Programming languages and translators

3.3 · Computers · revision notes

Low-level languages (machine code and assembly language) work directly with the processor, while high-level languages are closer to English and must be translated before they can run. You need the characteristics and uses of low-level languages, and how a compiler and an interpreter differ in the way they translate high-level code into machine code. Paper 1 questions are mostly 'Give', 'Explain' and comparison questions.

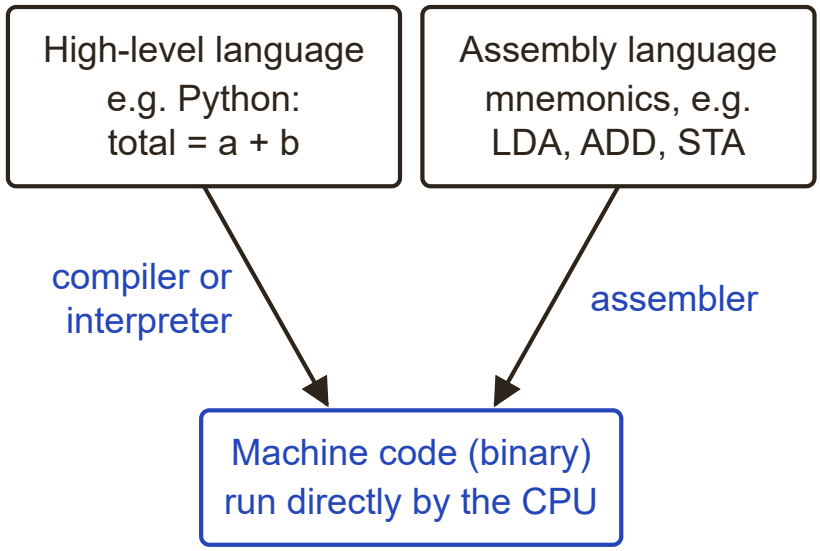
Key facts

- Machine code: binary instructions that the CPU runs directly
- Low-level: processor-specific, direct hardware control, efficient, hard to write and debug
- Compiler: whole program at once → executable file; runs fast; errors reported after translation
- Writing and testing → interpreter; finished program for users → compiler
- Assembly language: mnemonics; one instruction = one machine code instruction; translated by an assembler
- High-level: English-like, portable, one statement = many machine code instructions
- Interpreter: one line at a time, every run; stops at the first error; no executable file

Notes

Low-level languages

- **Machine code** is instructions in binary that the CPU executes directly. Each type of processor has its own set of instructions (its instruction set).
- **Assembly language** uses short codes called mnemonics, such as LDA, ADD and STA, in place of binary. Each assembly instruction matches one machine code instruction. An **assembler** translates it into machine code.



Everything must end up as machine code before the CPU can run it

- Characteristics: specific to one type of processor (not portable), direct access to memory locations and registers, and hard to read, write and debug.
- Purpose: programs that must be very fast, use very little memory or control hardware directly, such as embedded systems, device drivers and parts of an operating system.

High-level languages

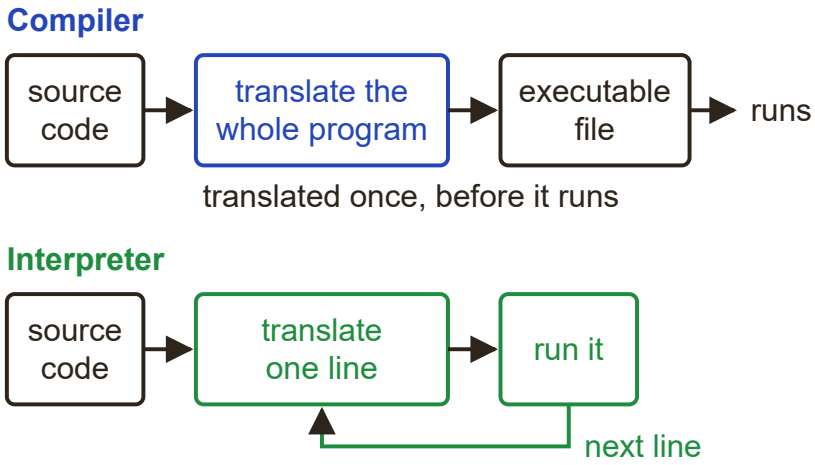
- Use English-like keywords such as if, while and print. Python is a high-level language.
- One statement is translated into many machine code instructions.
- Portable: the same source code can be translated to run on different types of processor.
- Quicker to write, and easier to read, debug and maintain. They come with built-in functions and libraries.
- They must be translated into machine code before the CPU can run them, by a **compiler** or an **interpreter**.

Compilers

- A compiler translates the **whole program** into machine code in one go, before it runs, and produces an **executable file**.
- The executable runs without the compiler or the source code, and runs quickly because it has already been translated.
- Errors are reported after the whole program has been translated, and the program cannot run until they are fixed.
- Users are not given the source code, so it is hard for them to copy or change it. The program must be compiled again after every change.

Interpreters

- An interpreter translates and runs the program **one line (statement) at a time**, every time it runs. No executable file is produced.



A compiler translates everything once and makes an executable; an interpreter translates line by line, every time

- It stops at the first error it finds and shows where it is, which is useful while writing and testing a program.
- Interpreted programs run more slowly: a line inside a loop is translated again every time the loop repeats.
- The interpreter must be installed on every computer that runs the program, and the source code has to be shared.

How to answer each type of question

Give characteristics of a type of language 1 to 2 marks **Grade 4**

1. Give short, separate facts, one per mark.
2. Use the key words: processor-specific, portable, English-like, mnemonics, binary, translated.

Example. (a) Give **two** characteristics of low-level languages. (2)
(b) State the name of the translator that converts assembly language into machine code. (1)

Model answer.

(a) Any two from: specific to one type of processor, so not portable (1); give direct control of memory and hardware (1); each assembly instruction matches one machine code instruction (1); written in binary or mnemonics, so hard to read and debug (1).
(b) An assembler (1)

Explain why a low-level language is used

2 to 4 marks | Grade 5

1. Give a characteristic of low-level languages.
2. Link it to the device or program in the question with 'so'.
3. For 'two reasons', repeat with a different characteristic.

Example. The program in a battery-powered smoke alarm is written in assembly language. Explain **two** reasons why. (4)

Model answer.

Assembly language gives direct control of the hardware, such as the smoke sensor and the buzzer (1), so the programmer can control exactly how and when the alarm responds (1). The code can be written to be very efficient, using little memory and few instructions (1), so it runs on a small, cheap, low-power processor and the battery lasts longer (1).
Also accept: the program only has to run on one type of processor (1), so the lack of portability does not matter (1).

Explain differences between a compiler and an interpreter

4 marks | Grade 6

1. Pick a difference: how much is translated at once, whether an executable is produced, how errors are reported, or speed of running.
2. Give both sides in one sentence using 'whereas'.
3. Repeat for a second difference.

Example. Explain **two** differences between the way a compiler and an interpreter translate high-level code. (4)

Model answer.

A compiler translates the whole program into machine code before it runs (1), whereas an interpreter translates and runs it one line at a time (1). A compiler produces an executable file that can be run without the compiler (1), whereas an interpreter produces no executable file, so the interpreter is needed every time the program runs (1).
Also accept: a compiler reports errors after translating the whole program, whereas an interpreter stops at the first error; compiled programs run faster, whereas interpreted programs are translated again each time they run.

Choose a translator for a situation 2 to 4 marks **Grade 7**

1. Decide what matters at this stage: finding errors quickly, or speed and protecting the code.
2. Name the translator.
3. Give a reason linked to that stage.

Example. A programmer is writing a game that will be sold to the public.
Explain which type of translator is most suitable:
(a) while she is writing and testing the game (2)
(b) for the version of the game that is sold. (2)

Model answer.

(a) An interpreter (1), because it stops at the first error and shows the line, so she can find and fix errors quickly without translating the whole program each time (1).

(b) A compiler (1), because it produces an executable file that runs quickly without a translator installed, and buyers are not given the source code, so they cannot easily copy or change it (1).

Don’t lose marks

- Saying an interpreter produces an executable file, or translates the whole program first.
- Saying high-level code runs directly on the CPU. It must be translated into machine code first.
- Saying assembly language is binary. It uses mnemonics; machine code is binary.
- Saying a compiled executable is portable. The high-level source code is portable; the executable only runs on the type of processor it was compiled for.
- Writing 'compilers are faster' without saying what is faster. The compiled program runs faster; the compiling itself takes time.

More tips

Memory tricks

- Compiler = translating a whole book before anyone reads it. Interpreter = a live translator, one sentence at a time.
- Assembler for assembly language: the names match.
- Low-level = close to the hardware. High-level = close to human language.
- Errors: an interpreter stops at the first one; a compiler reports them after translating the whole program.

Exam technique

- For differences, give both sides in one sentence, using 'whereas'.
- Be precise about translation: 'the whole program before it runs' versus 'one line at a time as it runs'.
- Use the context: embedded systems and device drivers point to low-level languages; a program sold to users points to a compiler.
- Do not just write 'easier': say why, e.g. English-like keywords make code easier to read and debug.

What each grade needs

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

Grade 3

State what machine code is

Instructions in binary that the CPU can execute directly, without translation.

Grade 4

Give characteristics of low-level languages

Specific to one type of processor, give direct control of hardware and memory, and are hard to read.

Grade 4

Give characteristics of high-level languages

English-like keywords, portable, one statement becomes many machine code instructions, must be translated.

Grade 5

Explain why low-level languages are used

They give direct control of hardware and very efficient code, e.g. for embedded systems and device drivers.

Grade 6

Describe how a compiler translates code

It translates the whole program into machine code before it runs, producing an executable file.

Grade 6

Describe how an interpreter translates code

It translates and runs one line at a time, every time the program runs, and stops at the first error.

Grade 7

Choose a suitable translator for a situation

An interpreter suits writing and testing; a compiler suits a finished program given to users.

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.