprietary software	29
Key terms	29
Section 6 exercises	29
Quick check	29
Practice	30
Exam-style	30
Section 7 – Algorithms (spec 2.1)	31
7.1 Computational thinking	31
7.2 Designing, creating and refining algorithms	32

Inputs, processes and outputs	32
Flowcharts	32
The OCR Exam Reference Language.....	33
Trace tables	33
Syntax errors and logic errors	34
7.3 Searching algorithms	34
Linear search	34
Binary search	35
7.4 Sorting algorithms.....	36
Bubble sort	36
Insertion sort.....	37
Merge sort.....	37
Choosing between them	38
Key terms	38
Section 7 exercises.....	39
Quick check.....	39
Practice	39
Exam-style	39
Section 8 – Programming fundamentals (spec 2.2).....	40
8.1 Variables, constants and assignment	40
8.2 Operators	41
8.3 The three constructs	41
8.4 Strings	42
8.5 Arrays	43
8.6 File handling.....	43
8.7 Records and SQL	43
8.8 Sub programs	44
8.9 Random numbers	45
Key terms	45
Section 8 exercises.....	46
Quick check.....	46
Practice	46
Exam-style	46
Section 9 – Producing robust programs (spec 2.3).....	47

9.1 Defensive design	47
9.2 Maintainability	48
9.3 Testing.....	48
Key terms	48
Section 9 exercises.....	49
Quick check.....	49
Practice	49
Exam-style	49
Section 10 – Boolean logic (spec 2.4)	50
10.1 The three gates	50
10.2 Combining gates	50
Key terms	51
Section 10 exercises.....	51
Quick check.....	51
Practice	51
Exam-style	52
Section 11 – Programming languages and IDEs (spec 2.5)	53
11.1 High-level and low-level languages	53
11.2 Translators: compilers and interpreters	53
11.3 IDEs	54
Key terms	54
Section 11 exercises.....	54
Quick check.....	54
Practice	54
Exam-style.....	54
Answers.....	55
Section 1 – Systems architecture.....	55
Section 2 – Memory and storage.....	55
Section 3 – Computer networks, connections and protocols	56
Section 4 – Network security	57
Section 5 – Systems software	57
Section 6– Ethical, legal, cultural and environmental impacts	57
Section 7 – Algorithms	58
Section 8 – Programming fundamentals	59
Section 9 – Producing robust programs	60
Section 10 – Boolean logic.....	60

Section 11 – Programming languages and IDEs	61
References and Bibliography	62
Primary specification and assessment sources	62
Legislation	62
Programming reference.....	63
Note on examination material.....	63

Section 1 – Systems architecture (spec 1.1)

Everything a computer does, from rendering a game to autocorrecting your texts into nonsense, comes down to one component executing very simple instructions unimaginably fast. This section is about that component: what the CPU is made of, how it works through a program, and why one CPU can be faster than another.

1.1 The CPU and the fetch-decode-execute cycle

The central processing unit's job is to carry out the instructions of a program, one after another, in a repeating loop called the fetch-decode-execute cycle. Fetch the next instruction from memory. Decode it to work out what it means. Execute it. Then go straight back to the start and do it again. A modern CPU runs this loop billions of times a second, and that relentless simple loop is the entire trick.

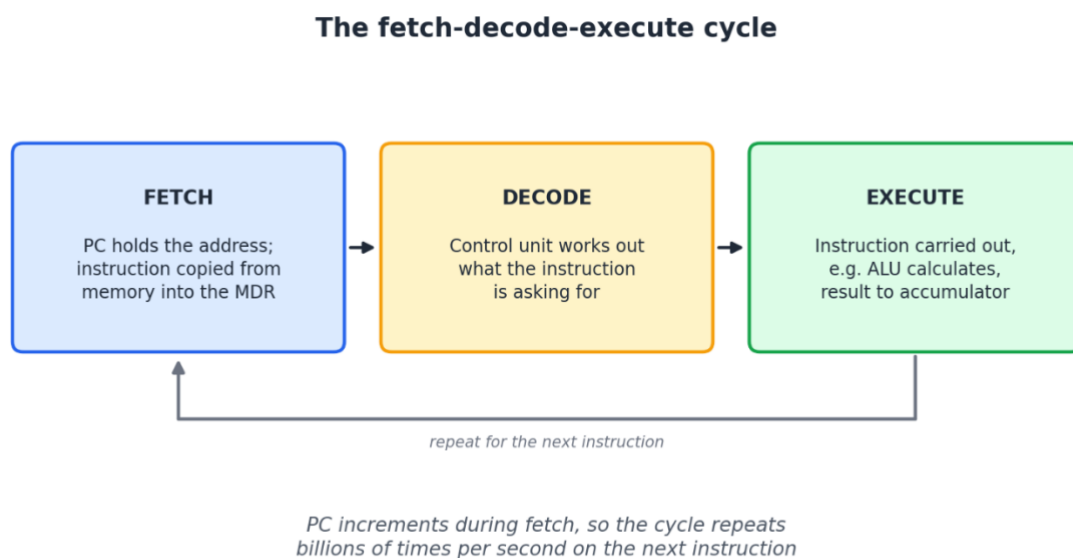


Fig 1.1 – The fetch-decode-execute cycle

1.2 Inside the CPU

The control unit (CU) is the manager. It decodes instructions, coordinates the other components, and sends the control signals that keep everything in step.

The arithmetic logic unit (ALU) is the calculator. It performs the maths (adding, subtracting) and the logical comparisons (is this bigger than that?) that instructions ask for.

Cache is a small amount of extremely fast memory built into the CPU itself. It stores the instructions and data used most often, so the CPU does not have to make the comparatively slow trip out to main memory every time.

Registers are tiny storage locations inside the CPU that hold one value each. Four of them have names you must know.

- Program counter (PC): holds the memory address of the next instruction to fetch. It increments during every cycle, which is how the CPU steps through a program.
- Memory address register (MAR): holds the address in memory currently being read from or written to.
- Memory data register (MDR): holds the actual data or instruction that was fetched from, or is about to be written to, that address.
- Accumulator (ACC): holds the results of calculations done by the ALU.

Common misconception. Students often say the MAR “stores instructions”. It never does; it stores addresses. The address goes in the MAR, and whatever lives at that address arrives in the MDR. Address in one, contents in the other.

1.3 Von Neumann architecture

Nearly every computer you have ever used follows a design described by John von Neumann in 1945, and its big idea sounds almost too simple: store the program's instructions in the same memory as its data. Before this, machines were rewired to change what they did. Afterwards, changing the program just meant loading different contents into memory, and the general-purpose computer was born.

Von Neumann architecture: one memory for data and instructions

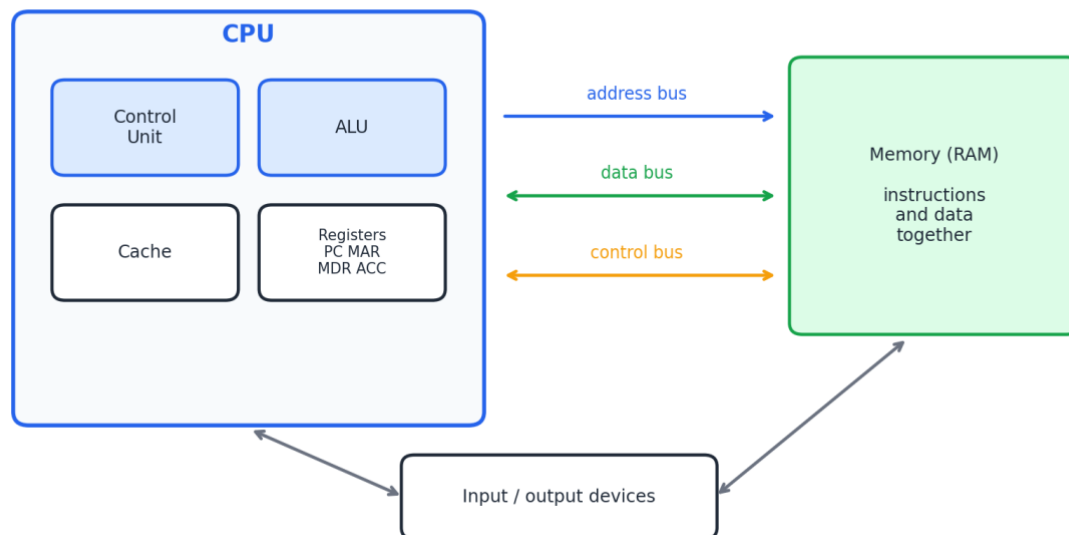


Fig 1.2 – Von Neumann architecture, with the CPU connected to memory by three buses

The CPU talks to memory across three buses: the address bus carries the location being accessed, the data bus carries the actual values in both directions, and the control bus carries the signals that coordinate it all, such as whether this is a read or a write.

1.4 What makes one CPU faster than another

Clock speed, measured in gigahertz (GHz), is the number of cycles the CPU performs per second. A 3.2 GHz processor runs 3.2 billion cycles every second, so a higher clock speed means more instructions completed in the same time.

Number of cores. Each core is a complete processing unit able to run its own fetch-decode-execute cycle, so a quad-core CPU can genuinely work on four things at once. The catch, and examiners love this catch, is that doubling the cores rarely doubles the speed: many tasks cannot be split neatly into parallel pieces, so some cores sit waiting.

Cache size. More cache means more of the frequently-used instructions and data are instantly available, so the CPU spends less time waiting on main memory.

Exam tip. Performance questions want the mechanism, not just the fact. “More cores is faster” scores little; “more cores allow more instructions to be processed simultaneously, provided the software can use them” scores well. Always say why.

1.5 Embedded systems

An embedded system is a computer built into a larger device to do one dedicated job: the controller in a washing machine, the thermostat on a wall, the engine management unit in a car. Because the job never changes, the hardware can be small, cheap and low-power, and the software is usually stored permanently and cannot be altered by the user. Compare that to your laptop, a general-purpose machine that will run whatever you install on it. The distinction, dedicated function versus general purpose, is precisely what exam questions test.

Key terms

Term	Meaning
CPU	The component that carries out the instructions of a program
Fetch-decode-execute	The repeating cycle by which the CPU processes each instruction
Control unit	Decodes instructions and coordinates the CPU's components
ALU	Performs arithmetic and logical operations
Cache	Small, very fast memory inside the CPU holding frequently used data
Register	A tiny storage location inside the CPU holding a single value
Von Neumann architecture	A design where instructions and data share the same memory
Clock speed	The number of FDE cycles per second, measured in GHz
Core	An individual processing unit within a CPU
Embedded system	A computer with one dedicated function built into a larger device

Section 1 exercises

Quick check

- 1. State the purpose of the program counter. [1]
- 2. Which register holds the results of ALU calculations? [1]
- 3. State what is meant by an embedded system. [1]

Practice

- 4. Describe what happens during the fetch stage of the fetch-decode-execute cycle. Name the registers involved. [3]
- 5. Explain why increasing cache size can improve CPU performance. [2]

Exam-style

- 6. A gamer is choosing between a 3.6 GHz dual-core CPU and a 2.8 GHz quad-core CPU. Discuss which they should choose, referring to how each factor affects performance. [4]
- 7. A smart doorbell contains an embedded system. Give two reasons why an embedded system is used rather than a general-purpose computer. [2]

Section 2 – Memory and storage (spec 1.2)

Before you start – from earlier sections

1. State the purpose of the program counter.
2. Give two factors, other than clock speed, that affect CPU performance.
3. What is an embedded system?

A computer needs somewhere to hold the program it is running right now, somewhere to keep your files when the power is off, and a way of turning everything, numbers, text, photos, music, into patterns of 1s and 0s. This is the biggest section in Component 01, and also the one with the most guaranteed exam marks, because the conversions and calculations in it are pure method: learn the method, collect the marks.

2.1 Primary storage: RAM, ROM and virtual memory

RAM (random access memory) holds the programs and data currently in use. It is fast, and it is volatile, meaning everything in it vanishes when the power goes off, which is why an unsaved essay dies with a power cut. More RAM lets a computer hold more programs open at once without slowing down.

ROM (read-only memory) is non-volatile and holds the small startup program (the bootstrap) that gets the computer going and loads the operating system. Its contents survive power-off and cannot normally be changed.

Virtual memory is the fallback for when RAM runs out. The operating system moves data that has not been used recently out of RAM into a reserved area of secondary storage, freeing RAM for what is active. It keeps everything running, but at a cost: secondary storage is far slower than RAM, so a machine leaning heavily on virtual memory feels like it is wading through treacle. The real fix is more RAM.

Common misconception. “ROM is where my files are saved” comes up every single year. Your files live in secondary storage. ROM holds the bootstrap and nothing of yours.

2.2 Secondary storage

Secondary storage is the non-volatile storage that keeps programs and data when the power is off. There are three technologies, and exam questions are almost always about choosing the right one for a scenario, so learn them as a set of trade-offs.

Type	How it works	Character
Magnetic (hard disk drive)	Data stored as magnetised spots on spinning platters, read by a moving arm	Huge capacity for the money; moving parts make it slower and vulnerable to knocks
Optical (CD, DVD, Blu-ray)	Data stored as pits on a reflective disc, read by laser	Cheap and very portable; low capacity and slow; easily scratched
Solid state (SSD, USB stick, memory card)	Data stored electronically in flash memory, no moving parts	Very fast, silent, durable, low power; more expensive per gigabyte

When a question asks you to recommend a device, judge it against the six characteristics OCR names: capacity, speed, portability, durability, reliability and cost. A video editor wants speed, so SSD. An archive of old CCTV footage wants cheap bulk capacity, so magnetic. Handing a client a copy of their wedding video wants cheap portability, so optical. Match the characteristic to the need and say so explicitly.

2.3 Units and binary numbers

Everything in a computer is stored as binary because hardware is built from switches, and a switch has exactly two states. A single 1 or 0 is a bit; 4 bits make a nibble; 8 bits make a byte. From there, OCR uses decimal prefixes: 1,000 bytes in a kilobyte (KB), 1,000 KB in a megabyte (MB), then gigabyte (GB), terabyte (TB) and petabyte (PB), each a thousand times the last. You will also see 1,024 used instead of 1,000, because 1,024 is 2^{10} , a neat binary number. OCR accepts either in conversions and calculations. This book uses 1,000 throughout because it is far quicker on a non-calculator paper; whichever you choose, stick to it within one answer.

Binary to denary and back

An 8-bit binary number is read against place values that double from the right: 128, 64, 32, 16, 8, 4, 2, 1. To convert binary to denary, add up the place values sitting under each 1. So 10110100 is $128 + 32 + 16 + 4 = 180$. To go the other way, work through the place values from the left, subtracting each one that fits: 75 takes a 64 (11 left), then an 8 (3 left), then a 2 and a 1, giving 01001011. The leftmost bit, worth 128 in a byte, is the most significant bit (MSB), because changing it changes the value the most. The rightmost bit, worth 1, is the least significant bit (LSB). A binary number with fewer than eight bits simply gains leading zeros: 11010 is the same as 00011010.

Binary addition

Adding binary numbers uses the same column method as primary school addition with four rules: $0+0=0$, $0+1=1$, $1+1=0$ carry 1, and $1+1+1=1$ carry 1. The interesting case is when adding two 8-bit numbers produces a ninth bit. There is nowhere for it to go, and this is called overflow: the result is too large for the space available, so the stored answer is wrong.

Binary shifts

Shifting every bit one place to the left multiplies the number by 2; one place to the right divides it by 2 (losing any remainder). A shift of two places multiplies or divides by 4, three places by 8, and so on. So 00001100 (12) shifted left twice becomes 00110000 (48). Watch for 1s falling off the end of a right shift, and for left shifts causing overflow.

2.4 Hexadecimal

Hexadecimal is base 16, using digits 0 to 9 and letters A to F (where A = 10 up to F = 15). Computer scientists use it because one hex digit represents exactly one nibble, so a whole byte is just two hex characters: far easier for humans to read, write and compare than eight bits. 11010111 splits into 1101 (13, which is D) and 0111 (7), giving D7. To convert hex to denary, the first digit is worth 16s: D7 is $13 \times 16 + 7 = 215$. To go from denary to hex, divide by 16: the whole-number answer is the first digit and the remainder is the second. For 215, 16 goes in 13 times (208) with 7 left over, so the digits are 13 and 7, which is D7. If dividing by 16 without a calculator feels risky, convert to binary first and split it into nibbles; both routes give the same answer. Two hex digits cover every value from 00 to FF, exactly 0 to 255.

Exam tip. Always split a byte into two nibbles and convert each nibble separately. Students who try to convert all 8 bits to denary and then to hex take three times as long and make twice the errors.

2.5 Characters

Text is stored by giving every character a number, and the mapping between characters and numbers is a character set. ASCII uses 7 bits, enough for 128 characters: fine for English, hopeless for the world. In OCR exams, ASCII codes are always written using 8 bits, so they start with a leading 0: A, code 65, appears as 01000001. Unicode extends the same idea with enough bits to cover essentially every alphabet, plus emoji. Two things about character sets earn marks: more bits per character means more characters available but larger file sizes, and the codes are logically ordered, so the code for B is one more than the code for A, which is exactly why you could sort strings alphabetically by comparing numbers. Text files follow the same logic as images and sound: file size in bits = bits per character $\times$ number of characters. A 2,000-character essay stored in 8-bit ASCII is $2,000 \times 8 = 16,000$ bits, which is 2,000 bytes, or 2 KB.

2.6 Images

A bitmap image is a grid of pixels, and each pixel is a binary number describing its colour. The number of bits per pixel is the colour depth: 1 bit gives two colours, 8 bits give 256, and 24 bits give around 16.7 million. Resolution is the number of pixels in the image. Raise either one and the image improves, and the file gets bigger, because:

A bitmap is just pixels, and pixels are just binary

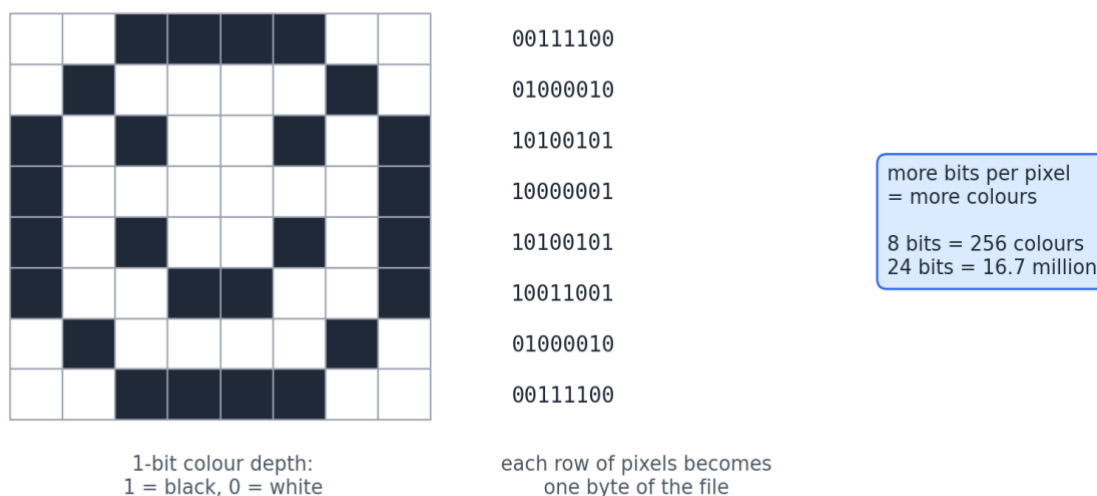


Fig 2.1 – An 8 × 8 bitmap at 1-bit colour depth, and the binary that stores it

file size in bits = colour depth × image height (px) × image width (px). Divide by 8 for bytes. Images also carry metadata, extra stored information such as the height, width, colour depth and the date the photo was taken, which the computer needs to rebuild the grid correctly.

2.7 Sound

Sound in the real world is an analogue wave: smoothly changing, infinitely detailed. Computers store it by sampling: measuring the height of the wave at fixed intervals and storing each measurement as a binary number.

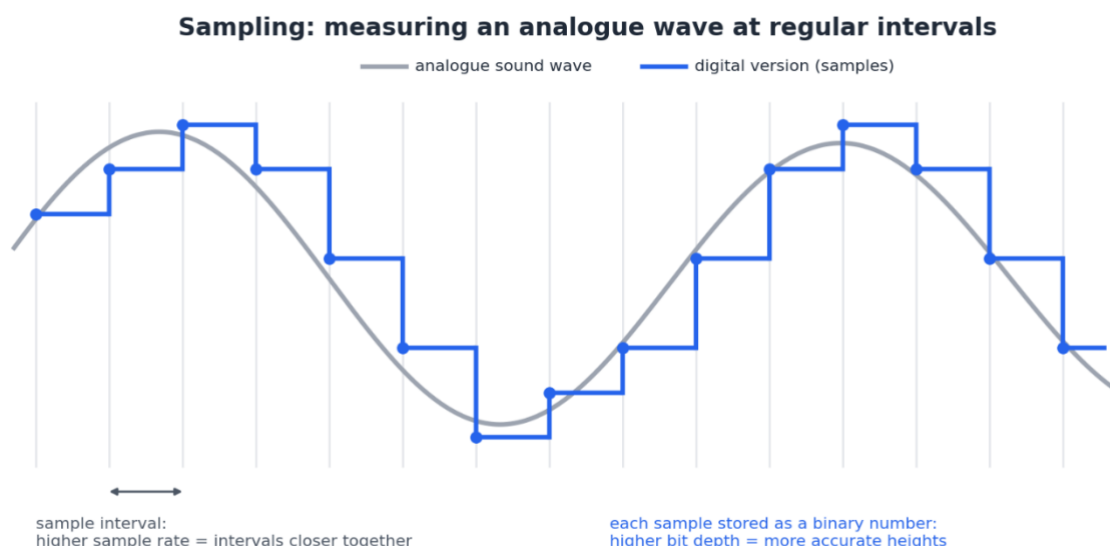


Fig 2.2 – Sampling an analogue wave: the digital version is a series of measurements

- Sample rate is how many measurements are taken per second, in hertz. CD audio uses 44,100 Hz. Higher rate, closer to the original wave, bigger file.

- Bit depth is how many bits store each measurement. More bits mean each sample's height is recorded more precisely. Higher depth, better quality, bigger file.

file size in bits = sample rate × duration (s) × bit depth. Divide by 8 for bytes, then by 1,000 for kilobytes (OCR also accepts 1,024). The pattern of this whole section should be jumping out at you by now: quality and file size are the same dial, and every representation question is about that trade-off. Capacity questions run the same calculation in reverse: work out one file's size, multiply by the number of files, put both figures in the same unit, then compare. Thirty 4 MB photos need 120 MB, which fits easily on a 2 GB memory card (2,000 MB).

2.8 Compression

Compression shrinks files so they take less storage and transfer faster over networks. Lossy compression permanently removes data the human eye or ear will not much miss; the file gets dramatically smaller, and the original can never be perfectly restored. It suits photos, music and video (JPEG, MP3). Lossless compression re-encodes the data more cleverly without discarding anything, so the original can be perfectly rebuilt; the saving is smaller. It is essential where every bit matters: text, spreadsheets, program code. Compress a Python file lossily and you no longer have a program.

Key terms

Term	Meaning
Volatile	Loses its contents when the power is turned off
RAM	Volatile primary storage holding programs and data in use
ROM	Non-volatile memory holding the bootstrap startup program
Virtual memory	Part of secondary storage used as overflow when RAM is full
Overflow	A result too large to fit in the number of bits available
Colour depth	The number of bits used to store each pixel
Sample rate	The number of sound samples taken per second
Metadata	Stored information about a file, such as an image's dimensions
Lossy compression	Reduces file size by permanently removing data
Lossless compression	Reduces file size with no data permanently removed

Try it in Python – Check your conversions

Python has binary and hexadecimal built in, which means it can mark your Section 2 practice for you.

```
print(bin(180))           # 0b10110100
print(int("01001011", 2)) # 75
print(hex(215))           # 0xd7
print(0b00001100 << 2)   # 48: a left shift of two places
```

Your task: Convert five numbers of your own on paper first, then check yourself. The 0b and 0x prefixes are just Python marking the base.

Section 2 exercises

Quick check

- 1. State what is meant by volatile memory. [1]
- 2. How many bits are in a nibble? [1]
- 3. Which type of secondary storage has no moving parts? [1]

Practice

- 4. Convert the binary number 10110100 to denary. Show your working. [1]
 - 5. Convert the denary number 75 to 8-bit binary. [1]
 - 6. Convert 11010111 to hexadecimal. [2]
 - 7. Add the binary numbers 10011010 and 01101100, and explain what has happened to the result. [3]
 - 8. The binary number 00001100 is shifted two places to the left. State the result in binary and explain the arithmetic effect of the shift. [2]
9. A 10-second sound clip is recorded at a sample rate of 8,000 Hz with a bit depth of 8 bits. Calculate the file size in kilobytes. [3]

Exam-style

- 10. A photographer needs portable storage to transfer large image files between studios daily. Recommend a suitable secondary storage type and justify your answer using at least two characteristics. [4]
- 11. A user complains their computer becomes very slow when many programs are open, even though it “still works”. Explain the role virtual memory is playing and state the hardware upgrade that would help. [3]
- 12. Explain why lossless rather than lossy compression must be used when compressing a program's source code. [2]

Section 3 – Computer networks, connections and protocols (spec 1.3)

Before you start – from earlier sections

1. Is RAM volatile or non-volatile?
2. Convert the binary number 00010100 to denary.
3. Give one reason computer scientists use hexadecimal.

A network is two or more computers connected to share data and resources. You use dozens of networks before lunch without noticing, which is rather the point: the machinery in this section is designed to be invisible. The exam makes it visible again.

3.1 LANs, WANs and performance

A local area network (LAN) covers a small geographical area, a home, a school, one office, and the infrastructure is owned by whoever runs it. A wide area network (WAN) connects LANs across large distances using infrastructure that is hired or leased, such as telephone lines and satellite links. The internet is the biggest WAN there is.

Five factors decide how well a network performs. Bandwidth is the amount of data that can pass through the connection per second; too little of it and everything queues. The number of users matters because they share that bandwidth: your school's connection might be fine at 8am and crawling by lunchtime for exactly this reason. The transmission media matter too: fibre optic carries more data, further, than copper cable, and wired connections generally outperform wireless ones. The error rate is how often data arrives damaged or not at all; every lost or corrupted packet has to be sent again, so a weak wireless signal or poor cabling slows everything down. Finally, latency is the delay between data being sent and being received, which is why a video call can lag even on a connection with plenty of bandwidth.

3.2 Client-server and peer-to-peer

Client-server vs peer-to-peer

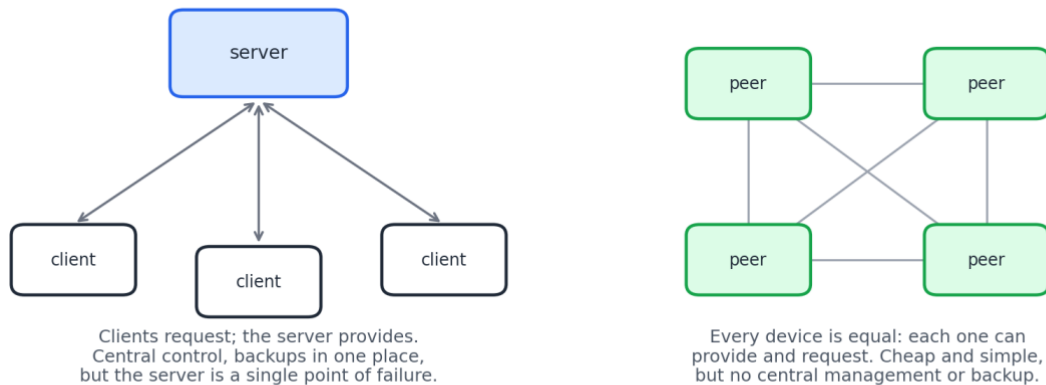


Fig 3.1 – Two ways to organise a network

In a client-server network, a powerful central machine (the server) provides services, files, email, websites, logins, and everyone else's machine (a client) request them. Management, security and backup all happen in one place, which is why organisations use it, but the server is expensive and if it fails, everyone feels it. In a peer-to-peer network every machine is equal, sharing files directly with the others. It is cheap and simple, ideal for a home, but there is no central control: backing up means backing up every machine, and security is only as good as the sloppiest peer.

3.3 Network hardware

- Wireless access point (WAP): allows wireless devices to connect to a network using Wi-Fi. In a typical home router, the WAP is the part that provides the wireless connection.
- Router: joins networks together and directs (routes) packets of data between them, most familiarly between your LAN and the internet.
- Switch: connects devices within a LAN, using each device's MAC address to send data only to its intended recipient rather than broadcasting to everyone.
- Network interface card (NIC): the component inside each device that lets it physically connect to the network at all.
- Transmission media: the cables (or airwaves) themselves, from cheap copper ethernet to fast, interference-proof fibre optic.

3.4 The internet, DNS and the cloud

The internet is a worldwide collection of interconnected networks. When you type a web address, your computer does not know where that site lives; it asks a Domain Name System (DNS) server, which translates the human-friendly name into the IP address of the web server hosting the site. Hosting means storing a website on a server so others can access it. The cloud is the same idea applied to your

own files and software: they are stored and run on someone else's servers, accessible anywhere with internet, with the trade-off that no connection means no files, and you are trusting the provider's security.

3.5 Topologies

Star vs full mesh topology

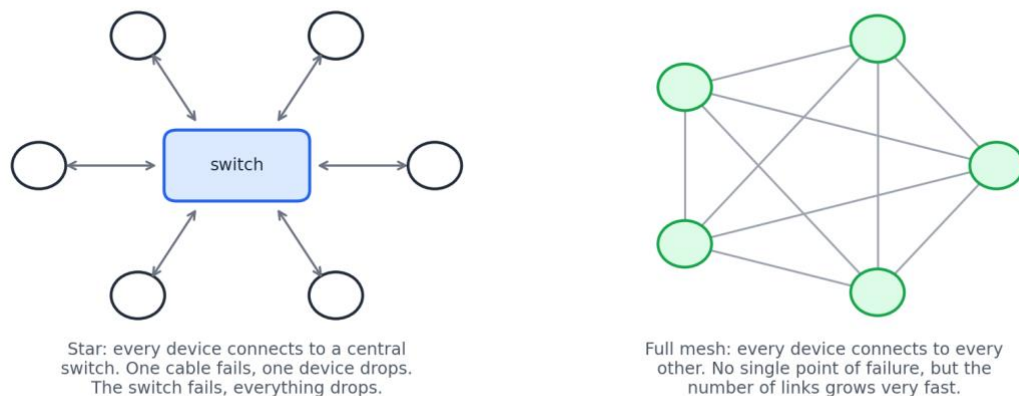


Fig 3.2 – Star and full mesh topologies

A topology is the layout of a network. In a star topology every device connects to a central switch: it performs well and one broken cable affects only one device, but the switch itself is a single point of failure. In a mesh topology devices connect to each other, in a full mesh, every device to every other, so data can route around any failure. The resilience is superb; the cabling cost in a wired full mesh grows absurd as devices are added, which is why full mesh is more practical for wireless networks.

3.6 Wired and wireless

Wired connections (ethernet) are faster, more reliable and more secure, since an attacker needs physical access to a cable. Wireless (Wi-Fi, and Bluetooth for short-range device-to-device links) wins on convenience and mobility, at the price of interference, walls, and signals that anyone nearby can receive. That last point is why encryption matters on wireless networks: the data is scrambled using a key so that intercepting the signal yields gibberish.

3.7 Addresses

An IP address identifies a device on a network and is assigned by the network, so it can change as a device moves between networks. IPv4 addresses are four numbers from 0 to 255 separated by dots (192.168.0.34); the world ran short of them, so IPv6 provides a vastly larger format written in hexadecimal. A MAC address identifies the device's network interface itself: assigned by the manufacturer, unique, and fixed for the life of the device. The switch on your LAN delivers data by MAC address; the wider internet routes it by IP address.

Common misconception. “The IP address is the device's permanent address” has it backwards. The MAC address is the permanent one, burned into the hardware. The IP address is more like a hotel room number: yours while you are there.

3.8 Protocols and layers

A standard is a set of agreed rules for an area of computing, such as how a network cable is wired, how characters are encoded or how a file format is laid out. Standards are why a laptop from one company connects to a router from another, and why any USB stick works in any computer: every manufacturer builds to the same rules, so their hardware and software can work together. Protocols are the standards for communication. A protocol is an agreed set of rules for communication; two devices following the same protocol can talk regardless of who made them. The ones you must know:

Protocol	Purpose
TCP/IP	The core pair: TCP splits data into numbered packets and checks they all arrive; IP routes each packet across the network
HTTP	Requests and transfers web pages
HTTPS	HTTP with encryption, for anything sensitive
FTP	Transfers files between computers
SMTP	Sends email between servers
POP	Retrieves email, downloading it to one device and removing it from the server
IMAP	Retrieves email while leaving it on the server, keeping every device in sync

These protocols are organised into layers, with each layer handling one part of the job and only talking to the layers directly above and below it. The value of layering is that each layer is self-contained: engineers can improve one layer without touching the others, and manufacturers can build products for one layer knowing they will work with everyone else's.

Key terms

Term	Meaning
LAN / WAN	A network over a small area / a network connecting sites over large distances
Bandwidth	The amount of data that can be transferred per second
Client-server	Clients request services from a central server
Latency	The delay between data being sent and being received
Peer-to-peer	All devices are equal and share directly with each other
DNS (Domain Name System)	Translates domain names (web addresses) into IP addresses
Topology	The layout of connections in a network
IP address	A network-assigned address identifying a device on a network
MAC address	A manufacturer-assigned, unchanging address identifying a network interface

Standard	An agreed set of rules that lets hardware and software from different manufacturers work together
Protocol	An agreed set of rules governing communication
Layer	A self-contained division of network communication with its own protocols

Section 3 exercises

Quick check

- 1. State one difference between a LAN and a WAN. [1]
- 2. Which piece of hardware directs packets between different networks? [1]
- 3. What does DNS do? [1]

Practice

- 4. Give two differences between an IP address and a MAC address. [2]
- 5. Explain why a full mesh topology is more commonly wireless than wired. [2]
- 6. A user wants their email available and synchronised on their phone, laptop and tablet. State whether POP or IMAP is suitable and justify your choice. [2]

Exam-style

- 7. A six-person startup is setting up its first office network and cannot afford a server. Recommend peer-to-peer or client-server, giving two justified reasons, and state one problem they should expect as the company grows. [4]
- 8. A school's internet performance collapses every day at 12:30. Explain the likely cause and one realistic solution. [3]
- 9. Explain one advantage of network protocols being organised into layers. [2]

Section 4 – Network security (spec 1.4)

Before you start – from earlier sections

1. State one difference between a LAN and a WAN.
2. Which hardware device routes packets between networks?
3. Give one difference between an IP address and a MAC address.

Every network is under attack, all the time, mostly by automated tools that do not know or care who you are. This section covers the forms those attacks take and the defences that blunt them. It is also, quietly, the section most likely to be useful to you personally for the rest of your life.

4.1 Threats

Malware is any software written to cause harm. Viruses attach themselves to files and spread when the files are shared; worms spread by themselves across networks with no help needed; trojans disguise themselves as legitimate software so you install them willingly; spyware secretly records what you do, including keystrokes; ransomware encrypts your files and demands payment for the key.

Social engineering skips the technology and attacks the people, on the reliable theory that humans are the weakest point of any system. Phishing, its most common form, sends messages that impersonate trusted organisations to trick users into revealing passwords or bank details. No firewall stops an employee who has been persuaded to open the door.

Brute-force attacks try every possible password, automatically, at enormous speed. Short and common passwords fall in seconds; long ones take centuries, which is the entire argument for long passwords.

Denial of service (DoS) floods a server with so many requests that genuine users cannot get through. Nothing is stolen; the harm is that the service stops working.

Data interception uses packet sniffing tools to eavesdrop on data as it travels across a network, which is especially easy on unencrypted public Wi-Fi.

SQL injection attacks a website through its own input boxes. If a site passes what users type directly into its database queries, an attacker can type fragments of SQL instead of a name, and the database will obediently execute them, potentially revealing or destroying its contents. You will understand exactly how this works after Section 8.

4.2 Defences

- Penetration testing: paying trusted experts to attack your own system and report the weaknesses they find, before hostile ones find them first.
- Anti-malware software: scans files against a database of known malware and quarantines what it finds. Only as good as its last update.

- Firewalls: monitor traffic entering and leaving the network, blocking anything that breaks the rules it has been given. The border checkpoint of the network.
- User access levels: give each user only the access their role needs. A student should not be able to open the staff folder; then a hacked student account cannot either. This limits the damage any single compromised account can do.
- Passwords: the longer, the better, and never reused. Best combined with other checks, such as a code sent to your phone.
- Encryption: scrambles data using a key so intercepted data is unreadable. It does not stop interception; it makes interception worthless.
- Physical security: locks, keycards and CCTV. No amount of software protects a server that someone can pick up and carry away.

Common misconception. Encryption does not prevent data being stolen, and anti-malware does not make attacks impossible. Almost every defence limits or contains harm rather than eliminating it, and answers that claim a defence makes a system “safe” overclaim. Examiners reward “reduces”, “limits” and “makes it harder”.

Exam tip. Threat-and-defence questions usually come as a scenario. Pair them precisely: phishing pairs with staff training, brute force with strong passwords, interception with encryption. A technically true but mismatched defence (“use a firewall” against phishing) earns nothing.

Key terms

Term	Meaning
Malware	Software written to cause harm
Ransomware	Malware that encrypts files and demands payment
Phishing	Fraudulent messages impersonating trusted sources to steal information
Brute-force attack	Trying every possible password until one works
Denial of service	Overloading a server so genuine users cannot access it
SQL injection	Entering database commands into a website's input boxes
Penetration testing	Authorised testing of a system to find its weaknesses
Firewall	Monitors and filters network traffic according to rules
User access levels	Restricting each user's access to what their role requires
Encryption	Scrambling data with a key so only key-holders can read it

Section 4 exercises

Quick check

- 1. State the difference between a virus and a worm. [1]
- 2. Which attack makes a service unavailable without stealing anything? [1]
- 3. What is the purpose of penetration testing? [1]

Practice

- 4. Explain why user access levels limit the damage caused by one compromised account. [2]
- 5. A cafe offers free unencrypted Wi-Fi. Describe the threat this poses to customers and the defence that addresses it. [3]

Exam-style

- 6. Employees at a company have received emails, apparently from their bank, asking them to confirm their login details. Identify the attack, explain why technical defences alone cannot fully prevent it, and recommend the most effective response. [4]
- 7. A hospital's patient records were encrypted by attackers demanding payment. Identify the type of malware and give two measures that would have reduced the impact of this attack. [3]

Section 5 – Systems software (spec 1.5)

Before you start – from earlier sections

1. Name two types of malware.
2. What does a firewall do?
3. What does encryption do to intercepted data?

Systems software is the software that runs the computer itself, as opposed to applications, which are the software you actually wanted the computer for. The two kinds you need are the operating system and utility software, and the theme of both is the same: enormous complexity hidden behind the scenes so that everything else can be simple.

5.1 The operating system

The operating system (OS) manages the hardware and provides the platform every other program runs on. OCR names five of its jobs, and “describe two functions of an operating system” is about as dependable as exam questions get.

- User interface: the layer you interact with, whether a graphical interface of windows and icons or a text-based command line. It exists so you can use the machine without knowing anything about the hardware underneath.
- Memory management and multitasking: the OS decides where in RAM each running program lives, keeps programs out of each other's memory, and switches the CPU's attention between programs so rapidly that they all appear to run simultaneously. They do not; it is very fast turn-taking.
- Peripheral management and drivers: every printer, mouse and webcam speaks its own dialect. A driver is the piece of software that translates between the OS and one specific device, which is why new hardware sometimes needs a driver installed before it works.
- User management: accounts, passwords, and controlling which users may access which files, the mechanism behind the access levels from Section 4.
- File management: organising data into files and folders, and handling naming, moving, copying, deleting and permissions.

Common misconception. Multitasking does not mean the CPU runs several programs at the same instant. A single core runs one instruction at a time; the OS just rotates between programs thousands of times a second. Genuine simultaneity needs multiple cores.

5.2 Utility software

Utilities are small tools that maintain the system. You need three.

- Encryption software scrambles the data on a disk so that a stolen laptop yields nothing readable without the key.
- Defragmentation reorganises the fragmented parts of files so that they are stored together and consolidates free space. On a magnetic hard disk drive (HDD), this can improve access speed

- because the read/write head has less physical movement to make. An important exam point: SSDs do not need defragmenting because they have no moving read/write head, so there is no seek time to reduce.
- Data compression utilities shrink files, using the lossy and lossless techniques from Section 2, to save storage space and speed up transfers.

Key terms

Term	Meaning
Operating system	Software managing the hardware and running other programs
User interface	The means by which the user interacts with the computer
Multitasking	The OS switching the CPU rapidly between programs
Driver	Software that lets the OS communicate with one specific device
Utility software	Small tools that maintain and optimise the system
Defragmentation	Reordering a hard disk so each file's parts are stored together

Section 5 exercises

Quick check

- 1. Name the software that translates between the OS and a specific peripheral. [1]
- 2. State two functions of an operating system. [2]

Practice

- 3. Explain how an operating system allows a single-core CPU to appear to run many programs at once. [2]
- 4. Describe what fragmentation is and how defragmentation software improves performance. [3]

Exam-style

- 5. A technician suggests running defragmentation software on a new laptop that has an SSD. Explain why this will not improve performance. [2]
- 6. A company issues laptops containing sensitive client data to travelling staff. Identify the most appropriate utility software and explain how it protects the data if a laptop is stolen. [3]

Section 6 – Ethical, legal, cultural and environmental impacts (spec 1.6)

Before you start – from earlier sections

1. State two functions of an operating system.
2. Which utility reorders a fragmented hard disk?
3. Why does an SSD not benefit from it?

This section has no algorithms and no calculations, and students routinely under-revise it, which is a gift to you: the questions are predictable, the marks are generous, and the skill being tested is making a balanced argument about how technology affects people. You already argue about technology daily. Here you get marks for it.

6.1 The impacts

Ethical issues are about right and wrong. Should an algorithm decide who gets a loan or a job interview? Is it acceptable that the companies holding your data profit from it? Automation raises the sharpest one: technology destroys some jobs while creating others, and the people who lose the old jobs are rarely the ones who get the new ones.

Legal issues are about what the law permits, covered by the legislation in 6.2 below: hacking, data theft, and using others' work without permission.

Cultural issues are about how technology changes the way we live. The digital divide is the exam favourite: people with good devices, fast internet and the skills to use them pull further ahead of people without, in education, in work, in access to services. When a government service moves online, it becomes easier for most and impossible for some.

Environmental issues cut both ways. Manufacturing devices consumes rare metals and energy, data centres draw astonishing amounts of power, and discarded electronics become e-waste, often exported to countries least able to process it safely. Yet technology also enables video calls that replace flights and smart systems that cut energy use. Balanced answers acknowledge both directions.

Privacy issues concern how much of your life is recorded: location history, browsing habits, purchases, faces on CCTV. The tension examiners want you to explore is between convenience and surveillance, since the same data that makes services helpful makes their holders powerful.

6.2 The legislation

Law	What it does
The Data Protection Act 2018	Governs personal data. Organisations must keep data secure, accurate, up to date, and use it only for the stated purpose; individuals gain rights over data held about them

Computer Misuse Act 1990	Makes it an offence to access a computer system without authorisation, to do so intending further crime, or to modify data without authorisation (including spreading malware)
Copyright, Designs and Patents Act 1988	Protects creative work, including software, music, images and text, from being copied or distributed without the holder's permission

Exam tip. Learn which law fits which scenario, because that pairing is the question. Guessing a friend's password and reading their messages: Computer Misuse Act. A company leaking customer records through carelessness: Data Protection Act. Sharing a cracked game: Copyright, Designs and Patents Act.

6.3 Open source and proprietary software

Software licences come in two broad kinds. Open-source software makes its source code available: anyone may inspect it, modify it and share it, usually free of charge, with a worldwide community fixing bugs and adding features. Proprietary software keeps its source code secret: you buy a licence to use it, may not modify it, and rely on the company for support and updates, in exchange for which you typically get polish, documentation and someone to phone when it breaks. Neither is simply better; the exam wants you to choose sensibly for a scenario. A school with no budget and a capable technician might thrive on open source; a business needing guaranteed support might happily pay for proprietary.

Key terms

Term	Meaning
Digital divide	The gap between those with and without access to technology
E-waste	Discarded electronic devices and their environmental harm
Data Protection Act 2018	UK law governing the handling of personal data
Computer Misuse Act 1990	UK law criminalising unauthorised access and modification
Copyright, Designs and Patents Act 1988	UK law protecting creative works including software
Open source	Software whose source code may be viewed, modified and shared
Proprietary	Software whose source code is private and licensed for use only

Section 6 exercises

Quick check

- 1. Which law would be broken by accessing a classmate's account without permission? [1]
- 2. State what is meant by the digital divide. [1]
- 3. Give one difference between open source and proprietary software. [1]

Practice

- 4. Give one positive and one negative environmental impact of computer technology. [2]
- 5. A company collects customers' addresses for deliveries, then sells the addresses to advertisers. Identify the law it breaches and explain why. [2]

Exam-style

- 6. A local council plans to move all its services online and close its walk-in centre. Discuss the impacts of this decision, considering at least two different stakeholder groups. [6]
- 7. A new charity needs office software but has almost no budget and no IT staff. Discuss whether open source or proprietary software is more appropriate. [4]

Section 7 – Algorithms (spec 2.1)

Before you start – from earlier sections

1. Which law criminalises unauthorised access to a computer system?
2. What is the digital divide?
3. What does it mean for software to be open source?

Before a single line of code gets written, somebody must work out what the program should do. That thinking stage is what this section is about. By the end of it you should be able to break a problem down with a structure diagram, plan a solution in flowcharts and pseudocode, follow somebody else's algorithm with a trace table, and explain how the standard searching and sorting algorithms work.

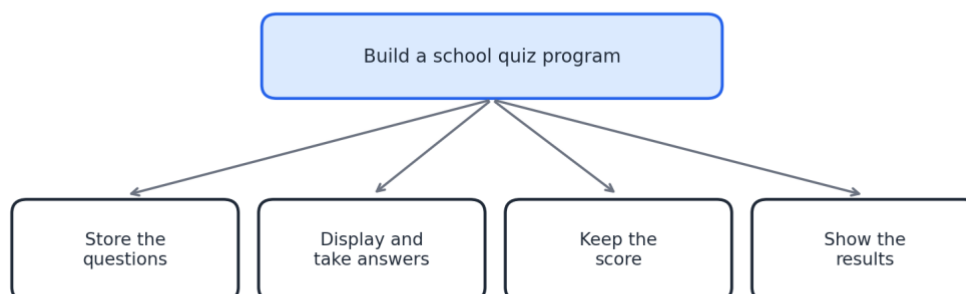
7.1 Computational thinking

Computational thinking is not “thinking like a computer”. Computers do not think. It means thinking about problems in a way that makes them solvable by a computer, and OCR names three ingredients you must know.

Abstraction means removing detail that does not matter so you can focus on what does. The London Underground map is the classic example: it throws away real distances, street layouts and geography, keeping only stations and the connections between them, because that is all a passenger needs. A weather app does the same thing when it shows you a cloud icon instead of ten thousand sensor readings.

Decomposition means breaking one intimidating problem into smaller problems that can each be solved, coded and tested separately. Once you see the parts, the whole thing stops being scary. Decomposition is drawn as a structure diagram: the whole problem sits in a box at the top and its sub-problems branch out underneath, like Fig 7.1. Any box that is still too big can be broken down again on the next level. OCR expects you to read and draw structure diagrams alongside flowcharts and pseudocode.

Decomposing one big problem into smaller ones



Each sub-problem is simple enough to plan, code and test on its own

Fig 7.1 – Decomposing a quiz program into four manageable sub-problems

Algorithmic thinking means working out the precise sequence of steps, and the order they must happen in, that will get you from the inputs to the outputs. If you have ever written down a method for a friend to follow (“first do this, then if that happens do this instead”), you have done it already.

Exam tip. A common one-mark question asks you to define one of these three terms. Learn the definitions word for word and be ready to spot which one is being used in a scenario. “The programmer ignores the colour of the cards because only their values matter” is abstraction, not decomposition.

7.2 Designing, creating and refining algorithms

Inputs, processes and outputs

Every algorithm can be described by what goes in, what happens to it, and what comes out. Get into the habit of identifying all three before you design anything. For a program that works out a grade: the input is a mark, the process is comparing that mark against grade boundaries, and the output is the grade. It sounds obvious, and it is, but exam questions regularly ask you to state the inputs and outputs of a scenario and students throw away marks by mixing them up.

Flowcharts

A flowchart shows an algorithm as a diagram. OCR expects you to know and use six symbols: the line, an arrow showing the direction of flow from one step to the next, plus the five shapes in Fig 7.2 (terminal, process, input/output, decision and sub program).

Flowchart symbols you are expected to use

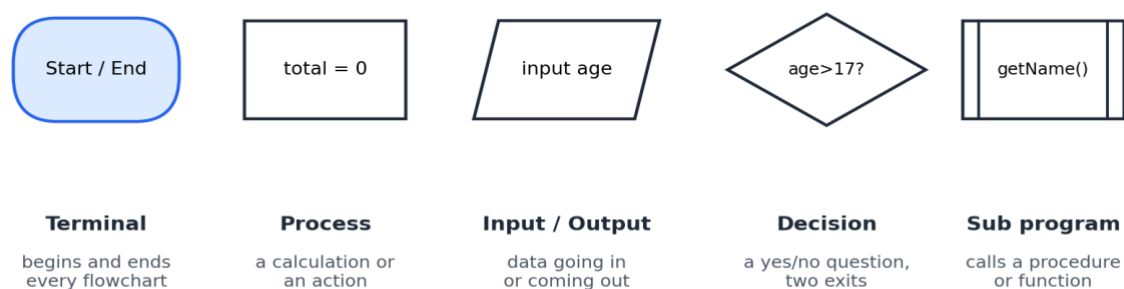


Fig 7.2 – The five flowchart symbols used in J277

Two rules will keep your flowcharts correct: every decision diamond must have exactly two labelled exits (usually Yes and No), and every route through the chart must eventually reach an end terminal. A loop is drawn simply by pointing an arrow back to an earlier step.

The OCR Exam Reference Language

Pseudocode is structured English for describing algorithms. Any sensible pseudocode gets credit when you are writing your own answers, but OCR sets its exam questions in a specific dialect called the Exam Reference Language (ERL), and you must be able to read it fluently. Everything in this book is written in ERL for that reason. Here is a single program that shows most of what you need.

```
// A guessing game: ERL tour in one program
const maxGuesses = 5
secret = random(1, 100)
guesses = 0
won = false

while guesses < maxGuesses AND won == false
    answer = input("Enter your guess: ")
    guess = int(answer)           // cast the string to a number
    guesses = guesses + 1
    if guess == secret then
        won = true
    elseif guess < secret then
        print("Too low")
    else
        print("Too high")
    endif
endwhile

if won == true then
    print("You got it in " + str(guesses) + " guesses")
else
    print("Out of guesses. It was " + str(secret))
endif
```

Notice the habits of the language: no semicolons, blocks closed with `endif` and `endwhile`, comparison written as `==`, comments starting with `//`, and casting done with `int()`, `str()`, `real()` and `bool()`. Count-controlled loops use `for i = 0 to 9 ... next i`, and condition-controlled loops use `while ... endwhile` or `do ... until`.

Exam tip. When a question says “you may answer in the OCR Exam Reference Language or any high-level language”, that is genuine. Python is fine. What loses marks is vagueness: “check if the number is right” is not an algorithm, `if guess == secret then` is.

Trace tables

A trace table lets you follow an algorithm by hand, recording the value of every variable at every step. It is how you find logic errors, and it is a guaranteed exam skill. Trace this short program:

```
total = 0
for i = 1 to 4
    total = total + i
next i
print(total)
```

i	total	OUTPUT
	0	
1	1	
2	3	
3	6	
4	10	
		10

Work one line at a time and only write a value in a column when it changes. The discipline matters: the most common trace-table mistake is doing the whole thing in your head and filling in what you assume the program does, rather than what it does. The entire point of the table is to catch the difference.

Syntax errors and logic errors

- A syntax error breaks the rules of the language, like a missing bracket or writing pint instead of print. The program will not run at all, and the translator will usually point at the offending line.
- A logic error is worse in a way: the program runs happily and produces the wrong answer. Writing `>` where you meant `>=` is a classic. Trace tables and testing are how you hunt these down.

7.3 Searching algorithms

You need two searching algorithms, and the interesting part is not how each one works but why you would pick one over the other.

Linear search

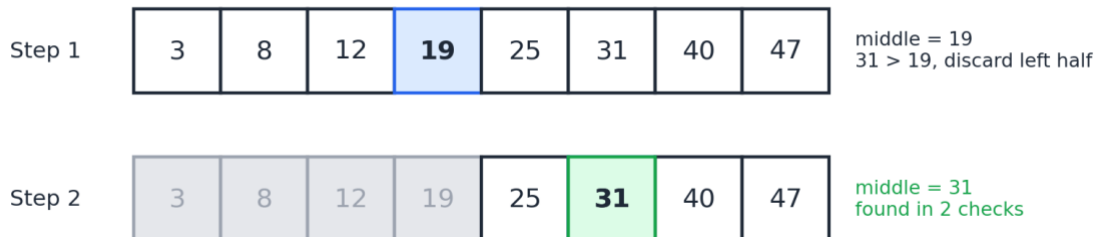
Start at the first item and check every item in turn until you find the target or run out of list. That is the whole algorithm. Its one great virtue is that the list does not need to be in any particular order.

```
// items holds 10 values; target is what we are looking for
found = false
i = 0
while found == false AND i < 10
    if items[i] == target then
        found = true
        print("Found at position " + str(i))
    else
        i = i + 1
    endif
endwhile
if found == false then
    print("Not in the list")
endif
```

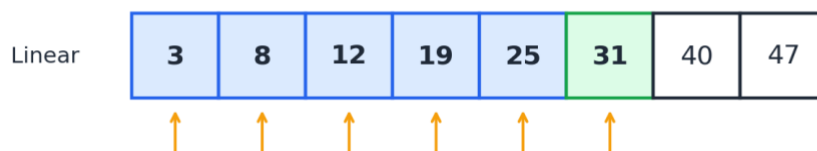
Binary search

Binary search only works on a sorted list, and that condition is the single most important fact about it. Check the middle item. If it is the target, done. If the target is bigger, the whole left half of the list is now irrelevant and gets discarded; if smaller, discard the right half. Repeat on whatever remains. Every comparison halves the problem, which is why it is dramatically faster on large lists.

Searching for 31 in a sorted list of 8 items



Binary search: 2 comparisons



Linear search: 6 comparisons, checking one item at a time

Fig 7.3 – Binary search finds 31 in two comparisons; linear search takes six

```
first = 0
last = 9
found = false
while found == false AND first <= last
    mid = (first + last) DIV 2
    if items[mid] == target then
        found = true
    elseif items[mid] < target then
        first = mid + 1
    else
        last = mid - 1
    endif
endwhile
```

The scale of the win is worth feeling. A million-item sorted list needs at most 20 comparisons with binary search. Linear search could need a million. But if the list is unsorted, binary search simply cannot be used, and sorting a list just to search it once is usually not worth the cost. That trade-off is exactly the kind of judgement OCR likes to ask about.

7.4 Sorting algorithms

Three sorting algorithms are on the spec. For each one you should be able to describe the method in words, show the state of a list after each pass, and compare their strengths. You are not expected to write full code for them from memory, but you are expected to complete or correct code you are given.

Bubble sort

Move through the list comparing each pair of neighbours, swapping them if they are out of order. One full sweep is called a pass, and each pass guarantees the largest remaining value “bubbles” to its final position at the end. Keep making passes until a pass happens with no swaps, which is the signal that the list is sorted.

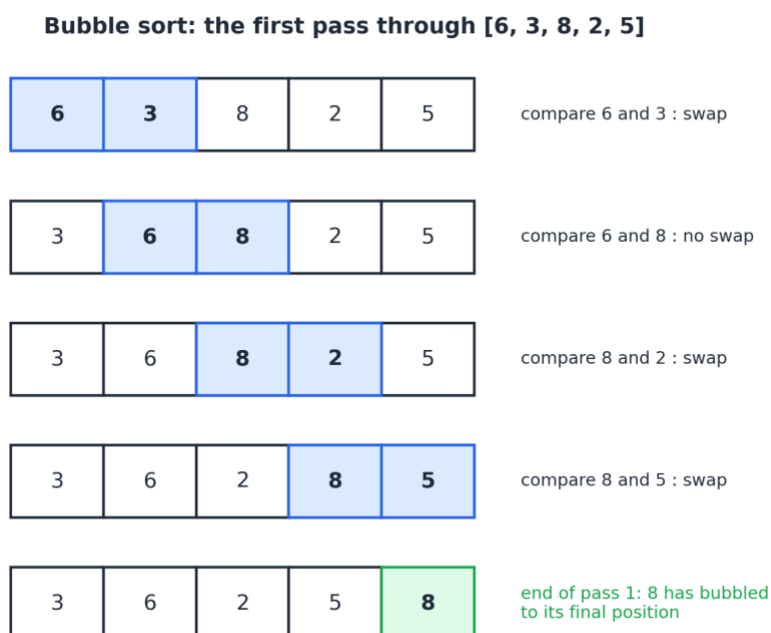


Fig 7.4 – One complete pass of bubble sort on [6, 3, 8, 2, 5]

```
// items holds 10 values
swapped = true
while swapped == true
  swapped = false
  for j = 0 to 8
    if items[j] > items[j + 1] then
      temp = items[j]
      items[j] = items[j + 1]
      items[j + 1] = temp
      swapped = true
    endif
  next j
endwhile
```

Insertion sort

Insertion sort is how most people sort a hand of playing cards without realising it. Treat the first item as a sorted section of one. Take the next unsorted item and slide it left into its correct place among the sorted items. Repeat until the sorted section is the whole list.

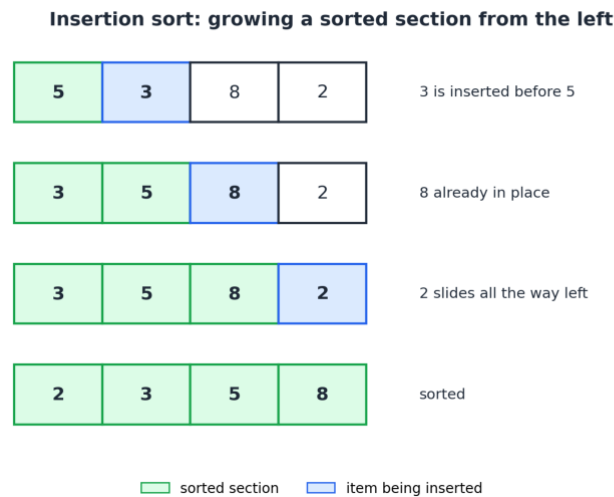


Fig 7.5 – Insertion sort growing a sorted section from the left

Merge sort

Merge sort takes a completely different approach, called divide and conquer. Keep splitting the list in half until every list contains a single item. A one-item list is sorted by definition, which sounds like a lawyer's trick but is the key to the whole thing. Then merge the lists back together in pairs, and because each pair being merged is already sorted, merging them in order is easy: repeatedly take whichever front item is smaller.

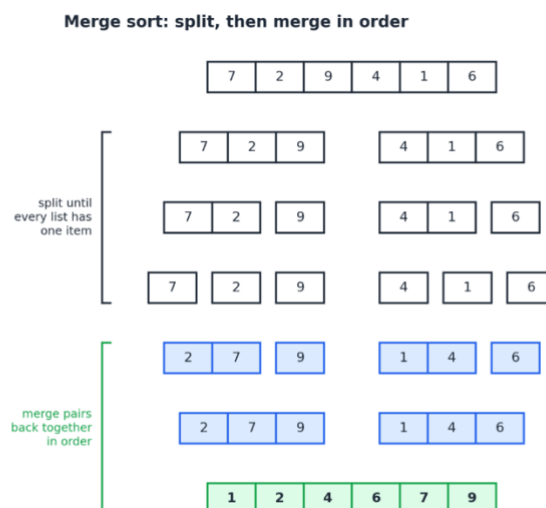


Fig 7.6 – Merge sort splitting [7, 2, 9, 4, 1, 6] and merging back in order

Choosing between them

Algorithm	Strengths	Weaknesses
Bubble sort	Very simple to understand and code; can detect an already-sorted list in one pass	Slowest of the three on most lists; lots of comparisons and swaps
Insertion sort	Simple; fast on lists that are small or nearly sorted; sorts as data arrives	Slows badly as unsorted lists grow
Merge sort	Consistently fast even on large lists; performance does not depend on the starting order	Harder to understand; needs extra memory for the merging stage

Exam tip. “Show the list after the first pass of a bubble sort” is a bread-and-butter question. Do it slowly on paper, one comparison at a time. Most lost marks here are rushing errors, not understanding errors.

Key terms

Term	Meaning
Abstraction	Removing detail that is not needed so you can focus on what matters
Decomposition	Breaking a problem into smaller sub-problems that can be solved separately
Structure diagram	A diagram showing a problem broken down into sub-problems, with the whole problem at the top
Algorithm	A precise sequence of steps that solves a problem
Trace table	A table recording the value of each variable as an algorithm is followed step by step
Linear search	Checking each item in turn from the start until the target is found
Binary search	Repeatedly checking the middle item of a sorted list and discarding the half that cannot contain the target
Pass	One complete sweep through a list during a sort
Syntax error	Code that breaks the rules of the language, so the program cannot run
Logic error	Code that runs but produces the wrong result

Try it in Python – Searches and sorts for real

The ERL in this section is for exams; Python is where it runs. Here is 7.3's linear search, translated almost word for word.

```
items = [3, 8, 12, 19, 25, 31, 40, 47]
target = 31
found = False
i = 0
while not found and i < len(items):
    if items[i] == target:
        found = True
```

```

        print("Found at position", i)
    else:
        i += 1
if not found:
    print("Not in the list")

```

Your task: Now implement binary search from 7.3 (first, last, mid, the while loop) and test it with a value in the list and one not in it. Then write bubble sort from 7.4 and check its output against Python's built-in `sorted()`.

Section 7 exercises

Quick check

- 1. Define abstraction. [1]
- 2. Name the flowchart symbol that has two exits. [1]
- 3. State the condition a list must meet before a binary search can be used on it. [1]
- 4. How does a bubble sort know the list is fully sorted? [1]

Practice

- 5. A programmer designing a card game ignores the pictures on the cards because only their values matter. State the computational thinking technique being used. [1]
- 6. Complete a trace table for the following algorithm, showing every value of `x` and the final output. [3]

```

x = 10
while x > 3
x = x - 4
endwhile
print(x)

```

- 7. The list [8, 5, 9, 1] is sorted into ascending order using a bubble sort. Show the list after each of the first two passes. [2]
- 8. The list [7, 2, 9, 4] is sorted using an insertion sort. Show the list after each unsorted item is inserted. [3]
- 9. The list [4, 1, 3, 2] is sorted using a merge sort. Show the lists after the splitting stage is complete, and after each merging stage. [3]
- 10. Write an algorithm in the OCR Exam Reference Language that asks the user for 10 numbers and outputs the largest one. [4]

Exam-style

- 11. A phone contacts app stores names in alphabetical order. Identify which searching algorithm the app should use and justify your choice. [3]

- 12. A company needs to sort a file containing two million customer records. Discuss whether a bubble sort or a merge sort is more suitable. [4]
- 13. The following program should add up the numbers 1 to 5 but contains a logic error. Identify the error and write the corrected line. [2]

```
total = 0
for i = 1 to 5
    total = i
next i
print(total)
```

Section 8 – Programming fundamentals (spec 2.2)

Before you start – from earlier sections

1. State the precondition of a binary search.
2. How does a bubble sort know it has finished?
3. What is the difference between a syntax error and a logic error?

Section 7 taught you to design algorithms. This section gives you the full toolkit for building them: the constructs, data types and techniques that every program, from a two-line script to a game engine, is assembled from. Everything is shown in the OCR Exam Reference Language, and everything here transfers directly to whichever language you program in for real.

8.1 Variables, constants and assignment

A variable is a named location in memory whose value can change while the program runs; a constant is one whose value is fixed when the program is written and cannot change while it runs. Using a constant for a value like a VAT rate does two jobs: it stops the value being changed accidentally, and it means a future change is made in exactly one place.

```
const vat = 0.2
price = real(input("Enter the price: "))
total = price * (1 + vat)
```

Every value has a data type, and you must know five: integer (whole numbers), real (numbers with decimal parts), Boolean (true or false), character (a single symbol) and string (text). Casting converts between them with `int()`, `real()`, `str()` and `bool()`, and you will need it constantly, because `input()` always hands you a string, even when the user typed a number. OCR's reference language uses `real()` to convert to a real number; `float()`, the Python name, is also accepted.

8.2 Operators

Kind	Operators	Notes
Arithmetic	+ - * / MOD DIV ^	MOD gives the remainder, DIV the whole-number quotient: 17 MOD 5 is 2, 17 DIV 5 is 3. ^ is exponentiation
Comparison	== != < <= > >=	== compares; = assigns. Mixing them up is the classic beginner error
Boolean	AND OR NOT	Combine conditions: if age >= 13 AND age <= 19

Exam tip. MOD is more useful than it looks. A number is even exactly when `number MOD 2 == 0`, and “every third row” in a program is `i MOD 3 == 0`. When a question involves remainders, cycles or divisibility, reach for MOD.

8.3 The three constructs

Every program ever written is built from just three patterns. Sequence: instructions executed in order, one after another. Selection: choosing between paths with `if/elseif/else` or `switch/case`. Iteration: repeating instructions with a loop, either count-controlled (`for`, when you know how many times) or condition-controlled, when a condition decides: a `while` loop repeats while its condition is true, and a `do until` loop repeats until its condition becomes true.

```
// selection
if mark >= 70 then
    print("Distinction")
elseif mark >= 40 then
    print("Pass")
else
    print("Fail")
endif

// count-controlled iteration
for i = 1 to 12
    print(str(i) + " x 7 = " + str(i * 7))
next i

// condition-controlled iteration
password = input("Enter password: ")
while password != "opensesame"
    password = input("Wrong. Try again: ")
endwhile

// selection with switch: one variable checked against several values
switch day:
    case "Sat":
        print("Weekend")
    case "Sun":
        print("Weekend")
```

```

    default:
        print("Weekday")
endswitch

// count-controlled iteration with step: prints 0, 5, 10 ... 50
for i = 0 to 50 step 5
    print(i)
next i

// condition-controlled, checked at the end: always runs at least once
do
    password = input("Enter password: ")
until password == "opensesame"

```

The difference between while and do until is when the condition is checked: while checks before each pass, so its body may run zero times; do until checks after, so its body always runs at least once. Constructs also nest, a loop inside a loop, an if inside a loop, and reading nested code confidently is a skill the exam tests directly. Follow the indentation: everything indented under a line belongs to it.

8.4 Strings

Strings can be joined (concatenated) with +, and come with built-in operations. The positions in a string are numbered from 0, a fact that will trip you up exactly once per week until it doesn't.

```

word = "Computer"
print(word.length)           // 8
print(word.upper)            // COMPUTER
print(word.lower)            // computer
print(word.substring(3, 4))  // "pute" (4 characters, starting at position 3)
print(word.left(4))          // "Comp"
print(word.right(3))         // "ter"
print(ASC("A"))               // 65: the character's code
print(CHR(66))               // "B": the code's character

```

8.5 Arrays

An array stores many values of the same type under one name, with each value reached by its index, counting from 0. A one-dimensional array is a list; a two-dimensional array is a grid, indexed by two numbers, and perfect for anything naturally arranged in rows and columns, like a chessboard or a seating plan.

```
array scores[5]
scores[0] = 82
scores[4] = 67           // the LAST index of a 5-element array

array board[3, 3]
board[1, 2] = "X"       // row 1, column 2

// the classic pattern: loop over an array to total it
total = 0
for i = 0 to 4
    total = total + scores[i]
next i
print("Average: " + str(total / 5))
```

Common misconception. An array declared with size 5 has indexes 0 to 4. Writing to index 5 is the single most common array error in exam answers, and examiners are specifically watching for it.

8.6 File handling

Programs use files so that data survives after the program ends. The lifecycle is always: open, use, close, and forgetting the close is a marks-losing habit.

```
// writing
newFile("scores.txt")
f = open("scores.txt")
f.writeLine("82")
f.close()

// reading a whole file, however long it is
f = open("scores.txt")
while NOT f.endOfFile()
    print(f.readLine())
endwhile
f.close()
```

8.7 Records and SQL

A database stores data in tables, where each row is a record (one entity, such as one student) and each column is a field (one attribute, such as their name). You interrogate a table with SQL, and at GCSE you need exactly one statement shape:

```
SELECT name, formGroup
FROM students
WHERE house = "Attenborough"
```

```
SELECT *
FROM students
WHERE surname LIKE "P%"
```

In the second query, `*` means every field, and `%` is a wildcard standing for any run of characters, so `"P%"` matches every surname starting with P. The specification only requires `SELECT`, `FROM` and `WHERE`; `LIKE` and the wildcards are widely taught and worth knowing but treat them as an extension.

Read it as a sentence: select these fields, from this table, where this condition holds. And now Section 4's SQL injection makes sense: if a website glues what a user typed straight into a statement like this, a user who types SQL gets their SQL run.

8.8 Sub programs

A sub program is a named, reusable block of code. A procedure does a job; a function does a job and returns a value to whoever called it. Sub programs take parameters, the values passed in for them to work with. They matter because they let you write a piece of logic once and use it everywhere, they make programs readable, and they let you test each part on its own, which is the payoff of the decomposition you did back in Section 7.

```
function celsiusToF(c)
    return c * 9 / 5 + 32
endfunction

procedure greet(name)
    print("Hello, " + name + "!")
endprocedure

greet("Ada")
print(celsiusToF(100))    // 212
```

Every variable has a scope, the part of the program where it can be used. A variable created inside a sub program is local: it exists only while that sub program runs and cannot be seen or changed from outside, so two sub programs can each have their own total without interfering. A variable declared in the main program with the keyword `global` can be used anywhere, including inside sub programs; constants follow the same rule, local or global. Use local variables wherever you can, because a value that every part of the program can change is very hard to track down when it goes wrong. Arrays can be passed into a sub program as parameters and returned from functions, just like single values.

```
global passMark = 50                // global: usable anywhere

function addBonus(marks, bonus)      // marks is an array passed in
```

```

    array newMarks[5]                // newMarks and i are local
    for i = 0 to 4
        newMarks[i] = marks[i] + bonus
    next i

    return newMarks                  // a whole array is returned
endfunction

array scores = [62, 45, 48, 90, 81]
scores = addBonus(scores, 5)
if scores[1] >= passMark then
    print("Now a pass")              // 45 + 5 = 50, so this prints
endif

```

8.9 Random numbers

`random(1, 6)` produces a random integer from 1 to 6 inclusive: a die. Random numbers power games, simulations and tests, and they combine naturally with everything above; the guessing game in Section 7.2 was one loop, one selection and one random number.

Key terms

Term	Meaning
Variable / constant	A named value that can change while running / one fixed for the whole run
Casting	Converting a value from one data type to another
Selection	Choosing between paths using a condition
Iteration	Repeating code with a count- or condition-controlled loop
Concatenation	Joining strings together with +
Array	Many values of one type stored under one name, accessed by index
Record / field	One row of a database table / one column of it
Procedure / function	A named reusable block / one that returns a value
Local / global variable	One usable only inside its sub program / one usable anywhere in the program
Parameter	A value passed into a sub program

Try it in Python – The guessing game, translated

The game from Section 7.2, in real Python. Notice how directly the ERL maps across: endwhile and endif simply became indentation.

```
import random
secret = random.randint(1, 100)
guesses = 0
won = False
while guesses < 5 and not won:
    guess = int(input("Enter your guess: "))
    guesses += 1
    if guess == secret:
        won = True
    elif guess < secret:
        print("Too low")
    else:
        print("Too high")
print("You got it!" if won else "It was " + str(secret))
```

Your task: Add a play-again loop around the whole game, and keep a running count of wins across rounds.

Section 8 exercises

Quick check

- 1. State the difference between a variable and a constant. [1]
- 2. Evaluate 23 MOD 4 and 23 DIV 4. [2]
- 3. What is the last valid index of the array declared by array names[8]? [1]

Practice

- 4. State the output of: word = "Keyboard" then print(word.substring(3, 5)). [1]
- 5. Explain why int() is needed in the line age = int(input("Age: ")). [2]
- 6. Write an SQL statement returning the name and price of every product in a table called products costing more than 10. [3]
- 7. Explain the difference between a while loop and a do until loop. [2]

Exam-style

- 8. Write a function isEven(num) that returns true if num is even and false otherwise. [3]
- 9. An array temps[7] holds one temperature per day of the week. Write an algorithm that outputs the highest temperature. [4]
- 10. Explain two advantages of using sub programs when writing a large program. [4]

Section 9 – Producing robust programs (spec 2.3)

Before you start – from earlier sections

1. What do MOD and DIV each give?
2. What is the last valid index of the array declared by array x[10]?
3. State the difference between a function and a procedure.

A robust program is one that keeps working when the world misbehaves: when users type their name in the age box, when a file is missing, when someone actively tries to break it. Beginners write programs for users who do everything right. Professionals assume the opposite, and this section is about making that shift.

9.1 Defensive design

Defensive design means anticipating what could go wrong and writing the program so it copes. Its first tool is input validation: checking that input is sensible before using it. You need five checks by name.

Check	Ensures	Example
Range check	The value falls between limits	Exam mark between 0 and 100
Type check	The value is the right data type	Age is an integer, not "twelve"
Length check	The input has an acceptable number of characters	Password at least 8 characters
Presence check	Something was entered at all	Email field not left blank
Format check	The input matches a required pattern	Postcode shaped like a postcode

```
// a range check in action: refuses to move on until the input is valid
mark = int(input("Enter mark: "))
while mark < 0 OR mark > 100
    mark = int(input("Must be 0 to 100. Try again: "))
endwhile
```

Beyond validation, defensive design includes anticipating misuse, such as sanitising input so that typed SQL cannot reach your database (Section 4's injection attack, closed off), and authentication, confirming users are who they claim to be with usernames, passwords and increasingly a second factor.

Common misconception. Validation checks that input is reasonable, not that it is true. A range check happily accepts an age of 25 from a 12-year-old. Verification, checking correctness, for example by double entry, is a different thing, and exam questions probe whether you know the difference.

9.2 Maintainability

Programs are read far more often than they are written, frequently by someone who is not the author, and often by the author six months later, who is functionally a stranger. Maintainable code is written for that reader: sub programs to break the logic into named, testable pieces; meaningful variable names (numberOfGuests, not n or x); consistent indentation so the structure is visible at a glance; and comments explaining anything whose purpose is not obvious. In the exam, “give two ways the maintainability of this program could be improved” is asked of code that is deliberately awful; the marks are for naming these techniques.

9.3 Testing

Testing is how you find errors before users do. Iterative testing happens during development, checking each module as it is built; final testing happens once the whole program is assembled, checking everything works together. You already know the two kinds of error from Section 7: syntax errors, which stop the program running, and logic errors, which let it run and be wrong.

A proper test plan attacks the program with four kinds of data:

- Normal: sensible values it should accept, such as a mark of 62.
- Boundary: values at the exact edges of the acceptable range, such as 0 and 100. Edges are where logic errors live; a program using > instead of >= fails only at the boundary.
- Invalid: values of the right type but outside the range, such as 105.
- Erroneous: values of the wrong type entirely, such as "twelve". The program should reject these gracefully, not crash.

Exam tip. When asked to design test data, always give the value and what the program should do in response. “Test with 100” earns little; “boundary test with 100, which should be accepted” earns the mark.

Key terms

Term	Meaning
Defensive design	Writing programs that anticipate and cope with problems
Validation	Checking that input is reasonable before it is used
Sanitisation	Cleaning input so it cannot be executed as code
Authentication	Confirming a user's identity
Maintainability	How easily a program can be understood and changed
Iterative / final testing	Testing during development / testing the finished whole
Boundary data	Test values at the exact limits of the acceptable range
Erroneous data	Test values of the wrong type, which should be rejected

Try it in Python – Validation that survives contact with users

Build the range check from 9.1, then attack it with your own test data.

```
while True:
    try:
        mark = int(input("Enter mark: "))
        if 0 <= mark <= 100:
            break
    except ValueError:
        pass
    print("Must be a whole number from 0 to 100.")
print("Accepted:", mark)
```

Your task: Test it with 62, 0, 100, 105 and the word twelve. try/except is how Python survives erroneous data instead of crashing.

Section 9 exercises

Quick check

- 1. Name the validation check that ensures a field was not left empty. [1]
- 2. State the difference between iterative and final testing. [1]

Practice

- 3. A program accepts percentages from 0 to 100. Give one item of normal, boundary, invalid and erroneous test data. [4]
- 4. Give two techniques that improve the maintainability of a program, and explain how each helps a future programmer. [4]

Exam-style

- 5. Write ERL code that repeatedly asks the user for a quantity until they enter a value from 1 to 20. [3]
- 6. Explain why testing only with normal data is insufficient, referring to where logic errors are most likely to appear. [3]

Section 10 – Boolean logic (spec 2.4)

Before you start – from earlier sections

1. Name two types of validation check.
2. What is boundary test data?
3. Give one way sub programs improve maintainability.

Beneath every device you own, decisions are being made by circuits that only understand true and false. Boolean logic is the algebra of those decisions, and at GCSE it comes down to three gates, their truth tables, and combining them.

10.1 The three gates

The three logic gates and their truth tables

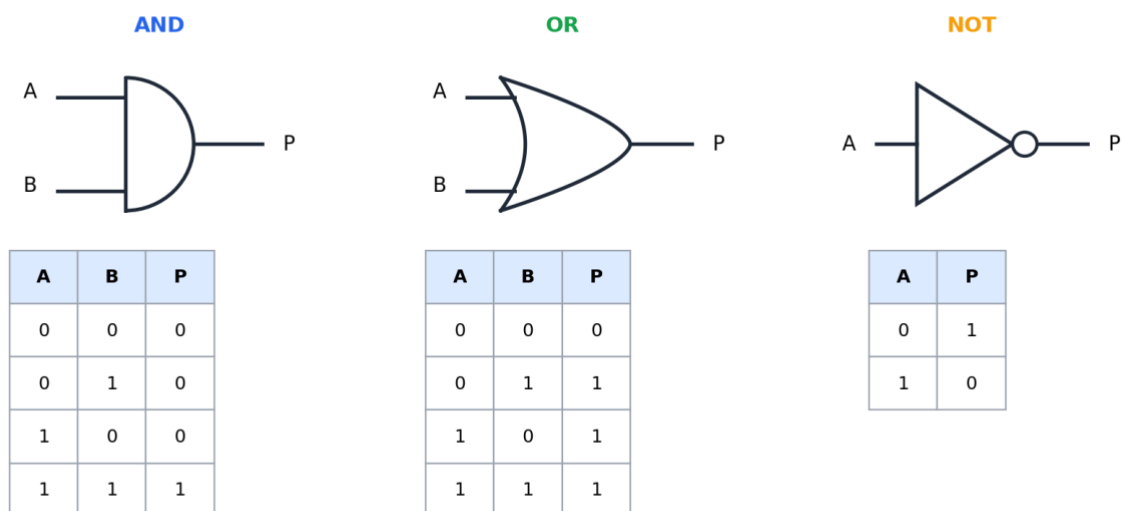


Fig 10.1 – AND, OR and NOT with their truth tables

- AND outputs 1 only when both inputs are 1. Written $P = A \text{ AND } B$. Think of two switches in series: both must be on.
- OR outputs 1 when at least one input is 1. Written $P = A \text{ OR } B$. Two switches in parallel: either will do.
- NOT outputs the opposite of its single input. Written $P = \text{NOT } A$. An inverter.

A truth table lists every possible combination of inputs and the output each produces. Two inputs give four rows; three inputs give eight. Fill in the input columns by counting in binary (00, 01, 10, 11) and you will never miss a row.

10.2 Combining gates

Real logic chains gates together, and the technique for analysing any combination is always the same: add a working column for each intermediate signal and build the table left to right. Take $P = (A$

AND B) OR NOT C. Work out A AND B for every row, then NOT C for every row, then OR those two columns:

A	B	C	A AND B	NOT C	P
0	0	0	0	1	1
0	0	1	0	0	0
0	1	0	0	1	1
0	1	1	0	0	0
1	0	0	0	1	1
1	0	1	0	0	0
1	1	0	1	1	1
1	1	1	1	0	1

Logic is also where programming and hardware meet: the expression in if (age >= 13 AND age <= 19) OR member == true then is exactly this section, wearing code.

Common misconception. In everyday English, “or” often means one or the other but not both (“tea or coffee?”). Boolean OR is inclusive: 1 OR 1 is 1. Bring the everyday meaning into a truth table and the last row goes wrong.

Key terms

Term	Meaning
Logic gate	A circuit producing an output determined by its input(s)
Truth table	A table showing the output for every combination of inputs
AND	Outputs 1 only when all inputs are 1
OR	Outputs 1 when at least one input is 1
NOT	Outputs the opposite of its input

Section 10 exercises

Quick check

- 1. How many rows are in the truth table of a circuit with three inputs? [1]
- 2. State the output of an OR gate when both inputs are 1. [1]

Practice

- 3. Draw or complete the truth table for $P = \text{NOT}(A \text{ AND } B)$. [3]
- 4. Write the logic statement for a burglar alarm that sounds when the system is armed AND either the door sensor OR the window sensor is triggered. [2]
- 5. Draw the logic diagram for $P = (A \text{ OR } B) \text{ AND NOT } C$. [3]

Exam-style

- 6. Construct the full truth table for $P = (A \text{ AND } B) \text{ OR } B$, and state, with a reason, which single input P behaves identically to. [4]

Section 11 – Programming languages and IDEs (spec 2.5)

Before you start – from earlier sections

1. What is the output of an AND gate with inputs 1 and 0?
2. How many rows has the truth table of a two-input circuit?
3. What does a NOT gate do?

You finish Component 02 with a step back: what actually happens between the code you write and the machine that runs it, and what tools make writing that code bearable.

11.1 High-level and low-level languages

High-level languages (Python, Java, and the ERL in this book) are written for humans: one line can express a whole idea, the code reads almost like English, and the same program can run on different types of machine. Low-level languages are written for the hardware. Machine code is the binary the CPU actually executes; assembly language is machine code made just barely human-readable, with short mnemonics standing in for binary instructions, and it is used when a programmer needs direct control of the hardware or maximum performance, such as in device drivers and embedded systems. The trade is always the same: high-level for ease and portability, low-level for control and efficiency.

11.2 Translators: compilers and interpreters

The CPU only runs machine code, so every high-level program must be translated. There are two ways to do it, and comparing them is one of the most reliable six-markers on the paper.

	Compiler	Interpreter
How it works	Translates the entire program in one go, producing an executable file of machine code	Translates and runs one line at a time, every time the program runs
Speed	Runs fast once compiled; nothing further to translate	Runs slower, since translation happens during execution
Errors	Reports all errors after compiling; nothing runs until the code compiles	Stops at the first error it meets, which makes errors easy to locate while developing
Distribution	The executable runs without the source code or the compiler	The source code and an interpreter are needed on every machine

Exam tip. Compiler questions reward precision about when translation happens. A compiled program is translated once, in advance; an interpreted program is translated every time it runs, line by line. Get those two clauses in and the comparison marks follow.

11.3 IDEs

An integrated development environment (IDE) gathers a programmer's tools into one application, and you need four of its features. The editor is where code is written, helping with colour-coding and auto-indentation. Error diagnostics underline mistakes and explain them, often before you even run the program. The run-time environment lets you run the program inside the IDE and watch its output. And the translator, the compiler or interpreter itself, is built in, so running your code is one click. None of these features write the program for you; all of them shorten the distance between making a mistake and finding it, which is most of what programming is.

Key terms

Term	Meaning
High-level language	A language written for humans, translated before the CPU can run it
Machine code	The binary instructions a CPU executes directly
Assembly language	A low-level language using mnemonics for machine instructions
Compiler	Translates a whole program in advance into an executable
Interpreter	Translates and executes a program one line at a time
IDE	One application combining the tools needed to develop programs

Section 11 exercises

Quick check

- 1. Name the only language a CPU can execute directly. [1]
- 2. Give two features of an IDE. [2]

Practice

- 3. Give one situation where a programmer would choose a low-level language, with a reason. [2]

Exam-style

- 4. A developer has finished a game and wants to sell it without customers seeing the source code. Explain whether a compiler or an interpreter should be used, giving two reasons. [4]
- 5. Describe how error diagnostics and the run-time environment of an IDE each help a student debug a program. [4]

Answers

Mark yourself honestly. If you got an answer wrong, go back to the section and find out why before moving on; the answer alone teaches you nothing.

Section 1 – Systems architecture

- 1. It holds the memory address of the next instruction to be fetched. [1]
- 2. The accumulator. [1]
- 3. A computer with a single dedicated function, built into a larger device. [1]
- 4. The address in the program counter is copied into the MAR [1]; the instruction stored at that address is copied from memory into the MDR [1]; the program counter is incremented ready for the next cycle [1].
- 5. A larger cache holds more of the frequently used instructions and data close to the CPU [1], so the CPU spends less time waiting for the slower main memory [1].
- 6. The 3.6 GHz CPU completes more cycles per second on each core, so individual tasks run faster [1]; the quad-core can process more instructions simultaneously [1], but only if the software is written to use multiple cores, and many games rely heavily on one or two [1]; a justified conclusion either way, e.g. the dual-core for typical games, earns the final mark [1].
- 7. Any two: it is cheaper to produce; it uses less power; it can be physically smaller; a dedicated device performing one function is simpler and more reliable; general-purpose features would never be used. [2]

Section 2 – Memory and storage

- 1. Memory that loses its contents when the power is turned off. [1]
- 2. 4. [1]
- 3. Solid state. [1]
- 4. $128 + 32 + 16 + 4 = 180$. [1]
- 5. 01001011 ($64 + 8 + 2 + 1$). [1]
- 6. Split into nibbles: $1101 = 13 = D$ [1]; $0111 = 7$; answer D7 [1].
- 7. $10011010 + 01101100 = 100000110$ [1 for correct column addition]. The result needs nine bits but only eight are available [1], so the ninth bit is lost: this is overflow, and the stored 8-bit result (00000110) is wrong [1].
- 8. 00110000 [1]. A left shift of two places multiplies the value by 4: 12 becomes 48 [1].
- 9. File size = sample rate $\times$ duration $\times$ bit depth = $8,000 \times 10 \times 8 = 640,000$ bits [1] = 80,000 bytes [1] = 80 KB [1]. (78.125 KB using 1,024, also accepted.)
- 10. 1 mark for SSD, plus 1 mark per justified characteristic, maximum 3 marks. Solid state, e.g. an external SSD or high-capacity USB drive [1]. Any three of: highly portable, being small and light, so easy to carry between studios [1]; durable, with no moving parts, so it survives daily transport and knocks [1]; fast read/write speeds, so large image files transfer quickly [1]; enough capacity for large image files [1]. [4]

- 11. RAM is full, so the OS is using virtual memory: moving inactive data out to secondary storage and back [1]. Secondary storage is much slower than RAM, and this constant swapping causes the slowdown [1]. Installing more RAM would help [1].
- 12. Lossy compression permanently removes data [1]; source code cannot lose a single character and still run correctly, so only lossless compression, which restores the file perfectly, is acceptable [1].

Section 3 – Computer networks, connections and protocols

- 1. A LAN covers a small area with infrastructure owned by the organisation; a WAN connects sites over large distances using hired infrastructure. (Either difference.) [1]
- 2. A router. [1]
- 3. It translates a domain name (web address) into the IP address of the server hosting it. [1]
- 4. An IP address is assigned by the network while a MAC address is assigned by the manufacturer [1]; an IP address can change when the device moves networks while a MAC address is fixed for the life of the device [1].
- 5. In a wired full mesh every pair of devices needs its own cable, and the number of cables grows rapidly with each added device, becoming impractical and expensive [1]; wireless devices connect to each other through the air, so full connectivity needs no extra cabling as devices are added [1].
- 6. IMAP [1], because it leaves messages on the server, so all three devices see the same synchronised mailbox; POP would download mail to one device and remove it from the server [1].
- 7. Peer-to-peer [1]: there is no expensive server to buy, and it is simple enough to set up without specialist staff [1]. As the company grows, they should expect problems with the lack of central management [1]: backups and security must be handled on every machine individually, which becomes unmanageable [1].
- 8. At 12:30 many users (students at lunch) share the network's fixed bandwidth simultaneously [1], so each user's share collapses [1]. Solutions include increasing the bandwidth of the connection or restricting non-essential traffic at peak times [1].
- 9. Each layer is self-contained [1], so one layer can be updated or replaced without changing the others (or: manufacturers can develop products for one layer knowing they will interoperate) [1].

Section 4 – Network security

- 1. A virus needs a host file and user action to spread; a worm spreads across networks by itself. [1]
- 2. Denial of service. [1]
- 3. To find and report a system's vulnerabilities, with authorisation, before real attackers exploit them. [1]
- 4. Each account can only access what its role requires [1], so an attacker who compromises one account can only reach that limited part of the system rather than everything [1].
- 5. Data travelling over the unencrypted Wi-Fi can be intercepted by packet sniffing [1], letting an attacker read customers' logins and messages [1]. Encrypting the network (or customers using HTTPS/a VPN) means intercepted data is unreadable without the key [1].
- 6. Phishing [1]. It attacks people rather than technology: no firewall or filter can reliably stop an employee being persuaded to hand over their own credentials, and some fraudulent messages will always get through [2 for the explanation]. The most effective response is staff training so employees recognise and report such emails [1].
- 7. Ransomware [1]. Regular backups kept separate from the network would allow records to be restored without paying [1]; plus any one of: up-to-date anti-malware, staff training against the phishing that usually delivers ransomware, or user access levels limiting how far it could spread [1].

Section 5 – Systems software

- 1. A (device) driver. [1]
- 2. Any two of: providing a user interface; memory management and multitasking; peripheral management; user management; file management. [2]
- 3. The OS switches the CPU between the open programs extremely rapidly, giving each a slice of time in turn [1]; the switching is so fast that all the programs appear to run simultaneously [1].
- 4. As files are saved, edited and deleted, their parts become scattered (fragmented) across the disk [1]; the read arm must travel between the parts, slowing access [1]. Defragmentation moves each file's parts back together and groups the free space, reducing arm movement and speeding the disk up [1].
- 5. An SSD has no moving parts [1], so there is no read arm travelling between fragments and therefore no time to save; access speed is the same wherever data is stored [1].
- 6. Encryption software [1]. The disk's contents are scrambled using a key [1], so a thief who steals the laptop cannot read the client data without that key [1].

Section 6– Ethical, legal, cultural and environmental impacts

- 1. The Computer Misuse Act 1990. [1]
- 2. The gap between those who have access to technology, connectivity and the skills to use them, and those who do not. [1]

- 3. Open-source code can be viewed and modified by anyone; proprietary source code is kept secret, and the software is licensed for use only. (Any valid difference.) [1]
- 4. Positive: e.g. video conferencing reduces travel and its emissions, or smart systems cut energy use [1]. Negative: e.g. e-waste, the energy demands of data centres, or the mining of rare metals for devices [1].
- 5. The Data Protection Act 2018 [1]: the addresses were collected for one stated purpose (deliveries) and used for another (sale to advertisers) without consent [1].
- 6. Level-marked. A strong answer considers at least two groups and both directions. For example: most residents gain faster, always-available services and the council saves money that can fund other services [1–2]; however, elderly residents and those without internet access or digital skills fall on the wrong side of the digital divide and may be unable to access essential services at all [1–2]; walk-in staff lose jobs, though some may be retrained; mitigations such as keeping a phone line, providing library terminals or offering training reduce the harm [1–2]. A reasoned conclusion is expected for full marks.
- 7. Open source is likely more appropriate [1]: the licence is free, which fits the budget [1]. The weakness is support: with no IT staff, the charity would rely on community forums rather than a company helpdesk [1]. A justified conclusion, e.g. open source with a volunteer or donated support arrangement, earns the final mark [1].

Section 7 – Algorithms

- 1. Removing unnecessary detail from a problem so that only the important parts remain. [1]
 - 2. The decision symbol (diamond). [1]
 - 3. The list must be sorted / in order. [1]
 - 4. A complete pass happens with no swaps. [1]
 - 5. Abstraction. [1] (Not decomposition: nothing is being broken into parts, detail is being ignored.)
 - 6. x takes the values 10, 6 and 2. Once x is 2, the condition $x > 3$ is false, so the loop stops and the output is 2. One mark for each value of x after the start (6 and 2), one for the correct output. [3]
 - 7. Pass 1: [5, 8, 1, 9]. Pass 2: [5, 1, 8, 9]. One mark per pass. A very common slip is doing pass 1 correctly and then rushing pass 2. [2]
 - 8. After inserting 2: [2, 7, 9, 4]. After inserting 9: [2, 7, 9, 4] (already in place). After inserting 4: [2, 4, 7, 9]. [3]
 - 9. After splitting: [4] [1] [3] [2]. First merge: [1, 4] and [2, 3]. Final merge: [1, 2, 3, 4]. [3]
10. One possible answer (marks for: sensible input with casting, a loop that runs the right number of times, a comparison that updates the largest value, and the final output): [4]

```
largest = int(input("Enter a number: "))
for i = 1 to 9
    num = int(input("Enter a number: "))
    if num > largest then
        largest = num
```

```

    endif
next i
print(largest)

```

- 11. Binary search [1], because the names are already sorted alphabetically [1], so the app can find a contact in far fewer comparisons than checking every name in turn, which matters as the contact list grows [1]. [3]
- 12. Merge sort is more suitable [1]. With two million records a bubble sort would require an enormous number of comparisons and swaps and would be far too slow [1], whereas merge sort remains fast on large lists regardless of the starting order [1]. A balanced answer also notes merge sort's cost: it needs additional memory during merging, but for a dataset this size the speed difference outweighs it [1]. [4]
- 13. The line `total = i` replaces the total instead of adding to it, so the program prints 5. [1]
Corrected line: `total = total + i`. [1]

Section 8 – Programming fundamentals

- 1. A variable's value can change while the program runs; a constant's value is fixed for the whole run. [1]
- 2. $23 \text{ MOD } 4 = 3$ [1]; $23 \text{ DIV } 4 = 5$ [1].
- 3. 7. [1]
- 4. board (5 characters starting at position 3: b-o-a-r-d). [1]
- 5. `input()` always returns a string [1]; casting with `int()` converts it to an integer so it can be used in calculations and numerical comparisons [1].
- 6. `SELECT name, price [1] FROM products [1] WHERE price > 10 [1]`.
- 7. A while loop checks its condition before each pass, so its body may run zero times [1]; a do until loop checks after, so its body always runs at least once [1].

8. One mark each for the function header with parameter, correct even/odd logic using MOD, and returning a Boolean: [3]

```

function isEven(num)
    return num MOD 2 == 0
endfunction

```

9. Marks for: initialising with the first element, looping over the remaining indexes, comparing and updating, and outputting the result: [4]

```

highest = temps[0]
for i = 1 to 6
    if temps[i] > highest then
        highest = temps[i]
    endif
next i
print(highest)

```

- 10. Code written once can be reused wherever needed [1], avoiding duplication and the errors that come from maintaining several copies [1]. Each sub program can be tested and debugged

- in isolation [1], making faults far easier to locate in a large program [1]. (Readability/teamwork also creditworthy.)

Section 9 – Producing robust programs

- 1. A presence check. [1]
 - 2. Iterative testing happens during development as each part is built; final testing checks the completed program as a whole. [1]
 - 3. Normal: 62, accepted. Boundary: 0 or 100, accepted. Invalid: 105 (or -1), rejected. Erroneous: “twelve”, rejected without crashing. One mark each, provided the expected outcome is stated. [4]
 - 4. Any two with explanations, e.g. meaningful variable names, so a future programmer can tell what each value is for without decoding it [2]; consistent indentation (or comments, or sub programs), so the structure and purpose of the code are visible at a glance [2].
5. Marks for taking input with casting, a condition covering both limits, and re-prompting inside the loop: [3]

```
quantity = int(input("Enter quantity: "))
while quantity < 1 OR quantity > 20
    quantity = int(input("Must be 1 to 20. Try again: "))
endwhile
```

- 6. Logic errors cluster at the edges of ranges, for example where > was written instead of >= [1]; normal data passes through both correct and faulty versions identically, so it cannot reveal the fault [1]; boundary, invalid and erroneous data test exactly the places where the program's handling is most likely to be wrong [1].

Section 10 – Boolean logic

- 1. 8. [1]
- 2. 1. [1]
- 3. NOT (A AND B): inputs 00, 01 and 10 all give P = 1; input 11 gives P = 0. One mark for correct rows, one for the working AND column, one for the final inversion. [3]
- 4. sounds = armed AND (door OR window). One mark for the AND structure, one for bracketing the OR pair. [2]
- 5. An OR gate with inputs A and B [1]; a NOT gate with input C [1]; the outputs of the OR gate and the NOT gate both feed into an AND gate, whose output is P [1]. The NOT gate must act on C alone, before the AND gate.
- 6. Rows (A, B, A AND B, P): 0 0 → 0, 0; 0 1 → 0, 1; 1 0 → 0, 0; 1 1 → 1, 1. [2 for a correct table with working column] P is identical to input B [1]: whenever B is 1 the OR makes P 1 regardless of A, and whenever B is 0 the AND is also 0, so P is 0 [1].

Section 11 – Programming languages and IDEs

- 1. Machine code. [1]
- 2. Any two of: editor, error diagnostics, run-time environment, translator. [2]
- 3. E.g. writing a device driver or software for an embedded system [1], because the programmer needs direct control of the hardware or maximum efficiency [1].
- 4. A compiler [1]: it produces a standalone executable, so customers receive machine code and never see the source [1]; the executable also runs without any translator installed and runs faster, since translation happened in advance [1]; an interpreter would require distributing the readable source code to every customer [1].
- 5. Error diagnostics highlight the line containing a mistake and describe the problem, so the student can locate and correct it quickly [2]; the run-time environment lets the student run the program immediately inside the IDE and observe its behaviour, so each attempted fix can be tested straight away [2].

References and Bibliography

This book has been written as an independent teaching and revision resource for OCR GCSE (9–1) Computer Science J277. The specification, assessment materials and official guidance published by Cambridge OCR have been used as the primary reference points for syllabus coverage, terminology, programming conventions and examination expectations.

The book is not endorsed by or affiliated with Cambridge OCR. All explanations, worked examples, exercises and teaching commentary are independently written unless otherwise stated.

Primary specification and assessment sources

Cambridge OCR (2026) *GCSE (9–1) Computer Science J277 Specification*. Version 3.1. Cambridge: Oxford Cambridge and RSA Examinations. Available at: <https://www.ocr.org.uk/Images/558027-specification-gcse-computer-science-j277.pdf> (Accessed: 22 October 2026).

Cambridge OCR (2024) *GCSE (9–1) Computer Science J277: Understanding the Assessment – Exploring Our Question Papers*. Version 2.0. Cambridge: Oxford Cambridge and RSA Examinations. Available at: <https://www.ocr.org.uk/Images/562109-understanding-the-assessment-exploring-our-question-papers.pdf> (Accessed: 22 October 2026).

Cambridge OCR (2026) *GCSE Computer Science (9–1) J277: Assessment Materials*. Cambridge: Oxford Cambridge and RSA Examinations. Available at: <https://www.ocr.org.uk/qualifications/gcse/computer-science-j277-from-2020/assessment/> (Accessed: 22 October 2026).

Cambridge OCR (2026) *Cambridge OCR Exam Reference Language (ERL)*. Cambridge OCR Customer Support. Available at: <https://support.ocr.org.uk/hc/en-gb/articles/30600830976786-Cambridge-OCR-exam-reference-language-ERL> (Accessed: 22 October 2026).

Cambridge OCR (2026) *GCSE Computer Science (9–1) J277: Frequently Asked Questions*. Available at: <https://www.ocr.org.uk/subjects/computing/frequently-asked-questions/gcse-9-1-computer-science-j277-2020/> (Accessed: 22 October 2026).

Legislation

Great Britain (1988) *Copyright, Designs and Patents Act 1988*. London: The National Archives. Available at: <https://www.legislation.gov.uk/ukpga/1988/48> (Accessed: 22 October 2026).

Great Britain (1990) *Computer Misuse Act 1990*. London: The National Archives. Available at: <https://www.legislation.gov.uk/ukpga/1990/18> (Accessed: 22 October 2026).

Great Britain (2018) *Data Protection Act 2018*. London: The National Archives. Available at: <https://www.legislation.gov.uk/ukpga/2018/12> (Accessed: 22 October 2026).

Programming reference

Python Software Foundation (2026) *Python 3 Documentation*. Available at: <https://docs.python.org/3/> (Accessed: 22 October 2026).

Python Software Foundation (2026) *The Python Tutorial*. Available at: <https://docs.python.org/3/tutorial/> (Accessed: 22 October 2026).

Note on examination material

Publicly available OCR specifications, assessment guidance, past examination papers, mark schemes and examiner reports have been consulted to ensure that terminology, depth of coverage and question style remain appropriate for OCR GCSE Computer Science J277. Questions and worked examples in this book are independently written and are not reproduced OCR examination questions unless explicitly identified as such.

All website addresses were checked on 22 October 2026.