

THEME A: CONCEPTS · A1.4 · HL ONLY

Translation — Mark scheme

33 marks · covers A1.4.1

How to mark. Each creditworthy point is on its own line and ends with a semicolon (;). A line beginning with ... is an *expansion* of the line above it, so the pair is worth the marks shown. A slash (/) separates acceptable alternative wordings. Words in (brackets) are not needed for the mark. Mark positively: credit what the student has got right rather than penalising what is missing, and accept any valid alternative not listed here. Do not award more than the maximum for a part. Where a later part uses a wrong answer from an earlier part correctly, award follow-through marks (FT). HL marks parts that assess higher level only content.

Q	ANSWER	MARKS
HL 1(a) A1.4.1	Award [2] max Award [1] per point up to [2] max. Translation is the process of changing source code written by a programmer into instructions the processor can execute; The source code is in a high-level language that people can read, and the result is machine code / machine instructions in binary; It is needed because a processor cannot execute a high-level language directly; The two common approaches are interpretation and compilation;	2
HL 1(b) A1.4.1	Award [3] max Award [1] per point up to [3] max. The interpreter reads the source code one line/statement at a time; Each line is translated into machine code and executed immediately, on the fly; No separate executable file is produced; the interpreter works from the source code every time the program runs; It stops at the first error it meets and reports it, and the program does not continue until that error is fixed; Because translation happens while the program is running, execution is generally slower than compiled code;	3
HL 2(a) A1.4.1	Award [3] max Award [1] per point up to [3] max. The compiler translates the whole program into machine code before any of it is run; The source code is broken into tokens and checked against the grammar of the language (lexical and syntax analysis); The output is a binary executable that the processor can run directly; Translation happens once, so the executable can be run many times without translating again; The executable is produced only if the program compiles without errors;	3
HL 2(b) A1.4.1	Award [3] max Award [1] per point up to [3] max. The contrast must be clear. A compiler checks the whole program before execution and reports the errors it finds together;	3

Q	ANSWER	MARKS
	No executable is generated until every error has been resolved, so the program cannot be run at all; An interpreter finds errors at runtime, stopping at the first one it reaches; Earlier lines of an interpreted program will already have run before the error is reported; The interpreter therefore gives quicker, more interactive feedback, while the compiler catches problems earlier and more completely;	
HL 3(a) A1.4.1	Award [3] max Award [1] per point up to [3] max. Compilation produces an executable made of machine code for one particular processor architecture and operating system; That executable will not run on a different platform, so the company must compile the program separately for Windows, macOS and Linux; Each build must be tested and maintained, which increases the work involved; An interpreted program is distributed as source code instead; ...(so) it runs on any platform that has a suitable interpreter installed, without being rebuilt;	3
HL 3(b) A1.4.1	Award [3] max Award [1] per point up to [3] max. The source code is compiled once into bytecode, an intermediate, lower-level representation; Bytecode is platform-independent: it is not machine code for any particular processor; Each operating system runs its own interpreter or virtual machine, which executes that same bytecode; Only the virtual machine differs between platforms, so one bytecode file can be distributed to all of them; This also runs faster than interpreting the original source code, because bytecode is a more compact and optimised form;	3
HL 4(a) A1.4.1	Award [3] max Award [1] per point up to [3] max. JIT is a hybrid approach that compiles code into machine code while the program is running, rather than beforehand; The bytecode begins by being interpreted, so the application starts without a long wait; The JIT compiler monitors which parts of the program are executed most often; Those frequently used sections are compiled into native machine code and the compiled version is kept and reused; Later calls to those sections run as machine code instead of being interpreted again;	3
HL 4(b) A1.4.1	Award [2] max Award [1] for the point and [1] for the expansion. The hot parts of the program are translated once and then reused as machine code; ...(so) the cost of translating the same instructions on every pass through a loop is avoided; The JIT compiler can optimise using information only available at runtime, such as the actual data types being used; ...(so) it can produce faster machine code than a translation decided in advance;	2
HL 5(a) A1.4.1	Award [3] max Award [1] for a correct choice and up to [2] for the justification. No marks for the choice alone. An interpreted language;	3

Q	ANSWER	MARKS
	There is no compile step between writing code and seeing it run, so the feedback loop is very fast; The team can change the code and immediately test the result, which suits frequent iteration on a first version; Dynamic typing reduces the amount of set-up code needed, so a working version is reached sooner; Execution speed matters little at this stage compared with development speed;	
HL 5(b) A1.4.1	Award [2] max Award [1] for a correct choice and [1] for the justification. No mark for the choice alone. A compiled language; The code is translated in advance into machine code, so no time is spent translating while the game is running; The compiler can optimise for the target hardware, which matters when every frame has a strict time budget;	2
HL 6 A1.4.1	Award [6] max This is an evaluate question (AO3), so both strengths and limitations are required, applied to this tool. Award [1] per point up to [6] max, but award a maximum of [4] if only one side is given, and a maximum of [5] if there is no supported conclusion. Arguments for an interpreted language The same source code runs on every operating system the scientists use, so only one version has to be written and maintained; There is no compile step, so a scientist can change the analysis and rerun it immediately, which suits code that changes frequently; Errors are reported as soon as they are reached, giving quick feedback during exploratory work; Interpreted languages such as Python have large scientific standard libraries and third-party modules, reducing how much must be written; Arguments against an interpreted language Translation happens while the program runs, so processing a large data set will be slower than the compiled equivalent; The cost is paid on every run, and these runs are long, so the penalty is repeated rather than one-off; Some errors are only found at runtime, which may be part-way through a long analysis; The interpreter must be installed and kept at a matching version on every machine; Conclusion (credit any supported judgement, for example) An interpreted language is the better choice here because cross-platform support and fast iteration match how the scientists work, and the speed penalty can be reduced by calling compiled library routines for the heavy computation; ...or: a bytecode-compiled or JIT-compiled language is a reasonable compromise, keeping portability while recovering much of the speed;	6

Total for this paper: 33 marks