COURSE : GCSE COMPUTER SCIENCE	**COMPILERS AND INTERPRETERS**
UNIT : 2.5.1 **PROGRAMMING LANGUAGES AND IDES**	

Why do we need compilers and interpreters?

When we write program code, we generally use that are something more-or-less like English (languages like Python; Java; Visual Basic; the C, C++, C# family…) – unless we're serious geeky nerds writing in assembler… So, we need something to translate our human-oriented code into the binary stuff the processor can – um – process.

The translators we use to convert our high-level language fall into two categories: **compilers** and **interpreters**.

What's the difference?

Interpreters

An interpreter translates the code line-by-line, when it is run. When you write programs in Python, the IDE runs it **interpretively** – that is to say it translates it a line at a time, as the machine code (binary) for each line is required. The interpreter also checks the **syntax** of your code, and alerts your attention to syntax errors (sometimes with more meaningful messages, sometimes leaving you to figure where the error in your code is!)

Compilers

A compiler translates the whole program into machine code to produce what is known as **object code**. (Your original program is known as **source code**.) The compiler will generate error messages if it finds anything wrong with the source code. Each time a change is made to the source code, the whole program needs to be re-compiled before it can be run again.

What's this IDE thing?

An IDE is an **Interactive Development Environment** – a tool which allows programmers to write/edit and test code, often with useful features like being able to help the programmer "spell check" some of their code before it is run; having features to highlight different parts of the code in different colours (command words, or string values specified within the code, for example); and will sometime highlight where the programmer has made syntax errors, even before the program is run or tested.

IDEs can have powerful features to enable both quick coding (like auto-fill of command words, completion of brackets and so on) and **debugging**.

The tool you use to write Python is an IDLE – or Interactive Development Learning Environment.

Compilers and Interpreters Compared

Compiler	**Interpreter**
Translates the **whole program** into **object code**.	Translates and executes the **source code** one line at a time.
Any change to the **source code** requires the entire program to be recompiled – at development stage, this can be very time-consuming.	Can quickly re-run any changes made to the **source code** since it is translated at **run-time** and the programmer does not have to wait for the entire program to be translated before part of it can be tested.
Compiled **object code** will run a lot faster than waiting for an interpreter to translate the program line-by-line every time it is run.	Running a program **interpretively** takes longer, as each line of **source code** needs to be translated every time it is run.
A compiler produces **executable** file, so the original source code does not need to be compiled again (unless it is changed).	The original **source code** needs to be interpreted (translated) every time the program is run.
Compiled code keeps the source code hidden meaning that a software house can sell copies of its finished programs for clients to use, without giving away any of their programming secrets.	If a program is to be run **interpretively** then the user neds to have all of the source code (and the interpreter) on their computer. Thus the source code can be available for the user to read, change (and copy…)
The object code can be run without needing any compiler or interpreter on the user's computer.	The user needs both the source code and the interpreter to run the program.
Advantages	***Advantages***
Finished code runs faster; users cannot see the source code. For example: When a games software house sells a product, they want it to run as fast as possible on their customers' machines; they also do not want to let anyone see their programming secrets!	Code can be run line-by-line, so changes can be made and tested quickly; also, if it's running line-by-line (and hence rather more slowly!) it's easier for a programmer to spot where something unexpected or unwanted is happening in their code.