Chapter 2 Getting Started with Python

Introduction

Python Programming language was developed by **Guido Van Rossum** in February 1991. It is based on or influenced with two programming languages:

1. ***ABC language***, a teaching language created as a replacement of BASIC, and
2. ***Modula-3***

Features of Python

1. **Easy to Use**
   - It is a very high level language and thus very programmer friendly.
   - Easy to use object oriented language with very simple syntax rules.

2. **Expressive Language**
   - Python's expressiveness means it is more capable to expressing the code's purpose than many other languages.

   e.g.

   In C++ : addition of two numbers

   ```
   int a=2; int b=3;
   int c = a +b ;
   ```

   In Python : addition of two numbers

   ```
   a= 2 , b=3 , a + b
   ```

3. **Interpreted Language**
   - It is an interpreted language, not a compiled language. This means that the Python installation interprets and executes the code line by line at a time.

4. **Completeness**
   - When you install Python, you get everything you need to do real work. All types of required functionality is available through various modules of ***Python standard library***.

5. **Cross Platform Language**
   - Python can run equally well on variety of platforms- Windows, Linux/UNIX, Macintosh, smart phones etc.

6. **Free and Open Source**
   - Python language is ***freely available*** i.e. without any cost (from www.python.org). It is open and available to everyone.

7. **Variety of Usage/ Applications**
   - It is being used in many diverse fields/applications like ***Web Applications, Game development, Database Applications etc.***

Disadvantages associated with Python

1. **Not the Fastest Language**

- It is an interpreted language not a fully compiled one. Python is first semi-compiled into an internal byte-code, which is then executed by a Python interpreter.

2. **Lesser Libraries than C,Java, Perl**

- Python library is still not competent with languages like C, Java, Perl as they have larger collections available.

3. **Not Easily Convertible**

- Translating a Python program into some other language is a problem, because of its weak syntax.

Interpretation and Compilation

1. **Interpreter:** This language processor converts an HLL program into machine language by converting and executing it **line by line.** If there is any **error** in any line, it **reports it at the same time** and program execution cannot resume until the error is rectified.

2. **Compiler:** It also converts the HLL program into machine language but the conversion manner is different. It converts the entire HLL program **in one go**, and **reports all the errors** at the end of the program along with the line numbers.

Working in Python

Before you start working in Python, you need to install Python on your computers. There are multiple **Python distribution** available today.

1. Default installation available from www.python.org is called CPython installation and comes with Python interpreter and Python IDLE.
2. Other popular Python distributions are: ***Anaconda Python distribution*** (comes preloaded with many packages and libraries) , ***Spyder IDE*** etc.

Working with Default Cpython distribution

- In this we can work in two modes : **(i) in Interactive mode (also called immediate mode)**
  (ii) in Script mode

Working in Interactive Mode

- Interactive mode of working means you type the command-one command at a time, and the Python executes the given command there and then and gives you output.
- In interactive mode , you type the command in front of Python command prompt >>> . For example, if you type 2 + 5 in front of Python prompt, it will give you result as 7.

```
>>> 2 + 5
7
```

- ***Interactive interpreter of Python is also called Python Shell.***

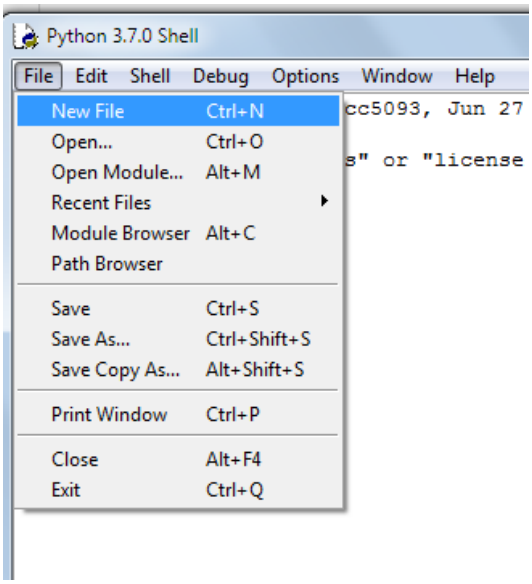
Working in Script Mode

- The problem associated with the interactive mode is that it does not save the commands entered in the form of program and the output is sandwiched between the command lines. The solution to this problem is the **Script Mode**.

Steps to work in Script Mode

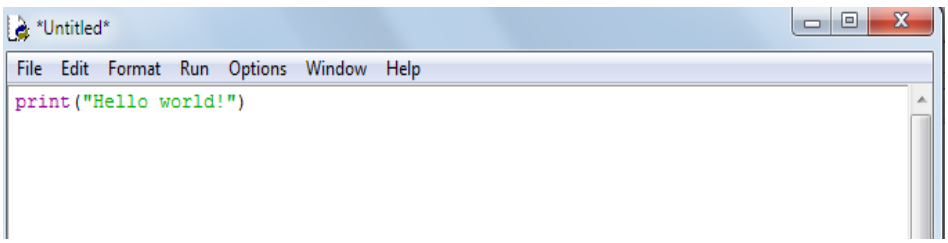
Step 1: Create Program File/ Script

(i) Open python Shell. **Go to File → New File.**

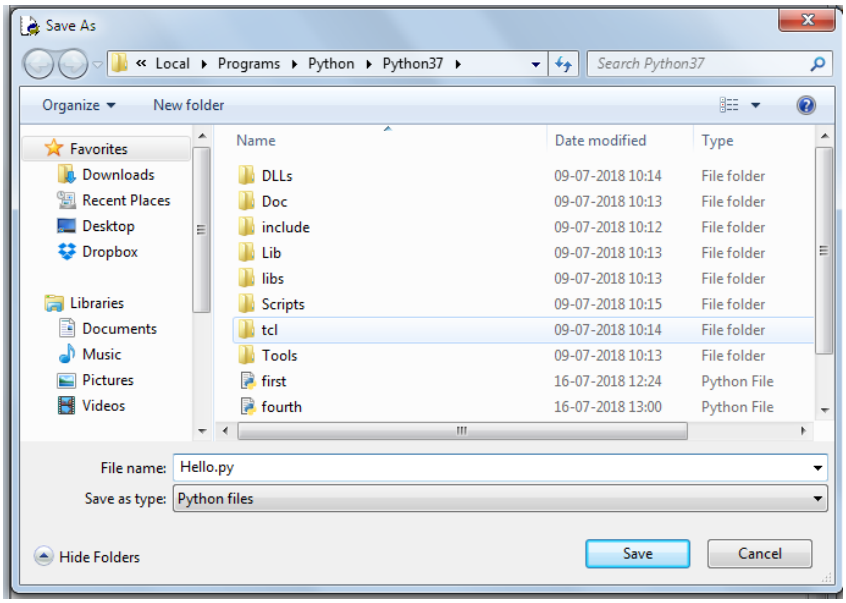


(ii) In the New Window that opens, type the commands you want to save in the form of a program(or script).

e.g. type the following line : **print("Hello World!")**

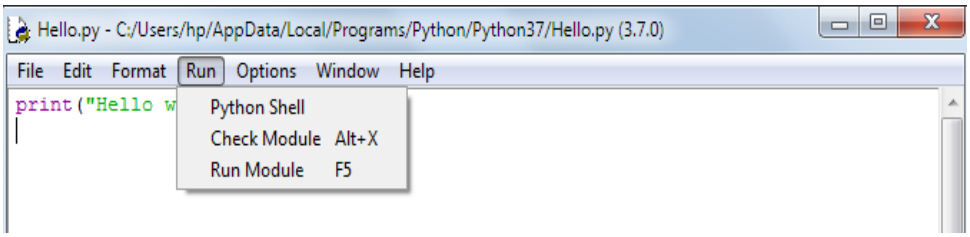


(iii) Click **File → Save** and then save the file with an extension .py . The Python program has .py extension.



Step 2: Run Program File/ Script

(i) Go to Option **Run → Run Module** in the Open program/script file's window. You may also press F5 key.



(ii) It will execute all the commands stored in program/script that you had opened and show you the complete output in a separate Python Shell window.

Analysing Script and Output

Type the following lines in script Mode :

```
#My first Program
print("Hello world!")
```

You will observe that Python gave you just one line's output as :

Hello World!

The reason being that any line that begins with a **# symbol is a comment in Python**. That is, it is for the Programmer's information only; Python will completely ignore all the lines starting with a #.

Understanding print( )

- To print or display output , Python 3.x provides print( ) function.

  E.g.1

  print("Hello World!")

  the above statement will print Hello World!

  E.g.2

  print(' My name is Dev ');

  Notice the **String(set of zero or more characters)** given in above two print statements. The first string is enclosed in double quotes and the second string is enclosed in single quotes. Both these strings are valid in Python.

- But the following print statement is not valid :

  print('Hey there")

Questions

Q1. Python is a Free and Open Source language. What do you understand by this feature?

Q2. Who developed Python Programming Language?

Q3. 'Python is an interpreted high level language'. What does it mean to you?

Q4. What is the difference between interactive mode and Script mode in Python ?

Q5. What will be the output of following code :

```
#This is a sample program
#to output simple statements
#print("Such as")
  print("Take every chance.")
  print("Drop every year")
```

Q6. Which of the following are not valid strings in Python ?

(a) "Hello" (b)'Hello' (c) "Hello' (d) 'Hello" (e) {Hello}

****

Chapter 3 Python Fundamentals

TOKENS

The smallest individual unit in a program is known as a Token or a lexical unit. Python has following tokens:

(i) Keywords
(ii) Identifiers
(iii) Literals
(iv) Operators
(v) Punctuators

KEYWORDS

- These are the words that convey a special meaning to the language compiler/interpreter. These are reserved for special purpose and must not be used as normal identifier names.

- e.g. if , elif , for , while , break , import etc.

IDENTIFIERS

- These are the fundamental building block of a program and are used as the general terminology for the names given to different parts of the program namely, variables etc.

Identifiers forming rules of java:

1. Identifiers can have alphabets, digits and underscore character.

2. They must not be a keyword.

3. They must not begin with a digit.

4. They can be of any length.

5. Python is case-sensitive i.e. upper-case and lower-case letters are treated differently.

Examples of some valid identifiers are:

myfile	price12
abc_12	_abc

LITERALS

These are referred to as constant-Values. These are data items that have a fixed value. Python allows several kinds of literals:

(i) String literals (ii) Numeric literals (iii) Boolean literals (iv) Special Literal None
(v) Literal Collections

String Literals

The text enclosed in quotes(single or double) forms a string literal in Python. E.g. ‘a’ , ‘abc’ , “abc” are all string literals in Python.

Non-graphic Characters

- Those characters that cannot be typed directly from keyboard e.g. backspace, tabs, etc.(No character is typed when these keys are pressed, only some action takes place)

- These non-graphic characters can be represented by using escape sequence. An escape sequence is represented by a backslash (\) followed by one or more characters.

Escape Sequence	What it does
\t	Horizontal Tab (TAB)
\v	Vertical Tab
\n	New line character
\"	Double quote
\'	Single quote

STRING TYPES IN PYTHON

1. **Single Line Strings –** created by enclosing text in single quotes(' ') or double quotes (" "). They must terminate in one line.

   Text1= 'hello world!'

2. **Multiline Strings** - for text spread across multiple lines. It can be created in two ways :

   (a) By adding a backslash at the end of normal single-quote/double quote strings.
   In normal strings, just add a backslash in the end before pressing Enter to continue typing text on the next line. For instance,

   ```
   Text1='hello\
   world'
   ```

   (b) By typing the text in triple quotation marks (No backslash needed at the end of line)
   Python allows to type multiline text string by enclosing them in triple quotation marks(both triple apostrophe or triple quotation marks will work)

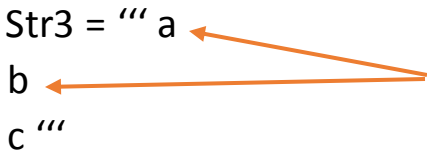
   ```
   Text1 = """ Hello
               World """
   ```

SIZE OF STRINGS

1. Python determines the size of a string as the count of characters in the string.

String value	Size
"abc"	3
'hello'	5
'\\'	1
'\n'	1
"\va"	2
"Seema\' s pen"	11

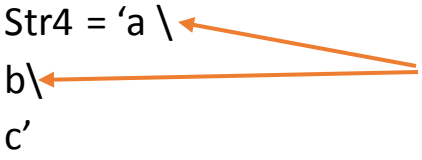
2. **For multiline strings created with triple quotes** – the EOL(end of line) character at the end of the line is also counted in the size. e.g.



These (enter keys) are considered as EOL (End of line) characters and counted in the length of multiline string.

size of the string Str3 is 5.

3. **For multiline strings created with single/double quotes and backslash character at the end of the line** - while calculating size, the backslashes are not counted in the size of the string.



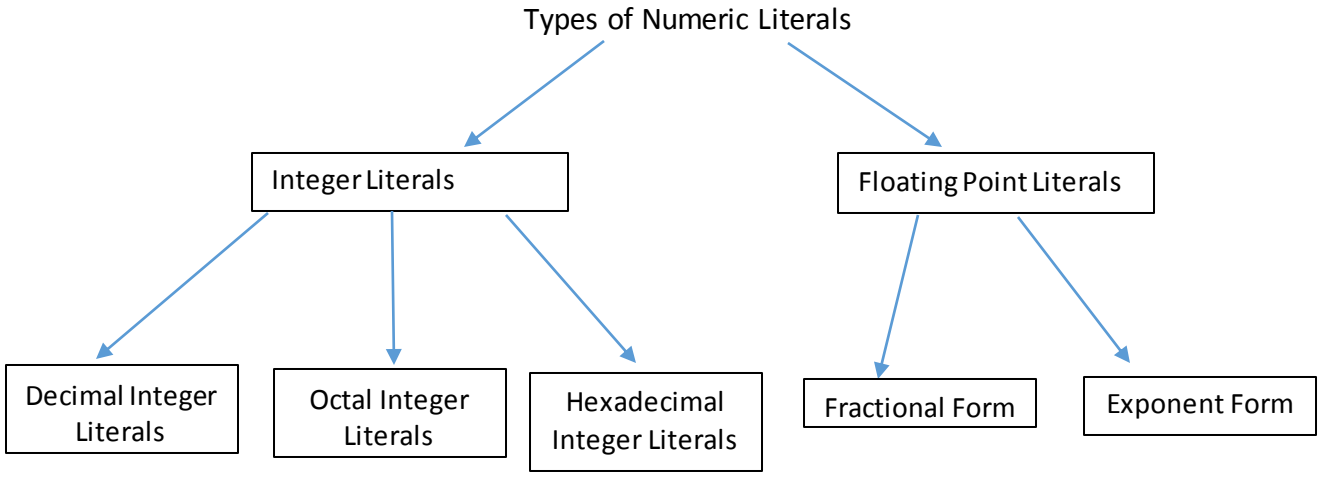
Backslashes are not counted in the size of Strings.

size of the string Str4 is 3.

***** to check the size of a string, you may also type* len(stringname) *command.***

NUMERIC LITERALS

- Fixed numeric values



INTEGER LITERALS

- It must have at least one digit and must not contain any decimal point. It may contain + or – sign. Commas cannot appear in an integer constant.

FLOATING-LITERALS (REAL LITERALS)

- Real literals are having fractional parts. **Rules of writing floating literals are :**

1. It must have at least one digit before a decimal point and at least one digit after the decimal point.

2. Commas cannot appear in a Floating literal.

3. It may contain either + or – sign.

- **Examples of valid floating literals are**: 2.0 , +17.5 , -13.0 , -0.000875 etc.

- Real literals may be written in two forms: **FRACTIONAL FORM , EXPONENT FORM**

REAL LITERAL IN EXPONENT FORM

A real literal in exponent form consists of two parts: **mantissa and exponent.**
e.g.

5.8 can be written in exponent form as: $0.58 \times 10^1 = 0.58E1$

Where mantissa part is 0.58(the part appearing before E) and exponent part is 1 (the part appearing after E).

BOOLEAN LITERALS

- A boolean literal in Python can either have value as True or as False.

SPECIAL LITERAL NONE

- The None literal is used to indicate absence of value.Python doesn't display anything when asked to display the value of a variable containing value as None.

e.g.

```
value1 = None
print(value1)   <--- This statement is not going to print anything.
```

OPERATORS

-Operators are tokens that trigger some computation when applied to variables and other objects in an expression.

TYPES OF OPERATORS

1. UNARY OPERATORS

-Those operators that require one operand to operate upon.

+ Unary plus (e.g. a=2 +a)

- Unary minus (e.g. a=-4 -a)

2. BINARY OPERATORS

- Those operators that require two operands to operate upon. Following are some binary operators:

ARITHMETIC OPERATORS

+ Addition

- Subtraction

* Multiplication

** exponent

/ Division

% Remainder/Modulus

// Floor Division

BITWISE OPERATORS

& Bitwise AND

| Bitwise OR

RELATIONAL OPERATORS

< Less than

> Greater than

<= Less than or equal to

>= Greater than or equal to

== Equal to

!= Not equal to

ASSIGNMENT OPERATORS

= Assignment

/= Assign quotient

+= Assign sum

*= Assign Product

%= Assign remainder

-= Assign difference

**= Assign Exponent

//= Assign Floor division

LOGICAL OPERATORS

and - Logical AND

or – Logical OR

PUNCTUATORS

- These are symbols that are used in programming languages to organize sentence structures.

- Most Common punctuators of Python programming language are :
  - ‘ (Single Quote)
  - “ (Double Quote)
  - # (Hash)
  - \ (forward slash)
  - ( ) (parenthesis)
  - [ ] (Square bracket)
  - { } (Curly bracket)
  - @ (at the rate)
  - , (comma)
  - : (colon)
  - . (dot)
  - = (Assignment)

EXPRESSIONS

- An expression is any legal combination of symbols that **represents a value**. An expression represents something, which **python evaluates** and which then produces a value.

- Some examples of expressions are:

15

2.9

} Expressions that are values only

a + 5

(3 + 5) / 4

} Complex expressions that produce a value when evaluated.

STATEMENT

- A statement is a programming instruction that does something i.e. some action takes place. E.g.

```
print("Hello")   # this statement calls print function
```

- While an **expression is evaluated, a statement is executed** i.e. some action takes place. And it is not necessary that a statement results in a value; it may or may not yield a value.
e.g.

```
a = 15
b= a – 10
print(a + 3)
```

COMMENTS

- Comments are the additional readable information, which is read by the programmers but ignored by Python interpreter.

- In python, comments begin with symbol # (hash) and end with the end of physical line.

Way of writing Comments

1. Add a # symbol in the beginning of every physical line part of the multi-line comments, e.g.

```
# Multiline comments are useful
# for additional information.
```

2. Type comment as a triple-quoted multi-line string e.g.

```
''' Multiline comments are useful
    for additional information. '''
```

} This type of multiline comment is known as **docstring.**

3. Those comment which start after a python instruction are called **Inline comments**. **e.g.**

```
print("Python")     # this statement is printing Python
```

VARIABLES

A variable in Python represents named location that refers to a value and whose values can be used and processed during program run.

e.g. the following statement creates a variable namely **marks** of **Numeric** type :

```
marks = 70
```

CREATING A VARIABLE

In python , to create a variable, just assign to its name the value of appropriate type. E.g.

```
Student = 'Jacob'     # variable Student is of string type
Age = 16              # variable Age is of numeric type
balance =23456.83     #variable balance is of numeric(floating) type
```

Lvalues and Rvalues

LValues : These are the objects to which you can assign a value or expression. LValues can come on lhs or rhs of an assignment statement.

RValues : These are the literals and expressions that are assigned to LValues.

e.g. a = 20
b = 10
a and b are LValues , whereas 20 and 10 are RValues.

MULTIPLE ASSIGMENT

1. **Assigning same value to multiple variables**
   You can assign same value to multiple variables in a single statement, e.g.
   ```
   a = b = c = 10
   ```
   It will assign value 10 to all three variables a, b, c.

2. **Assigning multiple values to multiple variables**
   You can even assign multiple values to multiple variables in single statement, e.g.
   ```
   x, y , z = 10 , 20 , 30
   ```
   It will assign the values **order wise**, i.e., first variable is given first value, second variable the second value and so on. That means, above statement will assign value 10 to x , 20 to y and 30 to z.

** ***While assigning values through multiple assignments***, python first evaluates the RHS(right hand side) expression(s) and then assigns them to LHS, e.g.

```
a , b , c  = 5, 10 , 7          #Statement 1
b , c , a = a+1 , b+2 , c-1     #Statement 2
print(a, b, c)
```

Statement 2 will first evaluate RHS i.e., a+1 , b+2 , c-1 which will yield 5 +1 , 10+2 , 7-1 = 6 , 12 , 6

The third statement print(a,b,c) will print 6 , 6 , 12

Question : Guess the output of following code fragment?

```
p , q = 3 , 5
q , r=p-2 , p + 2
print(p , q , r)
```

** expressions separated with commas are evaluated from left to right and assigned in same order e.g.

```
x = 10
y , y = x + 2, x + 5
```

firstly it will assign first RHS value to first LHS variable i.e., y =12

then it will assign second RHS value to second LHS variable i.e., y = 15

So if you print y after this statement y will contain 15.

Question : Guess the output of following code fragment?

```
x , x = 20 , 30
y , y = x + 10 , x + 20
print(x , y)
```

VARIABLE DEFINITION

- A variable is defined/created only when you assign some value to it. Using an undefined variable in an expression/statement causes an error called **Name Error**.

e.g.
```
print(x)
x = 20
print(x)
```

when you run the above code, it will produce an error for the first statement(line1) only -name 'x' not defined.

DYNAMIC TYPING

- A variable pointing to a value of a certain type, can be made to point to a value/object of different type. This is called Dynamic Typing.

e.g.

```
X =10
print(X)
X = "Hello World"
print(X)
```

Above code will yield the output as :

```
10
Hello World
```

*Note that variable X does not have a type but the value it points to does have a type. So you can make a variable point to a value of diiferent type by reassigning a value of that type; Python will not raise any error. This is called ***Dynamic Typing feature of Python***.

- Type of a variable can be found using the type( ) in following manner :

  type(<variable name>)

  e.g.

  >>> a = 10
  >>> type(a)
  <class 'int'>
  >>> b = 20.5
  >>> type(b)
  <class 'float'>

SIMPLE INPUT AND OUTPUT

- The built in function input( ) is used in the following manner to obtain the input from the user :

  variable_to_hold_the_value = input ("String to be displayed")

e.g.

name = input('What is your name? ')

```
>>> name=input('What is your name?')
What is your name? abc
>>> print(name)
 abc
```

The value that you type in front of the displayed prompt will be assigned to given variables, **name** in above case.

- ***The input( ) function always returns a value of String type***. So when number are also provided through input( ) function, it will also be of type String. e.g.

```
>>> age = input('Enter your age:')
Enter your age:22
>>> age+1
Traceback (most recent call last):
  File "<pyshell#6>", line 1, in <module>
    age+1
TypeError: can only concatenate str (not "int") to str
```

The above error was produced because we have tried to add 1 to string type of value i.e. '22' . Python cannot add on integer to a string. Since variable age received value 22 through input( ), it actually has '16' in it i.e., string value '16' ; thus you cannot add an integer to it.

READING NUMBERS

- Python offers two functions **int( ) and float( )** to be used with input( ) to convert the values received through input( ) into int and float types.

  e.g.

  <variable_name> = int(input("String to be displayed"))

  OR

  <variable_name> = float(input("String to be displayed"))

```
>>> age=int(input("Enter your age:"))
Enter your age:22
>>> age+1
23
```

```
>>> marks=float(input("Enter marks:"))
Enter marks:75.6
>>> marks+1
76.6
```

POSSIBLE ERRORS WHEN READING NUMERIC VALUES

(i) While inputting integer values using int( ) with input( ), make sure that the value being entered must be int type compatible.

```
>>> age=int(input("Enter yor age:"))
Enter yor age: 22.5
Traceback (most recent call last):
  File "<pyshell#18>", line 1, in <module>
    age=int(input("Enter yor age:"))
ValueError: invalid literal for int() with base 10: ' 22.5'
```

(ii) While inputting integer values using float( ) with input( ), make sure that the value being entered must be float type compatible.

```
>>> marks=float(input("Enter the Marks:"))
Enter the Marks: 56.7.2
Traceback (most recent call last):
  File "<pyshell#19>", line 1, in <module>
    marks=float(input("Enter the Marks:"))
ValueError: could not convert string to float: ' 56.7.2'
```

(iii) Values like 73 , 73. or .73 are all float convertible, hence python will be able to convert them to float and no error shall be reported if you enter such values.

```
>>> marks=float( input("Enter the Marks:"))
Enter the Marks:73
```

OUTPUT THROUGH PRINT STATEMENT

- The print( ) function is a way to send output to standard output device, which is normally a monitor. The syntax to use print( ) function is as follows :

  print(*objects, [sep=' ' , end = '\n'])

  ***objects** means it can be one or multiple comma separated objects to be printed.

- E.g. 1. print("Python is Wonderful.") will print Python is Wonderful.
- E.g. 2. print("Sum of 2 and 3 is", 2+3) will print Sum of 2 and 3 is 5
- E.g. 3. a=25
  print("Double of", a , "is", a*2) will print Double of 25 is 50.

FEATURES OF PRINT STATEMENT

1. It auto-converts the items to strings i.e., if you are printing a numeric value, it will automatically convert it into equivalent string and print it.
2. It inserts space between items automatically because the default value of **sep** argument is space (' '). The sep argument specifies the separator character. The print( ) automatically adds the sep character between the items/objects being printed in a line.
   e.g.

```
>>> print("My", "Name", "is", "Amit")
My Name is Amit
```

Four different string objects with no space in them are being printed.

But the output line has automatically spaces inserted in between them because default sep character is a space.

You can change the value of separator character with sep argument of print( ) as per this:

```
>>> print("My", "Name", "is", "Amit", sep='...')
My...Name...is...Amit
```

This time the print( ) separated the items with given sep character , which is '...'

3. It appends a newline character ('\n') at the end of the line unless you give your own **end** argument. **Consider the code given below :**

```
print("My name is Amit")
print("I am 16 years old")
```

It will produce output as :

```
My name is Amit
I am 16 years old
```

The **print()** works this way only when you have not specified any end argument with it because by default **print( )** takes value for **end** argument as **'\n' – the newline character**.

-If you explicitly give an end argument with a print( ) function then the print( ) will print the line and end it with the string specified with the end argument, e.g. the code

```
print("My name is Amit." , end='$')
print("I am 16 years old.")
```

will print output as :

This time the **print( )** ended the line with given **end** character, which is **'$'** here.

```
My name is Amit. $ I am 16 years old.
```

So the end argument determines the end character that will be printed at the end of print line.

Question: Guess the output of following code fragment?

```
a , b = 20 , 30
print("a= ", a , end= ' ')
print("b= " , b)
```

Question: Guess the output of following code fragment?

```
Name = 'Enthusiast'
print("Hello" , end= ' ')
print(Name)
print("How do you find python?")
```

****

Program 1 : Program to obtain three numbers and print their sum.

```
num1=int(input("Enter number 1:"))
num2=int(input("Enter number 2:"))
num3=int(input("Enter number 3:"))
sum = num1 + num2 + num3
print("Three numbers are :" , num1, num2, num3)
print("Sum is :" , sum)
```

Program 2 : Program to obtain length and breadth of a rectangle and calculate its area.

```
length= float(input("Enter length of the rectangle:"))
breadth = float(input("Enter breadth of the rectangle:"))

area = length * breadth
print("Rectangle specifications")
print("Length=" , length)
print("breadth=",breadth)
print("Area=",area)
```