# Languages and translators

2.5.1 · Algorithms and programming (Paper 2) · revision notes

Programs are written in high-level or low-level languages, and translators turn them into machine code the CPU can execute. You need the features of each level of language and the differences between a compiler and an interpreter. Expect 'give a reason', 'compare' and scenario questions about which translator to use.

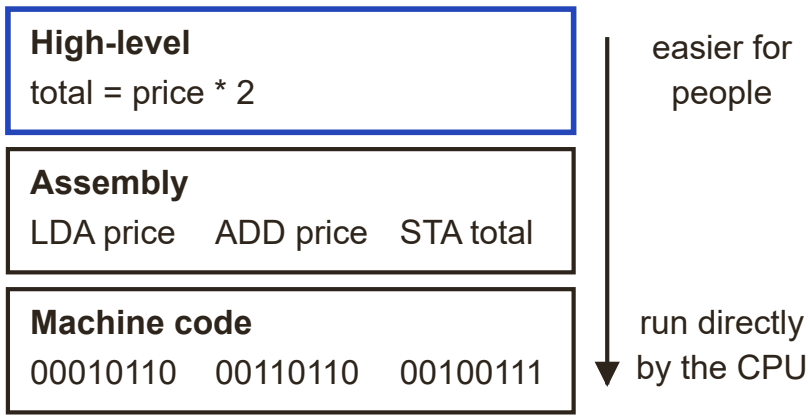
## Key facts

- High-level: English-like, easier to write and debug, portable
- Low-level: processor-specific, direct hardware control, fast and memory-efficient
- The CPU can only execute machine code
- Interpreter: line by line each time it runs; stops at the first error; slower; no executable
- Low-level: machine code (binary) and assembly language (mnemonics)
- Translator: converts source code into machine code
- Compiler: whole program at once → executable file; runs fast; errors reported after translating
- Develop with an interpreter; distribute with a compiler

## Notes

### High-level and low-level languages

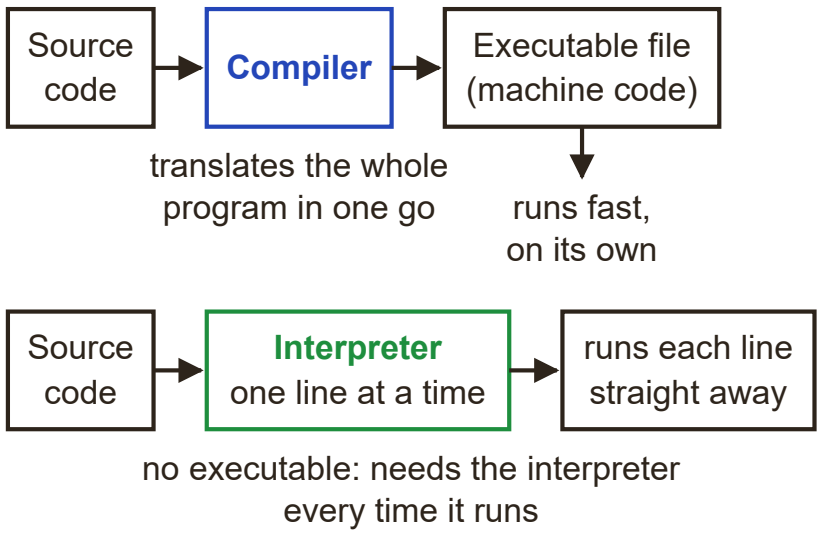
- **High-level languages** (such as Python and Java) use English-like keywords. One statement can stand for many machine code instructions.
- High-level code is easier to read, write and debug, and is **portable**: it can be translated to run on different types of processor.
- **Low-level languages** are **machine code** (binary instructions the CPU executes directly) and **assembly language** (short mnemonics such as ADD, each standing for one machine code instruction).
- Low-level code is specific to one type of processor. It gives direct control of the hardware and can be very fast and use little memory, so it suits embedded systems and device drivers, but it is harder to write and debug.




One statement in a high-level language stands for several low-level instructions

## Translators

- The CPU can only execute machine code, so source code must be translated into machine code first.
- A **compiler** translates the whole program in one go and produces an executable file (machine code).
- Compiler: the executable runs quickly and can be run without the translator or the source code, so the code stays private. But errors are only reported after the whole program has been translated, and the program must be compiled again after every change.
- An **interpreter** translates and runs the program one line at a time, every time it runs. No executable file is produced.
- Interpreter: it stops at the first error and shows where it is, which makes debugging easier, and code can be tested straight away. But the program runs more slowly, and the user needs the interpreter (and the source code) to run it.




Compiler and interpreter compared

## How to answer each type of question

**Give reasons for using a level of language**

2 to 4 marks | **Grade 5**

1. Pick a feature of that level of language.
2. Explain why it helps in the situation given.
3. One feature and reason per mark pair.

**Example.** Give **two** reasons why most programs are written in high-level languages. [2]

**Model answer.** Any two: they use English-like keywords, so are easier to read and write (1); they are easier to debug (1); they are portable, so the same code can be translated to run on different processors (1); one statement does the work of many machine code instructions, so programs are quicker to write (1).

### Explain why a translator is needed

2 marks | Grade 4

1. Say what the CPU can execute: only machine code.
2. Say that source code must be converted into machine code.

**Example.** Explain why a program written in a high-level language must be translated before it can be run. [2]

**Model answer.** The CPU can only execute machine code (binary instructions) (1). High-level source code is not machine code, so it must be converted into machine code first (1).

### Compare compilers and interpreters in a scenario

4 marks | Grade 7

1. Interpreter strengths: stops at the error line, runs code straight away: good for development.
2. Compiler strengths: fast executable, no translator needed, source code kept private: good for release.
3. Link each point to the scenario.

**Example.** A company is developing a new game. Explain why its programmers use an interpreter while writing the game, but a compiler to produce the version sold to customers. [4]

**Model answer.**

An interpreter stops when it reaches an error and shows the line, so bugs are easier to find (1), and code can be run and tested straight away without translating the whole program (1).
A compiler produces an executable file that runs faster (1), and customers can run it without a translator and cannot see the source code (1).

### Explain when a low-level language is used

2 marks | Grade 6

1. Say what low-level code gives: direct control of the hardware, speed or small size.
2. Link it to the device or task in the question.

**Example.** A programmer writes part of the software for a washing machine's control chip in assembly language. Give **two** reasons why. [2]

**Model answer.** It gives direct control of the hardware, such as the chip's registers (1). It produces fast code that uses very little memory, which suits the small chip (1).

## Don’t lose marks

- Saying a compiler 'runs' the program: it translates it into an executable.
- Saying an interpreter produces an executable file: it does not.
- Saying high-level code runs faster than low-level code: usually the reverse.
- Thinking assembly language is high level, or that the CPU can run it directly.

- Giving answers such as 'easier' without saying easier to do what (read, write, debug).

## More tips

### Memory tricks

- Compiler = translates the whole book before anyone reads it. Interpreter = translates live, one sentence at a time.
- High = Human-friendly; Low = close to the hardware.
- Develop and debug with an interpreter; sell and share with a compiler.

### Exam technique

- In compare questions, write matching pairs of points (compiler … whereas interpreter …).
- In scenario questions, say who benefits (the programmer or the customer) and how.
- Use the key words: machine code, executable, line by line, portable, source code.

## What each grade needs

What you need to be able to do, from the first marks up to the top grade.

**Grade 3** — **Give examples of high-level and low-level languages**
Python and Java are high level; machine code and assembly language are low level.

**Grade 4** — **Explain why translators are needed**
The CPU can only execute machine code, so source code must be translated.

**Grade 5** — **Give features of high-level languages**
English-like, easier to write and debug, and portable between processor types.

**Grade 5** — **Give features of low-level languages**
Processor-specific, direct control of hardware, fast and memory-efficient.

**Grade 6** — **Compare compilers and interpreters**
A compiler translates the whole program into an executable; an interpreter translates and runs line by line.

**Grade 7** — **Choose a translator for a scenario**
Interpreter for development and debugging; compiler for distributing a finished program.

Free notes, Full packs, practice papers and mock exams at chhetriacademy.com. Written for this site; not taken from past papers. Stuck? 1-to-1 tutoring with a free 30-minute trial lesson.