**Python:**Python was developed by **Guido van Rossum** in the **late eighties and early nineties** at the National Research Institute for Mathematics and Computer Science in the Netherlands.  
The language is named after the BBC show “**Monty Python’s Flying Circus**” and has nothing to do with reptiles.  
  
**Latest python version is 3.6** and the version which is in production is 2.7.  
  
Python provides interfaces to all major commercial databases.  
Python is an easy to learn, powerful programming language. It has efficient high-level data structures and a simple but effective approach to object-oriented programming. Python’s elegant syntax and dynamic typing, together with its interpreted nature, make it an ideal language for scripting and rapid application development in many areas on most platforms.

The Python interpreter and the extensive standard library are freely available in source or binary form for all major platforms from the Python Web site, <https://www.python.org/>, and may be freely distributed. The same site also contains distributions of and pointers to many free third party Python modules, programs and tools, and additional documentation.

The Python interpreter is easily extended with new functions and data types implemented in C or C++ (or other languages callable from C). Python is also suitable as an extension language for customizable applications.

You could write a UNIX shell script or Windows batch files for some of these tasks, but shell scripts are best at moving around files and changing text data, not well-suited for GUI applications or games. You could write a **C/C++/Java program**, but it can take a lot of development time to get even a first-draft program. Python is simpler to use, available on Windows, Mac OS X, and UNIX operating systems, and will help you get the job done more quickly.

Python is simple to use, but it is a real programming language, offering much more structure and support for large programs than shell scripts or batch files can offer. On the other hand, Python also offers much more error checking than C, and, being a ***very-high-level language***, it has high-level data types built in, such as flexible arrays and dictionaries. Because of its more general data types Python is applicable to a much larger problem domain than **Awk or even Perl**, yet many things are at least as easy in Python as in those languages.

Python allows you to split your program into modules that can be reused in other Python programs. It comes with a large collection of standard modules that you can use as the basis of your programs — or as examples to start learning to program in Python. Some of these modules provide things like **file I/O, system calls, sockets, and even interfaces to graphical user interface toolkits like Tk**.

Python is an **interpreted language**, which can save you considerable time during program development because **no compilation and linking are necessary**. The interpreter can be used interactively, which makes it easy to experiment with features of the language, to write throw-away programs, or to test functions during bottom-up program development. It is also a handy desk calculator.

Python enables programs to be written compactly and readably. Programs written in Python are typically much shorter than equivalent C, C++, or Java programs, for several reasons:

* the high-level data types allow you to express complex operations in a single statement;
* statement grouping is done by indentation instead of beginning and ending brackets;
* No variable or argument declarations are necessary.

*Python is extensible:* if you know how to program in C it is easy to add a new built-in function or module to the interpreter, either to perform critical operations at maximum speed, or to link Python programs to libraries that may only be available in binary form (such as a vendor-specific graphics library). Once you are really hooked, you can link the Python interpreter into an application written in C and use it as an extension or command language for that application.

Invoking **the interpreter** without passing a script file as a parameter brings up the following prompt −  
*$ python*  
Python 3.3.2 (default, Dec 10 2013, 11:35:01)  
[GCC 4.6.3] on Linux

Type "help", "copyright", "credits", or "license" for more information.

>>>

Below is one way to execute a python script.  
*$ python test.py*

Below is another way to execute a python script.  
*$ chmod +x test.py* # This is to make file executable

*$./test.py*

**Sample Python script:**

*#!/usr/bin/python3*

*print ("Hello, Python!")*

**Lines and Indentation**

Python doesn't use braces ({}) to indicate blocks of code for class and function definitions or flow control. Blocks of code are denoted by line indentation, which is rigidly enforced.  
The number of spaces in the indentation is variable, but all statements within the block must be indented the same amount.  
Thus, in Python all the continuous lines indented with same number of spaces would form a block.

#!/usr/bin/python3

import sys

try:

# open file stream

file = open(file\_name, "w")

except IOError:

print ("There was an error writing to", file\_name)

sys.exit()

print ("Enter '", file\_finish,)

print "' When finished"

while file\_text != file\_finish:

file\_text = raw\_input("Enter text: ")

if file\_text == file\_finish:

# close the file

file.close

break

file.write(file\_text)

file.write("\n")

file.close()

file\_name = input("Enter filename: ")

if len(file\_name) == 0:

print ("Next time please enter something")

sys.exit()

try:

file = open(file\_name, "r")

except IOError:

print ("There was an error reading file")

sys.exit()

file\_text = file.read()

file.close()

print (file\_text)

## Multi-Line Statements

Statements in Python typically end with a new line. Python does, however, allow the use of the line continuation character (\) to denote that the line should continue. For example −

total = item\_one + \

item\_two + \

item\_three

Statements contained within the [], {}, or () brackets do not need to use the line continuation character. For example −

days = ['Monday', 'Tuesday', 'Wednesday',

'Thursday', 'Friday']

## Quotation in Python

Python accepts single ('), double (") and triple (''' or """) quotes to denote string literals, as long as the same type of quote starts and ends the string.

The triple quotes are used to span the string across multiple lines. For example, all the following are legal −

word = 'word'

sentence = "This is a sentence."

paragraph = """This is a paragraph. It is

made up of multiple lines and sentences."""

## Comments in Python

A hash sign (#) that is not inside a string literal begins a comment. All characters after the # and up to the end of the physical line are part of the comment and the Python interpreter ignores them.

#!/usr/bin/python3

# First comment

print ("Hello, Python!") # second comment

This produces the following result −

Hello, Python!

You can type a comment on the same line after a statement or expression −

name = "Madisetti" # This is again comment

Python doesn't have multiple-line commenting feature. You should comment each line individually as follows −

# This is a comment.

# This is a comment, too.

# This is a comment, too.

# I said that already.

## Using Blank Lines

A line containing only whitespace, possibly with a comment, is known as a blank line and Python totally ignores it.

In an interactive interpreter session, you must enter an empty physical line to terminate a multiline statement.

## Waiting for the User

The following line of the program displays the prompt, the statement saying “Press the enter key to exit”, and waits for the user to take action −

#!/usr/bin/python3

input("\n\nPress the enter key to exit.")

Here, "\n\n" is used to create two new lines before displaying the actual line. Once the user presses the key, the program ends. This is a nice trick to keep a console window open until the user is done with an application.

## Multiple Statements on a Single Line

The semicolon ( ; ) allows multiple statements on the single line given that neither statement starts a new code block. Here is a sample snip using the semicolon −

import sys; x = 'foo'; sys.stdout.write(x + '\n')

## Multiple Statement Groups as Suites

A group of individual statements, which make a single code block are calledsuites in Python. Compound or complex statements, such as if, while, def, and class require a header line and a suite.

Header lines begin the statement (with the keyword) and terminate with a colon ( : ) and are followed by one or more lines which make up the suite. For example −

if expression :

suite

elif expression :

suite

else :

suite

**Variables:**

*#!/usr/bin/python3*

*counter = 100 # An integer assignment*

*miles = 1000.0 # A floating point*

*name = "John" # A string*

*print (counter)*

*print (miles)*

*print (name)*

In below ways also we can assign values to variables.

*a=b=c=1  
a,b,c=1,2,”string”*

Python has five **standard data types** −  
Numbers  
String  
List  
Tuple  
Dictionary

Python supports four different **numerical types −**  
Int  
Float  
Complex (Complex numbers) (Ex: 3.14j, 9.322e-36j, 3e+26J, 4.53e-7j)   
All integers in Python3 are represented as long integers. Hence there is no separate number type as long.

We can delete the number object by using **del statement**.  
var1 = 10

Var2 = 20

del var1, var2

**Strings:**Subsets of strings can be taken using the slice operator ([ ] and [:] ) with indexes starting at 0 in the beginning of the string and working their way from -1 at the end.

The plus (+) sign is the string **concatenation** operator and the asterisk (\*) is the **repetition** operator.   
  
For example −

*#!/usr/bin/python3*

*str = 'Hello World!'*

*print (str) # Prints complete string*

*print (str[0]) # Prints first character of the string*

*print (str[2:5]) # Prints characters starting from 3rd to 5th*

*print (str[2:]) # Prints string starting from 3rd character*

*print (str \* 2) # Prints string two times*

*print (str + "TEST") # Prints concatenated string*

Output:

Hello World!

H

llo

llo World!

Hello World!Hello World!

Hello World!TEST

**Lists:**Lists are the most versatile of Python's compound data types. A list contains items separated by commas and enclosed within square brackets ([]). To some extent, lists are **similar to arrays in C**.   
  
One difference between them is that all the **items belonging to a list can be of different data type**.

The values stored in a list can be accessed using the slice operator ([ ] and [:]) with indexes starting at 0 in the beginning of the list and working their way to end -1. The plus (+) sign is the list **concatenation** operator, and the asterisk (\*) is the **repetition** operator.

For example −

*#!/usr/bin/python3*

*list = [ 'abcd', 786 , 2.23, 'john', 70.2 ]*

*tinylist = [123, 'john']*

*print (list) # Prints complete list*

*print (list[0]) # Prints first element of the list*

*print (list[1:3]) # Prints elements starting from 2nd till 3rd*

*print (list[2:]) # Prints elements starting from 3rd element*

*print (tinylist \* 2) # Prints list two times*

*print (list + tinylist) # Prints concatenated lists*

*Output:*['abcd', 786, 2.23, 'john', 70.200000000000003]

abcd

[786, 2.23]

[2.23, 'john', 70.200000000000003]

[123, 'john', 123, 'john']

['abcd', 786, 2.23, 'john', 70.200000000000003, 123, 'john']

**Tuples:**   
A tuple is another **sequence data type** that is **similar to the list**. A tuple consists of a number of values separated by commas. Unlike lists, however, tuples are enclosed **within parentheses**.

The **main differences between lists and tuples** are:   
Lists are enclosed in brackets ( [ ] ) and their elements and size can be changed, while tuples are enclosed in parentheses ( ( ) ) and cannot be updated. Tuples can be thought of as read-only lists.

For example –  
*#!/usr/bin/python3*

*tuple = ( 'abcd', 786 , 2.23, 'john', 70.2 )*

*tinytuple = (123, 'john')*

*print (tuple) # Prints complete tuple*

*print (tuple[0]) # Prints first element of the tuple*

*print (tuple[1:3]) # Prints elements starting from 2nd till 3rd*

*print (tuple[2:]) # Prints elements starting from 3rd element*

*print (tinytuple \* 2) # Prints tuple two times*

*print (tuple + tinytuple) # Prints concatenated tuple*

Output:

('abcd', 786, 2.23, 'john', 70.200000000000003)

abcd

(786, 2.23)

(2.23, 'john', 70.200000000000003)

(123, 'john', 123, 'john')

('abcd', 786, 2.23, 'john', 70.200000000000003, 123, 'john')

The following code is invalid with tuple, because we attempted to update a tuple, which is not allowed. Similar case is possible with lists –

*#!/usr/bin/python3*

*tuple = ( 'abcd', 786 , 2.23, 'john', 70.2 )*

*list = [ 'abcd', 786 , 2.23, 'john', 70.2 ]*

*tuple[2] = 1000 # Invalid syntax with tuple*

*list[2] = 1000 # Valid syntax with list*

**Dictionary:**   
Python's dictionaries are kind of hash table type. They work like associative arrays or hashes found in Perl and consist of key-value pairs.   
A dictionary key can be almost any Python type, but are usually numbers or strings. Values, on the other hand, can be any arbitrary Python object.

Dictionaries are enclosed by curly braces ({ }) and values can be assigned and accessed using square braces ([]).

For example –

*#!/usr/bin/python3*

*dict = {}*

*dict['one'] = "This is one"*

*dict[2] = "This is two"*

*tinydict = {'name': 'john','code':6734, 'dept': 'sales'}*

*print (dict['one']) # Prints value for 'one' key*

*print (dict[2]) # Prints value for 2 key*

*print (tinydict) # Prints complete dictionary*

*print (tinydict.keys()) # Prints all the keys*

*print (tinydict.values()) # Prints all the values*

Output:

This is one

This is two

{'dept': 'sales', 'code': 6734, 'name': 'john'}

['dept', 'code', 'name']

['sales', 6734, 'john']

Dictionaries have no concept of order among elements. It is incorrect to say that the elements are "out of order"; they are simply unordered.

**Data type conversion:**

int(x [,base])—Converts x to an integer. Base specifies the base if X is a string.

Float(x)—Converts x to a float

Chr(x) —Converts an integer to a character  
str(x) —Converts x to string  
tuple(s) —Converts s to a tuple  
list(s) —Converts s to a list  
set(s) —Converts s to a set

**Operators:**

# Database Access: Python Database API supports a wide range of database servers such as −

Gadfly

mSQL

MySQL

PostgreSQL

Microsoft SQL Server 2000

Informix

Interbase

Oracle

Sybase

SQLite