A1 COMPUTER FUNDAMENTALS · A1.4 · HL ONLY

Translation: compilers and interpreters

How **source code** becomes machine code the CPU can run, the trade-offs between a **compiler** and an **interpreter**, and the stages they share. This whole subtopic is HL only.

01 Compiler

Translates	The whole program, before it runs.
Output	A standalone executable (machine code).
Speed	Usually runs fast.
Errors	Reported after a full pass.
Portable	Less so: tied to one CPU and OS.
Examples	C, C++, Rust, Go.

02 Interpreter

Translates	Line by line, while it runs.
Output	None: runs the source directly.
Speed	Usually slower.
Errors	Stops at the **first error**.
Portable	More so: bridges code and machine.
Examples	Python, JavaScript, Ruby, PHP.

03 Stages of translation

Lexical	Break the source text into tokens: keywords, names, numbers, and symbols.	BOTH
Syntax	Parse the tokens into a parse tree using the language's grammar rules.	BOTH
Semantic	Check the meaning: types match, variables are declared, operations are valid.	BOTH
Optimisation	Rewrite the code for speed and size. Interpreters usually skip this pass.	COMPILER
Code gen	Emit the target: machine code in an executable (compiler) or bytecode (interpreter).	BOTH

FINAL PASS BEFORE THE EXAM

Rapid exam tips

Six things that lose marks in Paper 1 if you slip on them. A1.4 is HL only. Skim before you walk in.

01

Compilers translate the whole program once into an **executable**; **interpreters** translate line by line at runtime.

02

Compiled code usually **runs** faster; interpreted code gives faster **development** and debugging.

03

It is the **interpreter** that stops at the **first error**, not the compiler.

04

Interpreters produce **no executable**. They run the source, often via **bytecode**, directly.

05

Bytecode is run by the interpreter; **machine code** is run by the CPU. Don't confuse them.

06

A1.4 asks you to **evaluate**: match the translator to the project. Execution speed vs development and portability. **HL only**.