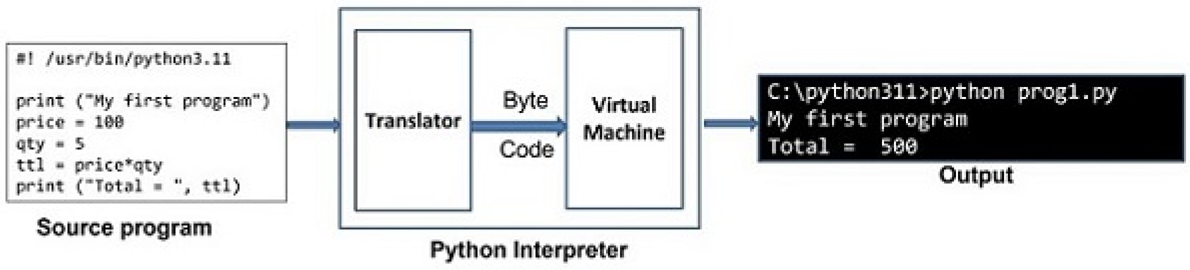
Python - Interpreter

Python is an interpreter-based language. In a Linux system, Python's executable is installed in **/usr/bin/** directory. For Windows, the executable (python.exe) is found in the installation folder (for example **C:\python311**).

This tutorial will teach you **How Python Interpreter Works** in interactive and scripted mode. Python code is executed by one statement at a time method. Python interpreter has two components. The translator checks the statement for syntax. If found correct, it generates an intermediate byte code. There is a Python virtual machine which then converts the byte code in native binary and executes it. The following diagram illustrates the mechanism:



```
#! /usr/bin/python3.11

print ("My first program")
price = 100
qty = 5
ttl = price*qty
print ("Total = ", ttl)
```

```
C:\python311>python prog1.py
My first program
Total =  500
```

Python interpreter has an interactive mode and a scripted mode.

Python Interpreter - Interactive Mode

When launched from a command line terminal without any additional options, a Python prompt **>>>** appears and the Python interpreter works on the principle of **REPL (Read, Evaluate, Print, Loop)**. Each command entered in front of the Python prompt is read, translated and executed. A typical interactive session is as follows.

```
>>> price = 100
>>> qty = 5
>>> total = price*qty
>>> total
500
>>> print ("Total = ", total)
Total = 500
```

To close the interactive session, enter the end-of-line character (ctrl+D for Linux and ctrl+Z for Windows). You may also type **quit()** in front of the Python prompt and press Enter to return to the OS prompt.

```
>>> quit()

$
```

The interactive shell available with standard Python distribution is not equipped with features like line editing, history search, auto-completion etc. You can use other advanced interactive interpreter software such as **IPython** and **bpython** to have additional functionalities.

Python Interpreter - Scripting Mode

Instead of entering and obtaining the result of one instruction at a time as in the interactive environment, it is possible to save a set of instructions in a text file, make sure that it has **.py** extension, and use the name as the command line parameter for Python command.

Save the following lines as **prog.py**, with the use of any text editor such as vim on Linux or Notepad on Windows.

```
print ("My first program")
price = 100
qty = 5
total = price*qty
print ("Total = ", total)
```

When we execute above program on a Windows machine, it will produce following result:

```
C:\Users\Acer>python prog.py
My first program
Total = 500
```

Note that even though Python executes the entire script in one go, but internally it is still executed in line by line fashion.

In case of any compiler-based language such as Java, the source code is not converted in byte code unless the entire code is error-free. In Python, on the other hand, statements are executed until first occurrence of error is encountered.

Let us introduce an error purposefully in the above code.

```
print ("My first program")
price = 100
qty = 5
total = prive*qty #Error in this statement
print ("Total = ", total)
```

Note the misspelt variable **prive** instead of **price**. Try to execute the script again as before −

```
C:\Users\Acer>python prog.py
My first program
Traceback (most recent call last):
  File "C:\Python311\prog.py", line 4, in <module>
   total = prive*qty
   ^^^^^
NameError: name 'prive' is not defined. Did you mean: 'price'?
```

Note that the statements before the erroneous statement are executed and then the error message appears. Thus it is now clear that Python script is executed in interpreted manner.

Python Interpreter - Using Shebang #!

In addition to executing the Python script as above, the script itself can be a selfexecutable in Linux, like a shell script. You have to add a **shebang** line on top of the script. The shebang indicates which executable is used to interpret Python statements in the script. Very first line of the script starts with **#!** And followed by the path to Python executable.

Modify the prog.py script as follows −

```
#! /usr/bin/python3.11


print ("My first program")
price = 100
qty = 5
total = price*qty
print ("Total = ", total)
```

To mark the script as self-executable, use the **chmod** command

```
$ chmod +x prog.py
```

You can now execute the script directly, without using it as a command-line argument.

```
$ ./hello.py
```

Interactive Python - IPython

IPython (stands for **Interactive Python**) is an enhanced and powerful interactive environment for Python with many functionalities compared to the standard Python shell. IPython was originally developed by Fernando Perez in 2001.

IPython has the following important features −

- IPython's object introspection ability to check properties of an object during runtime.
- Its syntax highlighting proves to be useful in identifying the language elements such as keywords, variables etc.
- The history of interactions is internally stored and can be reproduced.
- Tab completion of keywords, variables and function names is one of the most important features.
- IPython's Magic command system is useful for controlling Python environment and performing OS tasks.
- It is the main kernel for Jupyter notebook and other front-end tools of Project Jupyter.

Install IPython with PIP installer utility.

```
pip3 install ipython
```

Launch IPython from command-line

```
C:\Users\Acer>ipython
Python 3.11.2 (tags/v3.11.2:878ead1, Feb 7 2023, 16:38:35) [MSC v.1934
64 bit (AMD64)] on win32
Type 'copyright', 'credits' or 'license' for more information
IPython 8.4.0 -- An enhanced Interactive Python. Type '?' for help.
In [1]:
```

Instead of the regular >>> prompt as in standard interpreter, you will notice two major IPython prompts as explained below −

- In[1] appears before any input expression.
- Out[1]appears before the Output appears.

```
In [1]: price = 100
In [2]: quantity = 5
In [3]: total = price*quantity
In [4]: total
Out[4]: 500
In [5]:
```

Tab completion is one of the most useful enhancements provided by IPython. IPython pops up appropriate list of methods as you press tab key after dot in front of object.

IPython provides information (introspection) of any object by putting **?** in front of it. It includes **docstring**, function definitions and constructor details of class. For example to explore the string object var defined above, in the input prompt enter var?.

```
In [5]: var = "Hello World"
In [6]: var?
Type: str
String form: Hello World
Length: 11
Docstring:
str(object='') -> str
str(bytes_or_buffer[, encoding[, errors]]) -> str
Create a new string object from the given object. If encoding or
errors is specified, then the object must expose a data buffer
that will be decoded using the given encoding and error handler.
Otherwise, returns the result of object.__str__() (if defined)
or repr(object).
encoding defaults to sys.getdefaultencoding().
errors defaults to 'strict'.
```

IPython's magic functions are extremely powerful. Line magics let you run DOS commands inside IPython. Let us run the **dir** command from within IPython console

```
In [8]: !dir *.exe
 Volume in drive F has no label.
 Volume Serial Number is E20D-C4B9


 Directory of F:\Python311


07-02-2023 16:55           103,192 python.exe
07-02-2023 16:55           101,656 pythonw.exe
               2 File(s)    204,848 bytes
               0 Dir(s)  105,260,306,432 bytes free
```

Jupyter notebook is a web-based interface to programming environments of Python, Julia, R and many others. For Python, it uses IPython as its main kernel.