**Python Learning By Python Institute**

A program makes a computer usable. Without a program, a computer, even the most powerful one, is nothing more than an object. Similarly, without a player, a piano is nothing more than a wooden box.

Naturally, the computer will be able to compute this, but the computer is not aware of such things as distance, speed or time. Therefore, it is necessary to instruct the computer to:

* accept a number representing the distance;
* accept a number representing the travel time;
* divide the former value by the latter and store the result in the memory;
* display the result (representing the average speed) in a readable format.

These four simple actions form a program. Of course, these examples are not formalized, and they are very far from what the computer can understand, but they are good enough to be translated into a language the computer can accept.

***Note:***

*Language is the keyword.*

A complete set of known commands is called an instruction list, sometimes abbreviated to IL. Different types of computers may vary depending on the size of their ILs, and the instructions could be completely different in different models.

***Note:***

*Machine languages are developed by humans.*

A program written in a high-level programming language is called a source code (in contrast to the machine code executed by computers). Similarly, the file containing the source code is called the source file.

**Compilation vs. interpretation**

There are two different ways of transforming a program from a high-level programming language into machine language:

**COMPILATION** - the source program is translated once (however, this act must be repeated each time you modify the source code) by getting a file (e.g., an .exe file if the code is intended to be run under MS Windows) containing the machine code; now you can distribute the file worldwide; the program that performs this translation is ***called a compiler or translator***;

**INTERPRETATION** - you (or any user of the code) can translate the source program each time it has to be run; the program performing this kind of transformation is called an interpreter, as it interprets the code every time it is intended to be executed; it also means that you cannot just distribute the source code as-is, because the end-user also needs the interpreter to execute it.

***Note:***

*If the compiler finds an error, it finishes its work immediately. The only result in this case is an error message. The interpreter will inform you where the error is located and what caused it.*

*If the line looks good, the interpreter tries to execute it (note: each line is usually executed separately, so the trio "read-check-execute" can be repeated many times - more times than the actual number of lines in the source file, as some parts of the code may be executed more than once).*

*You may ask now: which is better? The "compiling" model or the "interpreting" model? There is no obvious answer. If there had been, one of these models would have ceased to exist a long time ago. Both of them have their advantages and their disadvantages.*

|  |  |  |
| --- | --- | --- |
| TYPE | COMPILATION | INTERPRETATION |
| ADVANTAGES | * the execution of the translated code is usually faster; * only the user has to have the compiler - the end-user may use the code without it; * the translated code is stored using machine language - as it is very hard to understand it, your own inventions and programming tricks are likely to remain your secret. | * you can run the code as soon as you complete it - there are no additional phases of translation; * the code is stored using programming language, not the machine one - this means that it can be run on computers using different machine languages; you don't compile your code separately for each different architecture. |
| DISADVANTAGES | * the compilation itself may be a very time-consuming process - you may not be able to run your code immediately after any amendment; * you have to have as many compilers as hardware platforms you want your code to be run on. | * don't expect that interpretation will ramp your code to high speed - your code will share the computer's power with the interpreter, so it can't be really fast; * both you and the end user have to have the interpreter to run your code. |

***Note:***

*Python is an interpreted language. This means that it inherits all the described advantages and disadvantages. Of course, it adds some of its unique features to both sets.*

*If you want to program in Python, you'll need the Python interpreter. You won't be able to run your code without it. Fortunately, Python is free. This is one of its most important advantages.*