Course Companion

International Baccalaureate · IB Diploma · Computer Science · HL

Every topic handout in one volume

全部专题讲义合集

exercises.lol

Contents 目录

1. **Computer fundamentals** 2
2. **Networks** 16
3. **Databases** 27
4. **Machine learning** 36
5. **Computational thinking** 45
6. **Programming** 52
7. **Object-oriented programming** 66
8. **Abstract data types (HL only)** 76
9. **Case study** 85
10. **Computational solution** 93

Computer fundamentals

International Baccalaureate · IB Diploma · Computer Science · HL

Scope and prerequisites

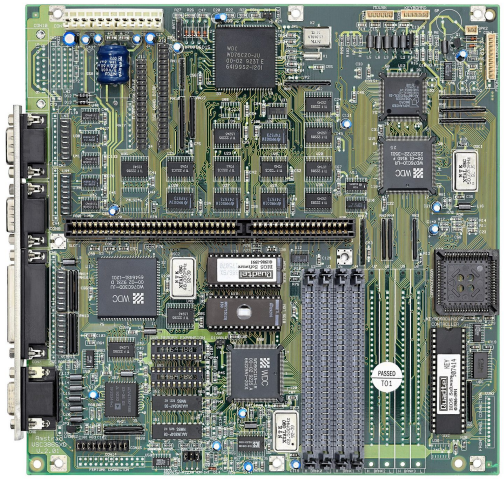
This handout teaches Theme A1, Computer fundamentals, for the course first assessed in 2027: hardware, data representation, logic and operating systems. Sections marked **AHL** are for Higher Level only.

Prerequisites: powers of 2 ($2^{10} = 1024$) and simple arithmetic.

Computer hardware

What happens when you tap an app?

You tap a photo app. Within a second its code is copied from storage into memory, the processor runs millions of instructions, the graphics chip draws the picture, and the operating system shares the processor with the music still playing. This handout follows that path.

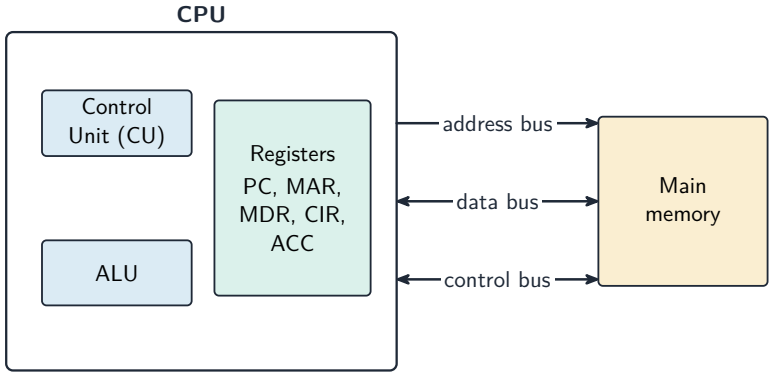


A motherboard: the CPU, memory and I/O all sit on one set of buses – the printed tracks running between them

The central processing unit

The **central processing unit** 中央处理器 (CPU) runs program instructions. Its main parts are:

Part	Job
control unit 控制单元 (CU)	fetches and decodes instructions, and sends control signals to the other parts
arithmetic logic unit 算术逻辑单元 (ALU)	does arithmetic (add, subtract) and logic (AND, compare)
registers 寄存器	tiny, very fast stores inside the CPU

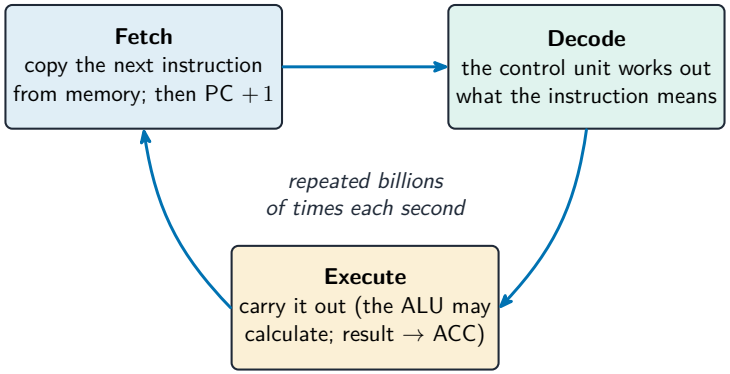


The CPU and main memory, joined by three buses

Register	Holds
program counter (PC)	the address of the next instruction
memory address register (MAR)	the address being read or written
memory data register (MDR)	the data or instruction just read, or about to be written
current instruction register (CIR)	the instruction being decoded and executed
accumulator (ACC)	the result of the last ALU operation

The CPU is joined to memory by **buses** 总线, sets of parallel wires: the **address bus** carries addresses from the CPU to memory, the **data bus** carries data both ways, and the **control bus** carries signals such as "read" and "write".

The fetch–decode–execute cycle



The cycle every CPU repeats

1. **Fetch**: the address in the PC is copied to the MAR; the instruction at that address travels on the data bus to the MDR, then to the CIR. The PC is increased by 1.
2. **Decode**: the CU works out what the instruction in the CIR means.
3. **Execute**: the instruction is carried out, for example the ALU adds and the result goes into the ACC.

The **clock speed** 时钟频率 is the number of cycles per second: a 3 GHz CPU has 3×10^9 clock ticks per second. Performance also depends on the number of **cores** 核心 (each core runs its own instructions at the same time) and on the size of the cache.

CPU and GPU

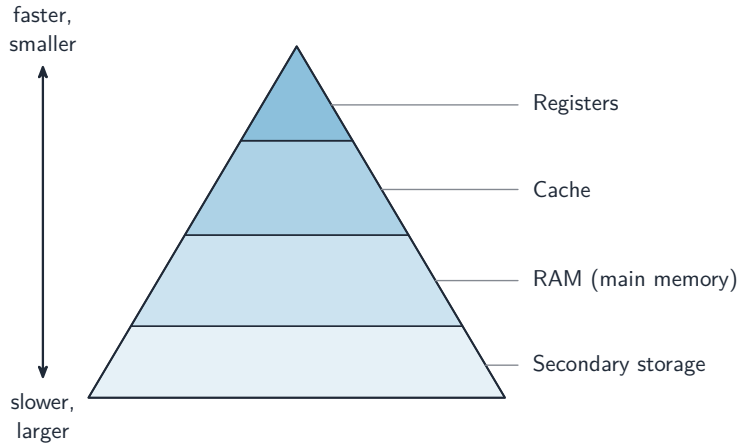
A **graphics processing unit** 图形处理器 (GPU) has thousands of simple cores that do the same operation on many pieces of data at once.

	CPU	GPU
cores	few, powerful	thousands, simple
good at	varied tasks with many decisions	the same calculation on lots of data
uses	running the OS and apps	graphics, video, machine-learning training

The CPU and GPU work together: the CPU runs the program and hands the parallel work, such as drawing each pixel, to the GPU.

Memory and storage

Type	Volatile?	Use
RAM 随机存取存储器	yes: loses its data without power	the programs and data in use now
ROM 只读存储器	no	the start-up instructions (firmware)
cache 高速缓存	yes	copies of the data the CPU uses most, inside or next to the CPU



The memory hierarchy: each level trades speed for size and cost

Cache works because programs reuse the same data and instructions again and again. Level 1 cache is the smallest and fastest; levels 2 and 3 are larger and slower, but all are faster than RAM. A **cache hit** finds the data in cache; a **cache miss** must fetch it from RAM.

Secondary storage 辅助存储器 keeps data without power:

Type	How it stores bits	Strengths	Weaknesses
hard disk (HDD)	magnetised spots on spinning platters	cheap per gigabyte	moving parts, slower
solid state (SSD)	electric charge in flash memory	fast, silent, robust	dearer per gigabyte
optical (DVD)	pits read by a laser	cheap to distribute	small, slow

Compression

Compression 压缩 makes files smaller, so they take less storage and transfer faster.

- **Lossless** 无损: the original can be rebuilt exactly. Needed for text and program files. Example, **run-length encoding** (RLE): `WWWWWBBBWW` becomes `5W3B2W`.
- **Lossy** 有损: removes detail people are unlikely to notice; the original cannot be rebuilt. Used for photos (JPEG), music (MP3) and video.

Cloud computing

Cloud computing 云计算 provides computing resources over the internet, paid for as they are used.

Service	The provider supplies	Example
SaaS (software as a service)	a finished application	web e-mail, online office apps
PaaS (platform as a service)	a platform to build and run your own apps	a hosted database and web server
IaaS (infrastructure as a service)	virtual machines, storage and networks	renting servers by the hour

Advantages: no hardware to buy, scales up and down, accessible anywhere. Disadvantages: needs a reliable connection, ongoing costs, and the data is held by another company.

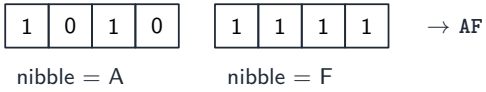
Data representation

Binary and hexadecimal

Computers store everything as **binary** 二进制 digits, **bits** 位, each 0 or 1. Eight bits make a **byte** 字节.



A seven-segment display shows one denary digit, often driven by BCD



A byte splits into two nibbles, and each nibble is one hex digit: 1010 1111 = AF

Place value	128	64	32	16	8	4	2	1
bits of 182	1	0	1	1	0	1	1	0

$128 + 32 + 16 + 4 + 2 = 182$. With n bits there are 2^n different values: 8 bits store 0 to 255.

Hexadecimal 十六进制 (base 16) uses the digits 0–9 and A–F (10–15). Each hex digit stands for exactly 4 bits, so hex is a short, readable way to write binary: `1011 0110` is `B6`, and $11 \times 16 + 6 = 182$. Hex is used for colour codes (`#FF8800`) and memory addresses.

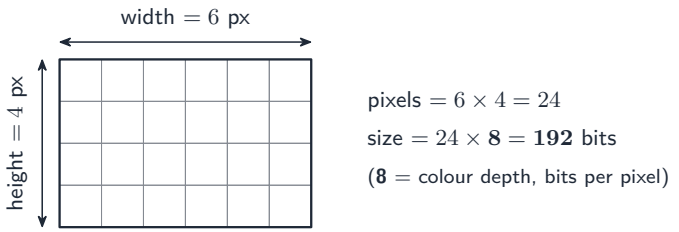
Unit	Size
kibibyte (KiB)	$2^{10} = 1024$ bytes
kilobyte (kB)	1000 bytes
megabyte (MB)	10^6 bytes
gigabyte (GB)	10^9 bytes

Negative numbers

Two's complement 补码 stores negative integers: the leftmost bit has a negative place value (-128 in 8 bits). To write -42: write 42 as `00101010`, flip every bit to `11010101`, then add 1: `11010110`. Check: $-128 + 64 + 16 + 4 + 2 = -42$. Eight bits store -128 to 127.

Text, images and sound

- **Text**: each character has a code. **ASCII** uses 7 bits (128 characters, English only); **Unicode** 统一码 assigns code points to characters and symbols, including Chinese; encodings such as UTF-8, UTF-16 and UTF-32 store those code points using different code-unit sequences.
- **Images**: a bitmap is a grid of **pixels** 像素; the **colour depth** 色深 is the number of bits per pixel. File size in bits = width × height × colour depth.
- **Sound**: the wave is measured at regular intervals. The **sample rate** 采样率 is samples per second; the **bit depth** is bits per sample. Size in bits = sample rate × bit depth × channels × seconds.



The same formula on small numbers: count the pixels, then multiply by the colour depth

Worked examples:

$$\text{image size} = \text{width} \times \text{height} \times \text{colour depth}$$

$$1920 \times 1080 \times 24 = 49\,766\,400 \text{ bits} = 6\,220\,800 \text{ bytes} \approx 6.2 \text{ MB}$$

$$\text{sound size} = \text{sample rate} \times \text{bit depth} \times \text{channels} \times \text{time}$$

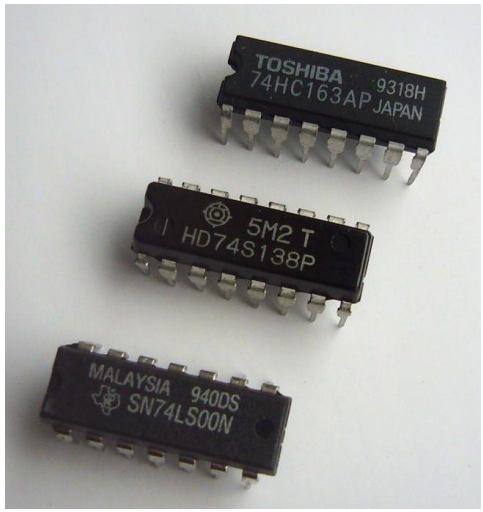
$$44\,100 \times 16 \times 2 \times 60 = 84\,672\,000 \text{ bits} = 10\,584\,000 \text{ bytes} \approx 10.6 \text{ MB}$$

A higher sample rate, bit depth or colour depth gives better quality and a bigger file.

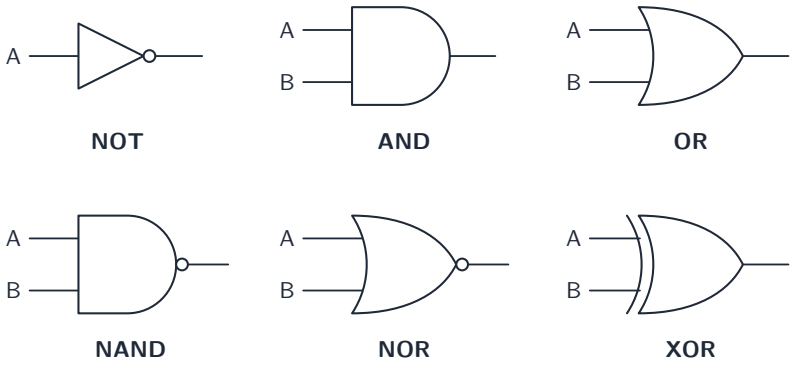
Computer logic

Logic gates

A **logic gate** 逻辑门 takes one or two binary inputs and gives one output. A **truth table** 真值表 lists the output for every combination of inputs.



A real logic chip: inside are logic gates like the ones on this page



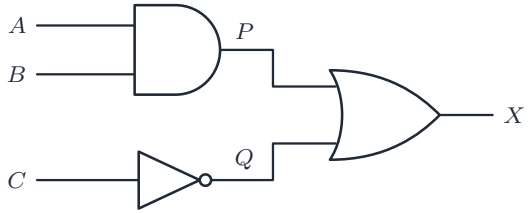
The six gates; a small circle on the output means NOT

A	B	**AND**	**OR**	**XOR**	**NAND**	**NOR**
0	0	0	0	0	1	1
0	1	0	1	1	1	0
1	0	0	1	1	1	0
1	1	1	1	0	0	0

NOT has one input and flips it: NOT 0 = 1. XOR is 1 when the inputs **differ**; XNOR is 1 when they are the same.

Logic circuits

Gates are joined into circuits. Work out a circuit's truth table by labelling each gate's output and filling in one column per label.



$$X = (A \ AND \ B) \ OR \ (NOT \ C)$$

A	B	C	$P = A$ **AND** B	$Q =$ **NOT** C	$X = P$ **OR** Q
0	0	0	0	1	1
0	0	1	0	0	0
0	1	0	0	1	1
0	1	1	0	0	0
1	0	0	0	1	1
1	0	1	0	0	0
1	1	0	1	1	1
1	1	1	1	0	1

Three inputs give $2^3 = 8$ rows; list them counting up in binary so none is missed.

Operating systems

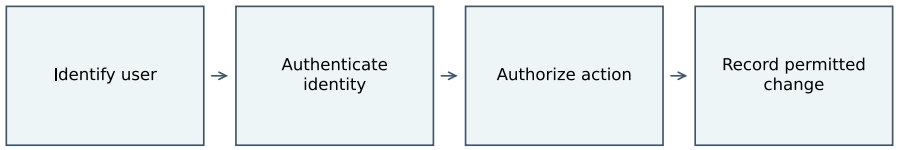
What an operating system does

An **operating system** 操作系统 (OS) manages the hardware and provides services to programs and users.



A phone runs a mobile operating system that manages memory, processes, files and the hardware for every app

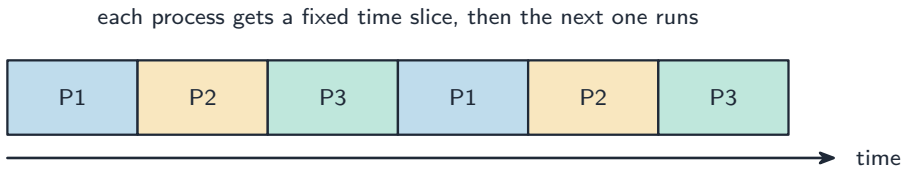
Function	What it does
process management 进程管理	starts and stops programs and shares the CPU between them
memory management 内存管理	gives each program its own area of RAM and protects it from others
file management	organises files in folders and controls who may read or write them
device management	controls hardware through **drivers** 驱动程序
security	checks users' identity and their permissions
user interface	a graphical or command-line interface



The OS checks who you are (authentication) before what you may do (authorization)

Scheduling

One CPU core runs one process at a time; the OS **scheduler** 调度程序 switches between processes so fast that they seem to run together.



Round-robin: each process gets a fixed time slice in turn, then the next runs (unlike first-come-first-served)

Method	Rule	Weakness
first come, first served	run each process to the end, in arrival order	a long job makes short ones wait
round robin	each process gets a fixed **time slice** 时间片 in turn	many switches if the slice is small
priority	the highest-priority ready process runs	low-priority processes may never run

Round robin, time slice 2 ms, processes P1 (5 ms), P2 (3 ms), P3 (1 ms) arriving together:

Time (ms)	**0–2**	**2–4**	**4–5**	**5–7**	**7–8**	**8–9**
runs	P1	P2	P3 (ends)	P1	P2 (ends)	P1 (ends)

P3 finishes at 5 ms, P2 at 8 ms, P1 at 9 ms.

Polling and interrupts

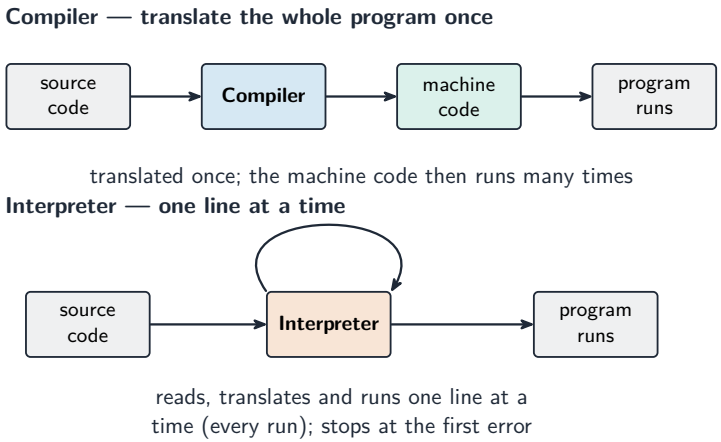
The CPU must know when a device needs attention.

- **Polling** 轮询: the CPU checks each device again and again. Simple, but wastes CPU time when nothing has happened.
- **Interrupt** 中断: the device sends a signal; the CPU finishes its current instruction, saves its state, runs the interrupt's handler, then carries on. Efficient for events that are rare or unpredictable, such as a key press.

AHL: Translation

Compilers and interpreters

The CPU runs only **machine code** 机器码. A program in a high-level language such as Python or Java must be translated.



A compiler translates the whole program once into machine code; an interpreter translates and runs one line at a time

	Compiler 编译器	**Interpreter 解释器**
when	translates the whole program before it runs	translates and runs one statement at a time
output	an executable file	no separate file
speed of running	fast	slower: translated every time
errors	all reported after compiling; nothing runs until they are fixed	stops at the first error it reaches
source code	not needed by users	needed on every machine

Java uses both: the compiler turns source code into **bytecode** 字节码, which a virtual machine interprets, or compiles **just in time** (JIT) while the program runs. The same bytecode then runs on any machine with the virtual machine.

Practice

1. State the role of (a) the program counter; (b) the ALU; (c) the address bus.
2. Describe the fetch stage of the fetch–decode–execute cycle, naming the registers used.
3. Explain why a GPU trains a machine-learning model faster than a CPU.
4. Explain why cache memory speeds up a computer.
5. Convert (a) 10011101 to denary; (b) 200 to binary; (c) 2F to denary.
6. Write -19 as an 8-bit two's complement number.
7. Calculate the file size, in bytes, of an 800×600 image with a colour depth of 16 bits.
8. Give the truth table for $X = \text{NOT}(A \text{ OR } B)$, and name the single gate that does the same.
9. State **two** functions of an operating system.
10. Processes P1 (4 ms) and P2 (3 ms) arrive together; round robin with a 2 ms slice. State when each finishes.
11. Explain why a keyboard uses interrupts rather than polling.
12. **AHL** Give **one** advantage of a compiler and **one** of an interpreter for a programmer.

Answers.

1. (a) holds the address of the next instruction; (b) does arithmetic and logic operations; (c) carries addresses from the CPU to memory.
2. The address in the PC is copied to the MAR; the instruction at that address is copied along the data bus into the MDR, then to the CIR; the PC is increased by 1.
3. Training repeats the same calculation on huge amounts of data; the GPU's thousands of cores do these calculations in parallel.
4. It holds copies of the most-used data and instructions next to the CPU, so they are read far faster than from RAM.
5. (a) $128+16+8+4+1 = 157$; (b) $200 = 128+64+8$: `11001000`; (c) $2\times16+15 = 47$.
6. 19 = `00010011`; flip: `11101100`; add 1: `11101101`.
7. $800 \times 600 \times 16 = 7\,680\,000$ bits $= 960\,000$ bytes.
8. Rows $00 \to 1$, $01 \to 0$, $10 \to 0$, $11 \to 0$: a NOR gate.
9. E.g. memory management, process management, file management, security.
10. P1 0–2, P2 2–4, P1 4–6 (ends), P2 6–7 (ends): P1 at 6 ms, P2 at 7 ms.
11. Key presses are rare and unpredictable; polling would waste CPU time checking, while an interrupt uses the CPU only when a key is pressed.
12. Compiler: e.g. the finished program runs faster, and the source code need not be shared. Interpreter: e.g. errors are found line by line while testing, and there is no wait to compile.

Exam tips

- Name registers in full the first time (program counter, PC) and give the direction of each bus.
- In the fetch stage, follow the order PC → MAR → memory → MDR → CIR, and say that the PC increments.
- For conversions, show the place values; for two's complement, show flip then add 1.
- File-size answers: show the formula, the numbers and the unit, and divide by 8 for bytes.
- Truth tables for n inputs need 2^n rows in binary order, with a column for every intermediate output.
- Compare CPU and GPU, polling and interrupts, or compiler and interpreter point by point, not in two separate lists.

Networks

International Baccalaureate · IB Diploma · Computer Science · HL

Scope and prerequisites

This handout teaches Theme A2, Networks, for the course first assessed in 2027: network types and devices, protocols and layers, addressing, data transmission and network security. All of it is taught here as SL and HL content.

Prerequisites: bits and bytes from A1 (1 byte = 8 bits).

Network fundamentals

What happens when you open a web page?

You type a web address on your phone at school. In under a second your request crosses the school Wi-Fi, a router, your internet provider and several countries, reaches a server, and the page comes back in hundreds of small pieces that are put together on your screen. Each step follows agreed rules. This handout explains them.



A home Wi-Fi router: it forwards packets between your devices and the internet

Types of network

A **network** 网络 is two or more devices joined to share data and resources.

Type	Covers	Example
LAN 局域网 (local area network)	one site	a school or office
WAN 广域网 (wide area network)	a large area, joining LANs	the internet; a bank's branches
PAN (personal area network)	one person's devices	a phone and earbuds over Bluetooth
VPN 虚拟专用网 (virtual private network)	a private, encrypted link over a public network	working from home on the school network

Network devices

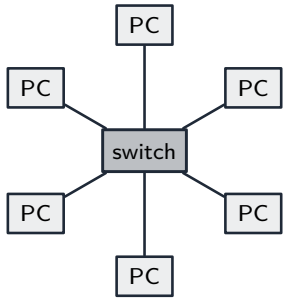
Device	Job
network interface card (NIC)	connects a device to the network; has a unique **MAC address** 物理地址
switch 交换机	joins devices in a LAN and forwards known unicast frames towards the destination port; broadcasts and unknown destinations may be flooded
router 路由器	joins networks and forwards packets between them, choosing a route
wireless access point (WAP)	lets wireless devices join a wired LAN
modem	converts digital data to a signal for the provider's line, and back
firewall 防火墙	checks traffic against rules and blocks what is not allowed
server	provides a service to other devices: files, web pages, e-mail

Client–server and peer-to-peer

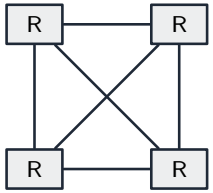
	Client–server 客户机-服务器	Peer-to-peer 对等网络
roles	servers provide services; clients request them	every device is equal and can share
control	central: security, backups, updates in one place	none: each user manages their own device
weakness	the server is a single point of failure; costly	hard to secure and back up
example	school log-ins, a web server	file sharing between two laptops

Topologies

A **topology** 拓扑结构 is the layout of the connections.



star



full mesh

Star: everything through the centre. Mesh: many alternative paths

- **Star**: easy to add devices; one broken cable affects one device; but if the central switch fails, the whole network fails.
- **Mesh**: many paths, so data can route round a failure; the internet's core is a partial mesh. Needs more cables and is costly.

Data travels over **wired** links (copper or fibre-optic cable: fast, reliable, secure) or **wireless** links (Wi-Fi, mobile: convenient, but slower, more interference and easier to intercept).

Protocols and layers

Why protocols?

A **protocol** 协议 is an agreed set of rules for communication: the format of the data, how errors are handled, and the order of messages. Devices from different makers can

talk only because they follow the same protocols.

The TCP/IP model

Networking is split into **layers** 层; each layer does one job and uses the layer below.



Encapsulation: each layer wraps the data from the layer above in its own header

Layer	Job	Protocols
application	the service the user sees	HTTP, HTTPS, DNS, SMTP
transport	splits data into segments; checks delivery and order	TCP, UDP
internet	addresses packets and routes them between networks	IP
link	sends frames over the physical connection	Ethernet, Wi-Fi

Layers let each part change on its own: a web browser works the same over Wi-Fi or cable, because only the link layer differs. Going down, protocols wrap data with control information such as headers and sometimes trailers (**encapsulation** 封装); the receiver removes the relevant information going up.

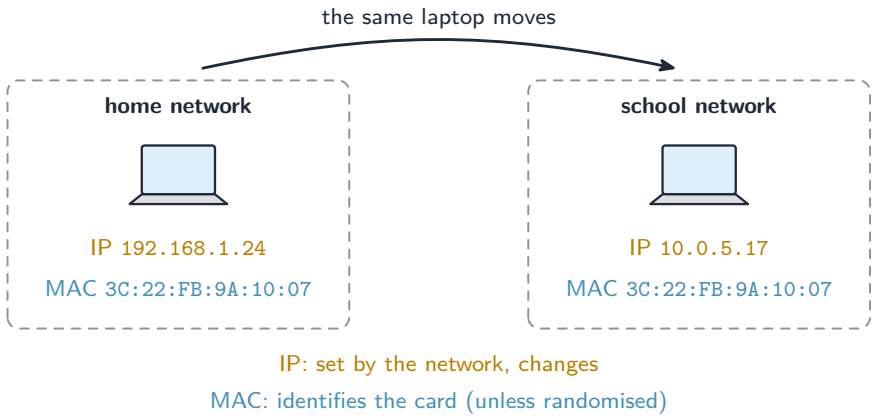
Protocol	Use
HTTP / HTTPS	web pages; HTTPS is encrypted
TCP	reliable delivery: numbered segments, acknowledgements, resends
UDP	datagrams without built-in delivery or ordering guarantees; checksums can detect corruption: video calls, games, live streams
IP	addresses and routes each packet
DHCP 动态主机配置协议	gives a device an IP address when it joins
DNS 域名系统	turns a domain name into an IP address
SMTP / IMAP	sending and fetching e-mail

TCP suits a web page or a file, where every byte must arrive. UDP suits a video call: a late packet is useless, so it is better to drop it than wait.

Addressing

IP and MAC addresses

	IP address 互联网协议地址	MAC address
identifies	a device's position on a network	the network card itself
set by	the network (often DHCP)	often the maker; software may configure or randomise it
changes?	may change with network configuration	may be changed or randomised
example	`192.168.1.24`	`3C:22:FB:9A:10:07`



The network sets the IP address, so it changes when the device moves; the MAC address belongs to the network card.

- **IPv4** uses 32 bits, about 4.3 billion addresses: too few for every device. **IPv6** uses 128 bits, written in hex such as `2001:db8::1`.
- A **private** address (such as `192.168.x.x`) works only inside a LAN; the router shares one **public** address with the whole LAN.
- A **static** address never changes (servers need one so they can be found); a **dynamic** address is lent by DHCP.

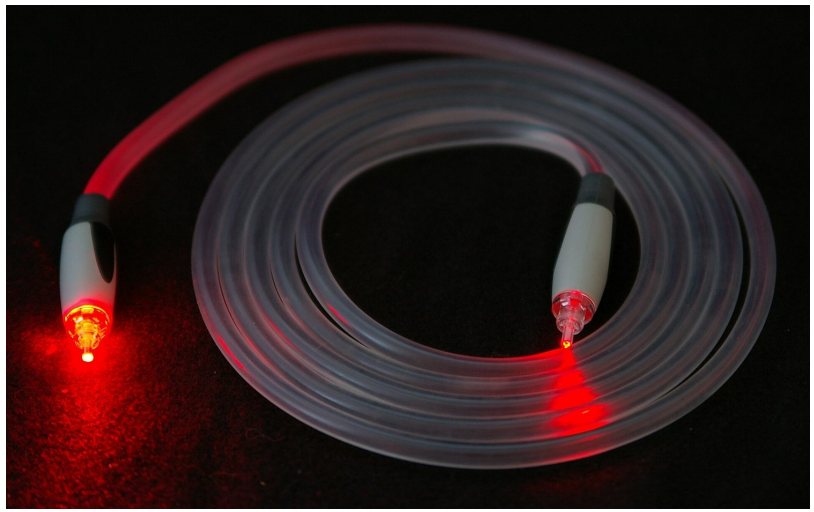
DNS: finding a server

1. You type `www.example.org`.
2. Your device asks a DNS server for its IP address.
3. If that server does not know, it asks other DNS servers in turn.
4. The IP address comes back, and your browser connects to it. The answer is stored (cached) to save time next visit.

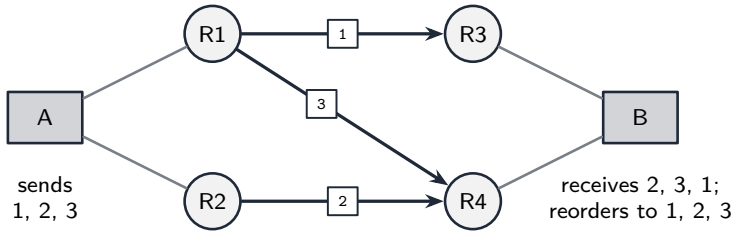
Data transmission

Packet switching

The internet uses **packet switching** 分组交换: a message is split into **packets** 数据包, each sent separately.



A fibre-optic cable carries data as pulses of light along thin glass fibres



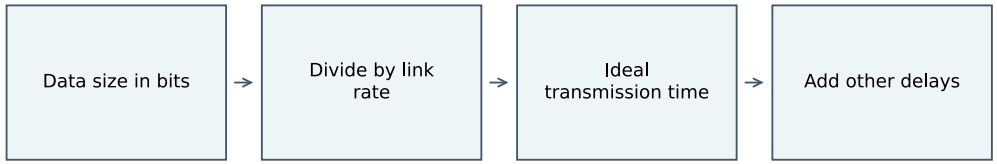
Packets can take different routes and arrive out of order

1. The data is split into packets. Each has a header with the source and destination IP addresses and a sequence number.
2. Each router reads the destination and forwards the packet along the best route it knows at that moment.
3. Packets may arrive out of order; TCP uses the sequence numbers to put them back, and asks again for any that are missing.

Advantages: lines are shared by many users, and packets route round a failed or busy link.

Speed: bandwidth, latency, throughput

- **Bandwidth** 带宽: the most data a link can carry per second, in bits per second.
- **Latency** 延迟: the delay before data starts to arrive, in milliseconds.
- **Throughput** 吞吐量: the data actually delivered per second; always at or below the bandwidth, because of other users, errors and resends.



The ideal transmission time; real transfers take longer

$$\text{time} = \frac{\text{data size in bits}}{\text{rate in bits per second}}$$

$$\text{time} = \frac{12 \times 10^6 \times 8 \text{ bits}}{8 \times 10^6 \text{ bits/s}} = 12 \text{ s}$$

for a 12 MB file on an 8 Mbit/s link, before latency and overhead are added. Note the units: file sizes are usually in **bytes**, link speeds in **bits** per second.

Network security

Threats

Threat	What happens
malware	harmful software, such as a virus or ransomware, damages or locks data
phishing	a fake message tricks a user into giving away a password
interception	data is read as it travels, for example on open Wi-Fi
unauthorised access	an attacker uses a weak or stolen password
denial of service	a server is flooded with requests until real users cannot reach it



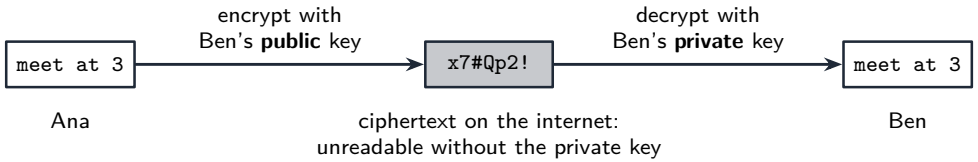
A fingerprint reader checks "something you are" – a feature of the person, not a password

Countermeasures

- **Firewalls** check each packet against rules: allowed addresses, ports and protocols.
- **Authentication** 身份验证 checks who a user is: something they know (password), have (phone) or are (fingerprint). **Multi-factor authentication** uses two or more. **Authorization** 授权 then decides what that user may do.
- Software updates close known weaknesses; backups allow recovery from ransomware; user training reduces phishing.
- **Encryption** 加密 makes intercepted data unreadable without the key.

Encryption

	Symmetric encryption 对称加密	Asymmetric encryption 非对称加密
keys	one shared secret key	a public key and a private key
speed	fast	slower
problem	the key must be shared safely	the public key must be authenticated and the private key protected



Anyone can lock with the public key; only the private key unlocks

HTTPS uses both: asymmetric encryption to agree a secret key safely, then fast symmetric encryption for the rest of the session. A **digital certificate** 数字证书, signed

by a trusted authority, binds a domain or identity to a public key; the browser checks the trusted signature chain, domain and validity. This authenticates the connection but does not guarantee that a site is honest.

Practice

1. State the difference between a LAN and a WAN, with an example of each.
2. State the job of (a) a switch; (b) a router; (c) a DHCP server.
3. Give **one** advantage and **one** disadvantage of a star topology.
4. Explain why a school uses a client–server network rather than peer-to-peer.
5. Explain why networking is organised in layers.
6. State whether TCP or UDP suits (a) downloading an app; (b) an online multiplayer game. Explain (b).
7. Describe what DNS does when you visit a website.
8. Explain why IPv6 was introduced.
9. Describe how a file is sent across the internet by packet switching.
10. A 30 MB video is sent over a 20 Mbit/s link. Calculate the ideal transmission time, and state why the real time is longer.
11. Explain the difference between symmetric and asymmetric encryption.
12. A user receives an e-mail asking them to "confirm your password" at a link. Name the threat and give **two** ways to reduce it.

Answers.

1. A LAN covers one site, such as a school; a WAN covers a large area and joins LANs, such as the internet.
2. (a) joins devices in a LAN and sends data only to the device it is for; (b) forwards packets between networks along a route; (c) gives each device an IP address when it joins.
3. Advantage: e.g. one cable failure affects only one device; easy to add devices. Disadvantage: if the central switch fails, the whole network fails.
4. Security, backups and software updates are managed centrally on the server; users can log in from any computer and find their files.
5. Each layer does one job with its own protocols, so one layer can change, such as cable to Wi-Fi, without changing the others; makers can build to one layer's standard.
6. (a) TCP; (b) UDP: it is fast with no resends, and a late game update is useless, so it is better to drop it than wait.
7. The browser asks a DNS server for the IP address of the domain name; if needed, other DNS servers are asked; the IP address is returned and the browser connects to it.
8. IPv4's 32 bits give only about 4.3 billion addresses, too few for every device; IPv6's 128 bits give far more.
9. The file is split into packets with headers (source, destination, sequence number); routers forward each packet by the best route; packets may arrive out of order and are reassembled by sequence number; missing ones are resent.
10. $30 \times 10^6 \times 8 = 240 \times 10^6$ bits; $240 \times 10^6 \div 20 \times 10^6 = 12$ s. Latency, other traffic, headers and resends lower the throughput.
11. Symmetric uses one shared key to encrypt and decrypt, which must be passed on safely; asymmetric encrypts with a public key and decrypts with a different, private key.
12. Phishing. E.g. train users to check links and senders; spam filters; multi-factor authentication so a stolen password alone is not enough.

Exam tips

- Name each device by its **job**, not just its name: "a router forwards packets between networks".
- In TCP/UDP answers, link the choice to the scenario: reliability for files, speed for live data.
- Describe packet switching in order: split, header, route, reassemble, resend missing.
- Keep bandwidth (the maximum), throughput (what you actually get) and latency (the delay) apart.
- Convert bytes to bits ($\times$ 8) before dividing by a speed in bits per second.
- For encryption, state which key encrypts and which decrypts.

Databases

International Baccalaureate · IB Diploma · Computer Science · HL

Scope and prerequisites

This handout teaches Theme A3, Databases, for the course first assessed in 2027: relational databases, design and normalisation, and SQL. Sections marked **AHL** are for Higher Level only.

Prerequisites: data types from B2 (integer, string, Boolean, real).

Why use a database?

One spreadsheet, many problems

A school library keeps loans in one spreadsheet: each row has the student's name, form, the book's title and author, and the date. When a student changes form, dozens of rows must be edited, and some are missed. Two rows now disagree about the same student. A database is designed to stop this.



A library card catalogue is a database on paper – records you can search and sort by index

Database fundamentals

A **database** 数据库 is an organised collection of data. A **database management system** 数据库管理系统 (DBMS) is the software that stores it, runs queries, controls access and makes backups.

Problem with flat files	How a relational database helps
data redundancy 数据冗余: the same data stored many times	each fact stored once
inconsistency 数据不一致: copies disagree	one copy, so nothing to disagree
hard to search and combine	SQL queries join tables
no control over who sees what	user accounts and permissions

Tables, records and keys

A **relational database** 关系数据库 stores data in **tables** 表 (relations). Each **record** 记录 (row) is one item; each **field** 字段 (column, attribute) is one property.

Key	Meaning	Example
primary key 主键	a field whose value is unique for every record	`student_id`
foreign key 外键	a field that holds the primary key of another table, linking them	`student_id` in Loan
composite key 复合键	a primary key made of two or more fields	(`order_id`, `item_id`)

Referential integrity 参照完整性 requires a non-null foreign key to match a referenced key (a primary or other suitable unique key); nullable foreign keys may represent no specified reference: a loan cannot refer to a student who does not exist.

Customer

customer_id PK	name
C1	Ana
C2	Ben
C3	Chen
C4	Dee

arrow: each FK value points to a PK

Order

order_id	customer_id FK
O1	C1
O2	C1
O3	C3
O4	C3
O5	C3
O6	C4

Join Customer to Order

inner join: matched pairs only
2 + 0 + 3 + 1 = 6 rows

left join: every customer kept
6 + 1 row for C2 (NULL order) = 7 rows

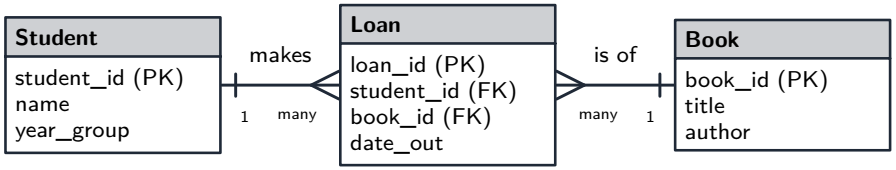
A foreign key value must match an existing primary key; the join type decides which unmatched rows survive.

A foreign key links each order to its customer

Database design

Entities and relationships

An **entity** 实体 is a thing we store data about: a student, a book. An **entity-relationship diagram** 实体关系图 (ERD) shows the entities and how they are related.



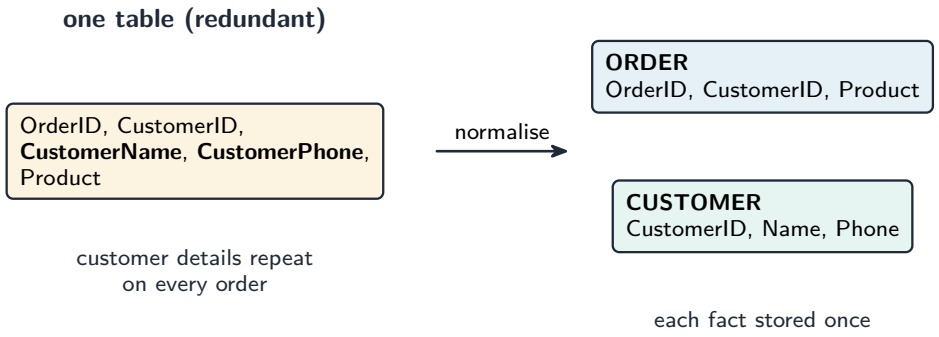
One student makes many loans; each loan is of one book

Relationship	Meaning	Example
one-to-one	each A has one B	a student and their locker
one-to-many	one A has many B	one student, many loans
many-to-many	many A, many B	students and books

A many-to-many relationship cannot be stored directly: it is split into two one-to-many relationships through a **link table** 连接表, here `Loan`.

Normalisation

Normalisation 规范化 organises tables to remove redundancy. Start from one unnormalised table:



Normalisation removes redundancy by splitting repeated data into its own table

order_id	customer	phone	items
501	Li Na	1381	pen, ruler
502	Li Na	1381	pen

Form	Rule	Result
first normal form 第一范式 (1NF)	every field holds one atomic value; no repeating groups	one row per item: key (`order_id`, `item`)
second normal form 第二范式 (2NF)	1NF, and no non-key field depends on only part of a candidate key	`customer` depends only on `order_id`: move it to an Order table; `price` depends only on `item`: move it to an Item table
third normal form 第三范式 (3NF)	2NF, with no transitive dependency of a non-key field on a candidate key	`phone` depends on the customer, not the order: move it to a Customer table

Result in 3NF:

- Customer(**customer_id**, name, phone)
- Order(**order_id**, customer_id)
- OrderLine(**order_id**, **item_id**, quantity)
- Item(**item_id**, description, price)

Now Li Na's phone number is stored once. A short test for 3NF: every non-key field depends on "the key, the whole key, and nothing but the key".

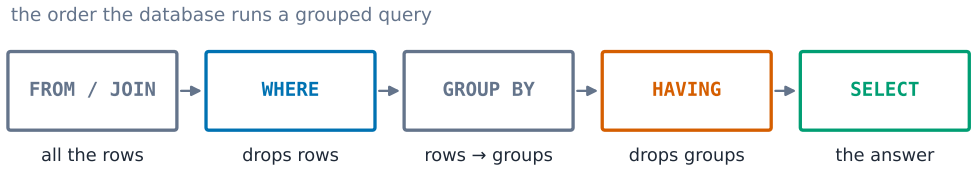
Choosing data types

Each field needs a type, such as `INTEGER`, `VARCHAR(50)` (text up to 50 characters), `DATE`, `BOOLEAN` or `DECIMAL(6,2)`. Phone numbers are stored as text, not integers: they can start with 0 and are never added up.

SQL

Querying data

SQL 结构化查询语言 (Structured Query Language) defines and queries relational databases. The examples use these tables:



COUNT(*) only exists after GROUP BY, so "more than 20 students" must be a HAVING filter.

The order a grouped query runs in: WHERE before the groups exist, HAVING after

Student

student_id	name	year_group
1	Ana	11
2	Ben	12
3	Chen	11

Book

book_id	title	author
10	Dune	Herbert
11	Emma	Austen
12	Holes	Sachar

Loan

loan_id	student_id	book_id	date_out
100	1	10	2026-09-01
101	1	12	2026-09-05
102	3	11	2026-09-06

```sql
SELECT name
FROM Student
WHERE year_group = 11
ORDER BY name DESC;
```

Result: Chen, Ana. `WHERE` filters rows; `ORDER BY … DESC` sorts Z to A.

```sql
SELECT Student.name, Book.title
FROM Loan
JOIN Student ON Loan.student_id = Student.student_id
JOIN Book ON Loan.book_id = Book.book_id;
```

Result: (Ana, Dune), (Ana, Holes), (Chen, Emma). A **join** 连接 combines tables by matching the foreign key with the primary key. Ben has no loans, so he does not appear.

```sql
SELECT student_id, COUNT(*) AS loans
FROM Loan
GROUP BY student_id;
```

Result: (1, 2), (3, 1). **Aggregate functions** 聚合函数 such as `COUNT`, `SUM`, `AVG`, `MAX` and `MIN` work on groups of rows.

Clause	Use
`WHERE`	filter rows: `=`, `<>`, `>`, `LIKE 'D%'`, `AND`, `OR`
`ORDER BY`	sort, `ASC` or `DESC`
`GROUP BY`	make groups for aggregate functions
`HAVING`	filter groups: `HAVING COUNT(*) > 1`

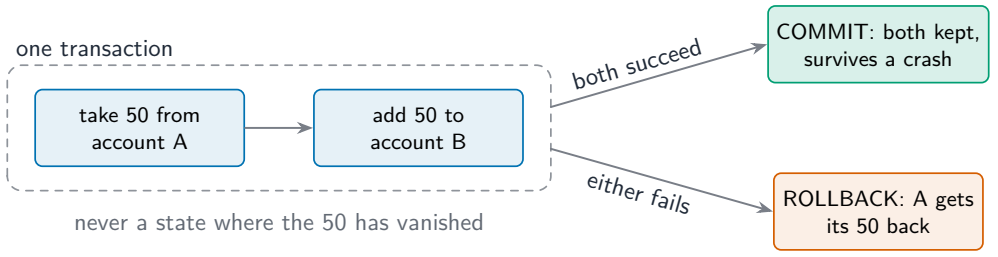
Changing data

```sql
INSERT INTO Book (book_id, title, author) VALUES (13, 'Wonder',
↪  'Palacio');
UPDATE Student SET year_group = 12 WHERE student_id = 3;
DELETE FROM Loan WHERE loan_id = 100;
```

An `UPDATE` or `DELETE` without `WHERE` changes **every** row.

Transactions

A **transaction** 事务 is a group of operations that must all succeed or all fail, such as moving money: take it from one account and add it to another. Transactions follow **ACID**:



All of it, or none of it

Property	Meaning
atomicity	all of it happens, or none of it
consistency	the database moves from one valid state to another
isolation	transactions running at once do not interfere
durability	once committed, a change survives a crash

AHL: Alternative databases and data warehouses

Beyond relational

Relational databases suit structured data with fixed fields. Other models suit other data:



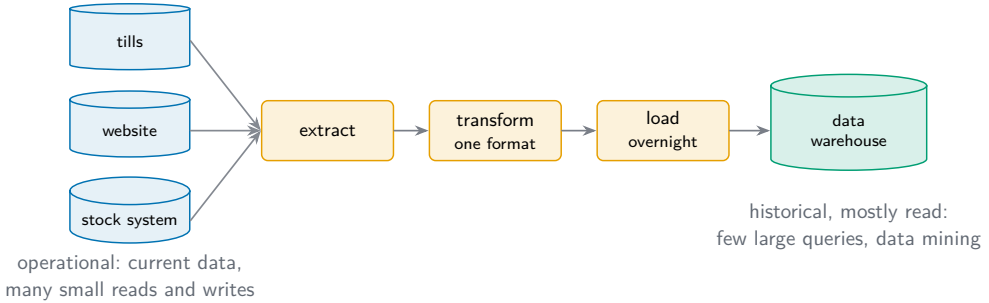
Rows of servers in a data centre, like those used for massively parallel computing

Model	Stores	Suits
document	flexible records, such as JSON documents	product catalogues with different fields
key-value	a value for each key	shopping carts, session data
graph	nodes and the links between them	social networks, route finding
spatial	locations and shapes	maps and delivery zones

These **NoSQL** databases scale across many servers easily, but give up some of the checks, such as referential integrity, that a relational DBMS enforces.

Data warehouses

A **data warehouse** 数据仓库 collects historical data from many sources into one store for analysis. Data is extracted, cleaned into one format, and loaded (**ETL**: extract, transform, load), usually overnight.



Extract, transform, load

	Operational database	Data warehouse
purpose	run the business now	analyse trends over time
data	current, changes constantly	historical, mostly read-only
queries	many small reads and writes	few large, complex queries

Data mining looks for patterns in the warehouse, such as which products are bought together. Because the data is about real people, it must be kept secure and used within the law.

Practice

1. State **two** problems of storing a school's data in one flat file.
2. Define primary key and foreign key, with an example from the library tables.
3. Explain why the many-to-many relationship between students and books needs the Loan table.
4. A table has fields (`booking_id`, `room`, `room_capacity`, `date`). Explain why it is not in 3NF and normalise it.
5. Write SQL to list the titles of all books by Austen.
6. Write SQL to list the names of students in year 11 with the number of loans each has made.
7. State the result of `SELECT COUNT(*) FROM Loan WHERE book_id > 10;` for the tables above.
8. Write SQL to change the author of book 12 to 'L. Sachar'.
9. Explain atomicity using a bank transfer.

10. **AHL** Suggest a suitable database model for a social network, and explain your choice.
11. **AHL** Explain **two** differences between a data warehouse and an operational database.

Answers.

1. E.g. data redundancy, the same student stored in many rows; inconsistency when one copy is updated and others are not; hard to search or control access.
2. A primary key uniquely identifies each record, such as `book_id` in Book. A foreign key is a field holding another table's primary key, such as `book_id` in Loan.
3. One student borrows many books and one book is borrowed by many students. A relational table cannot store this directly; the link table turns it into two one-to-many relationships, each loan linking one student and one book.
4. `room_capacity` depends on `room`, a non-key field: a transitive dependency. Split into Booking(**booking_id**, room, date) and Room(**room**, room_capacity).
5. `SELECT title FROM Book WHERE author = 'Austen';`
6. `SELECT Student.name, COUNT(*) FROM Student JOIN Loan ON Student.student_id = Loan.student_id WHERE Student.year_group = 11 GROUP BY Student.name;` giving (Ana, 2), (Chen, 1).
7. Loans 101 (book 12) and 102 (book 11): **2**.
8. `UPDATE Book SET author = 'L. Sachar' WHERE book_id = 12;`
9. The money must both leave one account and arrive in the other; if the system fails part-way, the whole transaction is undone, so money is never lost or created.
10. A graph database: people are nodes and friendships are links, so queries such as "friends of friends" follow links directly.
11. E.g. a warehouse holds historical data from many sources, an operational database current data; a warehouse is mostly read for large analysis queries, an operational database handles many small updates.

Exam tips

- Keys: say the primary key is **unique** for each record; a foreign key **links** to a primary key in another table.
- In normalisation answers, name the dependency you are removing, then show the new tables with their keys.
- Write SQL keywords in capitals, put text in single quotes, and end each statement with a semicolon.
- In a join, always give the `ON` condition matching the foreign key to the primary key.
- Check `UPDATE` and `DELETE` statements for a `WHERE` clause.
- Learn ACID as four words with one-line meanings, and illustrate with a transfer.

Machine learning

International Baccalaureate · IB Diploma · Computer Science · HL

Scope and prerequisites

This handout teaches Theme A4, Machine learning, for the course first assessed in 2027: what machine learning is, its main types, how models are judged, and its ethical issues. Sections marked **AHL** are for Higher Level only.

Prerequisites: reading a scatter graph and a percentage. The datasets here are small and fictional.

What is machine learning?

A program that learns from examples

A spam filter is not given a list of rules such as "block any e-mail containing 'prize'". Instead it is shown thousands of e-mails labelled "spam" or "not spam" and finds the patterns itself. When spammers change their wording, it is retrained on new examples. This is **machine learning** 机器学习 (ML): a program improves at a task by learning patterns from data rather than following rules written by a programmer.



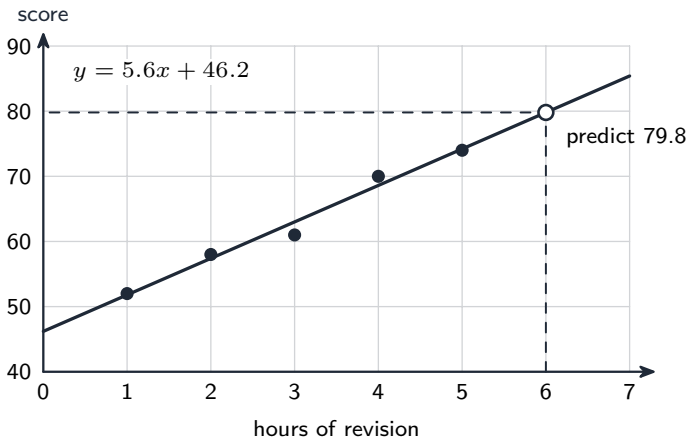
A self-driving car uses cameras and lidar sensors to see the road around it

ML is one part of **artificial intelligence** 人工智能 (AI), the wider field of making computers do tasks that seem to need human intelligence. **Deep learning** 深度学习 is a kind of ML that uses neural networks with many layers.

Types of machine learning

Type	Learns from	Typical task	Example
supervised learning 监督学习	labelled data: inputs with the right answers	classify, predict a number	spam or not; a house price
unsupervised learning 无监督学习	unlabelled data	find groups or patterns	grouping shoppers by habits
reinforcement learning 强化学习	rewards and penalties from trial and error	choose actions	a robot learning to walk; a game player

Supervised learning splits into **classification** 分类 (the answer is a category: cat or dog) and **regression** 回归 (the answer is a number: a score or a price).



Supervised learning: the model learns a line from labelled examples, then predicts a new case

Training and testing

1. Collect and label data.
2. Split it: most for **training** 训练 the model, the rest held back as **test data** 测试数据.
3. Train the model on the training data.
4. Measure it on the test data, which it has never seen.

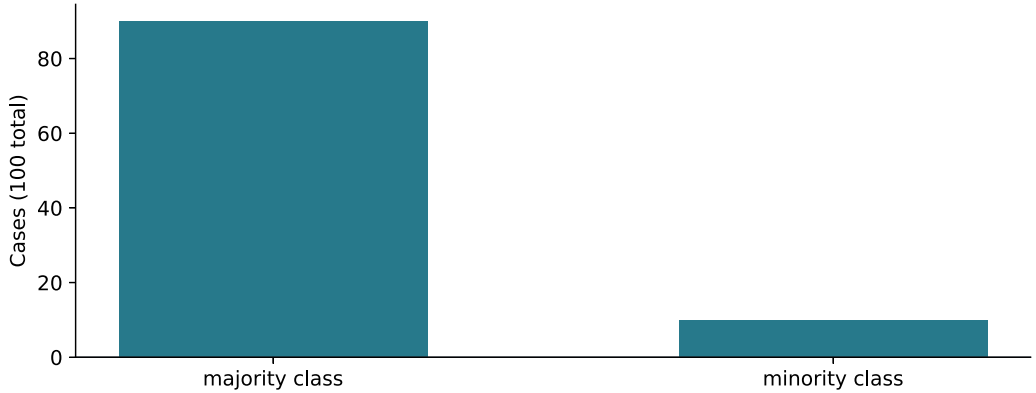
Testing on unseen data shows whether the model has learned the general pattern. A model that memorises its training data but fails on new data is **overfitting** 过拟合. A model too simple to capture the pattern is **underfitting**.

Measuring a classifier

Accuracy 准确率 is the share of predictions that are correct:

$$\text{accuracy} = \frac{\text{correct predictions}}{\text{total predictions}} \times 100\%$$

A model correct on 90 of 100 test cases has 90% accuracy. Accuracy can mislead when the classes are unbalanced:



With 90 of 100 cases in one class, always predicting that class scores 90% yet misses every minority case

A "model" that always answers "healthy" for a disease affecting 10% of patients scores 90% and finds no one who is ill. So check how the model does on each class, not just overall.

Hardware for machine learning

Training repeats the same arithmetic on huge amounts of data, so it runs on **GPUs** and special chips with thousands of cores working in parallel, often rented in the cloud. Using a trained model (**inference**) needs far less power and can run on a phone.

Ethics of machine learning

Who is affected?

Issue	What can go wrong	Example
bias 偏见	training data under-represents a group, so the model works worse for them	face recognition less accurate for darker skin
privacy	personal data collected or used without consent	photos scraped from social media
transparency	no one can explain why the model decided	a loan refused with no reason given
accountability	unclear who is responsible for a harmful decision	a self-driving car accident
jobs	tasks automated	checkout staff, translators
misuse	realistic fake images, audio or text	deepfakes spread as news



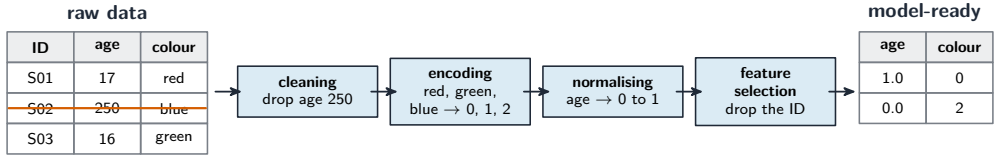
A CCTV control room: surveillance systems trade safety gains against privacy concerns

Ways to reduce harm: train on data that represents everyone the model affects; test accuracy separately for each group; keep a human in the loop for serious decisions; collect data with consent and remove what is not needed; explain decisions to the people affected.

AHL: Preparing data

Data preprocessing

Real data is messy. **Data preprocessing** 数据预处理 prepares it before training:



Data leakage: never keep a feature that would not be known when the prediction is made

Clean, encode, normalise and select features before training, and keep out any feature that leaks the answer

Step	What it does	Example
cleaning	fix or remove missing, duplicate and wrong values	delete a row with age 250
encoding	turn categories into numbers	red, green, blue → 0, 1, 2
normalisation (scaling)	put features on a similar range, such as 0 to 1	income in thousands and age in years
feature selection 特征选择	keep only the inputs that help prediction	drop a student's ID number
dimensionality reduction	combine many features into fewer	thousands of pixel values → a few patterns

A **feature** 特征 is one input variable, such as hours of revision; the **label** 标签 is the answer to be predicted, such as the score.

Watch for **data leakage** 数据泄漏: a feature that would not be known when the prediction is made, such as using "days in hospital" to predict whether a patient will be admitted. The model then looks excellent in testing and fails in real use.

AHL: Machine learning algorithms

Linear regression

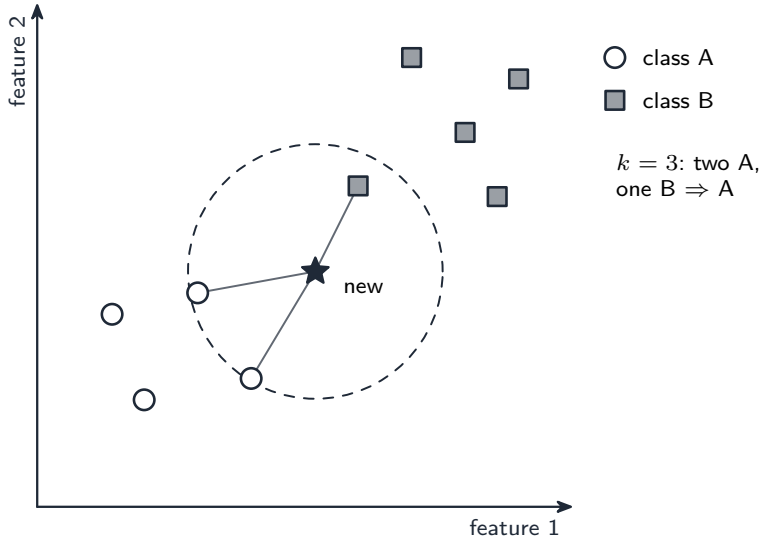
Linear regression 线性回归 fits the straight line $y = mx + c$ that best matches the data, then uses it to predict. For the revision data above, the best-fit line is $y = 5.6x + 46.2$:

$$y = 5.6 \times 6 + 46.2 = 79.8$$

so 6 hours of revision predicts a score of about 80. Predicting far outside the data, such as for 30 hours, is unreliable.

k-nearest neighbours

k-nearest neighbours 近邻算法 (k-NN) classifies a new item by the classes of the k training items closest to it.



With $k = 3$ the new point is classed A, although its single nearest neighbour is B

1. Work out the distance from the new item to every training item.
2. Take the k nearest.
3. Give the new item the most common class among them.

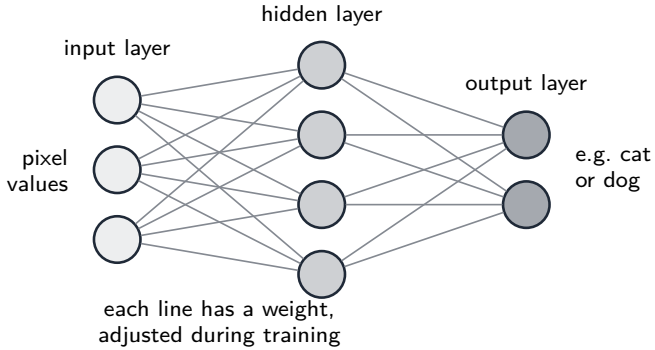
An odd k avoids ties between two classes. k-NN needs no training, but each prediction compares with every stored item, so it is slow on large datasets.

Decision trees and clustering

- A **decision tree** 决策树 asks a sequence of yes/no questions about the features, such as "Is the e-mail from a known contact?", ending at a leaf that gives the class. It is easy for people to follow and explain.
- **k-means clustering** 均值聚类 is unsupervised: choose k centres, put each item in the cluster of its nearest centre, move each centre to the mean of its cluster, and repeat until nothing changes. Used to group customers or documents.

Neural networks

An **artificial neural network** 人工神经网络 is made of layers of simple units (nodes). Each connection has a **weight** 权重; each node adds up its weighted inputs and passes the result on.



Training adjusts the weights until the outputs match the labels

Training compares the network's output with the right answer and adjusts every weight a little to reduce the error, over many thousands of examples. A **convolutional neural network** (CNN) has layers that scan small patches of an image to find edges, shapes and then objects, and is used for image recognition.

Precision and recall

For a classifier that flags positives (such as "has the disease"), compare predictions with reality in a **confusion matrix** 混淆矩阵:

	Actually positive	**Actually negative**
predicted positive	true positive (TP) 4	false positive (FP) 10
predicted negative	false negative (FN) 1	true negative (TN) 85

$$\text{accuracy} = \frac{TP + TN}{\text{total}} = \frac{4 + 85}{100} = 0.89$$

$$\text{precision} = \frac{TP}{TP + FP} = \frac{4}{14} \approx 0.29 \qquad \text{recall} = \frac{TP}{TP + FN} = \frac{4}{5} = 0.80$$

Recall asks "of the real positives, how many did we find?"; precision asks "of those we flagged, how many were right?". A disease screen wants high recall, so few sick patients are missed; a spam filter wants high precision, so real e-mails are not lost.

Practice

1. Define machine learning, and explain how it differs from a program with written rules.
2. State the type of ML for each: (a) grouping songs by similarity with no labels; (b) predicting tomorrow's temperature from past records; (c) a program learning to play chess by playing itself.
3. State whether each is classification or regression: (a) a house price; (b) whether a photo shows a cat.
4. Explain why a model is tested on data it was not trained on.
5. A model makes 72 correct predictions out of 80. Calculate its accuracy.
6. A model for a rare disease (2% of patients) has 98% accuracy. Explain why this may be useless.
7. Explain **two** ethical issues of using ML to shortlist job applicants, and suggest a way to reduce each.
8. **AHL** Explain what data leakage is, with an example.
9. **AHL** Use $y = 5.6x + 46.2$ to predict the score for 3.5 hours of revision.
10. **AHL** In the k-NN figure, state the class given to the new point with $k = 1$ and with $k = 3$.
11. **AHL** A classifier has TP = 30, FP = 10, FN = 20, TN = 140. Calculate its accuracy, precision and recall.

Answers.

1. ML is a program that improves at a task by learning patterns from data. A rule-based program follows rules a programmer wrote; an ML model finds its own rules from examples.
2. (a) unsupervised; (b) supervised (regression); (c) reinforcement.
3. (a) regression; (b) classification.
4. To check that it has learned the general pattern rather than memorised the training examples (overfitting); unseen data shows how it will work in real use.
5. $\dfrac{72}{80} \times 100\% = 90\%$.
6. Always predicting "healthy" would also score 98% while finding no sick patients; the model must be judged on how many real cases it finds.
7. E.g. bias: past hiring data may favour one group, so the model repeats it; train on balanced data and test results for each group. Transparency: applicants cannot learn why they were rejected; use an explainable model and keep a human reviewer.
8. Using information that would not be available when the prediction is made, so test results are falsely good; e.g. predicting who will fail a course using their final exam mark.
9. $5.6 \times 3.5 + 46.2 = 19.6 + 46.2 = 65.8$.
10. $k = 1$: the nearest point is B, so **B**; $k = 3$: two A and one B, so **A**.
11. Accuracy $= \dfrac{30 + 140}{200} = 0.85$; precision $= \dfrac{30}{40} = 0.75$; recall $= \dfrac{30}{50} = 0.60$.

Exam tips

- Name the type of learning from the data: labels → supervised; no labels → unsupervised; rewards → reinforcement.
- Say **why** test data is kept separate: to measure performance on unseen data and detect overfitting.
- Do not trust accuracy alone with unbalanced classes; mention per-class results.
- Ethics answers: name the issue, explain the harm **in the scenario**, then give a specific way to reduce it.
- **AHL** Show the formula before substituting for regression predictions, precision and recall.
- **AHL** For k-NN, state k, the neighbours found and the majority class.

Computational thinking

International Baccalaureate · IB Diploma · Computer Science · HL

Scope and prerequisites

This handout teaches Theme B1, Computational thinking, for the course first assessed in 2027. All of it is SL and HL content.

Prerequisites: none beyond basic arithmetic. Theme B2 then turns these ideas into Python code.

Thinking like a computer scientist

How do you plan a whole app?

Your school wants a canteen app: students order lunch on a phone, pay from a card balance, and the kitchen sees the orders. Nobody can write that in one go. Computational thinking is the set of habits that turns a big, vague problem into small, exact steps a computer can follow.

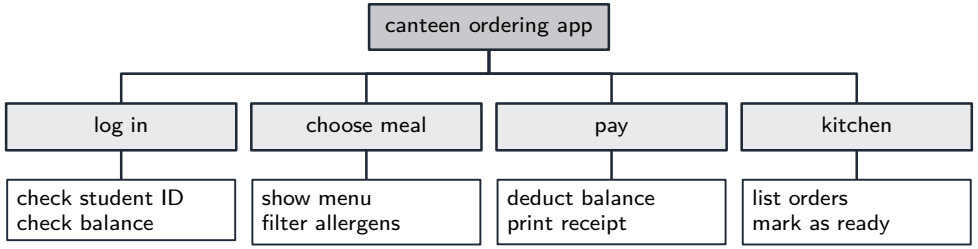


Computational thinking breaks a big problem into smaller, easier parts – like solving a jigsaw

Four approaches

Approach	What you do	Canteen example
decomposition 分解	split the problem into smaller sub-problems	log in, choose meal, pay, kitchen
pattern recognition 模式识别	spot what sub-problems have in common	every screen shows a list and takes a choice
abstraction 抽象	keep the details that matter, drop the rest	a meal is a name, a price and allergens, not its photo or recipe
algorithmic design 算法设计	write exact steps for each sub-problem	the steps to deduct a price from a balance

Decomposition



Each box is small enough to design, write and test on its own

Decomposition lets a team share the work, test each part separately, and reuse parts: the "check balance" step serves both logging in and paying.

Pattern recognition and abstraction

Pattern recognition finds the same structure in different places, so one solution can be reused. Calculating a total for a lunch order, a sports-kit order and a trip payment is the same pattern: add the prices, then deduct from a balance.

Abstraction builds a **model** 模型 that leaves out what does not affect the problem. A map of the school for fire drills shows exits and corridors, not wall colours. Too much abstraction loses something that matters: a canteen model with no allergens would be dangerous.

Algorithms

What makes an algorithm

An **algorithm** 算法 is a finite sequence of exact steps that solves a problem. A good algorithm is:

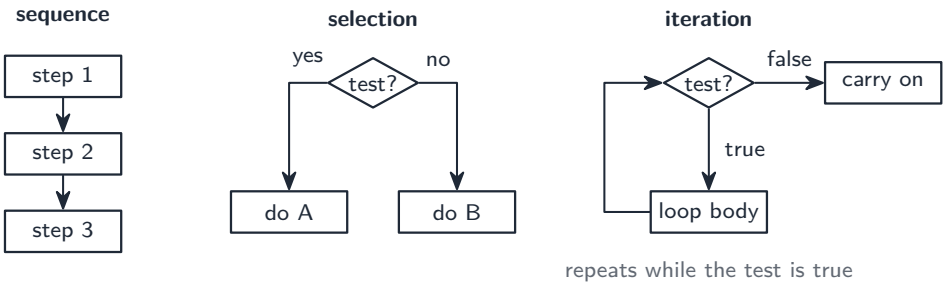
- **unambiguous**: each step has one meaning;
- **finite**: it stops after a limited number of steps;
- **correct**: it gives the right output for every valid input;
- defined by its **inputs** 输入 and **outputs** 输出.

"Add some salt" is not an algorithm step; "add 2 g of salt" is.

Sequence, selection and iteration

Every algorithm is built from three constructs:

Construct	Meaning	Flowchart shape
sequence 顺序	steps run one after another	arrows joining boxes
selection 选择	a decision chooses the path	a diamond with yes and no
iteration 迭代	steps repeat	an arrow looping back

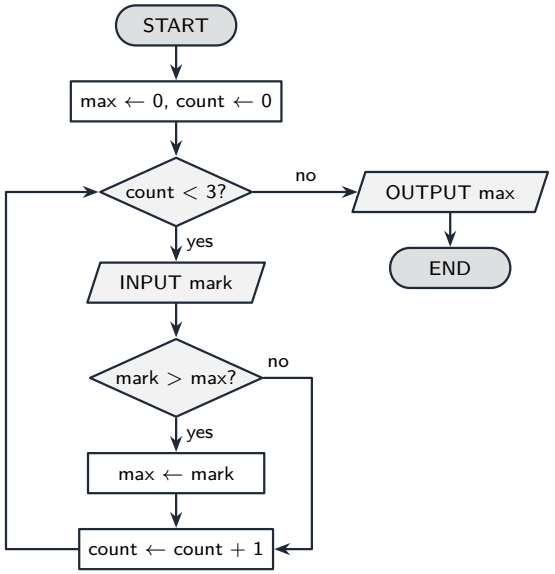


Sequence, selection and iteration: every algorithm is built from these three

Flowcharts

A **flowchart** 流程图 shows an algorithm as shapes joined by arrows.

Symbol	Use
rounded rectangle (terminator)	START and END
parallelogram	INPUT or OUTPUT
rectangle	a process, such as an assignment
diamond	a decision with yes and no exits



Find the largest of three marks: a loop with a decision inside it

`max ← 0` means "store 0 in max": the arrow is assignment.

Tracing a flowchart

A **trace table** 跟踪表 records each variable after each step. For the inputs 62, 85, 71:

`count`	`count < 3?`	`mark`	`mark > max?`	`max`	Output
0				0	
0	yes	62	yes	62	
1	yes	85	yes	85	
2	yes	71	no	85	
3	no				85

The algorithm outputs **85**. Note the trap: if every mark could be negative, starting `max` at 0 gives a wrong answer; starting it at the first mark is safer.

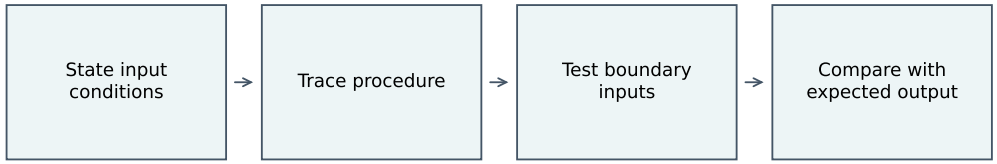
Pseudocode

The same algorithm written as **pseudocode** 伪代码, structured English that is close to code but not tied to one language:

```
max ← 0
count ← 0
loop while count < 3
    input MARK
    if MARK > max then
        max ← MARK
    end if
    count ← count + 1
end loop
output max
```

Checking that an algorithm works

Test data



Correctness evidence: state the inputs, trace, test the boundaries, compare with the expected output

Choose tests from the problem's **specification** 规格说明, not just the cases you expect. For a mark from 0 to 100:

Type	Purpose	Examples
normal data 正常数据	typical valid data	45, 72
boundary data 边界数据	the edges of the valid range, and just outside	0, 100, -1, 101
invalid data 无效数据	data that must be rejected	-20, 150, "ten"

One passing test proves only that the algorithm works for that input. When a test fails, decide whether the **algorithm** is wrong, the **implementation** (the code) does not match the algorithm, or the **requirement** never covered that case.

Efficiency

Two correct algorithms can differ greatly in speed. Finding a name in a register of 1000 by checking each in turn takes up to 1000 steps; halving a sorted register each time (binary search, B2) takes about 10. Theme B2 measures this with Big O.

Practice

1. Define decomposition, and decompose "organise a school sports day" into four sub-problems.
2. Explain abstraction using a metro map as the example.
3. Give **one** example of pattern recognition in a school app.
4. Explain why "stir until it looks ready" is not a good algorithm step.
5. Name the flowchart symbol used for (a) a decision; (b) OUTPUT; (c) START.
6. Trace the flowchart above for the inputs 40, 40, 90 and state the output.
7. Trace the flowchart for the inputs -5, -8, -2. State the output and explain the error.
8. A ticket costs 12 for ages 0 to 17 and 20 for ages 18 and over; ages above 120 are rejected. Give two normal, two boundary and two invalid test values.
9. Draw a flowchart that inputs numbers until 0 is entered, then outputs how many numbers were entered, not counting the 0.

Answers.

1. Decomposition is splitting a problem into smaller sub-problems that can be solved separately. E.g. events and timetable, registering students, scoring and results, equipment and staff.
2. A metro map keeps what a traveller needs, the stations, lines and connections, and drops real distances, streets and buildings.
3. E.g. every form in the app (sign-up, booking, feedback) checks that required fields are filled, so one checking routine can be reused.
4. It is ambiguous: "looks ready" means different things to different people, so the step does not have one exact meaning.
5. (a) diamond; (b) parallelogram; (c) rounded rectangle (terminator).
6. `max` becomes 40 after the first input; the second 40 is not greater than 40, so `max` stays 40; 90 is greater, so `max` becomes 90. Output **90**.
7. No mark is greater than 0, so `max` stays 0. Output **0**, which is not one of the marks: `max` should start at the first input, not 0.
8. Normal: e.g. 10, 35. Boundary: 17 and 18 (the price change), or 0 and 120. Invalid: e.g. -1, 121.

9. START → `count ← 0` → INPUT `n` → decision `n = 0`?; no: `count ← count + 1`, INPUT `n`, back to the decision; yes: OUTPUT `count` → END.

Exam tips

- Name the approach the question asks for, then apply it to **its** scenario: a generic definition alone earns little.
- For abstraction, say both what is kept and what is removed, and why.
- Use the correct flowchart shapes; every decision needs labelled yes and no exits.
- In a trace table, add a new row each time a variable changes, and put outputs in their own column.
- Boundary test data sits at both edges of the range and just outside them.
- Check an algorithm with unusual inputs: all equal, all negative, empty, or the largest allowed.

Programming

International Baccalaureate · IB Diploma · Computer Science · HL

Scope and prerequisites

This handout teaches Theme B2, Programming, for the course first assessed in 2027, using Python. Sections marked **AHL** are for Higher Level only.

Prerequisites: computational thinking from B1 (decomposition, abstraction, algorithms as flowcharts and pseudocode). You should be able to run a short Python program and read its output.

Programming fundamentals

Why does a working program fail on Monday?

A shop's checkout program has run for a month. On Monday a customer types `two` instead of `2` for the quantity and the program crashes. Nothing in the code changed: the input did. Good programs are written for every input they might meet, not only the ones the programmer tried.

Variables and data types

A **variable** 变量 is a named store for a value that can change while the program runs. A **constant** 常量 is a named value that should not change; in Python it is written in capitals by convention, such as `VAT_RATE = 0.2`.

integer	whole number: 7
real	decimal: 3.5
char	one letter: 'A'
string	text: "hello"
boolean	true or false

The basic data types: integer, real, char, string and boolean

Every value has a **data type** 数据类型, which decides what it can store and what operations work on it.

Type	Python	Example	Use
integer	`int`	`42`	counts, indexes
real	`float`	`3.75`	measurements, money in a calculation
Boolean	`bool`	`True`	flags, conditions
string	`str`	`"Ada"`	text

- `int("42")` turns text into an integer and `str(42)` turns a number into text. This **casting** 类型转换 is needed because `input()` always returns a string.
- `7 / 2` is `3.5`; `7 // 2` is `3` (**integer division** 整除); `7 % 2` is `1` (the remainder, **modulo** 取模).

Strings

A **string** 字符串 is a sequence of characters, indexed from 0.

```
code = "ib-2027"
len(code)        # 7
code[0]          # "i"
code[3:7]        # "2027": from index 3 up to, not including, 7
code.upper()     # "IB-2027"
code.find("-")   # 2; -1 if not found
"2027" in code   # True
```

Python strings are **immutable** 不可变的: `code.upper()` returns a new string and leaves `code` unchanged. Joining strings with `+` is **concatenation** 连接.

Exceptions: handling bad input

An **exception** 异常 is an error raised while the program runs, such as `ValueError` for `int("two")` or `ZeroDivisionError` for `5 / 0`. Unhandled, it stops the program. A `try … except` block catches it:

```
while True:
    text = input("Quantity: ")
    try:
        qty = int(text)
        break
    except ValueError:
        print("Enter a whole number.")
```

The loop repeats until the cast succeeds, so the program never crashes on `two`. A `finally` block, if present, runs whether or not an exception occurred: it is used to close files.

Debugging and trace tables

There are three kinds of error:

count	total	output
1	5	
2	12	
3	20	20

A trace table records each variable's value as the program runs

Kind	When it shows	Example
syntax error	before the program runs	a missing colon after `if x > 0`
runtime error	while it runs, as an exception	an index past the end of a list
logic error	never reported: the output is wrong	`<` where `<=` was meant

A **trace table** 跟踪表 records the value of each variable after each step, by hand. It is the standard way to find a logic error and a common exam question.

```
total = 0
for k in range(1, 5):
    if k % 2 == 0:
        total = total + k
print(total)
```

k	k % 2 == 0	total
		0
1	False	0
2	True	2
3	False	2
4	True	6

Output: `6`. Note that `range(1, 5)` stops **before** 5. Other debugging techniques are print statements at key points, breakpoints in a **debugger** 调试器 that pause the program and show the variables, and stepping through one line at a time.

Data structures

Static and dynamic structures

A **static data structure** 静态数据结构 has a size fixed when it is created, like an array in Java or C. A **dynamic data structure** 动态数据结构 grows and shrinks as the program runs, like a Python list.

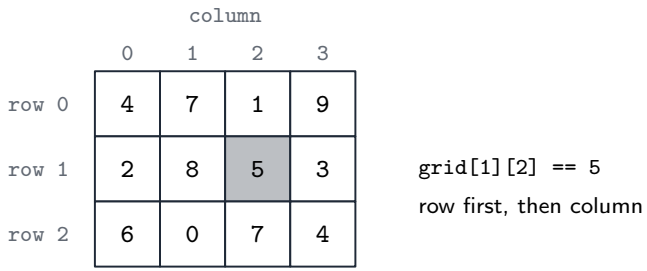
	Static	Dynamic
size	fixed	changes at run time
memory	reserved in advance, may be wasted	used as needed
access	direct by index, fast	may need extra work to grow
risk	overflow when full	running out of memory

One- and two-dimensional arrays

A one-dimensional **array** 数组 stores items of one type under one name, each reached by its **index** 索引. In Python a list plays this role:

```
marks = [67, 82, 45, 90]
marks[2]          # 45
marks[2] = 50     # change one item
marks.append(71)  # add at the end: [67, 82, 50, 90, 71]
len(marks)        # 5
```

A **two-dimensional array** 二维数组 is a table: a list of rows, where each row is a list. The first index picks the row and the second the column.



`grid[row][col]`: the row index comes first

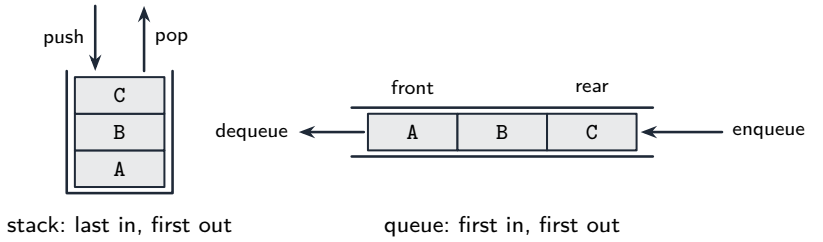
```
grid = [[4, 7, 1, 9],
        [2, 8, 5, 3],
        [6, 0, 7, 4]]

row_total = 0
for col in range(len(grid[1])):
    row_total = row_total + grid[1][col]
print(row_total)   # 2 + 8 + 5 + 3 = 18
```

To visit every cell, nest two loops: the outer one over the rows, the inner one over the columns.

Stacks and queues

Both are **abstract data types** 抽象数据类型: defined by their operations, not by how they are stored.



Stack: one end does everything. Queue: add at the rear, remove from the front

	Stack	Queue
order	**LIFO** 后进先出: last in, first out	**FIFO** 先进先出: first in, first out
add	`push` onto the top	`enqueue` at the rear
remove	`pop` from the top	`dequeue` from the front
look	`peek` at the top	`peek` at the front
uses	undo, back button, function calls	print jobs, keyboard buffer, task scheduling

```
stack = []
stack.append("A"); stack.append("B"); stack.append("C")    # push
stack.pop()        # "C"

from collections import deque
queue = deque()
queue.append("A"); queue.append("B"); queue.append("C")    # enqueue
queue.popleft()    # "A"
```

Before a `pop` or `dequeue`, check `isEmpty`: removing from an empty structure is an **underflow** 下溢. Pushing onto a full fixed-size stack is an **overflow** 溢出.

Programming constructs

Sequence, selection and iteration

Every algorithm is built from three constructs: **sequence** 顺序 (steps in order), **selection** 选择 (`if`, `elif`, `else`) and **iteration** 迭代 (loops).

Loop	Python	Use when
count-controlled	`for i in range(n):`	the number of repeats is known
condition-controlled	`while condition:`	repeat until something happens
over a collection	`for item in items:`	visit each item once

A `while` loop whose condition never becomes false is an **infinite loop** 无限循环. Nested selection or loops place one construct inside another; indentation shows which is inside which.

Functions, parameters and scope

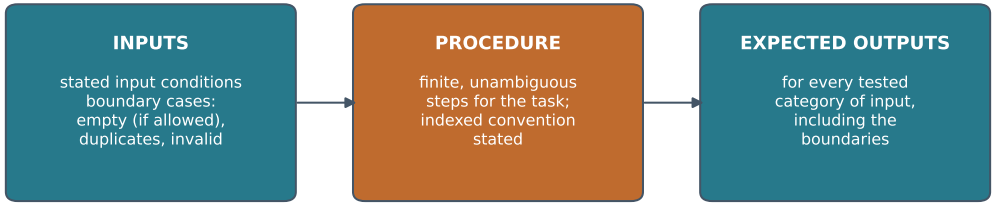
A **function** 函数 is a named block of code that can be called many times. It takes **parameters** 参数 and may **return** 返回 a value. Splitting a program into functions is **modularity** 模块化: each part is written, tested and reused on its own.

```
def mean(values):
    """Return the mean of a non-empty list of numbers."""
    total = 0
    for v in values:
        total = total + v
    return total / len(values)

print(mean([4, 8, 9]))   # 7.0
```

- A **local variable** 局部变量 such as `total` exists only inside the function while it runs.
- A **global variable** 全局变量 is defined at module level and can be read where Python scope and name-binding rules resolve to it; a local name can shadow it. Using many globals makes a program hard to test, because any function might change them.
- The docstring states the function's contract: what it needs (a non-empty list) and what it returns. `mean([])` breaks the contract and raises `ZeroDivisionError`.

A program contract: inputs, procedure, expected outputs



A failure is diagnosed as a wrong algorithm, a wrong implementation, or an incomplete requirement.

A function's contract: the inputs it accepts, including boundary cases, and the output it promises for each

Test a function against its contract: normal values, **boundary cases** 边界情形 at the edge of what it accepts (a one-item list), and invalid values it should reject (an empty list).

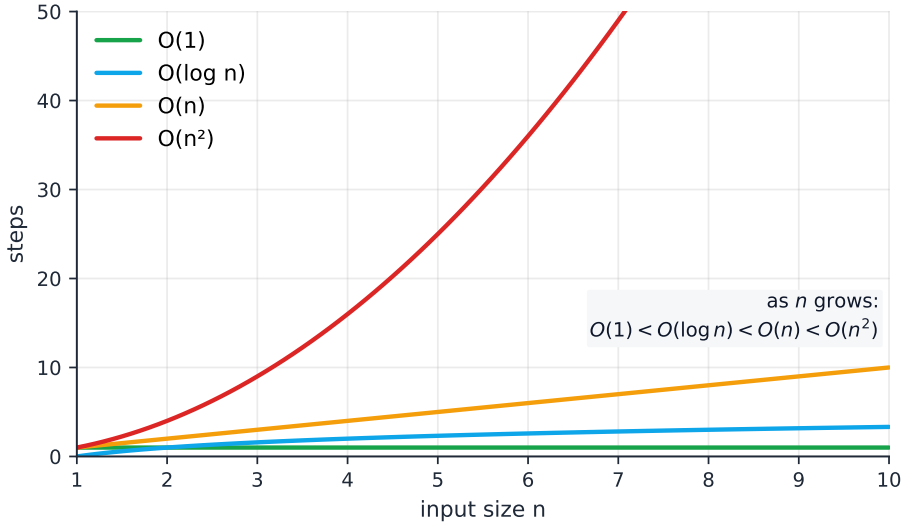
Algorithms

Measuring efficiency: Big O

Big O notation 大 O 表示法 gives an asymptotic upper bound for how a chosen cost grows with input size n. State whether the cost being analysed is worst-case, average-case or best-case. Constants and smaller terms are dropped.

Big O	Name	Example	Steps for $n = 1000$
$O(1)$	constant	reading `marks[2]`	1
$O(\log n)$	logarithmic	binary search	about 10
$O(n)$	linear	linear search	1000
$O(n^2)$	quadratic	bubble sort, selection sort	about 1 000 000

A single loop over the data is usually $O(n)$; a loop inside a loop over the same data is usually $O(n^2)$.



Growth, not speed on one input: $O(1) < O(\log n) < O(n) < O(n^2)$

Linear and binary search

A **linear search** 线性查找 checks each item in turn. It works on any list, sorted or not, and is $O(n)$.

```
def linear_search(items, target):
    for i in range(len(items)):
        if items[i] == target:
            return i
    return -1
```

A **binary search** 二分查找 needs a **sorted** list. It checks the middle item and throws away the half that cannot hold the target, so each probe halves what is left: $O(\log n)$.

```
def binary_search(items, target):
    low, high = 0, len(items) - 1
    while low <= high:
        mid = (low + high) // 2
        if items[mid] == target:
            return mid
        elif items[mid] < target:
            low = mid + 1
        else:
            high = mid - 1
    return -1
```

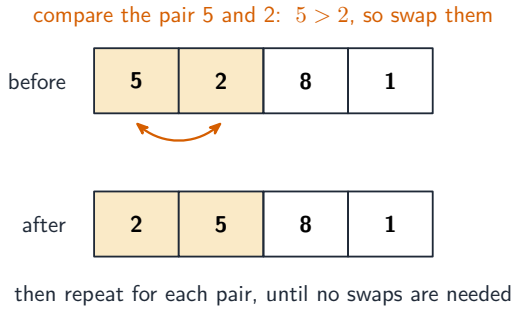


Each probe discards half of what remains

Trace for 23: `low` 0, `high` 8, `mid` 4 ($16 < 23$) → `low` 5; `mid` $(5+8)//2 = 6$ ($38 > 23$) → `high` 5; `mid` 5: found, return **5**. On an unsorted list binary search can miss a value that is there, so the contract must require sorted input.

Bubble sort and selection sort

A **bubble sort** 冒泡排序 compares each pair of neighbours and swaps them if they are in the wrong order. After each pass the largest remaining item has "bubbled" to the end.



One comparison in another example: bubble sort swaps neighbours that are out of order, then continues through the pass

```
def bubble_sort(a):
    n = len(a)
    for p in range(n - 1):
        swapped = False
        for i in range(n - 1 - p):
            if a[i] > a[i + 1]:
                a[i], a[i + 1] = a[i + 1], a[i]
                swapped = True
        if not swapped:
            break
```

The `swapped` flag stops early when a pass makes no swaps: the list is already sorted.

A **selection sort** 选择排序 finds the smallest item in the unsorted part and swaps it into the next place.

```
def selection_sort(a):
    for i in range(len(a) - 1):
        smallest = i
        for j in range(i + 1, len(a)):
            if a[j] < a[smallest]:
                smallest = j
        a[i], a[smallest] = a[smallest], a[i]
```

Passes of each sort on `[5, 1, 4, 2]`:

After pass	Bubble sort	Selection sort
1	`[1, 4, 2, 5]`	`[1, 5, 4, 2]`
2	`[1, 2, 4, 5]`	`[1, 2, 4, 5]`
3	no swaps: stop	`[1, 2, 4, 5]`

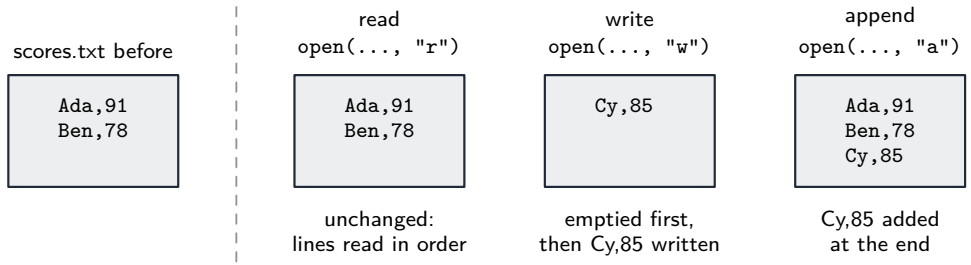
Both are $O(n^2)$. Selection sort makes at most $n-1$ swaps; bubble sort can stop early on a nearly sorted list.

File processing

Reading and writing text files

A program's variables are lost when it ends; a **file** 文件 keeps data between runs. Open the file with a **mode** 模式, use it, then close it.

Mode	Meaning	If the file exists
`"r"`	read	read from the start
`"w"`	write	**emptied** first
`"a"`	append	add to the end



Mode "w" destroys what was there; mode "a" keeps it

```
with open("scores.txt", "a") as f:
    f.write("Ada,91\n")

best_name, best = "", -1
with open("scores.txt", "r") as f:
    for line in f:
        name, score = line.strip().split(",")
        if int(score) > best:
            best_name, best = name, int(score)
print(best_name, best)
```

- `with` closes the file automatically, even if an exception occurs.
- Each line read ends in `"\n"`; `strip()` removes it before `split(",")` breaks the line into fields.
- Opening a missing file for reading raises `FileNotFoundError`: catch it with `try … except`.

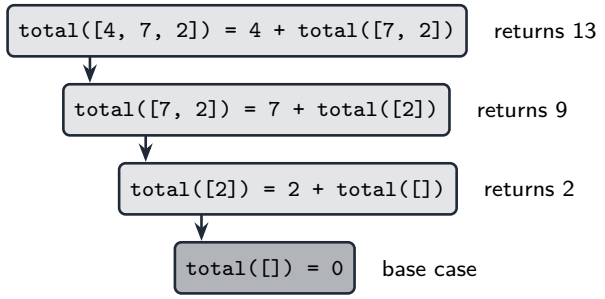
AHL: Recursion

A function that calls itself

A **recursive function** 递归函数 calls itself on a smaller version of the same problem. It needs two parts:

1. A **base case** 基线条件 that is answered directly, with no further call.
2. A **recursive case** 递归情形 that calls the function on a smaller input, moving towards the base case.

```
def total(items):
    if items == []:                      # base case
        return 0
    return items[0] + total(items[1:])   # recursive case
```



Calls go down to the base case; results come back up

Each call is placed on the **call stack** 调用栈 with its own local variables. The calls wait until the base case returns, then finish in reverse order: last called, first finished, like a stack.

Tracing a recursive call

```
def power(b, e):
    if e == 0:
        return 1
    return b * power(b, e - 1)
```

`power(3, 3)` → `3 * power(3, 2)` → `3 * 3 * power(3, 1)` → `3 * 3 * 3 * power(3, 0)` → `3 * 3 * 3 * 1` = **27**: four calls in all.

Recursion or iteration?

	Recursion	Iteration
code	often shorter, matches the problem's definition	often longer
memory	one stack frame per call	one set of variables
risk	**stack overflow** 栈溢出 if too deep or no base case	an infinite loop
natural for	trees, nested folders, divide and conquer such as **quicksort** 快速排序	simple counting and running totals

Quicksort is recursive: pick a pivot, split the list into items less than and greater than the pivot, then quicksort each part. Its average case is $O(n \log n)$, much faster than the $O(n^2)$ sorts above for large n.

Practice

1. State the data type best suited to each: (a) the number of students in a class; (b) whether a door is locked; (c) a postcode such as `"SW1A 1AA"`; (d) a temperature of 21.5 °C.
2. `s = "Programming"`. Give (a) `len(s)`; (b) `s[3:7]`; (c) `s.find("g")`.
3. Classify each error as syntax, runtime or logic: (a) `print("Hi"`; (b) a loop that adds one item too few; (c) `int("abc")`.
4. Complete a trace table for `x = 1` then `while x < 20: x = x * 3`, and state the final value of `x`.
5. The items 10, 20 and 30 are pushed onto an empty stack, then one item is popped and 40 is pushed. State the item on top.
6. The same items are put through a queue: enqueue 10, 20, 30, dequeue once, enqueue 40. State the item at the front.
7. For the list `[3, 8, 12, 19, 25, 31, 40]`, list the values that a binary search for 31 compares with.
8. Show the list `[6, 3, 8, 2]` after each pass of a bubble sort.
9. Write a Python function `count_over(marks, limit)` that returns how many marks in the list are greater than `limit`.
10. Explain why opening `"log.txt"` with mode `"w"` each time a program starts loses data, and state the fix.
11. **AHL** For `power` above, state the number of calls made by `power(2, 5)` and the value returned.
12. **AHL** Explain what happens when a recursive function has no base case.

Answers.

1. (a) integer; (b) Boolean; (c) string, as it holds letters and spaces; (d) real.
2. (a) 11; (b) `"gram"`; (c) 3.
3. (a) syntax: a missing bracket; (b) logic; (c) runtime: a `ValueError`.
4. `x` takes the values 1, 3, 9, 27; the loop stops when `x` is 27. Final value **27**.
5. Push 10, 20, 30; pop removes 30; push 40. Top: **40**.
6. Enqueue 10, 20, 30; dequeue removes 10; enqueue 40. Front: **20**.
7. `low` 0, `high` 6: `mid` 3, value 19 < 31, so `low` = 4; `mid` $(4 + 6)//2 = 5$, value **31**: found. Compared with 19 and 31.
8. Pass 1: `[3, 6, 2, 8]`; pass 2: `[3, 2, 6, 8]`; pass 3: `[2, 3, 6, 8]`.
9. One answer:

```
def count_over(marks, limit):
    count = 0
    for m in marks:
        if m > limit:
            count = count + 1
    return count
```

10. Mode `"w"` empties an existing file before writing, so the earlier log is lost. Open it with `"a"` to append.
11. `power(2, 5)` calls itself with `e` = 4, 3, 2, 1, 0: **6** calls, returning 2^5 = **32**.
12. It calls itself for ever; each call adds a frame to the call stack until memory runs out, a stack overflow (in Python a `RecursionError`).

Exam tips

- In a trace table, give a new row each time a variable changes, and show the output separately. Check where `range` stops.
- State the indexing convention (Python indexes from 0) before tracing an array or a search.
- For a binary search answer, say that the list **must be sorted** and give `low`, `high` and `mid` at every probe.
- Bubble sort and selection sort are both $O(n^2)$: compare them by swaps, early stopping and passes, not by Big O.
- "Explain" a stack or queue with its order (LIFO or FIFO), its two operations and a real use.
- When writing code, use clear names, close files with `with`, and handle bad input with `try … except`.
- **AHL** A recursion answer names the base case, the recursive case, and how each call moves towards the base case.

Object-oriented programming

International Baccalaureate · IB Diploma · Computer Science · HL

Scope and prerequisites

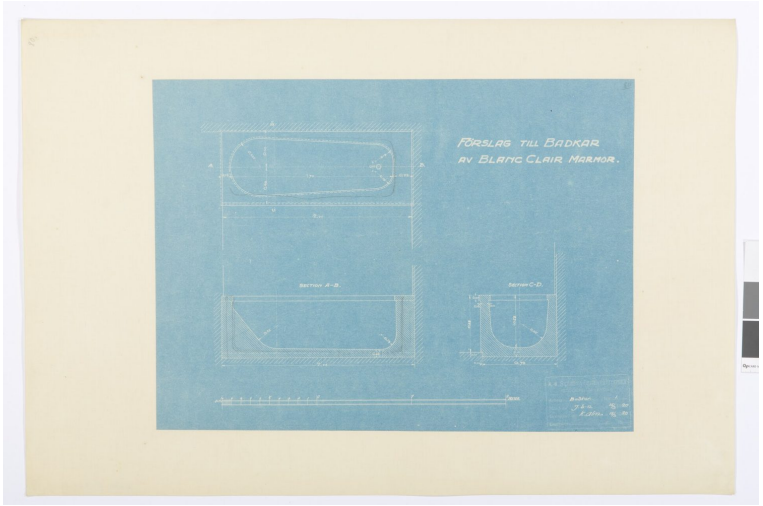
This handout teaches Theme B3, Object-oriented programming, for the course first assessed in 2027, using Python, with Java shown where its syntax differs. Sections marked **AHL** are for Higher Level only.

Prerequisites: B2 Programming (variables, selection, loops, functions and parameters). The examples are fictional models, not real bank systems.

Classes and objects

Why model the world as objects?

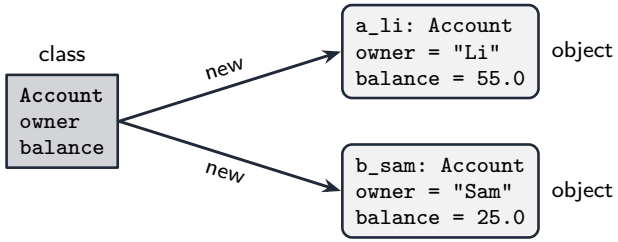
A school app tracks thousands of students. Each one has a name, a year group and a list of grades, and each can enrol on a course or receive a grade. Writing separate variables for every student is impossible. Instead we write the design once, as a class, and make one object per student.



A blueprint: a class is a template that defines how objects of that type are built

Class, object and instance

A **class** 类 is a blueprint: it defines the **attributes** 属性 (the data each object holds) and the **methods** 方法 (what each object can do). An **object** 对象 is one thing built from that blueprint, an **instance** 实例 of the class. Making an object is **instantiation** 实例化.



One class, many objects: the same attributes, each with its own values

- An object's **state** 状态 is the current values of its attributes; its **behaviour** 行为 is what its methods do.
- Objects of a class normally share its defined interface and may hold different instance values; Python also allows instance attributes to be added dynamically.

Writing a class in Python

```
class Account:
    def __init__(self, owner, balance):   # the constructor
        self.owner = owner                 # instance attributes
        self.balance = balance

    def deposit(self, amount):
        self.balance = self.balance + amount

    def get_balance(self):
        return self.balance

a_li = Account("Li", 40.0)      # instantiation
b_sam = Account("Sam", 20.0)
a_li.deposit(15.0)
print(a_li.get_balance(), b_sam.get_balance())   # 55.0 20.0
```

- The object-initialisation step is often taught with the **constructor** 构造函数. In normal Python construction, `__new__` creates the instance and `__init__` initialises its state. In Java it is a method with the class's name: `public Account(String owner, double balance)`.
- `self` is the object the method is working on: `a_li.deposit(15.0)` changes `a_li`'s balance only.
- A variable holds a **reference** 引用 to an object. After `c = a_li`, both names refer to the **same** object: a deposit through `c` is seen through `a_li`.

Instance and static members

An **instance attribute** 实例属性 belongs to one object. A **static** 静态 (class) attribute belongs to the class and is shared by every object.

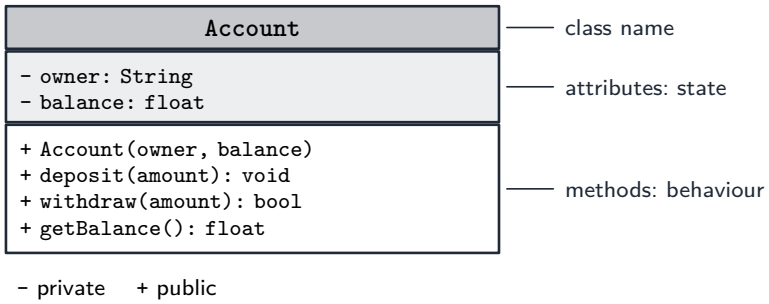
```
class Account:
    count = 0                          # static: one value for the whole
    ↪  class
    def __init__(self, owner, balance):
        self.owner = owner
        self.balance = balance
        Account.count = Account.count + 1
```

After two accounts are made, `Account.count` is 2. In Java a static member is marked `static` and is called on the class, such as `Math.sqrt(9)`.

Encapsulation

Hiding the state

Encapsulation 封装 bundles the data and the methods that act on it into one class, and hides the data so it can be changed only through those methods. The class then protects its own rules, such as "the balance is never negative".



A UML class diagram: name, attributes, methods; - private, + public

Access	Java	Python convention	Who can use it
private 私有	`private`	leading _ marks non-public use by convention	Java restricts access; Python's single underscore does not enforce privacy
public 公有	`public`	plain name	any code

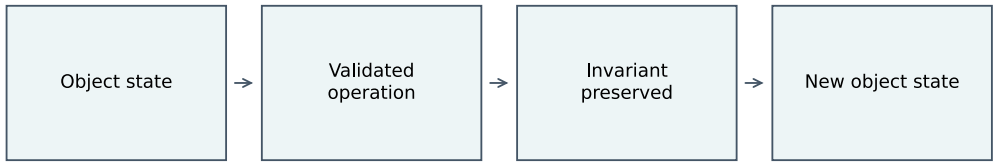
Accessors and mutators

Private attributes are read through an **accessor** 访问器 (getter) and changed through a **mutator** 修改器 (setter), which can check the new value first.

```
class Account:
    def __init__(self, owner, balance):
        self._owner = owner
        self._balance = max(0.0, balance)

    def get_balance(self):              # accessor
        return self._balance

    def withdraw(self, amount):         # mutator with validation
        if amount <= 0 or amount > self._balance:
            return False
        self._balance = self._balance - amount
        return True
```



A validated operation keeps the rule true: a rejected withdrawal leaves the state unchanged

Trace: `acc = Account("Li", 15.0)`; `acc.withdraw(30.0)` returns `False` and the balance stays 15.0; `acc.withdraw(15.0)` returns `True` and the balance becomes 0.0.

Advantages of encapsulation:

- **Data integrity**: invalid values are rejected in one place.
- **Maintenance**: the inside of a class can change without breaking code that uses only its public methods.
- **Teamwork**: a programmer can use a class from its public interface alone.

Designing with classes

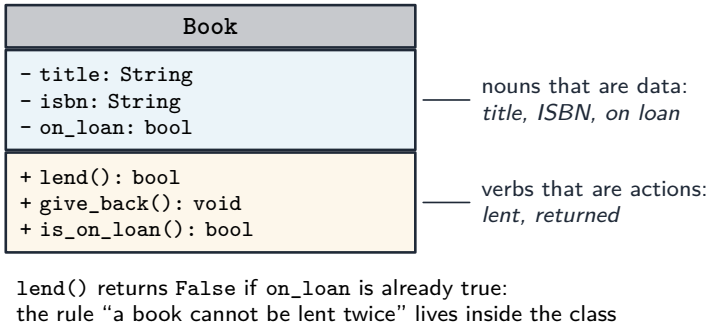
Choosing attributes and methods

To design a class from a scenario, pick out the **nouns** that are data (attributes) and the **verbs** that are actions (methods).

> A library lends books. Each book has a title, an ISBN and whether it is on loan. A book can be lent and returned.

Attributes (private)	Methods (public)
`title: String`	`lend(): bool`
`isbn: String`	`give_back(): void`
`on_loan: bool`	`is_on_loan(): bool`

`lend` returns `False` if the book is already on loan, so the rule "a book cannot be lent twice" lives inside the class.



Nouns become attributes, verbs become methods, and the rule lives inside the class

Advantages and limits of OOP

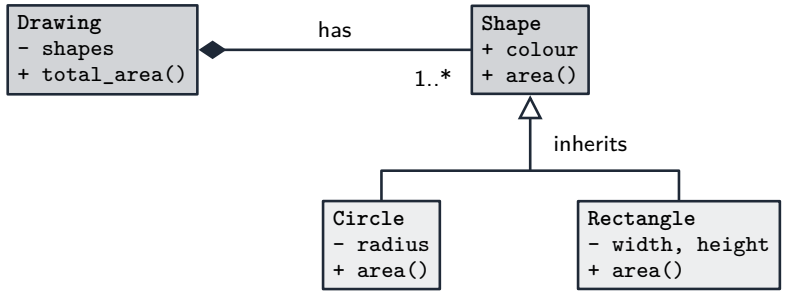
Advantages	Disadvantages
models real things, so designs are easy to discuss	more code and planning for a small program
classes are reused in other programs	objects can use more memory
encapsulation protects data	a poor class design is hard to change later

AHL: Inheritance and polymorphism

Inheritance

Inheritance 继承 lets a **subclass** 子类 take the attributes and methods of a **superclass** 父类 and add or change some. It models an "is a" relationship: a circle **is a**

shape.



Hollow triangle: inherits ("is a"). Filled diamond: composition ("has a")

```python
import math

class Shape:
    def __init__(self, colour):
        self.colour = colour
    def area(self):
        return 0.0

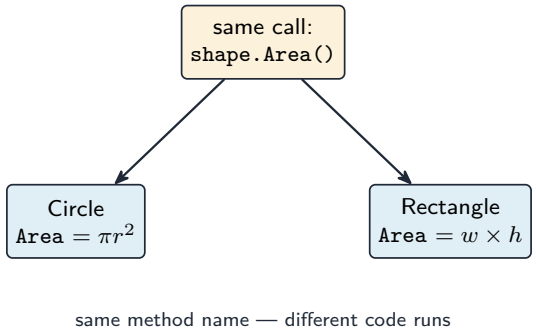
class Circle(Shape):
    def __init__(self, colour, radius):
        super().__init__(colour)        # run the superclass constructor
        self.radius = radius
    def area(self):                     # overrides Shape.area
        return math.pi * self.radius ** 2

class Rectangle(Shape):
    def __init__(self, colour, width, height):
        super().__init__(colour)
        self.width, self.height = width, height
    def area(self):
        return self.width * self.height
```

- `super()` calls the superclass's version, so the colour code is written once.
- In Java: `class Circle extends Shape`.

Polymorphism

Polymorphism 多态 means one call works on objects of different classes, each running its own version. A subclass method with the same name and parameters as the superclass's **overrides** 重写 it.



Polymorphism: the same method call runs each object's own code

```python
shapes = [Circle("red", 1.0), Rectangle("blue", 2.0, 3.0)]
for s in shapes:
    print(round(s.area(), 2))   # 3.14, then 6.0
```

The loop does not need to know which kind of shape it holds. A new subclass such as `Triangle` works with the same loop unchanged.

Overloading 重载 is different: several methods in one class share a name but take different parameters, such as two Java constructors `Account(String owner)` and `Account(String owner, double balance)`. Python does not overload; it uses default parameter values instead.

Abstract classes

An **abstract class** 抽象类 cannot be instantiated: it exists to be inherited. An **abstract method** 抽象方法 must be implemented before a subclass can be concrete; Java abstract declarations have no body, while a Python abstract method may supply a body for reuse. `Shape` should be abstract, since a plain shape has no sensible area.

```python
from abc import ABC, abstractmethod

class Shape(ABC):
    @abstractmethod
    def area(self):
        pass
```

`Shape()` now raises an error; `Circle` must define `area` or it cannot be instantiated either. In Java: `abstract class Shape` and `abstract double area();`.

■ Composition and aggregation

Relationship	Meaning	Example	UML
inheritance	"is a"	a `Circle` is a `Shape`	hollow triangle
composition 组合	"has a"; the part dies with the whole	a `Drawing` has its `Shapes`	filled diamond
aggregation 聚合	"has a"; the part lives on alone	a `Team` has `Players`	hollow diamond

Prefer composition when the relationship is "has a": a car has an engine, it is not a kind of engine. Deep inheritance chains are hard to change, because a change to a superclass affects every subclass.

■ Design patterns

A **design pattern** 设计模式 is a tested solution to a common design problem.

Pattern	Problem it solves	Idea
singleton	exactly one object is needed, such as a settings store	the class makes its single instance and always returns it
factory	the code should not depend on which subclass is made	one method decides which subclass to create
observer	several objects must react when one changes	the subject keeps a list of observers and notifies each

Practice

1. Explain the difference between a class and an object, using a phone model as an example.
2. For the `Account` class with `deposit`, state the output of: `x = Account("Ana", 10.0)`, `y = Account("Bo", 10.0)`, `x.deposit(5.0)`, `print(x.get_balance(), y.get_balance())`.
3. `p = Account("Kai", 30.0)`, `q = p`, `q.deposit(20.0)`. State `p.get_balance()` and explain.
4. Explain why `balance` is made private and changed only through `withdraw`.
5. Draw the UML class diagram for a `Thermostat` with a private `target` (an integer), a constructor, and public methods `raise_by(d)`, `lower_by(d)` and `get_target()`.
6. From the scenario "A cinema has screens. Each screen has a number and a number of seats; seats can be booked and cancelled", identify two attributes and two methods of `Screen`.
7. State **one** advantage and **one** disadvantage of OOP.
8. **AHL** A `Dog` and a `Cat` both inherit from `Animal`, which has `speak()`. Explain how polymorphism lets one loop make every animal in a list speak.
9. **AHL** Distinguish overriding from overloading.
10. **AHL** State whether each is inheritance, composition or aggregation: (a) a `Car` and its `Engine`, made and scrapped together; (b) a `Truck` and `Vehicle`; (c) a `Club` and its `Members`, who can leave and join other clubs.

Answers.

1. The class is the design (the phone model): it says what every phone has and can do. An object is one actual phone built from it, with its own state, such as its own battery level.
2. `15.0 10.0`: each object has its own balance.
3. **50.0**: `q = p` copies the reference, so `p` and `q` are the same object.
4. Encapsulation: no code outside the class can set the balance to an invalid value; `withdraw` checks every change, so the rule "never negative" always holds.
5. Three compartments: `Thermostat`; `- target: int`; `+ Thermostat(target)`, `+ raise_by(d): void`, `+ lower_by(d): void`, `+ get_target(): int`.
6. Attributes: `number`, `seats` (or `booked`). Methods: `book()`, `cancel()`.
7. Advantage: e.g. classes can be reused, or data is protected by encapsulation. Disadvantage: e.g. more planning and code for a small program.
8. Each subclass overrides `speak()`. The loop calls `a.speak()` on each `Animal`; at run time each object runs its own class's version, so dogs bark and cats meow without the loop testing the type.
9. Overriding: a subclass redefines a superclass method with the same name and parameters. Overloading: methods in one class share a name but take different parameters.
10. (a) composition; (b) inheritance (a truck **is a** vehicle); (c) aggregation.

Exam tips

- Use the words exactly: a **class** is the blueprint, an **object** is an instance, **instantiation** makes it.
- In a UML class diagram, keep three compartments and mark each member `-` private or `+` public, with types.
- When you "explain encapsulation", say both parts: data and methods bundled together, and the data hidden behind methods that validate it.
- In code questions, remember that assigning an object variable copies the reference, not the object.
- Mutators normally return nothing (`void`); accessors return a value and change nothing.

- **AHL** For inheritance answers, name the “is a” test; for composition, the “has a” test.
- **AHL** Polymorphism answers need an example: the same call, different classes, each running its own overridden method.

Abstract data types (HL only)

International Baccalaureate · IB Diploma · Computer Science · HL

Scope and prerequisites

This handout teaches Theme B4, Abstract data types, for the course first assessed in 2027, using Python. The whole theme is Higher Level only (AHL).

Prerequisites: B2 Programming (arrays, stacks, queues, recursion) and B3 Object-oriented programming (classes and references).

Abstract data types

Why does the structure matter?

A music app holds a playlist of 10 000 songs. Adding a song in the middle of an array means shifting thousands of items along; searching an unsorted list for a title checks them one by one. Choosing a different structure makes both jobs fast. This theme is about those structures.



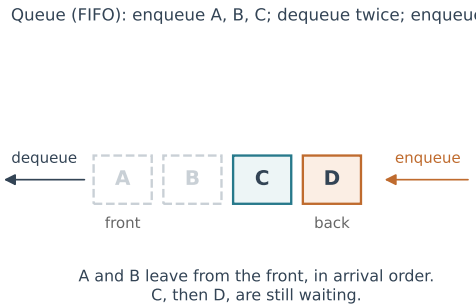
*A line of people is a queue you can see. You join at the **back** and are served from the **front**, so whoever waited longest is served first – that is exactly FIFO*

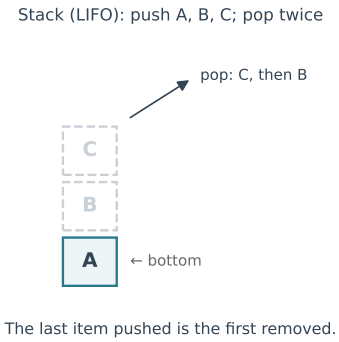


*A pile of books is a stack you can see. You can only add or take a book from the **top**, so the last one you put on is the first one you take off – that is exactly LIFO*

Interface and implementation

An **abstract data type** 抽象数据类型 (ADT) is defined by the values it holds and the operations on them, not by how it is stored. The stack from B2 is an ADT: `push`, `pop`, `peek` and `isEmpty`, last in first out. It can be **implemented** 实现 with an array or with a linked list; code that uses the stack does not need to know which.





Same items, two ADTs: the operations, not the storage, decide the order out

ADT	Key operations	Order or rule
stack	push, pop, peek	LIFO
queue	enqueue, dequeue	FIFO
linked list	insert, delete, traverse	follows the pointers
binary search tree	insert, search, traverse	left smaller, right larger
set	add, remove, contains	no duplicates, no order
hash table	put, get by key	position from a hash function

Linked lists

Nodes and pointers

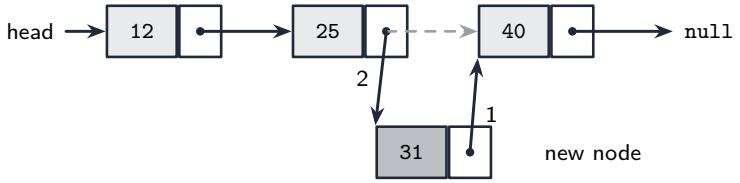
A **linked list** 链表 is a chain of **nodes** 节点. Each node holds data and a **pointer** 指针 (a reference) to the next node. The list is reached through its **head** 头指针; the last node points to `None` (null).

```
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None
```

To **traverse** 遍历 the list, follow the pointers from the head until `None`:

```
def show(head):
    current = head
    while current is not None:
        print(current.data)
        current = current.next
```

Inserting and deleting



Insert after a node: first point the new node onward, then redirect the old pointer

To insert 31 after the node holding 25 (called `prev`):

```
new = Node(31)
new.next = prev.next      # step 1: new node points to 40
prev.next = new           # step 2: 25 now points to the new node
```

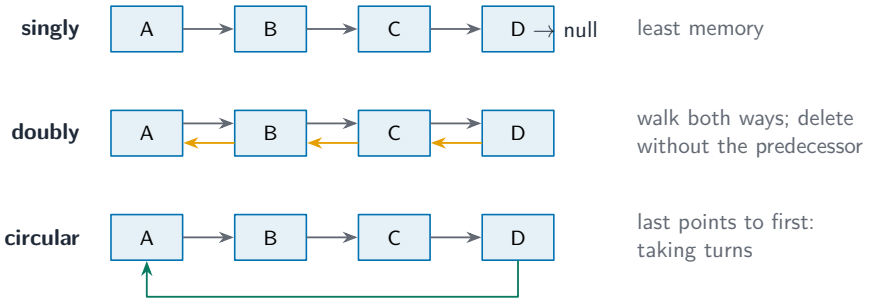
The order matters: doing step 2 first would lose the pointer to 40 and the rest of the list.

To delete the node after `prev`: `prev.next = prev.next.next`. The removed node is no longer reachable, and Python's garbage collector frees its memory.

No items are shifted, so insertion and deletion are $O(1)$ once the position is found; but finding a position means traversing from the head, $O(n)$. There is no index: reading the 500th item takes 500 steps.

Variants

Variant	Each node points to	Advantage
singly linked 单向链表	the next node	least memory
doubly linked 双向链表	the next and the previous node	traverse both ways; delete a node without its predecessor
circular 循环链表	the next; the last points back to the first	round-robin tasks, such as taking turns



Three linked-list variants

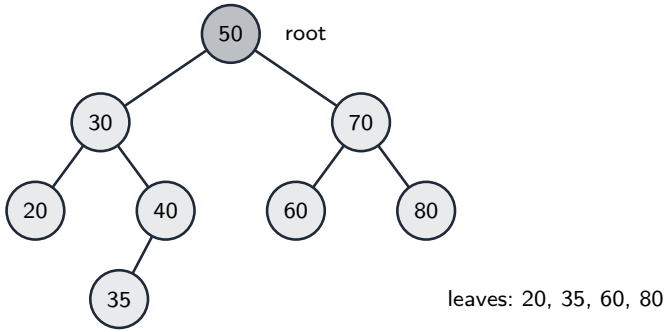
Linked list or array?

	Array	Linked list
access by index	$O(1)$	$O(n)$
insert or delete in the middle	$O(n)$: shift items	$O(1)$ at a known node
size	fixed (static) or resized	grows one node at a time
memory	data only	data plus a pointer per node

Binary search trees

Structure

A **binary tree** 二叉树 is made of nodes that each have at most two **children** 子节点, left and right. The top node is the **root** 根节点; a node with no children is a **leaf** 叶节点. In a **binary search tree** 二叉搜索树 (BST), every key in a node's left subtree is smaller than the node's key, and every key in its right subtree is larger.



Built by inserting 50, 30, 70, 20, 40, 60, 80, 35 in that order

Inserting and searching

To insert or find a key, start at the root and go left if the key is smaller, right if larger, until a match or an empty place.

Inserting 35: $35 < 50$, go left; $35 > 30$, go right; $35 < 40$, go left; empty, so 35 becomes the left child of 40.

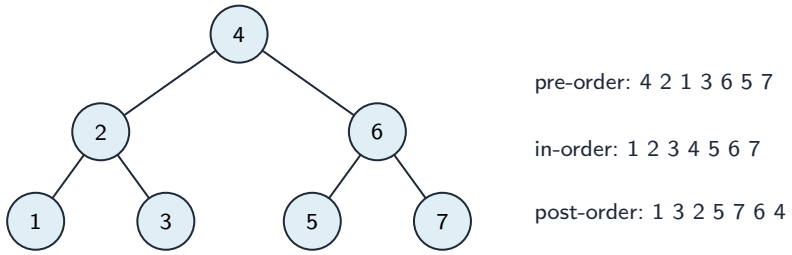
```
class TreeNode:
    def __init__(self, key):
        self.key = key
        self.left = None
        self.right = None

def contains(node, key):
    if node is None:
        return False
    if key == node.key:
        return True
    if key < node.key:
        return contains(node.left, key)
    return contains(node.right, key)
```

Each comparison drops one subtree, so a **balanced** 平衡 tree of n keys is searched in $O(\log n)$. Keys inserted already sorted (10, 20, 30, …) make a tree that is a single chain: search becomes $O(n)$, no better than a list.

Traversals

A **traversal** visits every node once. The three depth-first orders differ in when the node itself is visited:



Three depth-first traversals of a binary tree: pre-order, in-order (sorted order) and post-order

Traversal	Order	For the tree above
pre-order 前序遍历	node, left, right	50, 30, 20, 40, 35, 70, 60, 80
in-order 中序遍历	left, node, right	20, 30, 35, 40, 50, 60, 70, 80
post-order 后序遍历	left, right, node	20, 35, 40, 30, 60, 80, 70, 50

In-order traversal of a BST gives the keys in **sorted order**. Pre-order copies a tree (insert in that order to rebuild it); post-order deletes one (children before parents).

```
def in_order(node):
    if node is not None:
        in_order(node.left)
        print(node.key)
        in_order(node.right)
```

Sets

No duplicates, no order

A **set** 集合 holds distinct values with no order: adding a value already present changes nothing. Its main operation is a fast membership test.

```
a = {"maths", "physics", "cs"}
b = {"cs", "art"}
a | b     # union: {"maths", "physics", "cs", "art"}
a & b     # intersection: {"cs"}
a - b     # difference: {"maths", "physics"}
"art" in a   # False
```

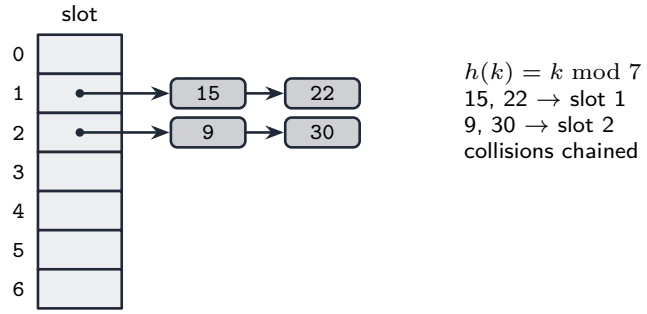
Operation	Result
union 并集	in either set
intersection 交集	in both sets
difference 差集	in the first but not the second
subset	every item of one is in the other

Python sets are built on hash tables, so `in` is $O(1)$ on average, against $O(n)$ for a list.

Hash tables

Position from the key

A **hash table** 哈希表 stores key–value pairs in an array. A **hash function** 哈希函数 turns each key into a slot index, so lookup is $O(1)$ on average under suitable hashing and load assumptions; collisions still require comparisons and worst-case lookup can be $O(n)$. A Python `dict` is a hash table.



Two keys in one slot is a collision; chaining keeps a list in the slot

With 7 slots and $h(k) = k \bmod 7$: $15 \bmod 7 = 1$, $22 \bmod 7 = 1$, $9 \bmod 7 = 2$, $30 \bmod 7 = 2$.

Collisions

A **collision** 冲突 happens when two keys hash to the same slot. Two common fixes:

Method	How it works	15, 22, 9, 30 with 7 slots
chaining 链地址法	each slot holds a linked list of items	slot 1: 15 → 22; slot 2: 9 → 30
linear probing 线性探测	try the next slot until an empty one	15 → 1; 22 → 2; 9 → 3; 30 → 4

With linear probing, a search for 30 starts at slot 2 and steps on until it finds 30 or an empty slot.

A good hash function is fast and spreads keys evenly. The **load factor** 负载因子 is $\frac{\text{items}}{\text{slots}}$; as it rises, collisions grow, so the table is enlarged and every key rehashed.

Practice

1. Explain what is meant by an abstract data type, using a queue as the example.
2. A linked list holds $8 \to 14 \to 21 \to$ None. Write the two statements that insert a node holding 17 after the node `p` that holds 14, in the correct order.
3. Explain why reading the kth item is slower in a linked list than in an array.
4. Give **one** use of a circular linked list.
5. Draw the BST made by inserting 40, 20, 60, 10, 30, 50, 70, 25.
6. For your tree in question 5, give the pre-order, in-order and post-order traversals.
7. State the number of comparisons needed to find 25 in your tree.
8. Explain what happens to search time if the keys 10, 20, 30, 40, 50 are inserted into a BST in that order.
9. $A = \{1, 3, 5, 7\}$ and $B = \{3, 4, 5\}$. State $A \cup B$, $A \cap B$ and $A - B$.
10. Insert the keys 12, 19, 5, 26 into a hash table of 7 slots with $h(k) = k \bmod 7$, using linear probing. State the slot of each key.
11. Explain why a hash table becomes slower as its load factor rises.

Answers.

1. An ADT is defined by its values and operations, not by its storage. A queue is defined by enqueue at the rear and dequeue from the front (FIFO), and can be built from an array or a linked list without changing the code that uses it.
2. `new.next = p.next` then `p.next = new`, where `new = Node(17)`.
3. A linked list has no index: the program must follow the pointers from the head, k steps. An array computes the item's address directly.
4. E.g. giving each player a turn in a game, or round-robin scheduling of processes.
5. Root 40; 20 left, 60 right; 10 and 30 under 20; 50 and 70 under 60; 25 the left child of 30.
6. Pre-order 40, 20, 10, 30, 25, 60, 50, 70. In-order 10, 20, 25, 30, 40, 50, 60, 70. Post-order 10, 25, 30, 20, 50, 70, 60, 40.
7. 40, 20, 30, 25: **4** comparisons.
8. Each key is larger than the last, so each goes to the right: the tree is one chain. A search may visit every node, $O(n)$, not $O(\log n)$.
9. $A \cup B = \{1, 3, 4, 5, 7\}$; $A \cap B = \{3, 5\}$; $A - B = \{1, 7\}$.
10. $12 \bmod 7 = 5$: slot 5. $19 \bmod 7 = 5$: taken, slot 6. $5 \bmod 7 = 5$: 5 and 6 taken, wrap round to slot 0. $26 \bmod 7 = 5$: 5, 6, 0 taken, slot 1.
11. More slots are full, so more keys collide; chains grow longer or probing visits more slots, and a search tends towards $O(n)$.

Exam tips

- When asked to "describe" an ADT, give its operations and its rule (LIFO, FIFO, ordered, no duplicates), not code.
- Linked-list insertion answers must give the pointer changes **in order**: new node first, then the old pointer.
- Draw BSTs neatly with smaller keys on the left; check by reading the in-order traversal, which must come out sorted.
- Learn the three traversals as node position: **pre** before the children, **in** between them, **post** after them.
- For hash tables, show the arithmetic ($k \bmod n$) for every key, and wrap round from the last slot to slot 0 when probing.
- Compare structures by Big O of the operation the question names: access, search, insert or delete.

Case study

International Baccalaureate · IB Diploma · Computer Science · HL

Scope and prerequisites

This handout teaches how to prepare for and answer questions on the **case study** 案例研究, for the course first assessed in 2027. All of it is SL and HL content.

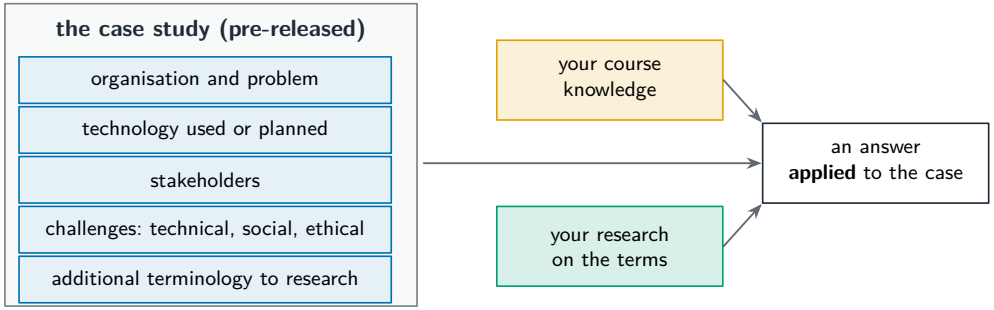
Each year the IB publishes a case study some months before the examinations, and some examination questions are based on it. Your teacher will give you the current case study and tell you which paper it is examined in; this handout teaches the method, using a short fictional case written for practice.

Prerequisites: Themes A and B. A case study can draw on any part of the course.

What the case study is

A scenario published in advance

A case study is a few pages about one real-world use of computing: a hospital, a farm, a transport network or a game studio, for example. It usually describes:



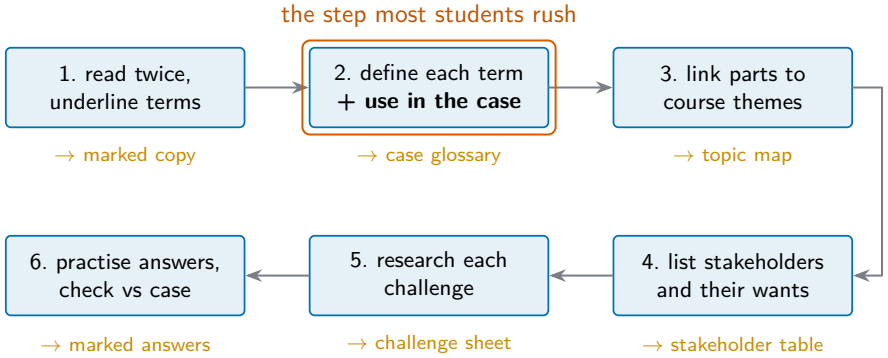
What the case contains and what you do with it

- the **organisation** and the problem it wants to solve;
- the **technology** it uses or plans to use;
- the **stakeholders** 利益相关者: the people and groups affected, such as users, staff, managers and the public;
- the **challenges** it faces, technical, social and ethical;
- a list of **additional terminology** 补充术语: terms that are outside the course but that you are expected to research.

The questions then test whether you can **apply** your course knowledge and your research to that scenario. A correct definition that never mentions the scenario usually earns little.

How to study it

You have months, not hours. Work through the case in stages and keep everything in one notebook or document.



Six stages, six products

Step	What you do	What you produce
1	Read the case twice; underline every technical term	a marked copy
2	Look up each term; write a definition in your own words and **how it is used in the case**	a case glossary
3	Link each part of the case to the course themes	a topic map
4	List the stakeholders and what each one wants	a stakeholder table
5	For each challenge, research how it could be solved, with advantages, disadvantages and alternatives	a challenge sheet
6	Write answers to practice questions, then check every claim against the case text	marked practice answers

Step 2 is the one most students rush. "Edge computing: processing data near where it is collected" is a definition; adding "in the case, each locker checks QR codes itself, so it works when the 4G link is down" makes it **case knowledge**.

A practice case: smart library lockers

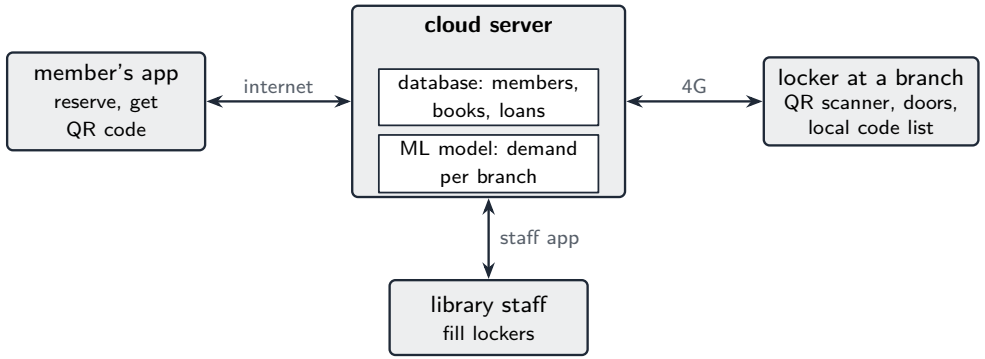
The scenario

This case is fictional and written for practice.

A city library has eight branches. Many members work long hours and cannot visit

while the branches are open. The library plans to install **smart lockers** outside each branch so that members can collect reserved books at any time.

A member reserves a book in the library's app. When staff have placed the book in a locker, the app shows a **QR code**. At the locker, the member scans the code and one door opens. Each locker connects to the library's **cloud server** over a 4G mobile network. The server holds a database of members, books and loans. A **machine learning** model on the server predicts how many reservations each branch will receive, so that staff can move stock in advance.



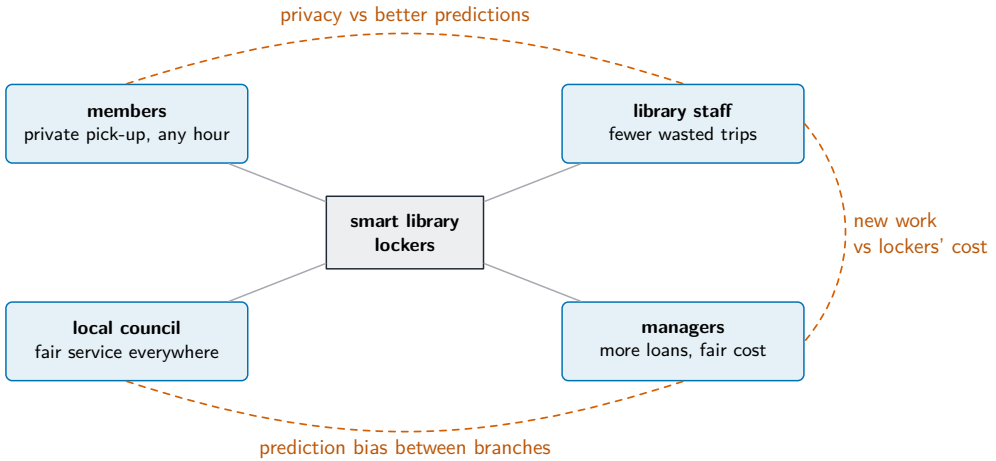
The smart-locker system: app, cloud server, lockers and staff

The library's challenges are:

1. Lockers must keep working when the 4G connection fails.
2. Reading history is personal data, but it would improve the predictions.
3. The model must not give poorer service to branches in poorer areas.
4. A QR code must not be usable by anyone except the member it was issued to.

Stakeholders

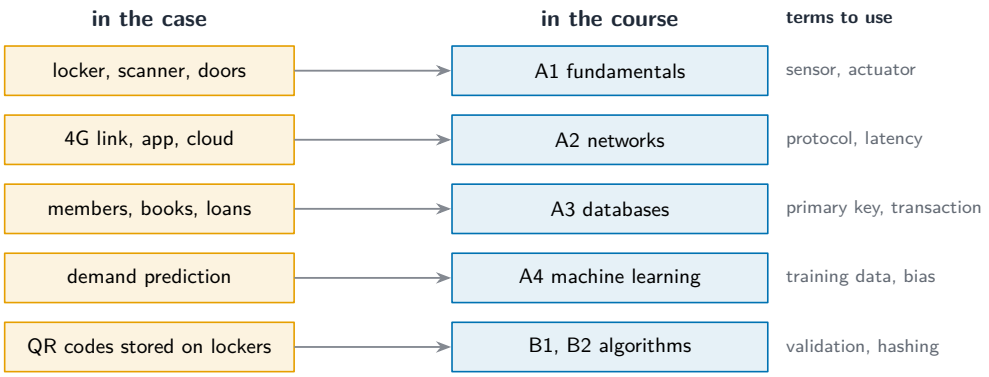
Stakeholder	What they want	A possible conflict
members	easy, private collection at any hour	privacy against better predictions
library staff	fewer wasted trips; clear tasks	new work filling lockers
library managers	more loans at a fair cost	cost of lockers and 4G
local council	a fair service in every area	prediction bias between branches



Who wants what, and where they clash

Linking the case to the course

Case feature	Course theme	Terms to use
locker, scanner, doors	A1 computer fundamentals	sensor, actuator, microcontroller
4G link, app, cloud server	A2 networks	protocol, latency, encryption
members, books, loans	A3 databases	primary key, relational database, transaction
demand prediction	A4 machine learning	training data, bias, supervised learning
QR codes, codes stored on the locker	B1, B2 algorithms and programming	validation, hashing, data structure



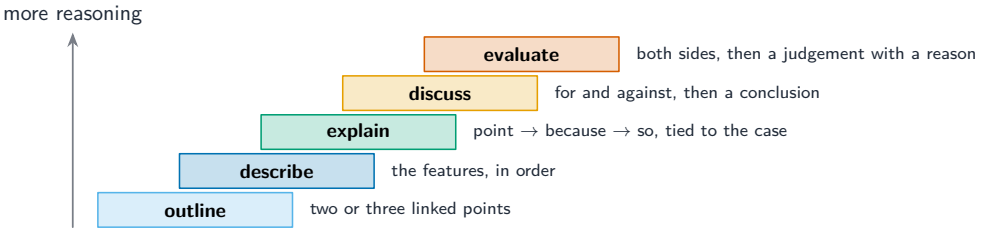
A topic map shows where questions come from

A topic map like this shows you where questions can come from, and which terms an examiner expects to see.

Answering case-study questions

Command terms

Every question starts with a **command term** 指令词 that tells you what kind of answer is wanted. Learn the IB's meaning of each.

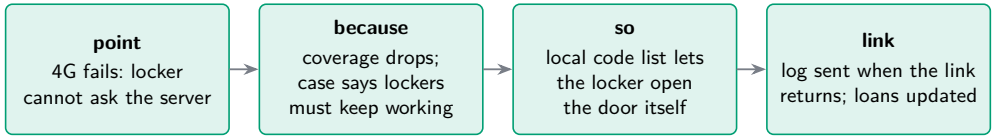


Command terms climb in demand

Command term	What it asks	Shape of a good answer
outline	a brief account	two or three linked points
describe	a detailed account	the features, in order
explain	a detailed account **with reasons or causes**	point → because → so, linked to the case
discuss	a balanced review of arguments	points for and against, then a conclusion
evaluate	weigh strengths and limitations	both sides, then a judgement with a reason

Worked example: an explain question

Explain why each locker should store a list of valid codes locally. (4 marks)



each step adds one idea **and** ties it to the case: 4 marks, not 1

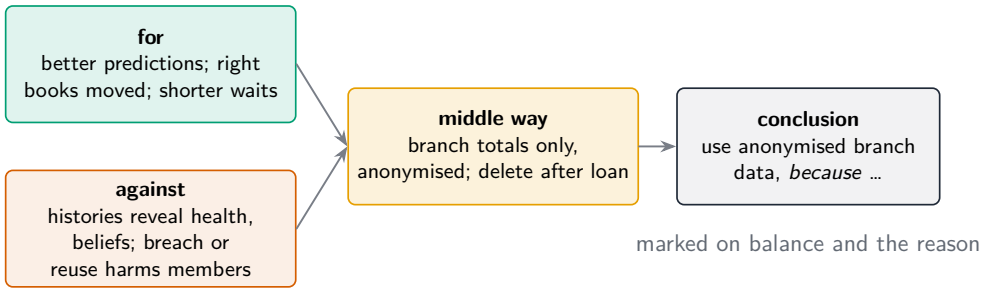
Four linked steps, four marks

1. **Point.** If the 4G link fails, the locker cannot ask the server whether a code is valid.
2. **Because.** Mobile coverage outside a building can drop, and the case says lockers must keep working when it does (challenge 1).
3. **So.** With a local list, received while the link was up, the locker can check the code itself and open the door; members are not turned away.
4. **Link.** When the link returns, the locker sends its log of openings to the server, so the loan records in the database are brought up to date.

Each step adds one idea **and** ties it to the case. An answer that says only "so it works offline" makes one point.

Worked example: a discuss question

Discuss whether the library should use members' reading histories to improve its demand predictions. (6 marks)



The shape of a discuss answer

For. Reading history shows which kinds of book each branch's members borrow, so the model could predict reservations more accurately; staff would move the right books in advance, and members would wait less.

Against. Reading history is personal and sensitive: it can reveal a member's health, beliefs or politics. If the database were breached, or the data used for another purpose, members would be harmed, and some might stop borrowing certain books.

A middle way. The model needs totals per branch, not individuals. The library could keep counts per branch and per subject, with no member IDs (**anonymised** 匿名化 data), and delete individual histories after the loan ends.

Conclusion. The library should use **branch-level, anonymised** data only, because it gives most of the accuracy benefit without the privacy risk, and it should tell members what is collected and why.

A discuss answer is marked on balance and on the conclusion. A conclusion that only repeats one side, or one that states no reason, loses marks.

Common weaknesses

- **Generic answers.** "Encryption keeps data safe" could be written about any system; say **what** is encrypted in the case and **from whom**.
- **Definitions without use.** Defining a term earns little unless the question asks for a definition.
- **One-sided discussions.** Give arguments on both sides before you judge.
- **New facts.** Do not invent details that the case does not give; if you must assume something, say so.

Practice

1. State **two** stakeholders in the smart-locker case and what each one wants from the system.
2. Identify the course theme that each of these belongs to: (a) the QR scanner; (b) the 4G link; (c) the table of loans.
3. Outline why a QR code shown in the app should change for each reservation.
4. Explain how training the demand model mainly on data from busy central branches could lead to unfair service.
5. Explain **one** advantage and **one** disadvantage of storing the database on a cloud server rather than on a server in the library.
6. Discuss whether the lockers should also accept a member's library card instead of a QR code.

Answers.

1. Any two, e.g. members: easy collection at any hour, with privacy; staff: clear tasks and fewer wasted trips; managers: more loans at a fair cost; the council: a fair service in every area.
2. (a) A1 computer fundamentals (an input device, a sensor); (b) A2 networks; (c) A3 databases.
3. A code used once and then made invalid cannot be reused if someone photographs or copies it, so only the member with the current code can open that door.
4. The model learns patterns from the data it is given; with few examples from quieter branches, its predictions for them are poorer, so the right books are moved there less often and those members wait longer. Fix: balance the training data and check accuracy for each branch.
5. Advantage: the cloud provider handles backups, uptime and scaling, so the library needs no server room or staff to run one. Disadvantage: the system depends on internet access, and personal data is stored by a third party, so the contract must cover security and where the data is kept.
6. For: a card is familiar and works without a phone. Against: a lost or stolen card would open any locker reserved by that member, unless a PIN is added. Conclusion with a reason, e.g. accept the card **with a PIN**, so members without smartphones are not excluded and a lost card alone is not enough.

Exam tips

- Learn the case **before** the exam: glossary, topic map, stakeholders, and an answer plan for each challenge.
- Every answer should name something from the case: a stakeholder, a device, a challenge.
- Read the command term first: "explain" needs reasons; "discuss" and "evaluate" need both sides and a conclusion.
- Use the course terms the examiner expects (latency, primary key, bias, encryption), each tied to the scenario.
- Do not add facts the case does not state; mark any assumption you need.

Computational solution

International Baccalaureate · IB Diploma · Computer Science · HL

Scope and prerequisites

This handout teaches the work behind the internal assessment, the **computational solution** 计算解决方案, for the course first assessed in 2027. All of it is SL and HL content.

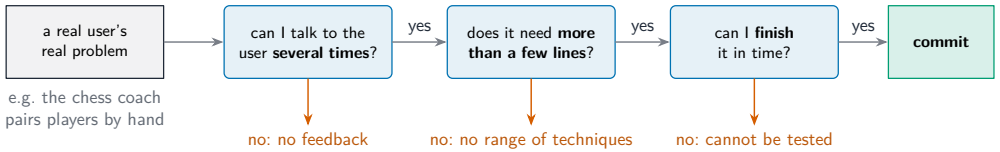
Your teacher will give you the current assessment criteria, word limit and deadlines; this handout does not repeat them, because they belong to the IB guide. It teaches how to do each stage well, with one fictional project as a running example.

Prerequisites: Theme B, especially B2 programming. The example project is invented for practice.

Choosing a problem

A real problem with a real user

A good project starts with a **real problem** for a real person or group, the **end user** 最终用户, whom you can talk to while you work. "A program that does maths" is not a problem; "the chess club's coach spends two hours each week pairing players by hand" is.



Three checks before you commit

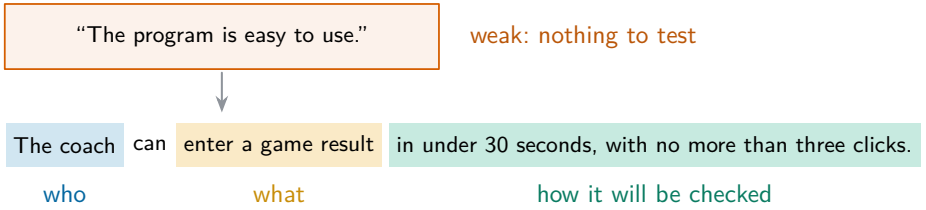
Check three things before you commit:

Question	Why it matters
Can I talk to the user several times?	you need their requirements and their feedback
Does it need more than a few lines of code?	you must show a range of techniques
Can I finish it in the time given?	an unfinished product cannot be tested

Running example. A school's chess club coach wants a program that records results, keeps a rating for each player and suggests fair pairings for each week's games.

Success criteria

Success criteria 成功标准 are the specific, testable things the finished product must do. You agree them with the user, and you **test against every one** at the end, so each must be clear enough to test.



Who, what, and how it is checked

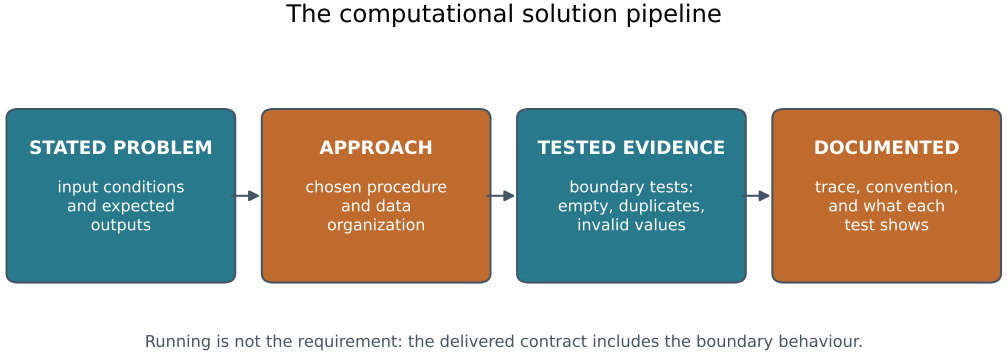
Weak criterion	Strong criterion
The program is easy to use.	The coach can enter a game result in under 30 seconds, with no more than three clicks.
It stores the players.	The program stores each player's name, year group and rating, and keeps them after it is closed.
Pairings are fair.	Each week, players are paired so that no two players meet twice in a row and rating gaps are as small as possible.
It is secure.	Only the coach, with a password, can change results.

A strong criterion names **who**, **what** and **how it will be checked**. Six to ten criteria are usual for a project of this size.

Planning and design

Decompose before you code

Break the problem into parts, then design each part: the data, the algorithms and the screens.



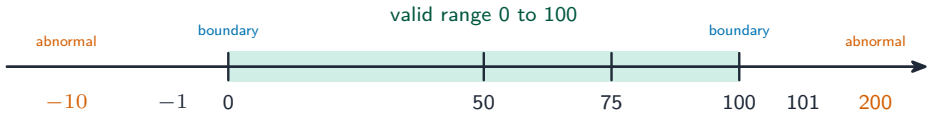
A computational solution runs from a stated problem through an approach and tested evidence to documentation

Running example. The chess program splits into: storing players and results; updating ratings after each game; making the week's pairings; the coach's screens.

Part	Design product
data	a table of the fields, their data types and an example; a file or database layout
algorithms	a flowchart or pseudocode for each non-trivial process, such as pairing
screens	a sketch of each screen, checked with the user
testing	a test plan, written **before** coding

A test plan

A **test plan** 测试计划 lists each test, the data used and the expected result. Use three kinds of test data:



Test data for a 0–100 field: normal inside, extremes at the boundaries, abnormal outside

- **normal data** 正常数据: typical values that should be accepted;
- **boundary data** 边界数据: values at the edge of what is allowed;
- **invalid data** 无效数据: values that should be rejected.

Test	Criterion	Data	Type	Expected result
1	result entry	win for player A	normal	A's rating rises; B's falls
2	stores year group	year group 7 (lowest allowed)	boundary	accepted
3	stores year group	year group 14	invalid	rejected with a message
4	pairing	9 players	boundary (odd number)	one player gets a bye; no pair repeats
5	security	wrong password	invalid	results cannot be changed

Each test points to a success criterion, so at the end you can show which criteria are met.

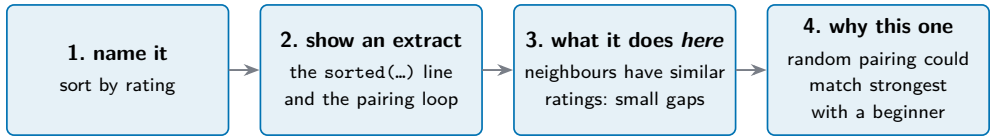
Development

Explaining your techniques

The product must work, but the write-up must also show **how** it works and **why** you chose each technique. For each important technique:



A programmer writes the code and tests it as they go



a list of every Python feature used is not an explanation

Explaining one technique

1. name it;
2. show a short, relevant extract of your code;
3. explain what it does in **this** product;

4. say why you chose it over an alternative.

Running example. Pairing sorts the players by rating, then pairs neighbours, so that rating gaps stay small:

```
def make_pairings(players):
    """Pair (name, rating) tuples; return the pairs and the bye."""
    ranked = sorted(players, key=lambda p: p[1], reverse=True)
    bye = None
    if len(ranked) % 2 == 1:
        bye = ranked.pop()      # lowest rating sits out
    pairs = []
    for i in range(0, len(ranked), 2):
        pairs.append((ranked[i][0], ranked[i + 1][0]))
    return pairs, bye
```

Explanation. Sorting by rating puts players of similar strength next to each other, so pairing neighbours gives the smallest rating gaps; this meets the fairness criterion. With an odd number of players, the lowest-rated player gets a bye, which test 4 checks. A random pairing would be simpler, but it could match the strongest player with a beginner.

A list of every Python feature you used is not an explanation. Choose the techniques that matter, such as a sort, a search, a file or database, validation, a class, and explain them well.

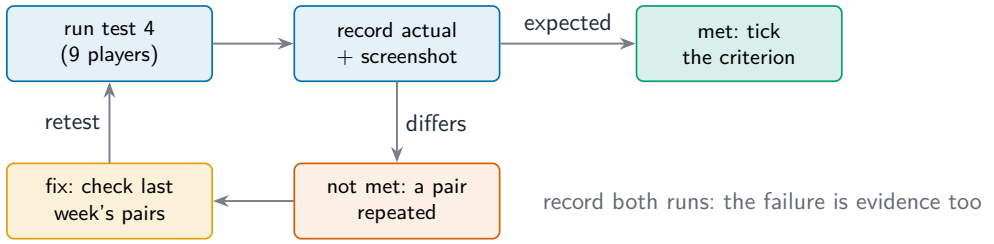
Using other people's work

If you use a library, a code example from a website, or an **AI tool** 人工智能工具, say where it came from and what you changed. This is part of **academic integrity** 学术诚信: work that is not yours must be **acknowledged** 注明出处. Your own explanation of how the code works shows that you understand it.

Testing and evaluation

Testing against the criteria

Carry out the test plan and record the **evidence**: a screenshot or a short video for each test, with the actual result beside the expected one. A failed test is not a disaster: fix the code, test again, and record both.

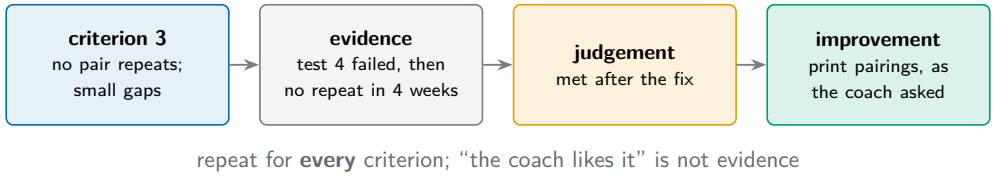


A failed test is evidence too

Test	Expected	Actual	Met?
3	year group 14 rejected	rejected: "Year group must be 7 to 13"	yes
4	9 players: one bye, no repeat pair	bye given; one pair repeated from last week	**no**: fixed by checking last week's pairs, then retested

Evaluation

An **evaluation** 评价 judges the finished product against **each** success criterion, using the test evidence and the user's feedback.



Evaluate criterion by criterion

Weak	Strong
The program works well and the coach likes it.	Criterion 1 is met: in testing the coach entered results in 12 seconds on average (test 1). Criterion 3 was only partly met: the first version repeated a pair (test 4); after the fix, no pair repeated over four trial weeks.

Then suggest **specific improvements**, linked to the evidence: "add an option to print the pairings, as the coach asked in feedback" is better than "add more features".

Practice

1. Rewrite this success criterion so that it can be tested: "The program should be fast."
2. For a criterion "The program accepts a player's age from 11 to 18", give one normal, one boundary and one invalid test value.
3. Explain why the test plan is written before the code.
4. A student's write-up lists "variables, loops, if statements, functions". Explain why this earns little, and what should be written instead.
5. The `make_pairings` function above is given 6 players with ratings 1500, 1420, 1610, 1380, 1550, 1490. List the pairs it returns.
6. A student used an AI tool to write a validation function. State what the student must do.

Answers.

1. E.g. "The pairings for up to 40 players appear within 2 seconds of the coach pressing *Pair*."
2. Normal: 15. Boundary: 11 or 18. Invalid: 10, 19, or a value such as "twelve".
3. It fixes, from the success criteria, what must be shown to work, so testing is planned rather than chosen after the fact to suit the product; it can also shape the design.
4. It names features without showing why they were used or how they solve the user's problem. Instead: choose a few important techniques, show a short code extract of each, and explain what it does in this product and why it was chosen.
5. Sorted from the highest: 1610, 1550, 1500, 1490, 1420, 1380. Pairs: (1610, 1550), (1500, 1490), (1420, 1380); no bye, as 6 is even.
6. Acknowledge the tool and what it produced, show what was changed, and explain in their own words how the function works.

Exam tips

- Pick a problem with a real user you can consult, and agree testable success criteria with them early.
- Every criterion → a test in the plan → evidence in testing → a judgement in the evaluation.
- Explain a few techniques well: code extract, what it does here, why it was chosen.
- Acknowledge every library, code source and AI tool.
- Keep the current assessment criteria from your teacher beside you, and check each section of your work against them.