1.7: TERMINOLOGY - INTERPRETER AND COMPILER

Contributed by Chuck Severance
Clinical Associate Professor (School of Information) at University of Michigan

Python is a *high-level* language intended to be relatively straightforward for humans to read and write and for computers to read and process. Other high-level languages include Java, C++, PHP, Ruby, Basic, Perl, JavaScript, and many more. The actual hardware inside the Central Processing Unit (CPU) does not understand any of these high-level languages.

The CPU understands a language we call *machine language*. Machine language is very simple and frankly very tiresome to write because it is represented all in zeros and ones:

```
001010001110100100101010000001111
11100110000011101010010101101101
...
```

Machine language seems quite simple on the surface, given that there are only zeros and ones, but its syntax is even more complex and far more intricate than Python. So very few programmers ever write machine language. Instead we build various translators to allow programmers to write in high-level languages like Python or JavaScript and these translators convert the programs to machine language for actual execution by the CPU.

Since machine language is tied to the computer hardware, machine language is not *portable* across different types of hardware. Programs written in high-level languages can be moved between different computers by using a different interpreter on the new machine or recompiling the code to create a machine language version of the program for the new machine.

These programming language translators fall into two general categories: (1) interpreters and (2) compilers.

An *interpreter* reads the source code of the program as written by the programmer, parses the source code, and interprets the instructions on the fly. Python is an interpreter and when we are running Python interactively, we can type a line of Python (a sentence) and Python processes it immediately and is ready for us to type another line of Python.

Some of the lines of Python tell Python that you want it to remember some value for later. We need to pick a name for that value to be remembered and we can use that symbolic name to retrieve the value later. We use the term *variable* to refer to the labels we use to refer to this stored data.

```
>>> x = 6
>>> print(x)
6
>>> y = x * 7
>>> print(y)
42
>>>
```

In this example, we ask Python to remember the value six and use the label *x* so we can retrieve the value later. We verify that Python has actually remembered the value using *print*. Then we ask Python to retrieve *x* and multiply it by seven and put the newly computed value in *y*. Then we ask Python to print out the value currently in *y*.

Even though we are typing these commands into Python one line at a time, Python is treating them as an ordered sequence of statements with later statements able to retrieve data created in earlier statements. We are writing our first simple paragraph with four sentences in a logical and meaningful order.

It is the nature of an *interpreter* to be able to have an interactive conversation as shown above. A *compiler* needs to be handed the entire program in a file, and then it runs a process to translate the high-level source code into machine language and then the compiler puts the resulting machine language into a file for later execution.

If you have a Windows system, often these executable machine language programs have a suffix of ".exe" or ".dll" which stand for "executable" and "dynamic link library" respectively. In Linux and Macintosh, there is no suffix that uniquely marks a file as executable.

If you were to open an executable file in a text editor, it would look completely crazy and be unreadable:

```
^?ELF^A^A^A^@^@^@^@^@^@^@^@^@^B^@^C^@^A^@^@^@\xa0\x82
^D^H4^@^@^@\x90^]^@^@^@^@^@^@4^@ ^@^G^@(^@$^@!^@^F^@
^@^@4^@^@^@4\x80^D^H4\x80^D^H\xe0^@^@^@\xe0^@^@^@^E
^@^@^@^D^@^@^@^C^@^@^@^T^A^@^@^T\x81^D^H^T\x81^D^H^S
^@^@^@^S^@^@^@^D^@^@^@^A^@^@^@^A\^D^HQVhT\x83^D^H\xe8
....
```

It is not easy to read or write machine language, so it is nice that we have *interpreters* and *compilers* that allow us to write in high-level languages like Python or C.

Now at this point in our discussion of compilers and interpreters, you should be wondering a bit about the Python interpreter itself. What language is it written in? Is it written in a compiled language? When we type "python", what exactly is happening?

The Python interpreter is written in a high-level language called "C". You can look at the actual source code for the Python interpreter by going to www.python.org and working your way to their source code. So Python is a program itself and it is compiled into machine code. When you installed Python on your computer (or the vendor installed it), you copied a machine-code copy of the translated Python program onto your system. In Windows, the executable machine code for Python itself is likely in a file with a name like:

```
C:\Python35\python.exe
```

That is more than you really need to know to be a Python programmer, but sometimes it pays to answer those little nagging questions right at the beginning.