What makes a language?

We can say that each language (machine or natural, it doesn't matter) consists of the following elements:

an alphabet: a set of symbols used to build words of a certain language (e.g., the Latin alphabet for English, the Cyrillic alphabet for Russian, Kanji for Japanese, and so on)

a lexis: (aka a dictionary) a set of words the language offers its users (e.g., the word "computer" comes from the English language dictionary, while "cmoptrue" doesn't; the word "chat" is present both in English and French dictionaries, but their meanings are different)

a syntax: a set of rules (formal or informal, written or felt intuitively) used to determine if a certain string of words forms a valid sentence (e.g., "I am a python" is a syntactically correct phrase, while "I a python am" isn't)

semantics: a set of rules determining if a certain phrase makes sense (e.g., "I ate a doughnut" makes sense, but "A doughnut ate me" doesn't)

The IL is, in fact, the alphabet of a machine language. This is the simplest and most primary set of symbols we can use to give commands to a computer. It's the computer's mother tongue.

Unfortunately, this mother tongue is a far cry from a human mother tongue. We both ( computers and humans) need something else, a common language for computers and humans, or a bridge between the two different worlds.

We need a language in which humans can write their programs and a language that computers may use to execute the programs, one that is far more complex than machine language and yet far simpler than natural language.

Such languages are often called high-level programming languages. They are at least somewhat similar to natural ones in that they use symbols, words and conventions readable to humans. These languages enable humans to express commands to computers that are much more complex than those offered by ILs.

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

Computer programming is the act of composing the selected programming language's elements in the order that will cause the desired effect. The effect could be different in every specific case – it's up to the programmer's imagination, knowledge and experience.

Of course, such a composition has to be correct in many senses:

alphabetically – a program needs to be written in a recognizable script, such as Roman, Cyrillic, etc.

lexically – each programming language has its dictionary and you need to master it; thankfully, it's much simpler and smaller than the dictionary of any natural language;

syntactically – each language has its rules and they must be obeyed;

semantically – the program has to make sense.

Unfortunately, a programmer can also make mistakes with each of the above four senses. Each of them can cause the program to become completely useless.

Let's assume that you've successfully written a program. How do we persuade the computer to execute it? You have to render your program into machine language. Luckily, the translation can be done by a computer itself, making the whole process fast and efficient.

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.

Due to some very fundamental reasons, a particular high-level programming language is designed to fall into one of these two categories.

There are very few languages that can be both compiled and interpreted. Usually, a programming language is projected with this factor in its constructors' minds - will it be compiled or interpreted?