CHAPTER 1

Introduction

Python is a high-level, interpreted, object-oriented and general-purpose programming language. It was created by **Guido van Rossum** (referred to its father) during 1985- 1991. Like Perl, Python source code is also available under the GNU General Public License (GPL).

The *General Public License* (*GPL*) is a widely-used free software license, which guarantees end users the freedom to run, study, share and modify the software.

General Purpose means it can be used for multiple application such as Data Science, Machine Learning, Desktop Application, Web Application, Scripts etc.

High Level Programming Language means human understandable language i.e. Human readable.

Python is a widely used dynamic programming language compared to other languages such as Java, Perl, PHP, and Ruby. It is often termed as a **scripting language**. It provides support for automatic memory management, multiple programming paradigms, and implements the basic concepts of *object-oriented programming (OOP)*.

Python is a strongly-typed procedural language along with support for a huge and broad standard library. The library of Python provides support for Artificial Intelligence, Natural Language Generation, Neural Networks, Data Analytics and other advanced fields of Computer Science

Python is derived from many other languages, including ABC, Modula-3, C, C++, Algol-68, SmallTalk, and Unix shell and other scripting languages. It has a unique syntax, which makes it different from other programming languages like Java, C++, and C.

Python programming is widely used in Artificial Intelligence, Natural Language Generation, Neural Networks and Data Analytics and other advanced fields of Computer Science. Python had deep focus on code readability.

History of 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.**
- **Guido Van Rossum was a Dutch programmer**, who wrote Python as a hobby programming project back in the late 1980s. Since then it has grown to become one of the most polished languages of the computing world.
- In his own words, Guido revealed the secret behind the inception of Python. He started working on it as a weekend project utilizing his free time during Christmas in Dec'1989. He originally wanted to create an interpreter, a descendant of the **ABC programming language** of which he was a contributing developer. And we all know that it was none other than Python which gradually transformed into a full-fledged programming language.
- Guido initially thought the Unix/C hackers to be the target users of his project. And more importantly, he was fond of watching the famous comedy series [The Monty Python's Flying Circus]. Thus, the name Python struck his mind as not only has it appealed to his taste but also to his target users.

"Python is an experiment in how much freedom programmers need. Too much freedom and nobody can read another's code; too little and expressiveness is endangered."

- Guido van Rossum

Python Features

Python's features include –

- **Easy-to-learn** – Python has few keywords, simple structure, and a clearly defined syntax. This allows the student to pick up the language quickly.
- **Easy-to-read and Expressive**– Python code is more clearly defined and expressive means that it is more understandable and readable.
- **Interpreted**– A program code is called *source code*. After writing a Python program, we should compile the source code using Python compiler. Python compiler translate the Python source code into an intermediate code called *byte code*. This byte code is then executed by PVM(Python **V**irtual **M**achine). Inside the PVM, an interpreter converts the byte code instructions into machine code so that the processor will understand It is an interpreter executes the code line by line at a time. This makes debugging easy and thus suitable for beginners.

- **Huge standard library** – Python's has large & broad library and provide rich set of module and functions for rapid application development**.**
- **Portable** – In Python, many types of computers and operating system are in use throughout the world and are connected to the Internet. For downloading programs through different platforms connected to the Internet, some portable, executable code is needed. Python does not have implementation dependent aspects and it yields or gives same result on any machine.
- **Extensible** – The programs or pieces of code written in C or C++ can be integrated into Python and executed using PVM. There are other flavours of Python where programs from other language can be integrated into Python. For example, *Jython* is useful to Integrate Java code into Python programs and run on JVM (**J**ava **V**irtual **M**achine). Similarly, *Iron Python* is useful to integrate .NET programs and libraries into python programs and run on CLR (**C**ommon Language **R**untime).
- **Databases** – Python provides a better interface to connect its programs to all major database like Oracle, MangoDB, Sybase and MySql.
- **GUI Programming** – Python supports GUI applications that can be created and ported to many system calls, libraries and windows systems, such as Windows MFC, Macintosh, and the X Window system of Unix.
- **Scalable** – A program would be scalable if it could be moved to another operating system or hardware and take full advantage of the new environment in terms of performance. Python programs are scalable since they can run on any plateform and use the features of the new plateform effectively.

Flavors of Python (Different Python Compiler)

Flavors of Python refer to different types of Python compilers. These flavors are useful to integrate various programming languages into Python. The following are some of them:

CPython: This is for implementation of Python programming language which is designed to run C code on Python plateform.

Jython : This is for implementation of Python programming language which is designed to run java on Java plateform.

IronPython: This is for the implementation of Python language for .NET framework. This is written in C# (C sharp) language. The Python program when compiled gives an intermediate language (IL) which runs in CLR (**C**ommon **L**anguage **R**untime) to produce the output.

PyPy : (Alternate implementation of Python runs faster than regular python due to presence of JIT (Just in time) compiler) Actually, PyPy is written in a language called *RPython* which was created in Python language. RPython is suitable for creating language interpreters. PyPy programs run very fast since there is JIT (Just In Time) compiler added to the PVM. Since the original Python uses only an interpreter in the Python Virtual Machine (PVM), the Python programs run slowly. To improve the speed of execution, a compiler called JIT is introduced into the PVM of PyPy. Hence PyPy program run faster than those of Python.

RubyPython: This is the bridge between the Ruby and Python interpreters. It encloses a Python interpreter inside Ruby applications.

Pythonxy: This is pronounced as Python xy and written as python(X,Y). This is the Python implementation that we get after adding scientific and engineering related packages.

AnacondaPython: When python is redeveloped for handling large-scale data processing, predictive and scientific computing, it is called Anaconda Python.

Python Version

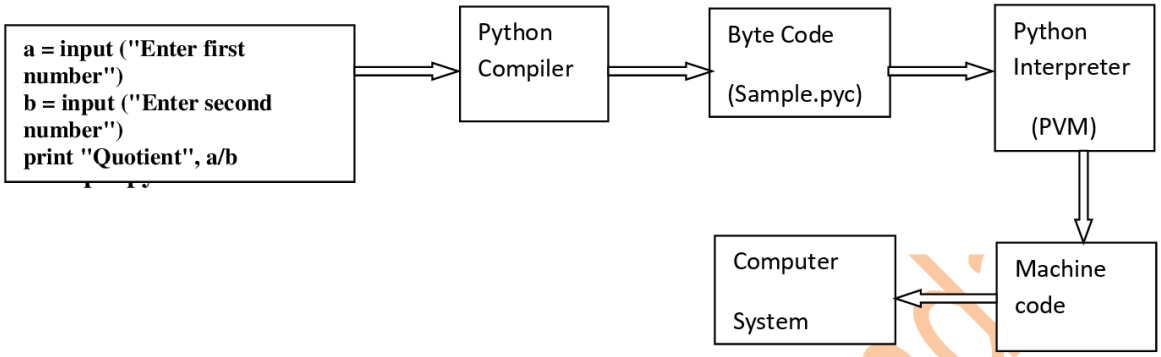
- Python 1.0 (1994 Jan)
- Python 2.0 (2000 Oct)
- Python 3.0 (2000 Dec)
- Python 3.6.3 (2016)
- Python 3.7.2 (2019)
- Python 3.7.4 (July 2019)
-

Note: Python does not support backward compatibility as feature in version 3x were not available in Python 2x.

Python Virtual Machine (PVM)

We all know that computers understand only machine code that comprises 1s and 0s. Since computer understands only machine code, it is imperative that we should convert any program into machine code before it is submitted to the computer for execution. For this purpose, we should take the help of a compiler. A compiler normally converts the program source code into machine code.

A Python compiler does the same task but in a slightly different manner. It converts the program source code into another code, called byte code. Each Python program statement is converted into a group of byte code instructions. Then what is byte code? Byte code represents the fixed set of instructions created by Python developers representing all types of operations. The size of each byte code instruction is 1 byte (or 8 bits) hence these are called byte code instructions.



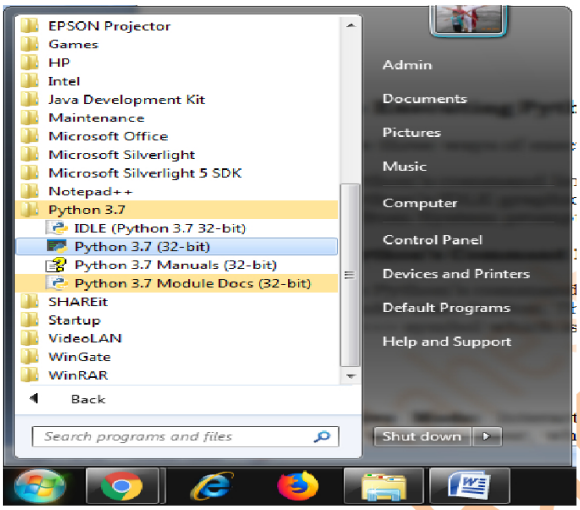
How to Executing Python Program

There are three ways of executing a Python program.

- ✓ Using Python's command line window
- ✓ Using Python's IDLE (**I**ntegrated **D**evelopment **E**nvironment) graphics window
- ✓ Directly from System prompt.

Using Python's Command Line Window

Click the Python's command line icon which already added to the program file menu under start button.



This will open Python's command line window. We can see **>>>** symbol which is call Python prompt. Type our first Python program at the **>>>** prompt, as shown below:

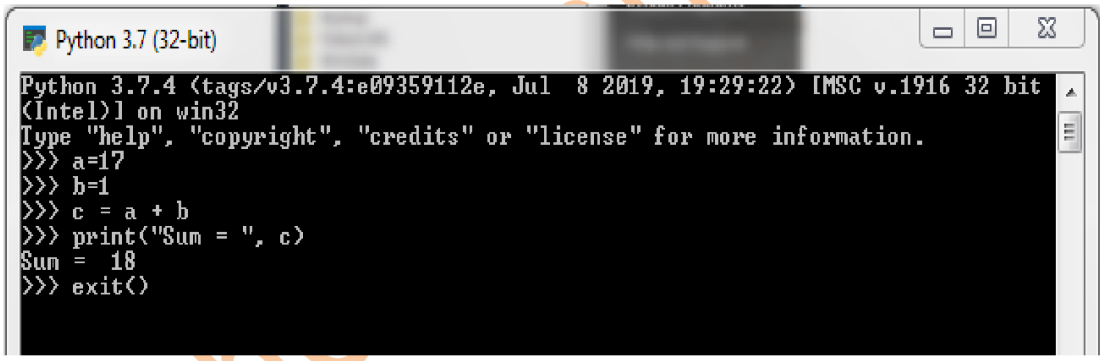
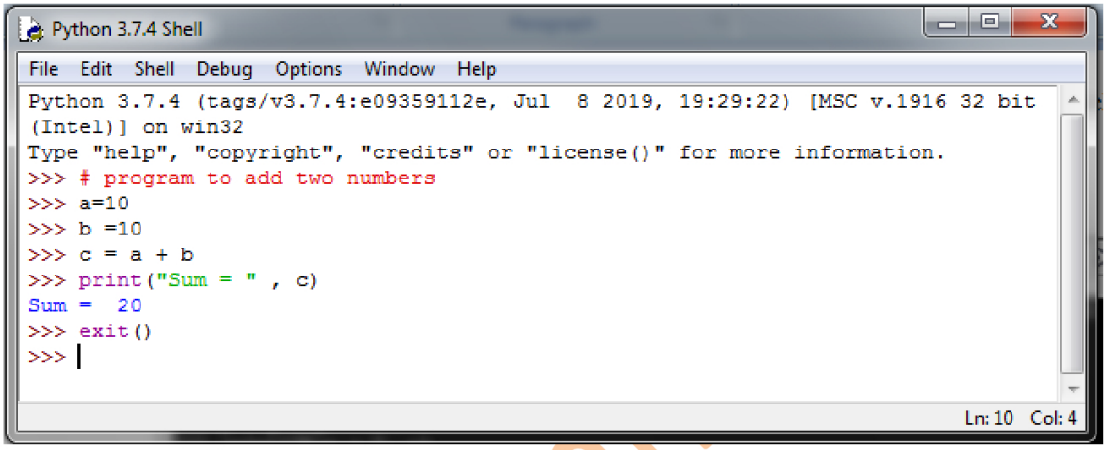


Figure: Python command prompt window

After typing the last line and pressing the Enter button, we can see the result of our program. After that, type exit() or quit() at the Python prompt. This will terminate the PVM and close the window.

Using Python's IDLE Graphic Window

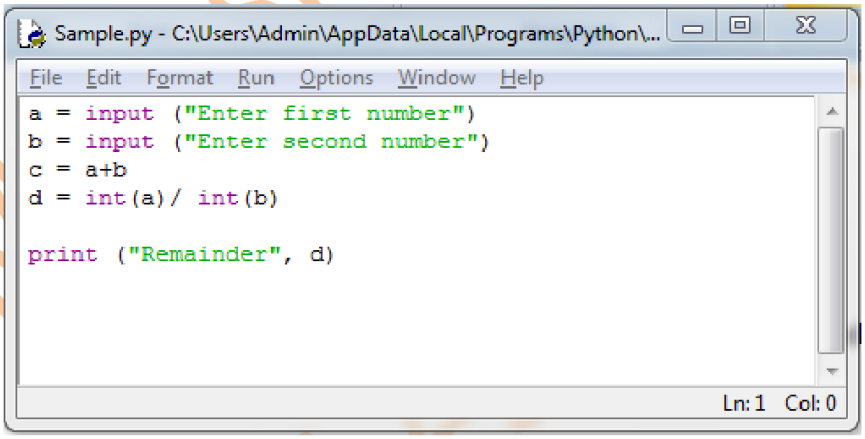
It is possible to execute a Python program by going to IDLE environment which provides graphical interface to the user. Click the Python's IDEL icon. This will open the IDLE window.



Running a program in IDLE window

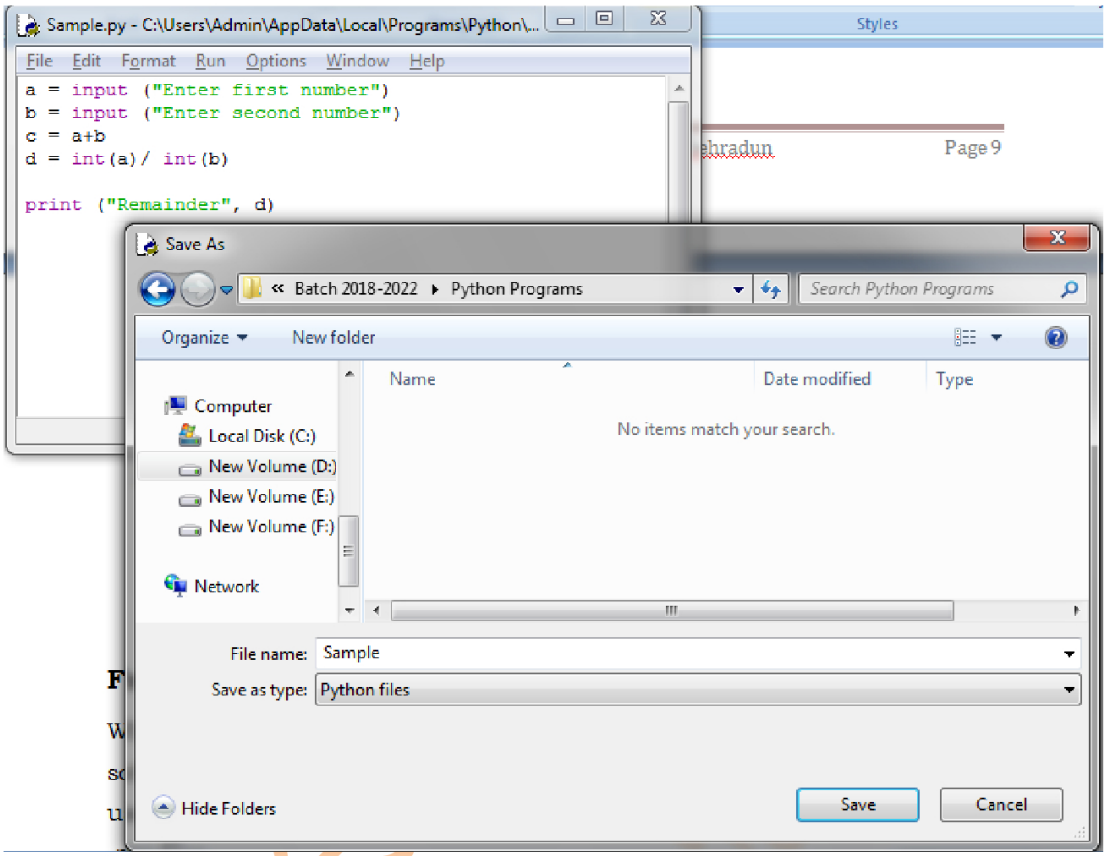
Running Directly from System Prompt

1. Open a text editor (notepad or Python IDLE mode editor) and type the program as shown in figure given below:



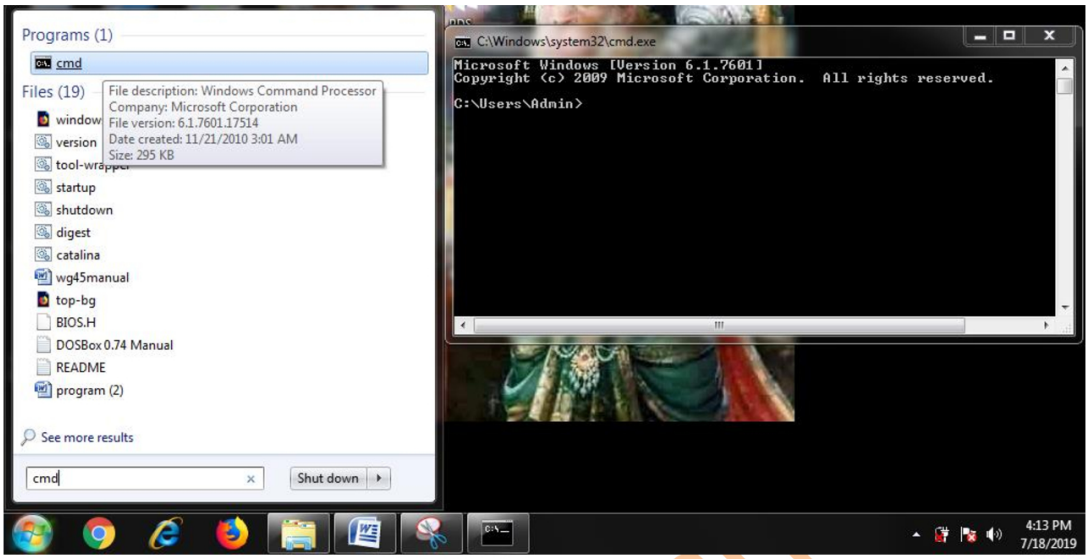
Typing Python program in IDLE mode editor

2. Save the program by clicking File→ Save As. Type the program name as "sample.py", as shown in figure given below



Saving the Python program in a folder

3. Open the command prompt (right click on the start button and then click on 'Run' and type 'command or 'cmd'), as shown in figure given below:



Going to windows or system command prompt window

4. Go to the director/folder where our Python program is saved. Suppose, we saved our program in 'D:' drive and in the 'Batch 2018-2021/Python programs' directory/folder. Now change the directory to 'D:\Batch 2018-2022\Python Programs' using the CD (change directory command) as:

C:\users\admin> d:

D:\>cd Batch 2018-2022

D:\Batch 2018-2022> cd Python Programs

D:\ Batch 2018-2022\Python Programs>

To see the directory contents and verify out 'sample.py' program is already available in that directory or not, we can display the directory contents as:

D:\ Batch 2018-2022\Python Programs>dir

The above 'dir' command will display the contents of the directory, as shown in figure given below:

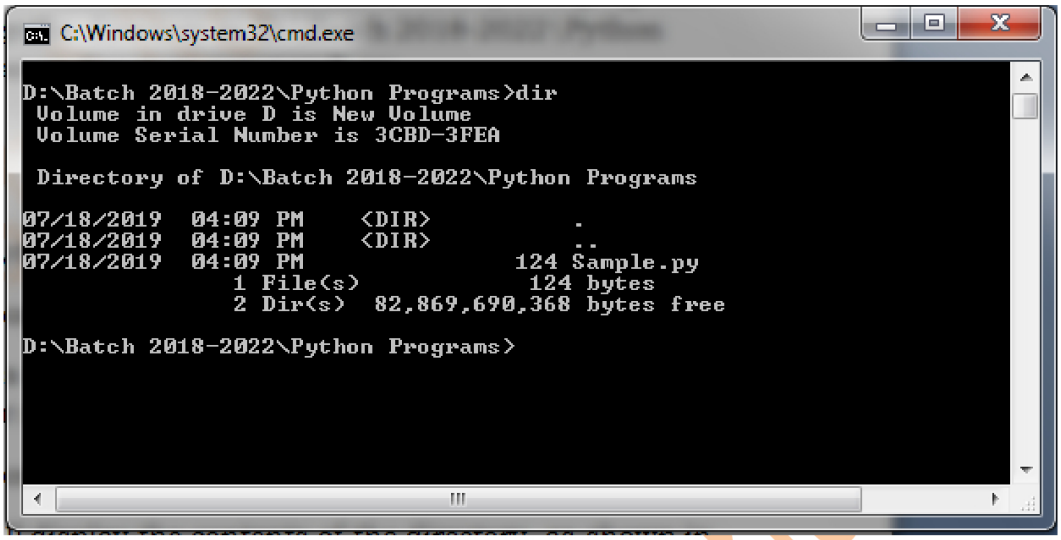
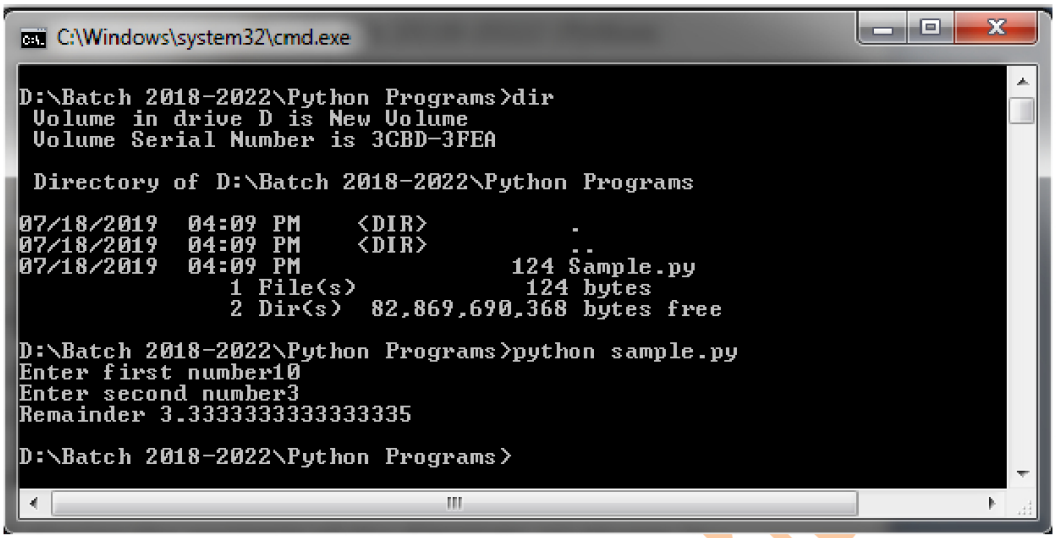


Figure: Entering the directory where Python Programs are saved.

Let's now execute the 'sample.py' program by calling the python command along with the file name at command prompt windows as:

D:\ Batch 2018-2022\Python Programs>python sample.py

Here, the command 'python' is a program that contain Python compiler as well as PVM. Hence, first of all, Python compiler compiles and converts our program into bytecode instructions. Then the PVM runs those byte code instructions to produce the final results. That means, we need not enter two separate commands to compile and then run our program. The single python command doest both tasks and gives the final results, as shown below:



Frozen Binaries

When a software is developed in Python, there are two ways to provide the software to end user. The first way is to provide the .pyc files to the user. The user will install PVM in his computer and run the byte code instruction of the .pyc files.

The other way is to provide the .pyc files, PVM along with the necessary Python library. In this method, all the .pyc files, related Python library and PVM will be converted into a single executable file (generally with .exe extension) so that the user can directly execute that file by double clicking on it. In this way converting the Python programs into true executables is called **frozen binaries**. But frozen binaries will have more size than that of simple .pyc files since they contain PVM and library files also.

Garbage Collection Python

Python's memory allocation and deallocation method is automatic. The user does not have to preallocate or deallocate memory similar to using dynamic memory allocation in languages such as C or C++. Python uses two strategies for memory allocation:

- Reference counting
- Garbage collection

Prior to Python version 2.0, the Python interpreter only used reference counting for memory management. ***Reference counting*** works by counting the number of times an object is referenced (or used). When an object is referenced twice, its reference count will be 2. When an object has some count, it is being used in the program and hence garbage collector will not remove it from memory. When an object is found with a reference count 0, garbage collector will understand that the object is not used by the program and hence it can be deleted from memory. Hence the memory allocated for that object is deallocated or free.

When references to an object are removed, the reference count for an object is decremented. When the reference count becomes zero, the object is deallocated. Ex-

```
# Literal 9 is an object
b = 9

# Reference count of object 9
# becomes 0.
b = 4
```

The literal value 9 is an object. The reference count of object 9 is incremented to 1 in line 1. In line 2 its reference count becomes zero as it is dereferenced. So garbage collector deallocates the object.

A ***reference cycle*** is created when there is no way the reference count of the object can reach to zero. A reference cycle is a cycle of references pointing to the first object from last object. For example, take three objects A, B and C. The object A refers to the object B whereas the B holds a reference to the object C. Now if the object C refers to the first object A, it will form a reference cycle.

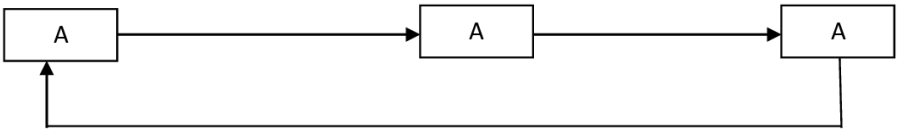


Figure: A reference cycle of three objects

There are two ways for performing manual garbage collection: time-based and event-based garbage collection.

Time-based garbage collection is simple: the garbage collector is called after a fixed time interval.

Event-based garbage collection calls the garbage collector on event occurrence. For example, when a user exits the application or when the application enters into idle state.

Comparisons between C and Python

Number	C	Python
2	C is a procedure oriented language	Python is object oriented language
3	C is compiler dependent language	Python is interpreter dependent language
4	Complied programs usually executes faster as compared to interpreted programs that why C is faster as compared to Python.	Interpreted programs usually executes slower as compared to compiled programs that why Python is slower as compared to C.
5	Error debugging is difficult as its compiler dependent language, it takes entire source code and compile it, then shows all error.	Error debugging is simple. as its interpreted language, it takes only one in instruction at time and compile and execute simultaneously. if error is there it shows instantly and stops execution, at that instruction.
6	C is Free form of language means	Python is an indentation language,

	you can write statements anywhere on the line.
	Example1:
	int a=10;
	float f=20.5;
	/*Works fine in C.*/
	
	Example2:
	int a=10;
	float f=20.5;
	/*Even this Works fine in C.*/
		means statements should start from the same column or same indent.
		Example1:
		a=10;
		f=20.5;
		/*Works fine in Python.*/
		
		Example2:
		a=10;
		f=20.5;
		/*this doesn't Work in Python as a and b not started in same coloum or same indent. you will get error as unexpected indent*/

7	In C you have to define variables type Explicitly.
	Example:
	int a;
	float f=20.5;
	/*In C we have to tell compiler a is integer and f is float by using keywords int and float respectively.*/
		In Python you don't have to define variables type Explicitly, Interpreter understands, variables type based on data whether it is integer or float or String.
		Example:
		a=10;
		f=20.5;

8	In C no need to initialize variable before we use.
	Example:
	int a;
	print("%d",a);
	//This works fine in c.
		In python you have to initialize the value of variable where you define it.
		Example:
		a;
		print(a);
		/*This will not work in Python,You will get error message
		a is not defined .*/

9	C doesn't support Function renaming mechanism means we can't use same function by two different names	Python supports Function renaming mechanism means we can use same function by two different names
		Example:
		def test():
		print("we are in test function");
		dummy=test;
		dummy();/*this will print we are in test function */
		/*Function dummy() will behave same as function test()*/

10.	Pointers concepts is available in C	Python does not use pointers.
11.	C does not have exception handling facility and hence C programs are weak.	Python handles exceptions an hence python programs are robust.
11	C does not contain a garbage collector	Automatic garbage collector is available in Python.
12.	It is compulsory to declare the data types of variables, array etc. in C	Type declaration is not required in Python.

Python Input and Output version (Version 3.7.4)

The purpose of computer is to process data and return results. It means that first of all, we should provide data to the computer. The given to the computer is called *input*.

INPUT?

Input is nothing but reading the data from the keyboard or any input devices, then we can call it as input.

The results by the computer are called *output*.

OUTPUT?

Output is nothing but displaying the data on the monitor or any output device, then we can call it as output.

So, we can say that computer takes input, processes that input and produces the output.

To provide input to a computer, Python provide some statements which are called Input statements. Similarly, to display the output, there are Output statements available in Python.

Output Statements

To display output or results, Python provides the print() function. This function can be used in different formats which are discussed hereunder:

"print()" function can be used for displaying the data on monitor or console.

New print format is introduced in Python 3.x version

The difference between version 2.x and version 3.x is, parenthesis is not used in version 2.x, but parenthesis is used in version 3.x

The print() Statement

When the print() function is called simply, it will throw the cursor to the next line. It means that a blank line will be displayed.

The print("String") Statement

A string represents a set of characters. When a string is passed to the print() function, the string is displayed s it is.

The basic syntax for print function is "print ('statements'). print syntax is used to print the every statement in new line.

Note: Please remember that in case of strings, double quotes and single quotes have the same meaning and hence can be used interchangeably.

See the example below:

```
>>> print ('string in single quotes vrsion3')
string in single quotes vrsion3
>>>
>>> print ("string in double quotes vrsion3")
string in double quotes vrsion3
>>>
>>> print ('''string in double quotes vrsion3''')
string in double quotes vrsion3
>>>
```

Below is the syntax to print the multiple statements in single line. To print the multiple statements, we need to separate each statement with comma ‘,’

```
>>> print ('Hello','In Python 3.x','In single quotes','Multiple statements')
Hello In Python 3.x In single quotes Multiple statements
>>>
>>> print ("Hello","In Python 3.x","In double quotes","Multiple statements")
Hello In Python 3.x In double quotes Multiple statements
>>>
>>> print ('''Hello''','''In Python 3.x''','''In triple quotes''','''Multiple statements''')
Hello In Python 3.x In triple quotes Multiple statements
>>>
```

We can use escape sequence characters inside the print() function. An escape sequence is a character that contains a special meaning. For example, ‘n\’ indicates new line. ‘\t’ represents tab space.

print(“This is the \n first line”)

Output:

This is the
first line

To escape the effect of escape sequence, we should add one more ‘\’ (backslash) before the escape sequence character. For example, ‘\\n’ display ‘\n’ and ‘\\t’ display ‘\t’ .

print("this is the \\n first line")
Output
This is the\n first line.

We can use repetition operator (*) to repeat the strings in the output as:

print(4* 'GEHU')

Output:

GEHUGEHUGEHUGEHU

The operator '+' when used on numbers will perform addition operation. The same '+' will not do addition on operation on strings as it is not possible to perform arithmetic operations on strings. When we use '+' on strings, it will join one string with another string. Hence '+' is called concatenation operator when used on string.

print("Graphic Era=" + "Dehradun")

Output:

Graphic Era=Dehradun

The '+' operator in the preceding statement joined the two strings without any space in between. We can also write the preceding statement by separating the string using ',' as:

print("Graphic Era=" , "Dehradun")

Output:

Graphic Era= Dehradun

In this case, a space is used by the print() function after each string. In the output, observe the single space after the string 'Graphic Era='. When the print() function sees a comma, it will assume that the values are different and hence a space should be used between them for clarity.

The print(variables list) Statement

We can also display the values of variables using the print() function. A list of variables can be supplied to the print() function as:

```
>>>
>>> a=35
>>> b=54
>>> print (a)
35
>>> print (b)
54
>>> print (a,b)
35 54
>>> print('complex num',2+1j)
complex num (2+1j)
>>> print (5.4)
5.4
>>>
```

Observe that the values in the output are separated by a space by default. To separate the output with comma, we should use '*sep*' attribute as shown below. '*sep*' represents separator. The format is "sep=" which are used to separate the values in the output.

```
Python 3.7.4 (tags/v3.7.4:e09359112e,
(Intel)] on win32
Type "help", "copyright", "credits" o
>>> a=8
>>> b =1
>>> print(a,b, sep=",")
8,1
>>> print(a, b, sep=':')
8:1
```

When several print() functions are used to display output, each print() function will display the output in a separate line as shown below:

print("Do")
print("Good")
print("Have")
print("Good")

Output:

Do
Good
Have
Good

Each print() function throws the cursor into the next line after displaying the output. This is the reasons we got the output in 3 lines. We can ask the print() function not to throw the cursor into the next line but display the output in the same line. This is done using 'end' attribute. The way one can use it is end=" characters" which indicate the ending characters for the line. See the example:

```
print("Hello", end='\t');
print("Dear", end='\t ');
print("How are u?", end='\t ');
```

Output:

Hello Dear How are u?

Format specifier can also be used to print the variable data.

```
>>> a=78
>>> b=6.7
>>> s="Hello"
>>> print("%s number is: %d and float is: %f"%(s,a,b))
Hello number is: 78 and float is: 6.700000
>>>
>>>
```

Input Statements

To accept input from keyboard, Python provides the input() function. This function takes a value from the keyboard and **returns it as a string**. For example,

To accept input from keyboard, Python provides the input() function. This function takes a value from the keyboard and **returns it as a string**. For example,

```
str = input( )  #this will wait till we enter a string
GEHU            #enter this string
```

print(str)
GEHU

It is better idea to display a message to the user that the user understands what to enter. This can be done by writing a message inside the input() functions as:

str = input('Enter a number: ')
Enter a number: 18
print(x)
18

Once the value comes into the variable 'str', it can be converted into 'int' or 'float' etc. This is useful to accept numbers as:

str = input('Enter a number: ')
Enter a number: 18
x = int(str) # str is converted into int
#x = float(str) # str is converted into float

print(x)
18

We can use the int() or float() function before the input() function to accept an integer from the keyboard as:

x = int(input("Enter a number: "))
#x = float(input("Enter a number: "))
Enter a number: 18
print(x)
18

Lest explore the input() statement in more detail with some examples:

#Pyhton program to convert numbers from other systems into decimal number system.

str = input("Enter hexadecimal number: ")
n = int(str,16) #inform the number base is 16
print("Hexadecimal to Decimal = ", n);

```
print("Hexadecimal no. = {0} and Equivalent Decimal no. = {1}".
format(str,n));

str = input("Enter Octal number: ")
n = int(str,8)  #inform the number base is 8
print("Octal to Decimal = ", n);
print("Octal no. = {0} and Equivalent Decimal no. = {1}". format(str,n))

str = int(input("Enter Binary number: "), 2) #inform the number base is 2
print("Binary =to Decimal = ", n);
print("Binary no. = {0} and Equivalent Decimal no. = {1}". format(str,n));
```

To accept more than one input in the same line, we can use a for loop along with the input() function in the following format:

```
a,b = [ int(x) for x in input("Enter two numbers: ").split()]
```

In the previous statement, the input() function will display the message 'Enter two numbers: ' to the user. When the user enters two values, they are accepted as a strings. These strings are divided wherever a space is found by split() method. So, we get two strings as element of list. These strings are read by for loop and converted into integers by the int() function. These integers are finally stored into a and b.

The split() method by default splits the values where a space found. Hence, while entering the numbers, the user should separate them using a space. The square bracket [] around the total expression indicates that the input is accepted as elements of list.

A Python program to accept 3 integers in the same line and display their sum.

```
a,b,c = [int(x) for x in input("Enter three numbers: ").split()]
print("Sum of {0} , {1} & {2} is = {3}" . format(a,b,c,(a+b+c)))
```

Output:

```
Enter three numbers: 10 20 30
Sum of 10, 20 & 30 is = 60
```

The eval() function takes a string and evaluates the results of the string by taking it as a Python expression. For example, let's take a string: "a+b-9" where a = 8 and b = 10. If we pass the string to the eval() function, it will evaluate the string and returns the result. Consider the following example:

```
a,b=5,10
result = eval("a+b-8")
print(result()
```

Output:
9

We can use the eval() function along with input() function. Since the input() function accept a value in the form of a string, the eval() function receives that string and evaluates it.

Consider the following example:

```
result = eval(input("Enter an expression: "))
print("Result = %d " %x)
```

Output:
Enter an expression: 8 + 10 -9
Result = 9

Command Line Arguments to the program

Till now, we have taken input in python using input() function. There is another method that uses command line arguments. The command line arguments must be given whenever we want to give the input before the start of the script, while on the other hand, input() function is used to get the input while the python program / script is running.

The command line arguments in python can be processed by using either 'sys' module or 'argparse' module.

For example, we write a program by the name 'argsumdemo.py' that takes two numbers and adds them.

```
#to add two numbers, save this as argsumdemo.py
import sys
#convert args into integers and add them
sum = int(sys.argv[1] + sys.argv[2])
print("Sum = ", sum)
```

We can supply the two numbers as input at the time of running the program at command prompt as:

C:\ D:\Batch 2018-2022\Python Programs>python argssumdemo.py 10 8

Output:

Sum =22

Please note the following points about the above program:

- They sys module in Python is one of many available modules of library code.
- The sys.argv takes the command line arguments in the form of a list. It is basically holding the command line arguments of our program. Don't forget that the counting starts at zero (0) not one (1).
- The first element in the list is the name of the file. (argssumdemo.py)
- The arguments always come in the form of a string even if we type an integer in the argument list. We need to use int() function to convert the string to integer.

sys.argv

'argssumdemo.py'	'10'	'8'
argv[0]	**argv[1]**	**argv[2]**

Figure: Command line args are stored as string in argv list

If we want to find the number of command line arguments, we can use the len() function as: len(argv). The following program reads the command line arguments entered at command line prompt and display them.

```
#to display command line args. Save this as cmd.py
import sys
n = len(sys.argv)     # n is the number of arguments
```

```
args = sys.argv      #args list contains arguments
print("No. of command line args = ", n)
print("The args are: ", args)
print("The args one by one:")
for a in args:
        print(a)
```

C:\ D:\Batch 2018-2022\Python Programs>python cmd.py 10 8 GEHU

Output: