Course Companion

# AP Computer Science Principles

Every topic handout in one volume

全部专题讲义合集

exercises.lol

## Contents 目录


1. **Creative Development** ..... 2
2. **Data** ..... 12
3. **Algorithms and Programming** ..... 22
4. **Computer Systems and Networks** ..... 40
5. **Impact of Computing** ..... 50






## Collaboration



*A jigsaw in progress: collaboration and modular design piece the solution together*

In 1999 NASA lost the Mars Climate Orbiter, a spacecraft that had cost well over a hundred million dollars. The code was not badly written. One team's software gave its numbers in US units, and another team's software read them as metric units. Each piece worked alone; the failure was between the people. This unit is about how programs are really made: by groups, for users, in repeated cycles, with errors found and fixed along the way.

Computing is a **collaborative** 协作 activity. Working in a team brings more **perspectives** 视角, catches more errors, and produces better programs than working alone. Good collaboration uses **consensus building** 达成共识, clear communication, and each member's strengths. **Pair programming** 结对编程 – two people at one computer, one typing and one reviewing – is a common practice. On the exam, you should be able to explain how collaboration improved a program (more ideas, fewer **bugs** 缺陷, wider **testing** 测试).

A **computing innovation** 计算创新 is anything new that has a program as an essential part. It can be physical (a self-driving car), software (a photo editor), or a concept (online shopping). When a question says "computing innovation", any of the three will do.

**Why different people matter.** People with different backgrounds notice different problems. A team that includes different perspectives is more likely to avoid **bias** 偏见, because someone in the room will ask "what about users like me?". Developers

also talk to **users** 用户 directly: what users say helps the team understand what the program is for and who it must work for.

**Skills and tools.** Good teams practise communication, consensus building, **conflict resolution** 解决冲突 and **negotiation** 协商. Online tools such as shared documents and shared code stores let people give feedback from anywhere.

> **Worked example.** Two students build a quiz app. Mei writes the scoring procedure while Tom reviews each line as she types; later Tom asks five classmates to try the app, and two of them find that a blank answer crashes it. How did collaboration help?

1. Tom reviews the code as it is typed (pair programming), which can catch errors early.
2. The classmates are **users** with different habits; their tests found a case neither programmer had thought of, the blank answer.
3. Each benefit is named **and** tied to the program: "more testers found the blank-answer crash" earns credit where "teamwork is good" does not.

> **Worked example: a disagreement.** Two teammates disagree: Sara wants a timer on each quiz question, and Li says a timer will worry nervous users. Which is the best next step?

- Sara and Li vote and the loser gives way: quick, but it ignores the users.
- Each builds their own version and they never share it: the team splits.
- **They ask some users, then agree a rule together**, for example a timer that users can turn off.

The third is **consensus building**: the decision comes from evidence, and both people accept it.

## Program Function and Purpose

Every program is written for a **purpose** 目的 – it solves a problem or pursues an interest. A program takes **input** 输入, processes it, and produces **output** 输出. Inputs can come from a user, a device, a file, or another program; outputs can be visual, audible, textual, or a signal to a device. Being able to state a program's purpose, and describe its inputs and outputs clearly, is a core skill (and part of the Create performance task).

Some words the exam uses exactly:

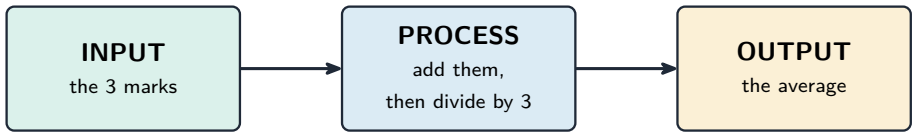
- A **program** 程序 (also called **software** 软件) is a collection of statements that does a task when a computer runs it. A **code segment** 代码段 is one part of a program.
- A program's **behavior** 行为 is how it acts while it runs, usually described by how a user interacts with it.



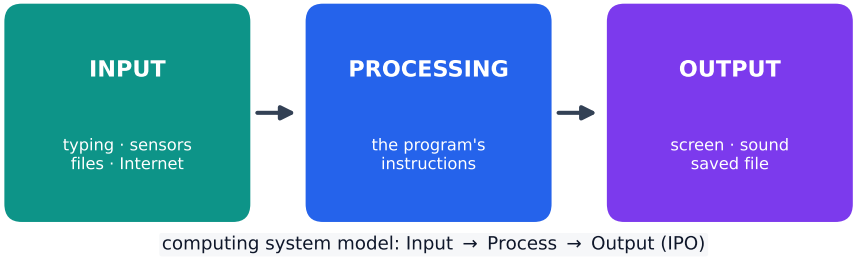
exercises.lol exercises.com.cn

- Input and output can each be **tactile** 触觉的 (touch, vibration), audio, visual or text.
- Output usually depends on the input, and it can also depend on **prior state**: values the program already holds.

You can describe a program broadly (what it does), or in more detail (what it does **and** how its statements do it). A written response usually needs the second.




*Every program decomposes into input, processing, and output*




*Every program follows the input-processing-output model*

> **Worked example.** A step-counter app reads movement from the phone's motion sensor, counts steps, and shows the day's total and a badge when the user reaches 10,000.

| Ask | Answer |
|---|---|
| Purpose: **why** it exists | to help users stay active by tracking their walking |
| Function: **what** it does | counts steps and compares the total with a goal |
| Input | readings from the motion sensor; the user's goal |
| Output | the step total on screen; a badge at 10,000 |

The badge also shows **prior state** at work: the same step of input gives a badge only if the stored total has just reached the goal.

A common mistake is to give the function as the purpose: "it counts steps" says what the program does, not the problem it solves.

## Program Design and Development



*A programmer debugging at a multi-monitor workstation – iterative design and testing*

Programs are built through an **iterative** 迭代的 process, not in one straight line: investigate the problem and users, design (often with a **diagram** 图示 or written plan), **implement** 实现 in code, and test – then repeat. A large problem is broken into smaller pieces (**decomposition** 分解). **Comments** 注释 and clear naming document the design so others (and your future self) can understand it. Development is **incremental** 增量的 – build and test a small piece, then add the next.

Common phases are investigating and reflecting, designing, building a **prototype** 原型 (an early working model), and testing. The order can be planned in advance or more exploratory, and a team may go back to an earlier phase at any time.

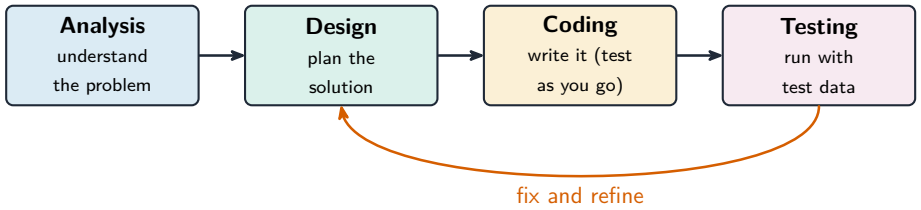
> **Worked example: iterative or incremental?** The two words sound alike, and the exam separates them.

| What the team does | Word | Why |
|---|---|---|
| Testers say the text is too small, so the team returns to the design and changes it | iterative | feedback sends them back to revise |
| The team makes questions display, tests that, then adds scoring, tests that, then adds a timer | incremental | one piece at a time, each working before it joins the whole |

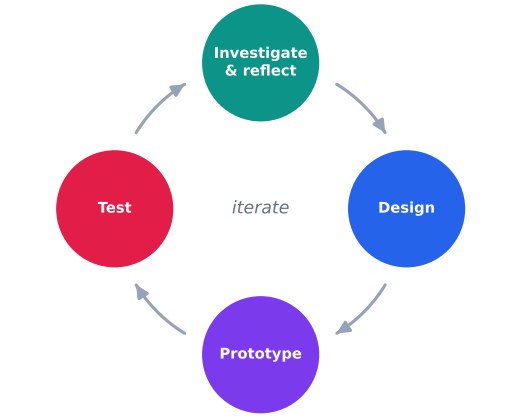
A real project usually does both.



exercises.lol exercises.com.cn




*The stages of program development, with testing feeding back to fix and refine*




*Software is built by an iterative, incremental development process*

### Investigating what users actually need

Before any code is written, the developer investigates the problem and the people who will use the program. Ways to do that:

- **surveys** 调查问卷 sent to potential users, which collect data from many people quickly;
- **interviews** 访谈 and direct **observation** 观察 of users doing the task by hand;
- **user testing** 用户测试, where people try an early version while the team watches;
- **studying existing solutions** to see what already works and what frustrates people.

The findings are turned into a design. Two artefacts do that: a **program requirements** 程序需求 list saying exactly what the program must do, and diagrams representing the layout of the **user interface** 用户界面 – sketches showing which controls appear where, and what each one does when used. Designing the interface on paper

first is cheaper than discovering after coding that the buttons are in the wrong place.

A program's **specification** 规格说明 defines its requirements. The design phase then plans how to meet them. It may include brainstorming, a **storyboard** 故事板 (a sequence of sketches of what the user sees), dividing the program into modules, and planning how it will be tested.

### ■ Events, and programs that wait

Not every program runs straight through from top to bottom. An **event** 事件 is generated when a key is pressed, a mouse is clicked, a program is started, or any other defined action occurs – and an event can change the **flow of execution** 执行流程 by triggering code attached to that event, called an **event handler** 事件处理程序. In many graphical programs, events wait in a queue until the current handler finishes.

This is why a program with a graphical interface can appear to be doing nothing: it is waiting for the next event. The order in which those events arrive depends on user actions and other event sources, so the same program can run its blocks in a different order each time it is used.

This style is called **event-driven programming** 事件驱动编程: statements run when they are triggered, not one after another from the top. The event also supplies input data, such as which key was pressed.

### ■ Documentation and credit

**Program documentation** 程序文档 is a written description of what a code segment, event, procedure or program does and how it was developed. **Comments** are one form: they are written for people and do not change how the program runs. Not every programming environment allows comments, so documentation sometimes lives in a separate file.

Write documentation **throughout** development, not at the end: it helps you and your teammates keep the program correct.

Code from someone else must be **acknowledged**: say in the documentation where it came from, with the original author's name if you know it. This applies to code written with a partner too.

## Identifying and Correcting Errors

A **bug** is an error in a program; **debugging** 调试 is finding and fixing it. The exam names four kinds:

- a **syntax error** 语法错误 breaks the language's rules, so the program will not run;
- a **runtime error** 运行时错误 crashes the program while it runs (e.g. dividing by zero);
- a **logic error** 逻辑错误 lets it run but gives the wrong result;
- an **overflow error** 溢出错误 happens when a number is outside the range the computer can handle.



exercises.lol exercises.com.cn

| count | total | output |
|---|---|---|
| 1 | 5 | |
| 2 | 12 | |
| 3 | 20 | 20 |

*A trace table records each variable's value as the program runs, to find bugs*

Find bugs by **test cases** 测试用例 with different inputs, adding **print statements** 打印语句 to see values, and **hand-tracing** 手动跟踪 the code. A **debugger** 调试器 stops the program at a chosen line so you can look at each variable, and a **visualization** 可视化 such as a chart can make a wrong pattern easy to see. Choose the test inputs deliberately: they should demonstrate the different expected outcomes **at or just beyond the extremes** – the minimum and maximum values the program should accept, and a value just outside each of them. A program that works on ordinary data very often fails on an empty list, a zero, or a value one past the end of a range, so those are the inputs worth trying first. Fixing one bug at a time and re-testing is the reliable method.

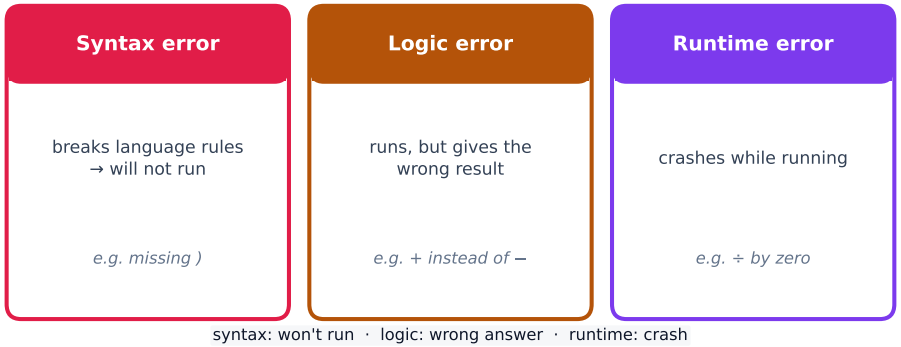
> **Worked example: choosing test inputs.** A club sign-up page should accept ages 13 to 18. The requirements tell you the correct result, so you can choose inputs and say what you expect:




*Test both ends of the range, and one step past each end*

Five tests are enough: 12, 13, 15, 18 and 19. If the code says `age > 13`, only the test with 13 shows the fault; age 15 passes either way.

**Exam skill:** be able to name the type of an error and describe a testing strategy that would catch it – a recurring multiple-choice and Create-task theme.




*Three common kinds of programming error: syntax, logic, and runtime*

**Worked example.** A program meant to print the average of two numbers instead runs `avg = a + b / 2`. Tracing the **order of operations** 运算顺序, `/` runs before `+`, so it computes $a + \frac{b}{2}$ rather than the average. Add parentheses to fix it: `avg = (a + b) / 2`. Testing with $a = 4$, $b = 6$ confirms the fix – the buggy line gives $4 + 3 = 7$, the corrected line gives $\frac{10}{2} = 5$. Testing with known inputs is exactly how you find and confirm a **logic error**.

**Worked example: name the error.**

| What happens | Type | Reason |
|---|---|---|
| `DISPLAY(score` has no closing bracket; nothing runs | syntax | a language rule is broken |
| `list[0]` stops the program with an error message | run-time | AP lists start at index 1, and this only fails while running |
| A discount of 10% is taken as 10 dollars | logic | the program runs and the answer is wrong |
| A 4-bit counter at `1111` has 1 added | overflow | 16 is outside the range |

### ■ Written responses about your own program

Section II of the exam gives 60 minutes for two questions with four written prompts, all about the program you submitted for the Create task. You see your own code segments in your **Personalized Project Reference**, so answers must be about **your**



exercises.lol exercises.com.cn

program, not programs in general. Imagine this procedure is in yours:

```
PROCEDURE Average(numbers)
{
   sum ← 0
   FOR EACH n IN numbers
   {
      sum ← sum + n
   }
   RETURN(sum / LENGTH(numbers))
}
```

Past prompts have asked for the following.

1. **A modification that causes a logic error, and how behavior changes.** "If `sum ← 0` is changed to `sum ← 1`, the procedure still runs, but every total is one too large. `Average([2, 4])` returns 3.5 instead of 3."
2. **A call the procedure accepts but handles wrongly.** "`Average([])`: the empty list is accepted, but `LENGTH(numbers)` is 0, so the last line divides by zero and a run-time error stops the program." State the call, the wrong behavior and the reason: three separate things.
3. **Documentation for another programmer.** "A comment above the procedure saying that `numbers` must not be empty. Another programmer who reuses `Average` then knows to check for an empty list before calling it."
4. **An unexpected input.** Name one real input to your program and say what your program actually does next. If it rejects the input, say which line does that.

**Common mistakes.** Describing what a modification is without saying what changes when the program runs. Giving a call without specific argument values. Answering about a different procedure from the one the prompt names.

## Practice

Answer each before reading the answers below.

1. A step-counter app counts steps and shows a daily total so users can meet a goal. State the program's purpose and one of its functions.
2. Classify each error as syntax, logic, run-time or overflow: (a) a missing closing bracket; (b) an average computed as `sum / (count + 1)`; (c) dividing by a variable that is zero while the program runs; (d) adding 1 to the largest value a fixed number of bits can store.
3. A procedure returns `"pass"` for scores of 60 or more. Choose four test inputs and explain your choices.
4. Give one benefit of collaboration when designing a program.
5. Distinguish iterative development from incremental development.

6. Give two reasons for writing comments and other program documentation.

## Answers

1. Purpose: help users meet a daily activity goal. Function: for example, it adds 1 to the total each time a step is detected, or displays the total.
2. (a) Syntax; (b) logic; (c) run-time; (d) overflow.
3. 59, 60 and 61 test the boundary, where errors are most likely; an extreme such as 0 or 100 checks the limits of valid input.
4. Different perspectives help find errors and avoid bias, so the program works for more users (for example, pair programming catches mistakes earlier).
5. Iterative development repeats design, test and refine cycles, revising the program after feedback. Incremental development builds and tests the program in small pieces, adding one working part at a time.
6. It explains what code does and why, so others (and the author later) can understand, maintain and debug it; it also records sources of borrowed code.

## Exam tips

- Assessment includes multiple-choice questions and the **Create** component, with written responses on the AP exam. Explain your reasoning clearly, not just your result.
- Know the benefits of collaboration and how diverse perspectives reduce bias in a program.
- Use precise vocabulary (iterative, incremental, program requirements) when you describe a design process.
- Test at and just beyond the extremes of the input range, and say the expected output for each test.
- Give and take **feedback** 反馈 constructively; credit collaborators and sources.
- Break a large problem into smaller modules that a team can build in parallel.



exercises.lol exercises.com.cn

# Data

AP Computer Science Principles

## Binary Numbers



*Binary digits on a display – all digital data is ultimately stored as 0s and 1s*

On 4 June 1996 the first Ariane 5 rocket was destroyed about 37 seconds after launch. Its guidance software tried to put a large number, stored in 64 bits, into a space of only 16 bits. The number did not fit, the guidance system failed, and years of work ended in the sky. The way a computer stores a value has limits, and those limits have real results. This unit explains how data become bits, and how bits become knowledge.

Computers store everything as **bits** 位 – each a `0` or `1`. A group of 8 bits is a **byte** 字节. Numbers are stored in **binary** 二进制 (base 2), where each place is a power of two $(1, 2, 4, 8, 16, \dots)$ instead of the powers of ten in **decimal** 十进制. For example, binary `1011` is $8 + 2 + 1 = 11$.

The rule behind this: number the positions from the right, starting at 0. Each bit is worth its value (0 or 1) times $2^{\text{position}}$. Decimal works the same way with powers of 10.

| place value | 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |
|---|---|---|---|---|---|---|---|---|
| bits | **1** | 0 | 0 | **1** | 0 | **1** | **1** | 0 |

$150 = 128 + 16 + 4 + 2 \;\Rightarrow\; \texttt{10010110}$

*An 8-bit place-value chart: the 1s sit under the values that add to the number*

**Worked example.** To convert binary `1101` to decimal, write the place values `8 4 2 1` under the bits `1 1 0 1` and add the ones that have a `1`: $8 + 4 + 0 + 1 = 13$. Going the other way, convert `19` to binary by subtracting the largest power of two that fits: $19 - 16 = 3$, then $3 - 2 = 1$, then $1 - 1 = 0$, so the bits sit at the `16`, `2`, and `1` places $\rightarrow$ `10011` (check: $16 + 2 + 1 = 19$).

**Worked example (compare and order).** Put these in order, smallest first: binary `1100`, decimal 11, binary `1010`.

Change them all to decimal: `1100` is $8 + 4 = 12$ and `1010` is $8 + 2 = 10$. The order is `1010` (10), decimal 11, `1100` (12). For two binary numbers of the same length there is a quicker way: read from the left, and at the first place they differ, the one with the `1` is larger.

Because a computer has a **finite** number of bits, it can represent only a limited range of values. This causes two effects tested on the exam:

- **Overflow error** 溢出错误: a number too large for the available bits cannot be stored correctly.
- **Round-off (rounding) error** 舍入错误: many numbers with **decimals** (real numbers), such as 0.1, can only be **approximated**, because infinitely many real values must map onto finitely many bit patterns.

On the exam, the AP pseudocode has no fixed size for integers: they are limited only by the computer's memory. So expect overflow only when a question states a fixed number of bits. Round-off is easy to see in many real languages, where `0.1 + 0.2` gives `0.30000000000000004`.

**Worked example (how many bits?).** With $n$ bits there are $2^n$ different patterns; each extra bit **doubles** the count.

| Question | Working | Answer |
|---|---|---|
| Largest whole number in 4 bits | `1111` is $8 + 4 + 2 + 1$ | 15 |
| Patterns in 3 bits | $2^3$ | 8 |
| Bits to give 100 students different IDs | $2^6 = 64$ is too few; $2^7 = 128$ | 7 bits |
| Patterns gained from 6 bits to 7 bits | $128 - 64$ | 64 more, twice as many |

In a 4-bit system, $15 + 1$ needs `10000`, five bits; with only four, the result cannot be stored: an overflow error.

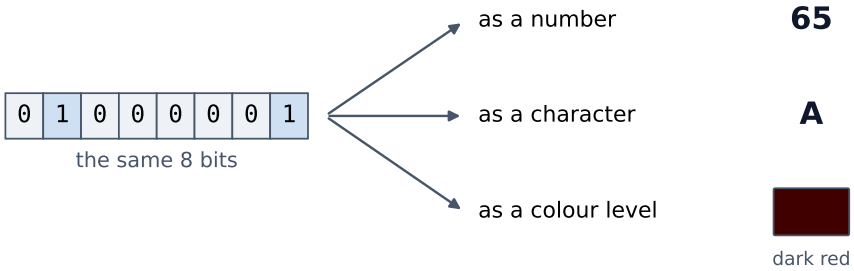
All data – text, images, sound – is ultimately encoded as binary. An image is a grid of **pixels** 像素, each stored as numbers for its colors; sound is stored as numbers sampled many times per second.



exercises.lol exercises.com.cn

## Bits as abstraction

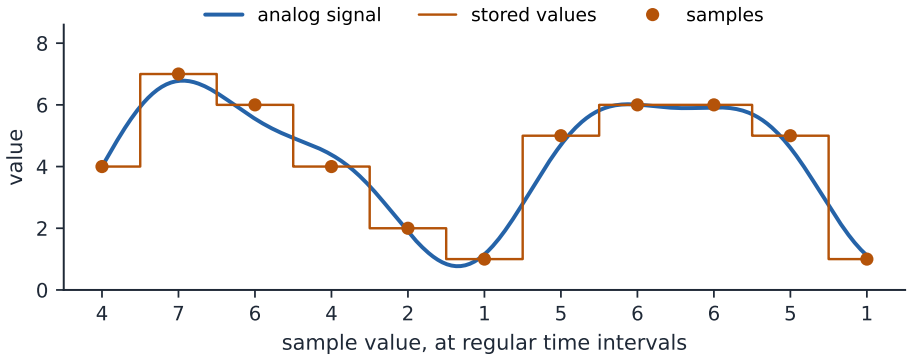
**Abstraction** 抽象 means reducing complexity by keeping the main idea and hiding details that do not matter for the question. When you see the letter A on a screen, you do not need to think about bits. Bits are grouped to stand for bigger ideas such as numbers, characters and colors, and **the same bits can mean different things in different contexts**:




*The bits do not say what they mean; the program that reads them decides*

## Analog and digital

**Analog** 模拟 data change smoothly, with no steps: the pitch of a song, the colors of a painting, the position of a runner. **Digital** 数字 data are stored as separate values. To store analog data, a device takes **samples** 采样: it measures the signal at regular intervals and stores each measurement as bits.




*Sampling: discrete measurements approximate the original signal*

The result is an approximation, so digital data of this kind are themselves an abstraction. More samples per second, and more bits for each sample, give a closer match, and a larger file.

**Worked example (the size of sound).** One minute of mono sound is sampled 44,100 times per second, with 16 bits per sample. Ignoring headers and compression, how large is the file?

1. Samples: $44100 \times 60 = 2646000$.
2. Bits: $2646000 \times 16 = 42336000$.
3. Bytes: $42336000 \div 8 = 5292000$, about 5.3 million bytes.

This is why sound and video are so often compressed.

## Data Compression

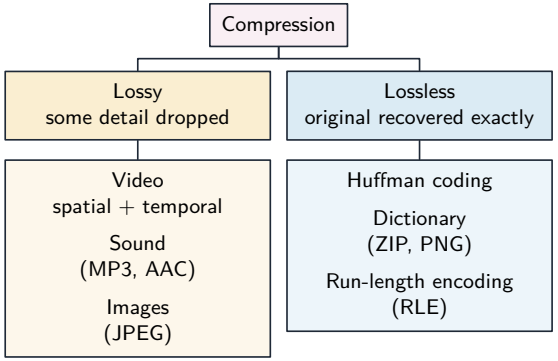


*Hard-disk platters and head: data compressed and stored as magnetic patterns*

**Compression** 压缩 reduces the number of bits needed to store or send data. Two kinds:

- **Lossless compression** 无损压缩 lets you restore the **exact** original data (used for text and programs, where every bit matters).
- **Lossy compression** 有损压缩 throws away some data to shrink the size **further** (used for photos, music, video, where a small quality loss is acceptable).






*Compression methods: lossless versus lossy, with common examples*

Choosing between them trades **size against fidelity**: lossless keeps everything but saves less; lossy saves more but loses detail permanently. Prefer lossless when the data must be exact.

| Situation | Choice | Reason |
|---|---|---|
| Sending a program file | lossless | one wrong bit can stop it working |
| Storing a medical scan | lossless | quality matters most |
| A live video call on a slow link | lossy | transmission time matters most |
| A website with thousands of photos | lossy | a small file matters more than perfect detail |

Questions of this kind ask which is **most important** in the situation: exact reconstruction, or size and speed.

**Worked example (run-length encoding).** A row of image pixels reads `WWWWWWBBBWWWW`, 13 symbols. **Run-length encoding** 游程编码 stores each run as a count and a symbol: `6W3B4W`, 6 symbols. Decoding gives back exactly `WWWWWWBBBWWWW`, so this is **lossless**. It saves space only when the data has long runs: `WBWBWB` becomes `1W1B1W1B1W1B`, twice as long. No lossless method can shrink every possible input.

How much a file shrinks depends on two things: how much **redundancy** 冗余 (repeated or predictable content) it has, and which algorithm is used. Notice also that **fewer bits does not always mean less information**: `6W3B4W` holds everything that the 13-symbol row held.



exercises.lol exercises.com.cn

## Extracting Information from Data

**Data** 数据 becomes useful when we extract **information** 信息 from it – patterns, trends, and answers to questions. Large data sets can reveal correlations a small one cannot, but data must be **cleaned** (fixing errors and inconsistencies) and often **transformed** or **filtered** first. A **correlation** 相关性 between two things does not prove that one **causes** the other – a key caution. **Metadata** 元数据 (data about data, like a photo's date and location) helps organize and search large collections.

Often one source is not enough to reach a conclusion, and data from several sources must be combined.

> **Worked example (what metadata can tell you).** A phone photo has this metadata: date, time, location, file size, camera model.

| Question | From the metadata alone? |
|---|---|
| Where and when was it taken? | Yes |
| How much storage does it use? | Yes |
| Who is in the photo? | No: that is in the data, the picture itself |
| Is the sky blue in the photo? | No: same reason |

Editing descriptive metadata such as the date or camera model need not change the image pixels. Metadata are data about other data and may be stored inside the same file. Some metadata, such as orientation, can affect how the picture is displayed.

> **Worked example (correlation is not cause).** A city's data shows that on days with more ice-cream sales, more people are treated for sunburn. Ice cream does not cause sunburn: both rise on hot, sunny days. The heat is a third factor behind both. To claim a cause you need more than a pattern in the data, such as an experiment that changes one thing and keeps the rest the same.

### ■ Why data are hard to work with

Data sets of every size bring the same problems:

| Problem | Example | What to do |
|---|---|---|
| Not uniform | `NYC`, `New York` and `new york` in one column | **clean** them: one spelling, same meaning |
| Incomplete | the age column is blank in some rows | find the data, or leave those rows out and say so |
| Invalid | age is `250` | check that values are possible |
| Separate sources | one file lists names, another lists ID numbers | find a way to match the rows |

**Cleaning data** 数据清洗 makes data uniform **without changing their meaning**. It would be wrong to "clean" a real low score up to the average.

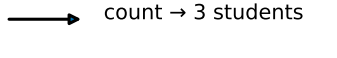


exercises.lol exercises.com.cn

**More data does not remove bias.** A city puts an online survey on its website to ask whether homes have good Internet. People without Internet cannot answer. Doubling the number of replies still leaves them out: the problem is **how** the data were collected, not how much there is.

Very large data sets may also need more than one computer, so **scalability** 可扩展性 matters: what a system can do with data depends on its computing power and storage.

| name | grade | club |
|---|---|---|
| Ana | 10 | robotics |
| Ben | 11 | music |
| Chen | 10 | music |
| Dara | 10 | robotics |
| Eli | 12 | robotics |




filter, then count, sum or average: questions answered from the data

*Filter the rows you need, then count, sum or average them*

## Using Programs with Data

Programs process data at scales humans cannot. Common operations are **filtering** 过滤 (keeping only rows that meet a condition), **cleaning** (removing errors), and **visualizing** 可视化 (charts and graphs that make patterns visible). Combining data from multiple sources can reveal more, but raises **privacy** 隐私 concerns. Interactive tools and visualizations let people explore data and draw their own conclusions.

Most data work is built from four actions:

| Action | Example |
|---|---|
| **Transform** every element | double each number in a list; add a parent's email to each student record |
| **Filter** | keep only positive numbers; keep only students in the band |
| **Combine or compare** | add up a list; find the highest GPA |
| **Visualize** | draw a chart of the results |

**Worked example (filter, then combine).** What does this display if `numbers` is `[3, -1, 4, -5]`?

```
total ← 0
FOR EACH n IN numbers
{
   IF (n > 0)
   {
      total ← total + n
   }
}
DISPLAY(total)
```

The `IF` filters out `-1` and `-5`; adding combines the rest. It displays $3 + 4 = 7$.

This work is **iterative**: a chart suggests a pattern, the pattern suggests a new question, and the program is changed to ask it. Programs can also group similar rows without being told the groups, called **clustering** 聚类, or put each row into a known category, called **classifying** 分类. Search tools and spreadsheets are everyday versions of the same ideas.

> **Worked example (what can the data answer?).** A school stores one row per student:

| Name | Grade | Transport | Distance (km) |
|---|---|---|---|
| Ana | 10 | bus | 6.5 |
| Ben | 11 | bike | 2.0 |
| Chen | 10 | bus | 8.0 |

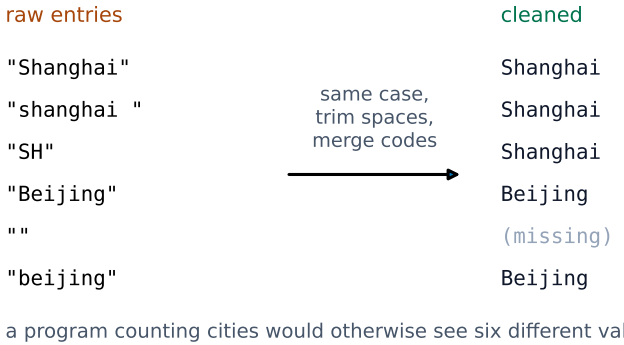
1. "What is the mean distance for students who come by bus?" **Can** be answered: filter Transport = bus, then average Distance: $\frac{6.5+8.0}{2} = 7.25$ km.
2. "How many grade 10 students cycle?" **Can** be answered: filter on two columns, then count: 0.
3. "Why do students choose the bus?" **Cannot**: no column records reasons.
4. "How long is each journey in minutes?" **Cannot**: time is not stored, and distance alone does not give it.

Exam questions of this kind turn on one check: is every piece of information the question needs in a column?

**Exam skill:** be able to explain how a program helps find information in a large data set, and why correlation shown in the data does not establish causation.



exercises.lol exercises.com.cn




*Cleaning data makes entries consistent before a program processes them*

## Practice

Answer each before reading the answers below.

1. Convert the binary number 10110110 to decimal, and the decimal number 45 to binary.
2. How many different values can 6 bits represent? How many bits are needed to give 200 students each a unique ID?
3. Should a text document be compressed with lossless or lossy compression? What about streaming video? Explain.
4. A 4-bit unsigned variable holds 15, and 1 is added. What happens?
5. Run-length encode `AAAABBBCCD`. Is the result shorter?
6. Give two examples of metadata stored with a digital photograph, and one privacy risk.
7. Data show that ice-cream sales and drownings rise together. Can we conclude that ice cream causes drowning? Explain.

## Answers

1. $128 + 32 + 16 + 4 + 2 = 182$; $45 = 32 + 8 + 4 + 1 = 101101_2$.
2. $2^6 = 64$. $2^7 = 128 < 200 \le 256 = 2^8$, so 8 bits.
3. Text: lossless, because every character must be restored exactly. Video: lossy is acceptable, because small losses are hard to notice and the file becomes much smaller.
4. An overflow error: 16 needs 5 bits, so the result cannot be stored (it may wrap around to 0).
5. `4A3B2C1D`: 8 characters instead of 10, so it is shorter.

6. Date and time taken, camera model, GPS location. The location can reveal where someone lives.
7. No. Correlation does not show causation; a third variable (hot weather) increases both.

## Exam tips

- Convert confidently between **binary** and **decimal**, and order numbers given in both.
- Remember a bit is one binary digit and a byte is 8 bits; $n$ bits represent $2^n$ values.
- Explain that all data – numbers, text, images, sound – is stored as binary, and that finite bits cause **overflow** and **round-off**.
- Distinguish **lossless** from **lossy** compression and when each is appropriate.
- Show the analog-to-digital idea: **sampling** turns a continuous signal into discrete values.
- Before you answer a "can the data show this?" question, check that every fact needed is in a column or in the metadata.



exercises.lol exercises.com.cn

# Algorithms and Programming

AP Computer Science Principles

Code below uses the **AP CSP pseudocode** – the exam's language-neutral reference. Assignment is written `a ← expression`, and list indices start at **1**.

## Variables and Assignments

A **variable** 变量 is a named place that holds a value. The **assignment** 赋值 operator stores the value on the right into the variable on the left:

```
a ← 5
b ← a + 3      // b is now 8
```




*A variable is a named store whose value can change*

A variable holds one value at a time; assigning again **replaces** it. Variables let a program store input, remember results, and reuse them.

**Worked example (a failed swap).** What is displayed?

```
a ← 3
b ← 7
a ← b
b ← a
DISPLAY(a)
DISPLAY(b)
```

| Line | a | b |
|---|---|---|
| `a ← 3` | 3 | – |
| `b ← 7` | 3 | 7 |
| `a ← b` | 7 | 7 |
| `b ← a` | 7 | 7 |

It displays `7 7`: the old value of `a` was replaced before it was copied. A swap needs a third variable: `temp ← a`, `a ← b`, `b ← temp`.

## Data Abstraction

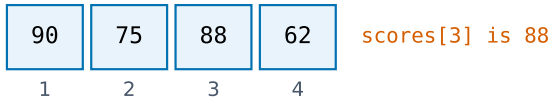
**Data abstraction** 数据抽象 lets you manage complexity by giving a single name to a collection of data – for example, a **list** rather than dozens of separate variables. It hides detail: you use the named collection without worrying about how it is stored. Lists (below) are the course's main data abstraction.

without a list

```
score1 ← 90
score2 ← 75
score3 ← 88
score4 ← 62
```

with a list

```
scores ← [90, 75, 88, 62]
```




scores[3] is 88

one name for the whole collection: a loop can process any number of items

*One list replaces many separate variables; list indexes start at 1*

## Mathematical Expressions

Programs compute with the operators `+`, `-`, `*`, `/`, and `MOD` (the **remainder** 余数 of a division, e.g. `17 MOD 5` is 2). Expressions follow the usual order of operations. `MOD` is especially useful for testing divisibility (`n MOD 2 = 0` means `n` is even) and for wrapping values around a range.

| | | |
|---|---|---|
| `17 MOD 5` | → 2 | 17 = 3 × 5 + 2 |
| `20 MOD 5` | → 0 | divides exactly |
| `4 MOD 7` | → 4 | smaller than 7: the remainder is 4 |
| `n MOD 2` | → 0 or 1 | 0 means n is even |
| `2 + 3 * 4` | → 14 | * before + |
| `(2 + 3) * 4` | → 20 | brackets first |

*MOD gives the remainder; multiplication comes before addition*



exercises.lol exercises.com.cn

**Worked example.** `MOD` has the same priority as `*` and `/`, so it is done before `+` and `-`:

| Expression | Steps | Value |
|---|---|---|
| `17 MOD 5 + 3 * 2` | `2 + 6` | 8 |
| `(17 + 3) MOD 5` | `20 MOD 5` | 0 |
| `472 MOD 10` | remainder after dividing by 10 | 2, the last digit |
| `(5 + 4) MOD 7` | day 5 plus 4 days, in a 7-day week | 2 |
| `3 MOD 8` | 8 goes 0 times, remainder 3 | 3 |

The fourth row shows week wrap-around: applying `MOD 7` keeps day numbers from 0 to 6. The last row shows that a smaller dividend is its own remainder. Note that `/` in AP pseudocode is ordinary division, so `7 / 2` is `3.5`.

## Strings

A **string** 字符串 is an ordered sequence of characters, like `"hello"`. Programs join strings (**concatenation** 拼接) and find their **length**. Strings represent text – names, messages, sequences – and are a common program input and output.




characters 2 to 4: "LAN"

```
concat("sun", "flower")  →  "sunflower"
```

a substring is a part of a string; concatenation joins strings

*Characters are numbered from 1; concatenation joins strings end to end*

The exam reference sheet has no string procedures, so a question defines the ones it uses, for example `concat(str1, str2)`, which joins two strings, and `substring(str, start, length)`, which returns `length` characters of `str` starting at position `start`.

**Worked example.** With those definitions and `word ← "computer"`:

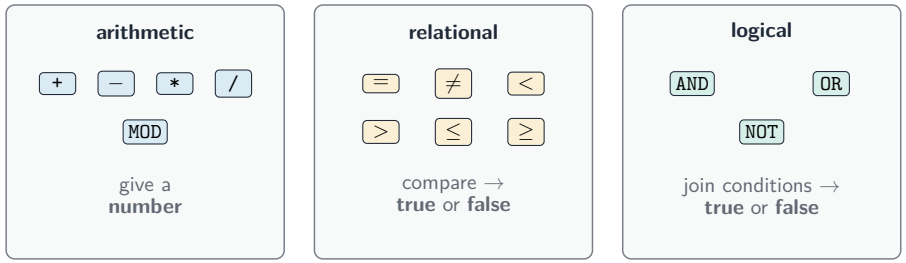
1. `substring(word, 4, 3)` starts at character 4, `p`, and takes 3 characters: `"put"`.
2. `substring(word, 1, 1)` is `"c"`.
3. `concat(substring(word, 1, 3), "et")` is `concat("com", "et")`, which is `"comet"`.

Positions start at 1, so the last character of a string `s` is at position `LENGTH(s)`; but always read the question's own definition first.

## Boolean Expressions

A **Boolean expression** 布尔表达式 evaluates to `true` or `false`. It uses **relational operators** (=, ≠, <, >, ≤, ≥) and **logical operators** `NOT`, `AND`, `OR`:

- `NOT` reverses a value,
- `AND` is true only when **both** sides are true,
- `OR` is true when **at least one** side is true.




*The three families of operators: arithmetic, relational, and logical*

These conditions drive every decision and loop.

> **Worked example.** With `age ← 15` and `member ← false`, evaluate `(age ≥ 13 AND age ≤ 19) AND NOT member`:

1. `age ≥ 13` is true; `age ≤ 19` is true; so the bracket is **true**.
2. `NOT member` is `NOT false`, **true**.
3. `true AND true` is **true**.

| p | q | p AND q | p OR q | NOT p |
|---|---|---|---|---|
| true | true | true | true | false |
| true | false | false | true | false |
| false | true | false | true | true |
| false | false | false | false | true |

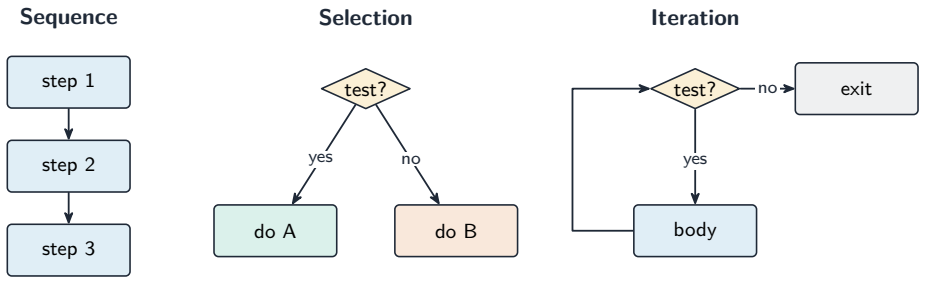
The opposite of `x > 5` is `x ≤ 5`, not `x < 5`: the value 5 itself belongs to the opposite.



## Conditionals

A **conditional (selection)** 条件语句 chooses which code to run. `IF` runs a block only when its condition is true; `ELSE` gives an alternative:

```
IF (score ≥ 60)
{
    DISPLAY("Pass")
}
ELSE
{
    DISPLAY("Fail")
}
```




*Selection chooses between paths based on a condition*

**Worked example.** Two separate `IF`s are **not** the same as `IF … ELSE`. For `n ← 12`:

```
IF (n > 10)
{
    n ← n - 5
}
IF (n < 10)
{
    n ← n * 2
}
DISPLAY(n)
```

The first test is true, so `n` becomes `7`; the second `IF` then tests the **new** value, `7 < 10`, which is true, so `n` becomes `14`. Written with `ELSE`, only the first branch would run and the program would display `7`.

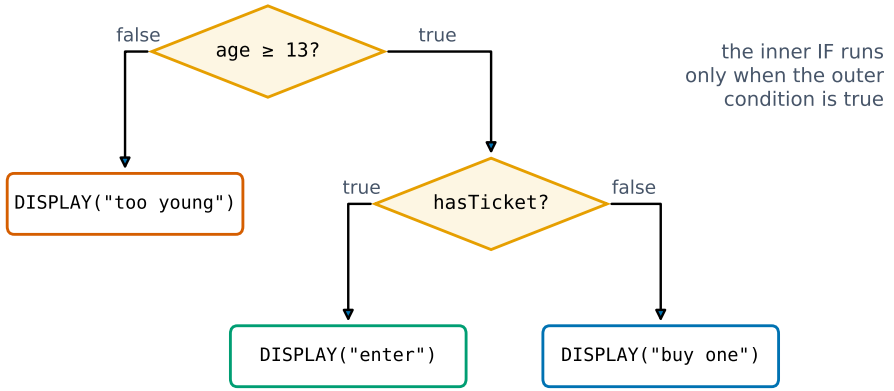


exercises.lol exercises.com.cn

## Nested Conditionals

A **nested conditional** 嵌套条件 places one `IF` inside another (or chains `ELSE IF`) to choose among **more than two** paths. Only the first matching branch runs:

```
IF (g ≥ 90)      { grade ← "A" }
ELSE IF (g ≥ 80) { grade ← "B" }
ELSE             { grade ← "C" }
```




*A nested conditional: the inner decision runs only in one branch of the outer one*

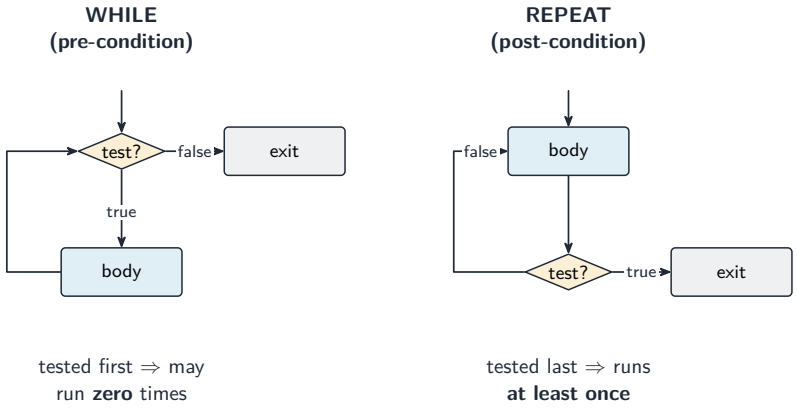
## Iteration

**Iteration (a loop)** 迭代 repeats instructions. AP pseudocode has two forms:

```
REPEAT 5 TIMES        // a fixed count
{
    DISPLAY("hi")
}

REPEAT UNTIL (found)  // until a condition becomes true
{
    ...
}
```






tested first ⇒ may run **zero** times

tested last ⇒ runs **at least once**

*A pre-condition (WHILE) loop tests before the body, so it may run zero times*

A loop that never meets its stopping condition is an **infinite loop** 无限循环.

`REPEAT UNTIL (condition)` checks the condition **before** each pass and stops as soon as it is true; if it is already true at the start, the body runs **zero** times.

**Worked example.** What is displayed, and how many times does the body run?

```
n ← 1
REPEAT UNTIL (n > 20)
{
    n ← n * 2
}
DISPLAY(n)
```

| Pass | `n` before | `n` after |
|---|---|---|
| 1 | 1 | 2 |
| 2 | 2 | 4 |
| 3 | 4 | 8 |
| 4 | 8 | 16 |
| 5 | 16 | 32 |

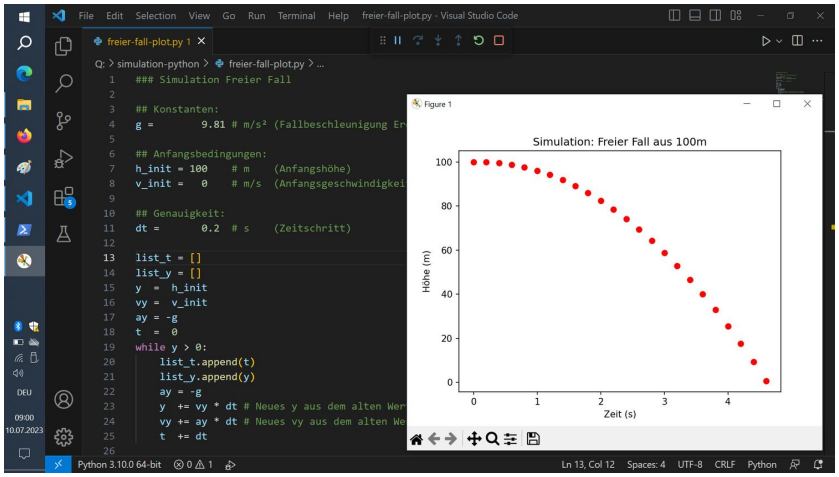
After pass 5, `n > 20` is true, so the loop stops: the body ran **5** times and the program displays `32`. If the body were `n ← n * 1`, `n` would stay 1 and the loop would never end.



exercises.lol exercises.com.cn

## Developing Algorithms

**An algorithm is not the same thing as code.** Beyond visual and textual programming languages, an algorithm can be expressed in a **variety of ways**: in **natural language** (ordinary sentences), as a **diagram** such as a flowchart, or in pseudocode. Those forms are for people – they let you check the logic and agree on it before any language is chosen, and the same algorithm can then be written in any language.

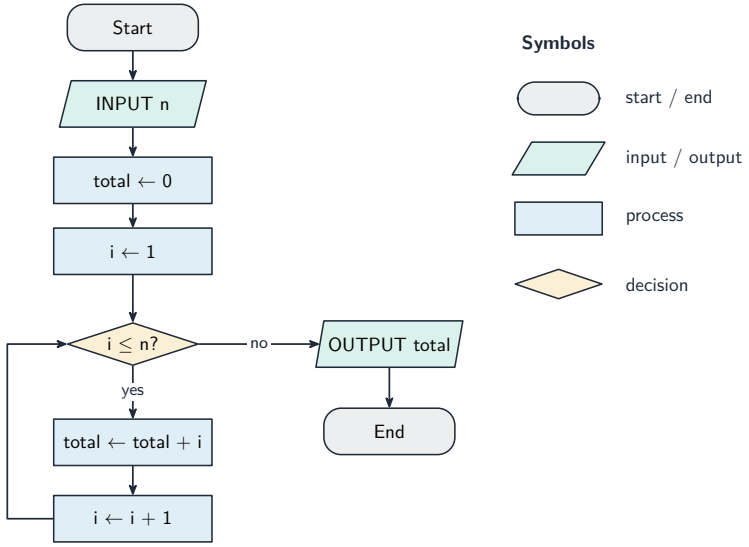



*Python source code on a screen – algorithms are precise, ordered instructions*

When you do write it in a programming language, **clarity and readability are important considerations**, not decoration: meaningful variable names, consistent indentation and comments explaining *why* rather than *what*. The program has to be read and modified later by someone – often you – and an algorithm nobody can follow cannot be maintained or debugged.

An **algorithm** 算法 is a finite sequence of steps that solves a problem, built from **sequencing**, **selection**, and **iteration**. Different algorithms can solve the same problem, and you should be able to combine and modify existing algorithms (for example, count the values in a list that meet a condition, or find the largest). Trace an algorithm by hand to check it is correct.



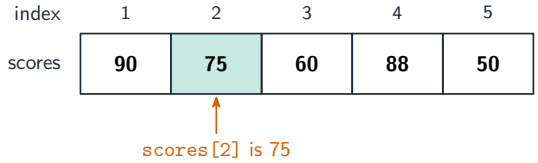



*A flowchart lays out an algorithm using the standard symbols*

## Lists

A **list** 列表 is an ordered collection of values under one name, the course's key data abstraction. AP pseudocode indexes from **1**:

```
scores ← [88, 74, 95]
DISPLAY(scores[1])          // 88
scores[2] ← 80              // replace the 2nd value
APPEND(scores, 60)          // add to the end
INSERT(scores, 1, 100)      // insert at index 1
REMOVE(scores, 3)           // delete the 3rd element
LENGTH(scores)              // how many elements
```




*A list holds many values in one variable, each found by its index*



exercises.lol exercises.com.cn

Traverse a list with a loop to sum, count, search, or find a maximum:

```
total ← 0
FOR EACH x IN scores
{
    total ← total + x
}
```

**Worked example (tracing a list traversal).** This procedure should count the scores of at least 80:

```
count ← 0
FOR EACH x IN [88, 74, 95, 80]
{
    IF (x ≥ 80)
    {
        count ← count + 1
    }
}
DISPLAY(count)
```

| x | x ≥ 80? | count after |
|---|---|---|
| 88 | true | 1 |
| 74 | false | 1 |
| 95 | true | 2 |
| 80 | true | 3 |

It displays 3. Two traps the exam sets: delete `count ← 0` and `count` has no starting value, so the result is undefined; write `x > 80` and the last score is missed, so it displays 2. Trace the boundary value every time.

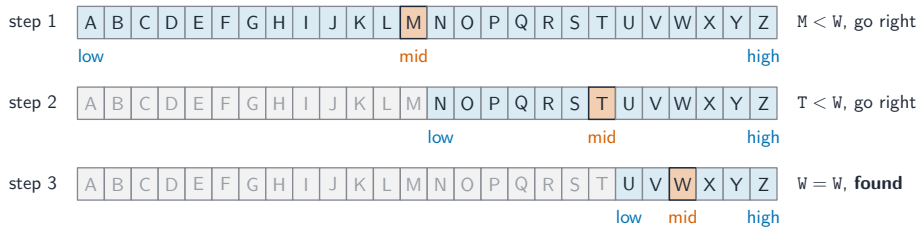
## Binary Search

**Binary search** 二分搜索 finds a value in a **sorted** list far faster than checking each element. It looks at the middle element, then discards the half that cannot contain the target, repeating until found. Each step **halves** the search space, so a list of $n$ items takes about $\log_2 n$ steps. It **requires the data to be sorted** first.






*A phone book: binary search halves the remaining pages each step*




*Binary search halves the range at each step (the list must be sorted)*

**Worked example.** Searching a sorted list of 8 items, binary search halves the range each step: $8 \to 4 \to 2 \to 1$ in three halvings, and the one item left must still be checked, so at most 4 comparisons ($\log_2 8 + 1$), whereas a linear search could take up to 8. The advantage grows explosively: about 1,000 items need only $\approx 10$ binary-search steps (but up to 1,000 linear ones), and 1,000,000 items need just $\approx 20$. Halving is what makes it a **reasonable-time** algorithm.

## Calling Procedures

A **procedure (function)** 过程 is a named, reusable block of code. **Calling** it runs its code with the **arguments** you supply, and it may **return** a value:

```
sum ← Add(3, 4)      // call, passing 3 and 4
```



exercises.lol exercises.com.cn

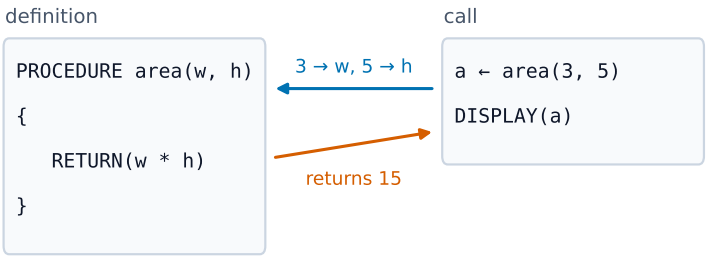
Procedures let you use code without knowing its inner workings – **procedural abstraction** 过程抽象.

**Worked example.** What is displayed?

```
PROCEDURE Mystery(x, y)
{
    IF (x > y)
    {
        RETURN(x - y)
    }
    RETURN(y - x)
}
DISPLAY(Mystery(3, 10))
DISPLAY(Mystery(Mystery(9, 4), 2))
```

1. `Mystery(3, 10)`: `3 > 10` is false, so it returns `10 - 3`, which is `7`.
2. The inner call `Mystery(9, 4)` returns `5`; then `Mystery(5, 2)` returns `3`.

It displays `7 3`. The procedure returns the distance between two numbers. `RETURN` ends the procedure at once, so the second `RETURN` runs only when the `IF` was false.



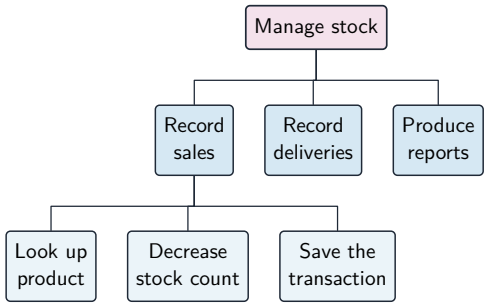

*Arguments are passed in, the body runs, and RETURN sends a value back*



## Developing Procedures

You **define** a procedure with a name, **parameters** (inputs), and a body, and optionally `RETURN` a result:

```
PROCEDURE Add(a, b)
{
    RETURN(a + b)
}
```




*Decomposing a program into procedures and sub-procedures*

**Worked example.** Write a procedure `CountAbove(list, limit)` that returns how many values in `list` are greater than `limit`.

```
PROCEDURE CountAbove(list, limit)
{
    count ← 0
    FOR EACH item IN list
    {
        IF (item > limit)
        {
            count ← count + 1
        }
    }
    RETURN(count)
}
```

`CountAbove([4, 9, 12, 7], 6)` returns `3`. The parameters make it general: the same procedure works for any list and any limit, which is the point of a procedure. Displaying `count` inside the loop, or forgetting `RETURN`, are the usual mistakes.

Writing your own procedures reduces repetition, breaks a big problem into named pieces, and makes programs readable and easier to test – the essence of **abstraction** 抽象.

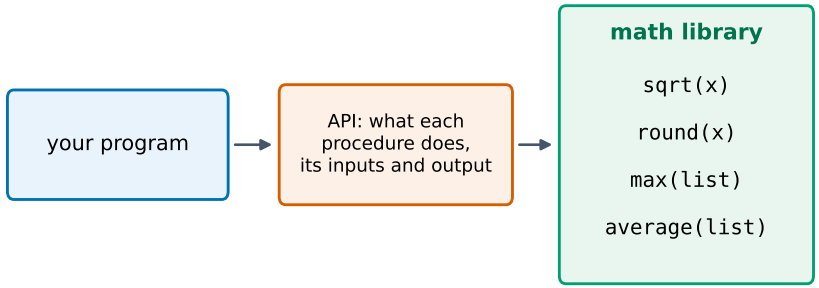


exercises.lol exercises.com.cn

## Libraries

A **library** 库 is a collection of ready-made procedures that others can reuse. An **API** (Application Program Interface) 应用程序接口 documents what each procedure does, its parameters, and its result – so you can use it without seeing its code. Libraries save time and let you build on existing, tested work.

**The documentation is part of the library.** Documentation for an API or library is **necessary** in order to understand the behaviours it provides and how to use them – what each procedure expects as parameters, what it returns, and what it does at the edges. Without it you would have to read the source, which defeats the point of abstraction; with it you can use a procedure correctly without knowing how it works inside.




*A library offers tested procedures that you call through their documentation*

## Random Values

`RANDOM(a, b)` returns a random integer from `a` to `b` (inclusive), letting a program produce **unpredictable** results – for games, sampling, or simulations. Each call may give a different value, so a program using randomness behaves differently each run.

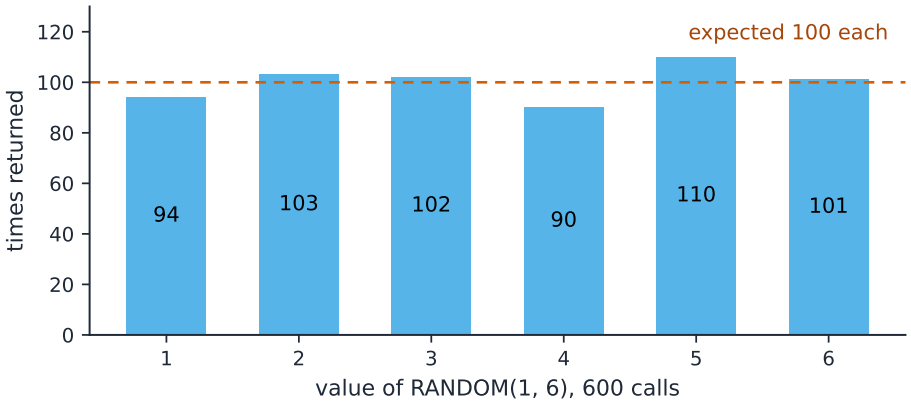
**Worked example.** A game gives a prize 30% of the time:

```
IF (RANDOM(1, 10) ≤ 3)
{
    DISPLAY("Prize!")
}
```

`RANDOM(1, 10)` has 10 equally likely values; 1, 2 and 3 are 3 of them, so the chance is $\frac{3}{10} = 30\%$. `RANDOM(0, 10) ≤ 3` would be wrong: it has 11 values, 4 of them at most 3. To list every possible output of a program with `RANDOM`, consider all values the call can return and trace the branches they trigger. The endpoints alone can miss an output produced only by an interior value.



exercises.lol exercises.com.cn

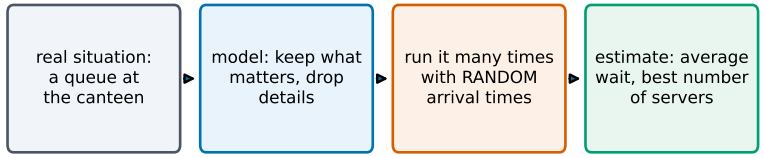



*RANDOM(1, 6) returns each value from 1 to 6 about equally often*

## Simulations

A **simulation** 模拟 is a program that models a real-world process to study it safely and cheaply. Simulations **simplify** reality (they leave out detail) and often use **randomness** to imitate chance events. They let you test scenarios that would be too costly, slow, or dangerous in real life – but their results are only as good as their assumptions.

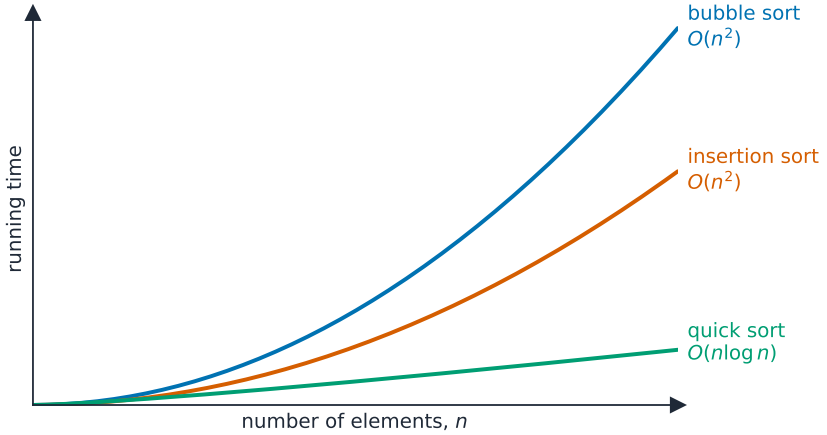
**A simulation is a way of doing science, not just a picture.** Because it can be run many times, cheaply and with one variable changed at a time, a simulation **facilitates the formulation and refinement of hypotheses** about the object or phenomenon under consideration: you propose an explanation, run the model, compare the result with reality, and adjust either the hypothesis or the model. That is why a simulation's simplifications matter – a result only supports a hypothesis about the real world to the extent that what was left out does not matter.




*A simulation simplifies a real situation, runs many times and estimates outcomes*

## Algorithmic Efficiency

**Efficiency** 效率 is how much time (or memory) an algorithm needs as its input grows. A **reasonable-time** algorithm's work grows like a polynomial of the input size (e.g. linear or quadratic); an **unreasonable-time** algorithm grows far faster (e.g. doubling with each added item), becoming impractical for large inputs. A faster algorithm can make a previously impossible problem solvable. Sometimes an exact answer takes too long, so a **heuristic** 启发式 – an approach that finds a good-enough answer quickly – is used instead.




for large $n$: $O(n \log n)$ (quick/merge) beats $O(n^2)$ (bubble/insertion)

*How the running time of an algorithm grows with the input size n*

## Undecidable Problems

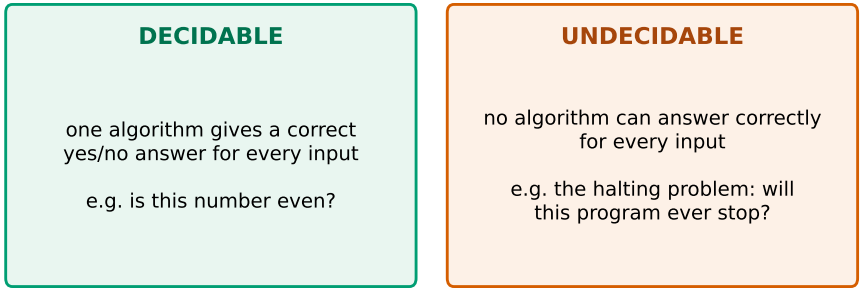
Some problems are **undecidable** 不可判定: no algorithm can solve **every** case of them with a correct yes/no answer. This is a fundamental limit of computing – not a matter of needing a faster computer, but a proof that no such algorithm can exist.

The best-known example is the **halting problem** 停机问题: deciding, for any program and its input, whether the program will eventually stop or run for ever. A tool can answer this for many programs, but it has been proved that no algorithm answers it correctly for **all** of them. An undecidable problem may still be solved for particular cases; what is impossible is one algorithm for every case.

**Exam skill:** be able to determine a code segment's result by tracing it, compare two algorithms' efficiency (reasonable vs unreasonable time), and recognize procedural and data abstraction in a program.



exercises.lol exercises.com.cn




*Decidable and undecidable problems*

## Practice

Answer each before reading the answers below.

1. What does this display?

```
a ← 3
b ← 7
a ← b
b ← a
DISPLAY(a)
DISPLAY(b)
```

2. Evaluate `17 MOD 5`.
3. After `nums ← [4, 8, 15]`, `APPEND(nums, 16)` and `INSERT(nums, 2, 23)`, what is `nums`, and what is `nums[1]`?
4. What is the greatest number of items a binary search checks in a sorted list of 1000 items? What must be true of the list?
5. Write a procedure `countAbove(list, n)` that returns how many items in `list` are greater than `n`.
6. `RANDOM(1, 6)` simulates a die. What is the probability that it returns 6? Give one advantage of using a simulation.
7. Explain what makes an algorithm run in a reasonable or an unreasonable time, and what a heuristic is.
8. What is an undecidable problem? Name one.

## Answers

1. `7` then `7`: after `a ← b`, `a` is 7, and `b ← a` copies 7 back into `b`.
2. 2.
3. `[4, 23, 8, 15, 16]`; `nums[1]` is 4 (indices start at 1).
4. 10, since $2^{10} = 1024 \geq 1000$. The list must be sorted.
5. One answer:

```
PROCEDURE countAbove(list, n)
{
    count ← 0
    FOR EACH item IN list
    {
        IF(item > n)
        {
            count ← count + 1
        }
    }
    RETURN(count)
}
```

6. $\frac{1}{6}$. A simulation can test situations that are too expensive, slow or dangerous to try for real.
7. Reasonable time grows like a polynomial in the input size (such as $n^2$); unreasonable time is not bounded by a polynomial (for example, $2^n$). A heuristic finds an approximate solution quickly when an exact one is too slow.
8. A problem for which no algorithm can always give a correct yes-or-no answer for every input; the halting problem.

## Exam tips

- Know a variable is a named store for a value and trace how **assignment** updates it step by step.
- Read the AP **pseudocode** carefully – `a <- expression` assigns, and lists are **1-indexed** on the exam reference sheet.
- Distinguish a variable from a **list** (a collection accessed by index) and use list operations correctly.
- Evaluate expressions with the right precedence and boolean logic (`AND`, `OR`, `NOT`).
- Pick clear, meaningful variable names – the written tasks reward readable code.

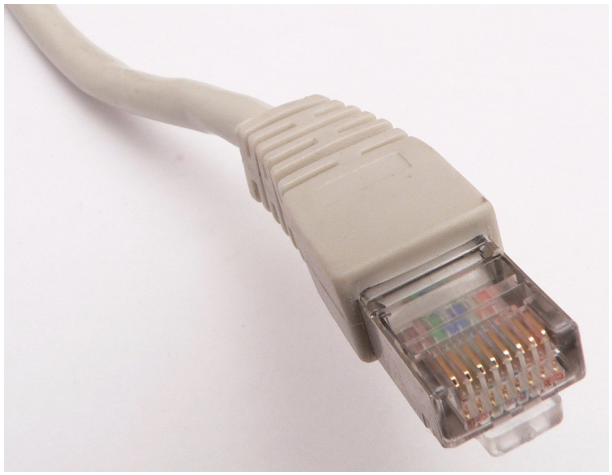


exercises.lol exercises.com.cn

# Computer Systems and Networks

AP Computer Science Principles

## The Internet

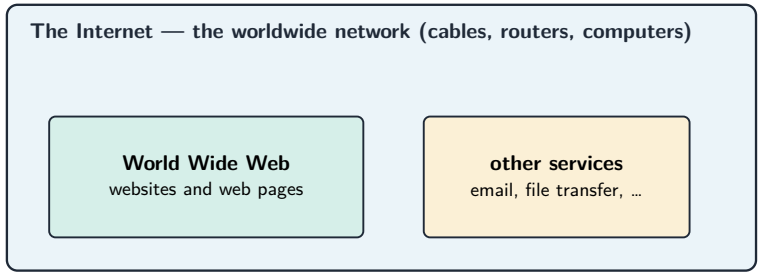


*An RJ45 Ethernet connector – physical links carry packets across the Internet*

In January 2008 two undersea cables near Alexandria, Egypt, were cut. Internet service across Egypt, the Gulf and India became very slow for millions of people, yet the Internet did not stop: traffic found other routes around the break. No person chose those new routes. The network was built to do this by itself, and this unit explains how.

The **Internet** 互联网 is a global **network** 网络 of networks. Data travels as **packets** 数据包 – small chunks that are sent separately and reassembled at the **destination** 目的地. Two ideas make it work at scale:

- **Protocols** 协议 are agreed rules for communication. **IP** (Internet Protocol) addresses and **routes** 路由 packets; **TCP** reassembles them in order and re-requests lost ones; **HTTP** carries web pages; **DNS** translates a name like `example.com` into an **IP address** IP 地址.
- **Redundancy** 冗余 and routing: there are **many** possible paths between two points, so if one path fails, packets take another. This makes the Internet **fault-tolerant** 容错.




*The internet is the worldwide network; the web is one service running on it*

The Internet is designed to be **open** 开放的 and **scalable** 可扩展的 – built on **standards** 标准 anyone can use, so it keeps working as billions of **devices** 设备 join. **Bandwidth** 带宽 is the amount of data a connection can carry per second.

## Devices, systems, networks and paths

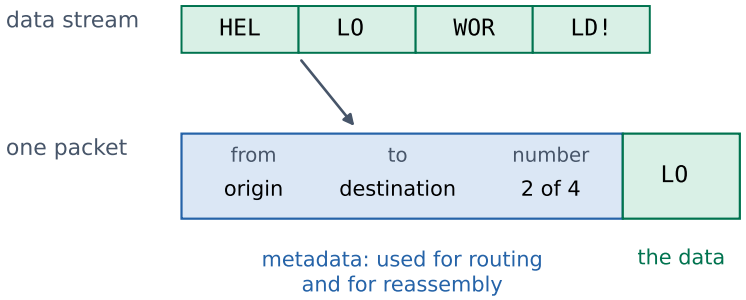
These terms are defined exactly on the exam, each building on the last:

- A **computing device** 计算设备 is a physical thing that can run a program: a computer, tablet, server, router or smart sensor.
- A **computing system** 计算系统 is a group of devices and programs working together for a common purpose.
- A **computer network** 计算机网络 is a group of connected devices that can send or receive data. It is one type of computing system.
- A **path** 路径 is a sequence of directly connected devices that starts at the sender and ends at the receiver. **Routing** is finding a path.

**How packets travel.** A **router** 路由器 reads each packet's destination **IP address** and forwards it toward that address. Routing is **dynamic**: each packet takes whatever path is best at that moment, so packets of one message can take **different paths** and **arrive out of order**. TCP puts them back in order using their sequence numbers.

Information travels as a **data stream** 数据流, split into chunks. Each chunk goes inside a packet together with **metadata** 元数据, which is used for routing and for putting the chunks back together:






*A packet holds a chunk of the data plus the metadata needed to deliver it*

Packets may arrive in order, out of order, or **not at all**. Suppose packets 1 to 5 are sent and 2, 1, 5, 3 arrive. TCP notices that 4 is missing, asks for it again, and gives the program all five in order. UDP passes on what arrived and does not ask for packet 4.

- **TCP** aims to provide **reliable** 可靠的, ordered byte-stream delivery by detecting missing data and retransmitting it. A connection can still fail, so delivery is not guaranteed under every network failure.
- **UDP** sends datagrams without built-in delivery acknowledgements, ordering or retransmission. It can avoid waiting for retransmission, but it is not always faster. An application may add its own reliability; time-sensitive video and games may prefer a missing update to a late one.
- Protocols are **layered** 分层的: for example, HTTP/1.1 uses TCP over IP. Other HTTP versions can use a different transport. Each layer has a role.
- **Bandwidth** is the **maximum** amount of data that can be sent in a fixed time, usually measured in bits per second; **latency** 延迟 is the **delay** for one packet to arrive.

> **Worked example: how many addresses?** An IP address is a binary number of fixed length, so $n$ bits give $2^n$ different addresses.

1. **IPv4** uses 32 **bits** 位: $2^{32} \approx 4.3$ billion addresses – too few for every device on Earth.
2. **IPv6** uses **128 bits**: $2^{128}$ addresses.
3. IPv6 has $\dfrac{2^{128}}{2^{32}} = 2^{96}$ times as many addresses as IPv4.
4. Each **extra bit doubles** the number of addresses: going from 32 to 33 bits gives $2 \times 2^{32}$ addresses, not $2^{32} + 1$.

> **Worked example: bandwidth.** How long does a 200-megabyte video take to send over a link with a bandwidth of 50 megabits per second?



exercises.lol exercises.com.cn

1. Change bytes to bits: $200 \times 8 = 1600$ megabits.
2. Divide by the rate: $\frac{1600}{50} = 32$ seconds.

This is the **shortest possible** time, because bandwidth is a maximum. **Common mistake:** forgetting the factor of 8. File sizes are usually given in bytes; bandwidth is given in bits.

### ■ The Internet is not the Web

| | **The Internet** | **The World Wide Web 万维网** |
|---|---|---|
| What it is | a network of connected networks | a system of linked pages, programs and files |
| Protocols | IP, TCP, UDP and others | HTTP |
| Relationship | carries the Web and much else, such as email | runs on top of the Internet |

So "the Web is down" and "the Internet is down" are different claims. A video call or an email can work with no web page involved.

### ■ Why the Internet can grow

The protocols are **open**, which means nonproprietary: no one needs to ask permission or pay to use them, so anyone can connect another device. Routing is not fixed in advance. **Scalability** is the capacity of a system to change in size to meet new demands, and because of these two facts the Internet has grown from a few computers to billions.

## Fault Tolerance



exercises.lol exercises.com.cn

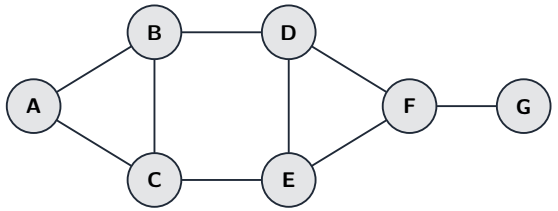



*A Wi-Fi router: the local gateway that forwards packets toward the wider Internet*

A system is **fault-tolerant** if it keeps working even when part of it fails. The Internet achieves this through **redundant** connections: because packets can be routed along multiple paths, the failure of one router or cable does not stop communication – traffic simply reroutes. Fault tolerance costs extra resources (the redundant paths) but greatly improves **reliability** 可靠性. A single path with no backup is not fault-tolerant.

Why it matters: parts of a large system fail at unexpected times, often several together. Fault tolerance lets people keep using the network while repairs happen. Having **more than one path** between any two connected devices is the usual form of **redundancy**, and it also helps the Internet grow, because new devices need not depend on any single link.

> **Worked example: which failures break the network?** In the network below, the lines are connections between computers.




1. **Can A still reach F if one connection fails?** Yes. Every connection on one route has a **backup route** 备用路由: if A–B fails, A–C–E–F still works.
2. **Smallest number of failures that cut A off from F? Two**: for example A–B **and** A–C, because those are A's only two connections.

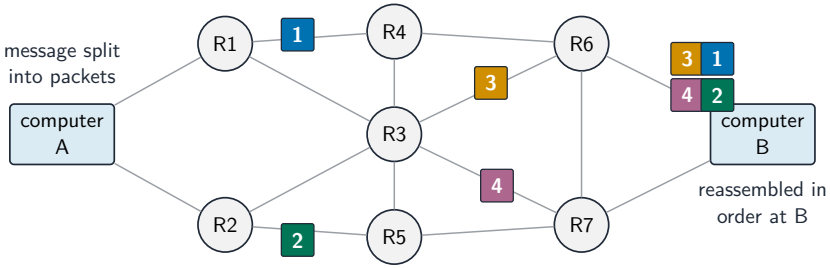
3. **Is G fault-tolerant?** No. G has **one** connection, F–G. If it fails, G cannot reach anyone, so F–G is a **single point of failure** 单点故障.
4. **How would you fix that?** Add one more connection to G, for example E–G. Now G has two paths, and a single failure cannot cut it off.

**Exam skill:** to find whether a failure disconnects two computers, cross out the failed connections and look for **any remaining path** between them.

> **Worked example: what redundancy costs.** Five computers are joined in three ways.

| Layout | Connections | Failures it can always survive |
|---|---|---|
| A line, each joined to the next | 4 | 0 |
| A ring | 5 | 1 |
| Every computer joined to every other | 10 | 3 |

The ring costs one extra connection and survives any single failure. The full layout survives more but needs twice as many connections. Redundancy always trades extra resources for reliability.




*Packet switching sends packets by independent routes, so the network tolerates a failed link*

## Parallel and Distributed Computing

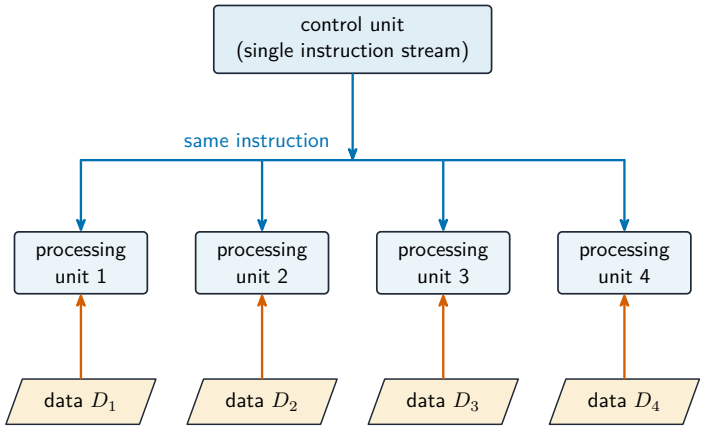


exercises.lol exercises.com.cn



*Data-centre server racks: distributed computing spreads work across many machines*

- **Sequential computing** 顺序计算 runs one operation at a time.
- **Parallel computing** 并行计算 splits a task into parts that run **at the same time** on multiple **processors** 处理器, finishing faster.
- **Distributed computing** 分布式计算 uses **many computers** connected by a network to work on one problem – essential for problems too big for a single machine.



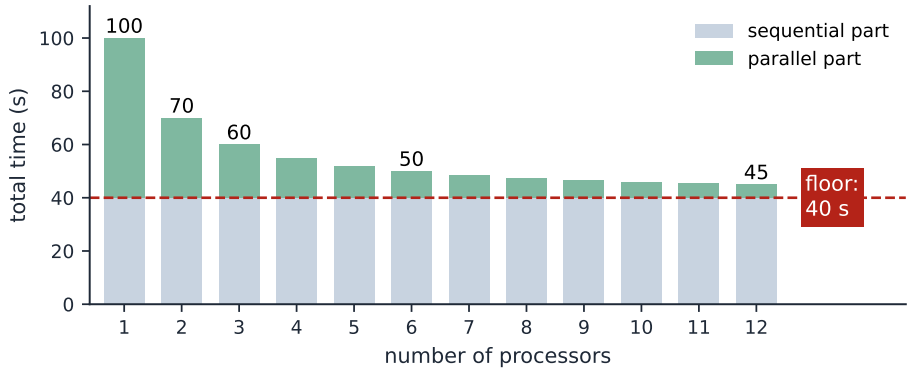

*Parallel computing: many processors work at the same time*

A parallel solution's **speedup** 加速比 is the sequential time divided by the parallel time. Speedup is limited: parts that **must** run in sequence cannot be sped up by adding processors, so doubling the processors rarely doubles the speed.

Two rules give almost every answer:

- A **sequential** solution takes as long as the **sum** of all its steps.
- A **parallel** solution takes as long as its sequential steps **plus the longest** of the steps that run at the same time.

> **Worked example.** A task has a part that **must** run sequentially, taking 40 seconds, plus a parallelizable part that takes 60 seconds on one processor – so on a single processor the whole task takes $40+60 = 100$ seconds. Spread the parallel part across 3 processors and it takes $\frac{60}{3} = 20$ seconds, so the total parallel time is $40 + 20 = 60$ seconds and the speedup is $\frac{100}{60} \approx 1.67$. The 40-second sequential part is a floor: even with infinitely many processors the task can never finish in under 40 seconds.




*Each extra processor helps less: the sequential part sets a floor*

So at some point adding processors no longer helps in any useful way. The first extra processor saves 30 seconds; going from 6 to 12 processors saves only 5. The speedup here can never reach $\frac{100}{40} = 2.5$.

> **Worked example: two processors, three independent processes.** Processes P, Q and R take 50, 40 and 30 seconds and do not depend on each other. Each processor runs one process at a time.

1. **Sequentially** (one processor): $50 + 40 + 30 = 120$ seconds.
2. With **two** processors, try each split and take the **longer** processor's time: $\{50\}, \{40, 30\} \to 70$; $\{50, 30\}, \{40\} \to 80$; $\{50, 40\}, \{30\} \to 90$.
3. The best split finishes in **70 seconds**: one processor runs P while the other runs Q then R.
4. Speedup $= \dfrac{120}{70} \approx 1.71$.



> **Worked example: steps that must wait.** Steps X (20 s) and Y (30 s) can run at the same time, but step Z (10 s) needs both results first. Two processors are available.

1. Sequentially: $20 + 30 + 10 = 60$ seconds.
2. In parallel: X and Y run together and finish after the longer one, 30 seconds. Then Z runs: $30 + 10 = 40$ seconds.
3. Speedup $= \dfrac{60}{40} = 1.5$.

**Common mistake:** adding every step in the parallel case, or dividing the total by the number of processors. Neither follows the rule, unless the work really does split evenly with no sequential part.

**Why distributed computing?** Some problems cannot be solved on one computer at all, because they would take too long or need too much storage. Search engines and weather forecasts are examples. Spreading the work over many devices solves much larger problems more quickly. Parallel solutions also **scale** better than sequential ones: when the data grow, you can add processors. The cost is that the parts must be coordinated, and any part that must run in sequence still limits the speed.

**Exam skill:** given the times for the sequential and parallel portions of a task, be able to calculate the total parallel time and the speedup.

## Practice

Answer each before reading the answers below.

1. Explain how a file is sent across the Internet in packets, and why the packets may arrive out of order.
2. Name the protocol that does each job: gives devices addresses and routes packets; makes sure all packets arrive and are reassembled; turns domain names into IP addresses.
3. Devices A, B and C are connected in a line: A–B–C. Is the network fault-tolerant? How could you make it so?
4. Three tasks take 40, 30 and 50 s. Find the time on one processor, the shortest time on two processors running in parallel, and the speedup.
5. How many bits are in an IPv4 address and an IPv6 address? Why was IPv6 introduced?
6. Explain the difference between the Internet and the World Wide Web.

## Answers

1. The file is split into packets, each with a source address, destination address and sequence information. Packets can take different routes, so they may arrive out of order; the receiver reassembles them.



exercises.lol exercises.com.cn

2. IP; TCP; DNS.
3. No: if B fails, A cannot reach C. Adding a direct link from A to C gives a redundant path.
4. One processor: $40 + 30 + 50 = 120$ s. Two: 50 s on one and $40 + 30 = 70$ s on the other, so 70 s. Speedup $= 120 \div 70 \approx 1.7$.
5. IPv4 has 32 bits ($2^{32}$, about 4.3 billion addresses); IPv6 has 128 bits. IPv4 addresses were running out.
6. The Internet is the network of connected devices and the protocols that carry data. The Web is one service that runs on it: linked pages accessed with HTTP.

## Exam tips

- Describe how data travels in **packets** over a redundant, fault-tolerant network of independent routers.
- Know that **protocols** (IP, TCP, HTTP) are agreed rules, and that open standards let different systems **interoperate** 互操作.
- Explain **scalability** 可扩展性 and how the Internet grows without **central control** 集中控制.
- Contrast **bandwidth** (rate) with **latency** (delay), and describe the **DNS** name-to-address lookup.
- Discuss the **digital divide** 数字鸿沟 and security basics (**encryption** 加密, **authentication** 身份验证) in plain terms.



exercises.lol exercises.com.cn

# Impact of Computing

AP Computer Science Principles

## Beneficial and Harmful Effects



*A pile of e-waste – computing has environmental costs as well as benefits*

In November 2017 the fitness app Strava published an updated world map of where its users had run and cycled. In January 2018 its privacy risks drew public attention. The aim was friendly: to show popular routes. No names were on it. But in empty desert areas the bright lines traced the outlines of military bases, because soldiers there wore fitness trackers. The programmers had not planned this result, and the map exposed sensitive activity patterns. This unit is about such effects: who gains, who loses, and how to protect yourself.

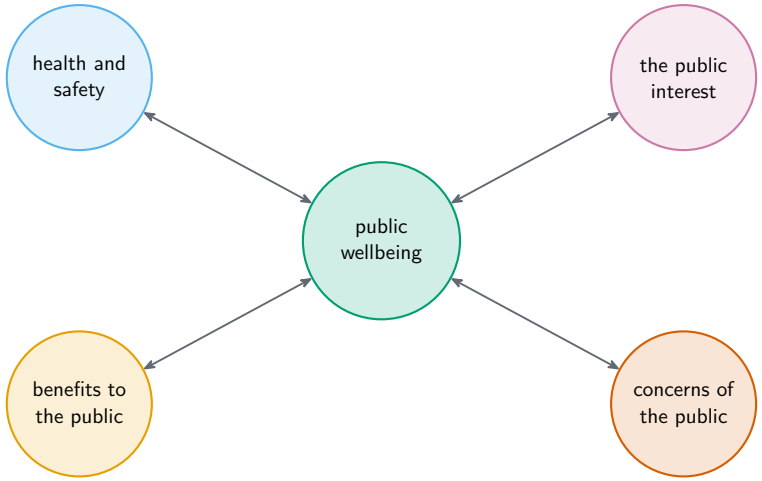
Every computing innovation can be used in ways that **help** and ways that **harm** – often the same technology does both. A social network connects people **and** can spread misinformation; automation raises productivity **and** can remove jobs. Effects are frequently **unintended**: creators cannot foresee every use. When you evaluate a computing innovation, weigh its benefits and harms on people and society, and remember that harms are not always deliberate.

The **same effect** can be seen as good and bad, by different people or by one person. A phone that shows work email lets you leave the office early, and it also means that work never quite stops. Computing also changes how people do tasks: few people now use a paper map.

**Innovations go beyond their purpose.** Three examples from the course:

- The **World Wide Web** 万维网 was first meant only for scientists to share information quickly.
- **Targeted advertising** 定向广告 helps businesses reach customers, but it can be misused against individuals and groups.
- **Machine learning** 机器学习 and **data mining** 数据挖掘 have led to advances in medicine, business and science, but what they find has also been used to discriminate against groups.

Responsible programmers try to think about unintended uses, but no one can foresee them all. When a program spreads quickly to many users, its effects can go beyond anything its creator can control.




*Computing affects the public's wellbeing in several ways*

**Computing also generates creativity in other fields**, which the CED asks for as a benefit in its own right, not merely as convenience: modelling and imaging in **medicine**, simulation in **engineering**, new forms in the **arts** and music, and new kinds of **communication** entirely. An innovation's beneficial effects are often in a field far from computing.

> **Worked example.** Give one beneficial and one harmful effect of map apps that suggest the fastest route.

- **Beneficial**: drivers save time and fuel, and delivery firms plan routes more cheaply.
- **Harmful, and unintended**: routing many drivers through quiet side streets brings heavy traffic to homes that never had it.

Each effect names **who** is affected and **how**. "It is convenient" or "it can be hacked" are too general to earn credit.



exercises.lol exercises.com.cn

## The Digital Divide

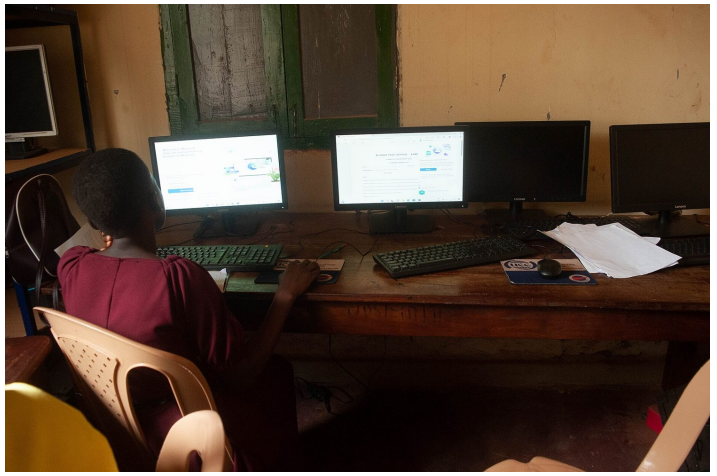
The **digital divide** 数字鸿沟 is the unequal access to computing and the Internet across groups – by income, geography, age, or country. Those with access gain education, jobs, and services; those without fall further behind. The divide is shaped by economic, social, and geographic factors, and efforts to close it (affordable devices, public access, infrastructure) aim to make computing's benefits fairer.

The divide raises issues of **equity** 公平, access and influence, both across the world and inside one town. It affects whole groups and also single people, and the actions of individuals, organizations and governments can make it wider or narrower. A government that pays for rural broadband narrows it; a school that moves all its homework online, when some families have no connection, can widen it.

> **Worked example.** Which situation shows the digital divide?

- **A** a student chooses not to use social media;
- **B** a village has no broadband, so its students cannot join online lessons;
- **C** a website is slow during a sale;
- **D** a company stores data in the cloud.

**B**: a group lacks **access** because of where it is. A is a choice, not a lack of access; C and D are about performance and storage.



*Public library computers: digital divide is about unequal access to devices, connectivity and skills*




*A rural satellite dish: geography still shapes who gets fast internet and who waits*

## Computing Bias



*A self-driving car on a city street – autonomous systems raise safety and bias questions*

**Bias** 偏见 can be built into computing systems – often unintentionally. If the **data** used to build a system reflects existing prejudice, or if the designers' assumptions are one-sided, the system can produce **unfair** results (for example, a hiring tool that favors one group). Bias can enter at every stage – data collection, design, and use – so systems should be tested for fairness across different groups. Recognizing that "the computer said so" is not the same as "fair" is an important habit.



Bias reaches a program in two main ways: through the **data** it uses, or through the rules in the **algorithm** itself. A loan program that never asks about family background can still be unfair if it gives low scores to certain postal codes, because where people live is often linked to income and background. Collecting more data of the same kind does not fix bias; programmers must look for it and act to reduce it.

> **Worked example.** A voice assistant is trained on recordings of adults only. It often misunderstands children.

1. **Where the bias entered**: data collection; the training data left out a group of users.
2. **The effect**: children get worse service, although no one meant this.
3. **A fix**: collect recordings from children too, and test accuracy for each age group before release.

> **Worked example.** A hiring algorithm is trained on a company's past hires, who were mostly one group, and it then rejects qualified applicants from other groups. Name the problem and its cause: this is **computing bias**, caused by biased training data – the model learned the historical pattern instead of a fair rule. A full-mark exam answer states the harm (qualified people are unfairly rejected) **and** its cause (the bias came from the data, not the code).




illustrative: the less a group appears in the data, the worse the program serves it

*A program trained on unrepresentative data works worse for the groups it rarely saw*

## Crowdsourcing

**Crowdsourcing** 众包 obtains input, ideas, or funding from a **large group of people**, usually online. It harnesses the knowledge and effort of many – mapping projects, product reviews, citizen science, and **crowdfunding** all rely on it. The Internet makes crowdsourcing possible at a scale and speed never before achievable, letting a



exercises.lol exercises.com.cn

project draw on contributors worldwide.

**Citizen science** 公民科学 is research done partly by many people, most of whom are not scientists, who send in data using their own devices: for example, phone recordings of bird songs. **Crowdfunding** 众筹 is a form of crowdsourcing that connects a business or a good cause with money from many small donors. Public data help in the same way: when information is open to everyone, more people can spot a problem and help solve it.

> **Worked example.** A citizen-science site asks volunteers to sort millions of telescope photos of galaxies by shape.

- **Benefit**: thousands of people finish in weeks what a few scientists would take years to do.
- **Limitation**: volunteers make mistakes, so each photo is shown to several people and the majority answer is kept.



*A Wikipedia edit-a-thon: crowdsourcing pools many people's work into a shared resource*

## Legal and Ethical Concerns

Computing raises questions of law and ethics:

- **Intellectual property** 知识产权 and **copyright** 版权 protect creators' work; using it may require permission or a license. **Open-source** 开源 and **Creative Commons** licenses let creators share work under stated terms.
- **Plagiarism** 抄袭 – using others' work as your own – is unethical and often illegal.
- Collecting and using personal data raises **privacy** questions about consent and misuse.



exercises.lol exercises.com.cn



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

### The three "open" terms, which are not the same thing

| Term | What it means |
|---|---|
| **open source** | programs that are **made freely available and may be redistributed and modified** by anyone. The licence grants those rights explicitly – free of charge is not the same as open source, and a free program you may not modify is not open source. |
| **open access** 开放获取 | research and other content made available online **without charge**, so a reader does not need a subscription. It says nothing about the right to modify. |
| **Creative Commons** 知识共享 | a family of licences a creator applies to their own work to grant specific permissions in advance – for example "you may reuse this if you credit me" or "you may reuse this but not commercially". |

All three are ways of **granting** rights the creator holds by default under copyright. That is why they matter for the exam: copyright usually arises automatically for qualifying work. Reuse must follow the licence or have permission, unless the work is in the public domain or a legal exception applies.

Just because something is technically possible does not make it legal or ethical.

> **Worked example: may you use it?**

| You want to use | Allowed? | What you must do |
|---|---|---|
| A photo under a Creative Commons licence that asks for credit | Yes | give the credit it asks for |
| Open-source code in your Create task | Yes, under its licence | follow its licence terms and acknowledge it in your documentation |
| An open-access research paper | Yes, to read; reuse depends on the rights | cite it, and check its licence or an applicable exception before copying or adapting it |
| A song from the charts in your app | Not without permission | ask the owner, or choose a licensed song |

Material from someone else should **always be cited**, even when the licence allows free use. Presenting it as your own is plagiarism in every case.

Other legal and ethical questions the course names: software that gives access to digital downloads and streaming of media, algorithms that include bias, and devices that collect data by **monitoring** activity all the time.

### ■ What is recorded while you browse

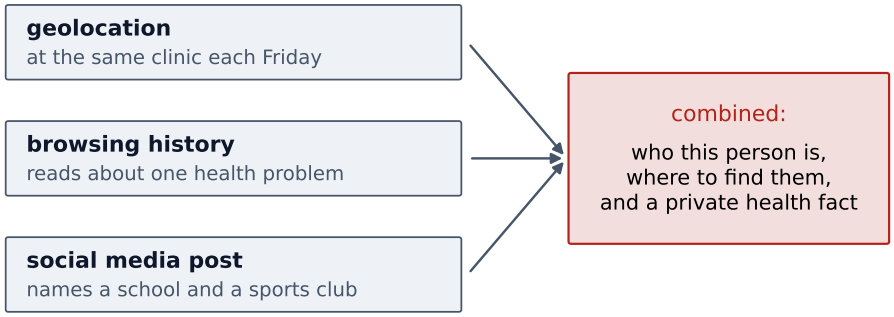
Two collection routes the CED names, and neither needs you to type anything:

- **Websites can record and maintain a history of the individuals who have viewed their pages** – which pages, when, how long, and from which device.
- **Search engines can use your search history** to suggest websites, and to sell **targeted marketing**: the advertisement follows the search, which is why a product looked up once then appears for weeks.

Neither is inherently malicious, and both are the mechanism behind services people find useful. The point the exam wants is that **data collected for one purpose can be combined and used for another**, often without the person realising they agreed to it.

Devices, websites and networks can also collect your **location**. Separate pieces of personal data, such as location, **cookies** (small files a website keeps on your device) and browsing history, can be joined to learn much more about someone:






*Each piece looks harmless; together they reveal a person*

## Safe Computing

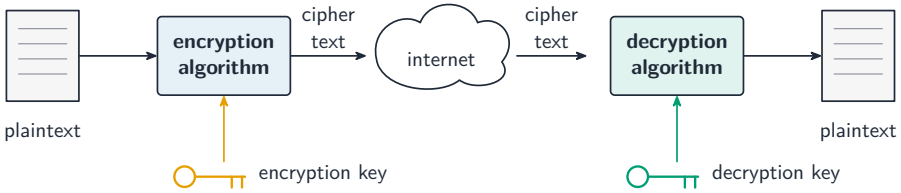
Protecting personal data is a shared responsibility. Key ideas:

- **Personally identifiable information (PII)** 个人身份信息 is information that identifies, links to, or describes a person: name, address, ID numbers, phone number, age, race, medical and financial information, and **biometric data** 生物特征数据 such as a fingerprint. It should be shared carefully, because it can be misused for **identity theft** 身份盗窃 or stalking.
- Threats include **phishing** 网络钓鱼 (tricking you into revealing information), **malware** 恶意软件, and weak passwords.
- **A malicious link can be disguised** on a web page or in an email message: the text you see and the address it actually goes to are separate, so a link reading `www.yourbank.com` can point anywhere. Hover to see the real destination before clicking, and be most suspicious of a link that arrives unexpectedly and creates urgency.
- Defenses include strong, unique **passwords**, **multi-factor authentication** 多因素认证, **encryption** 加密 (scrambling data so only authorized people can read it), and keeping software updated.

**Encryption** is the central tool for keeping data private in transit and storage. Being a responsible computer user means protecting your own and others' information.



exercises.lol exercises.com.cn




*Encryption scrambles plaintext with a key; only the key can decrypt it*

PII is not only a risk. Stored payment details make buying online quicker, and location makes a map app useful. The risk is that once information is online it is **hard to delete**: posts can be copied and forwarded, and an employer may see them years later.

## Proving who you are

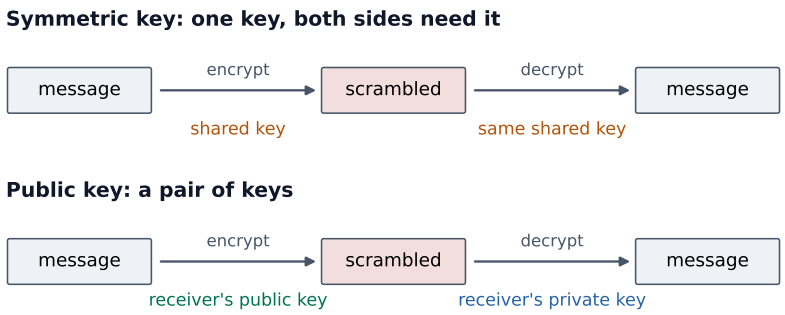
**Authentication** 身份验证 protects devices and data from people who should not have access. A **strong password** 强密码 is easy for you to remember and hard for anyone else to guess, even someone who knows you. **Multi-factor authentication** asks for evidence from at least two of three categories:

| Category | Meaning | Example |
|---|---|---|
| Knowledge | something you know | a password |
| Possession | something you have | a code sent to your phone |
| Inherence | something you are | a fingerprint |

Each step is one more layer that an attacker must break. A password and a second password are both "knowledge", so they are weaker than a password and a phone.

## Two kinds of encryption

**Decryption** 解密 is the reverse of encryption: turning coded data back into readable form.




*Symmetric key encryption uses one key; public key encryption uses a pair*



- **Symmetric key encryption** 对称密钥加密 uses **one key** for both jobs, so both people must already share it secretly.
- **Public key encryption** 公钥加密 uses a pair: a **public key** 公钥 that anyone may have, and a **private key** 私钥 that only the receiver keeps.

> **Worked example.** Ana wants to send Ben a secret message using public key encryption. Which key does she use? **Ben's public key.** She never needs Ben's private key, which is the point of the method. Only Ben can decrypt the message, with his private key. Ana's own keys play no part.

How does Ana know the public key really is Ben's? A **certificate authority** 证书颁发机构 issues **digital certificates** 数字证书 that confirm who owns a key, and web browsers trust these authorities.

The mathematics of encryption is not tested.

## How attackers get in

| Method | What happens |
|---|---|
| **Phishing** | a false message tricks you into giving personal information |
| **Keylogging** 键盘记录 | a program records every key you press, to capture passwords |
| **Rogue access point** 恶意接入点 | a wireless access point gives unauthorized access to a network, so data on it can be read or changed |
| Malicious links and attachments | unexpected emails, even from a friend whose account was taken over, install harmful software |
| Untrustworthy downloads | free software from unknown sites contains malware |

**Malware** is any software meant to damage a system or take partial control of it. A **computer virus** 计算机病毒 is one kind: it copies itself and often attaches to a legitimate program.

To defend yourself, keep software updated, because all real systems have flaws that can be exploited and updates fix them. Use scanning software against viruses and malware. Review the **permission settings** 权限设置 of your apps, so that they collect only what you agree to.

> **Worked example: name the attack.** At a café, Wei joins a free network called `Cafe_WiFi`, set up by a stranger, and the stranger reads Wei's unencrypted messages. This is a rogue access point. Phishing is wrong, because no one tricked Wei into typing personal information; Wei's data were read as they crossed the network.

**Exam skill:** be able to identify the beneficial and harmful effects of a given innovation, explain a privacy or security risk, and name a safe-computing practice that addresses



exercises.lol exercises.com.cn

it.

## Practice

Answer each before reading the answers below.

1. Give two causes of the digital divide.
2. A facial-recognition system works poorly for people with darker skin. Explain a likely cause and one way to reduce the bias.
3. Give an example of crowdsourcing and one benefit.
4. Distinguish phishing, keylogging and a rogue access point.
5. Explain how multifactor authentication protects an account even when a password is stolen.
6. A student wants to use a song in a video project. Explain how a Creative Commons licence could allow this.

## Answers

1. Any two: cost of devices or Internet service; lack of infrastructure in rural areas; lack of digital skills; government restrictions.
2. The training data included too few images of people with darker skin. Use more representative training data and test accuracy across groups.
3. Example: volunteers classifying galaxy images, or reporting traffic in a navigation app. Many people provide data or solve problems quickly and cheaply.
4. Phishing tricks a user into giving personal information through a fake message or site. Keylogging records keystrokes to capture passwords. A rogue access point is an unauthorized wireless network used to intercept data.
5. An attacker also needs a second factor, such as a code from the user's phone or a fingerprint, which a stolen password does not provide.
6. The creator chooses a licence that allows reuse under stated conditions, such as giving credit; the student follows those conditions without asking for separate permission.

## Exam tips

- Argue **both** the beneficial and harmful effects of a computing innovation – a balanced answer scores best.
- Use correct terms for data concerns: **PII**, privacy, security, and **algorithmic bias**.
- Explain how **crowdsourcing** and large data sets create value and raise new risks.
- Distinguish the **digital divide** (access) from bias (fairness) and give a concrete example of each.



exercises.lol exercises.com.cn

- Tie each effects answer to the innovation in the question. Create written responses concern your own submitted program.
- For public key encryption, the sender uses the **receiver's public** key and the receiver uses their own **private** key.
- Match the attack to how it works: phishing tricks a person, keylogging records keys, a rogue access point sits on the network.