Python Programming Language

- Python is a high-level, general-purpose and a very popular programming language.
- Python programming language (latest Python 3) is being used in web development, Machine Learning applications, along with all cutting edge technology in Software Industry.
- Python Programming Language is very well suited for Beginners, also for experienced programmers with other programming languages like C++ and Java.

- some facts about Python Programming Language:

1. Python is currently the most widely used multi-purpose, high-level programming language.
2. Python allows programming in Object-Oriented and Procedural paradigms.
3. Python programs generally are smaller than other programming languages like Java. Programmers have to type relatively less and indentation requirement of the language, makes them readable all the time.

4. Python language is being used by almost all tech-giant companies like – Google, Amazon, Facebook, Instagram, Dropbox, Uber… etc.
5. The biggest strength of Python is huge collection of standard library which can be used for the following:

- Machine Learning
- GUI Applications (like Kivy, Tkinter, PyQt etc.)
- Web frameworks like Django (used by YouTube, Instagram, Dropbox)
- Image processing (like OpenCV, Pillow)
- Web scraping (like Scrapy, BeautifulSoup, Selenium)
- Test frameworks
- Multimedia
- Scientific computing
- Text processing and many more..

- **Writing our first program:**
- Just type in the following code after you start the interpreter.

- # Script Begins
- print("GeeksQuiz")
- # Scripts Ends

Output:

- GeeksQuiz

- ***Line 1: [#* Script Begins]** In Python, comments begin with a #. This statement is ignored by the interpreter and serves as documentation for our code.
- ***Line 2: [print("GeeksQuiz")]*** To print something on the console, print() function is used. This function also adds a newline after our message is printed(unlike in C). Note that in Python 2, "print" is not a function but a keyword and therefore can be used without parentheses. However, in Python 3, it is a function and must be invoked with parentheses.
- ***Line 3: [#* Script Ends]** This is just another comment like in Line 1.

- **Python Language advantages and applications**
- **Advantages :**
  1) Presence of third-party modules
  2) Extensive support libraries(NumPy for numerical calculations, Pandas for data analytics etc)
  3) Open source and community development
  4) Easy to learn
  5) User-friendly data structures
  6) High-level language
  7) Dynamically typed language(No need to mention data type based on value assigned, it takes data type)
  8) Object-oriented language
  9) Portable and Interactive
  10) Portable across Operating systems

- **Applications :**
  1) GUI based desktop applications(Games, Scientific Applications)
  2) Web frameworks and applications
  3) Enterprise and Business applications
  4) Operating Systems
  5) Language Development
  6) Prototyping

- Assignment
- Short note on python advantages ad disadvantages

- **Organizations using Python :**
  1) Google(Components of Google spider and Search Engine)
  2) Yahoo(Maps)
  3) YouTube
  4) Mozilla
  5) Dropbox
  6) Microsoft
  7) Cisco
  8) Spotify
  9) Quora

- Variables and Data Structures
- In other programming languages like C, C++, and Java, you will need to declare the type of variables but in Python you don't need to do that.
- Just type in the variable and when values will be given to it, then it will automatically know whether the value given would be an int, float, or char or even a String.

```
# Python program to declare variables
myNumber = 3
print(myNumber)

myNumber2 = 4.5
print(myNumber2)

myNumber ="helloworld"
print(myNumber)
```

Output:

3 4.5 helloworld

Input and Output

- input() function is used to take input from the user.

```
# Python program to illustrate
# getting input from user
name = input("Enter your name: ")

# user entered the name 'harssh'
print("hello", name)
```

Output:

hello harssh

Python3 program to get input from user

```
# accepting integer from the user
# the return type of input() function is string ,
# so we need to convert the input to integer
num1 = int(input("Enter num1: "))
num2 = int(input("Enter num2: "))
 num3 = num1 * num2
print("Product is: ", num3)
```

Output:

Enter num1: 8 Enter num2: 6 ('Product is: ', 48)

Selection

Selection in Python is made using the two keywords ‘if’ and ‘elif’ and else (elseif)

```
# Python program to illustrate
# selection statement
  num1 = 34
if(num1>12):
    print("Num1 is good")
elif(num1>35):
    print("Num2 is not goooooo....")
else:
    print("Num2 is great")
```

Output:

Num1 is good

Functions

- You can think of functions like a bunch of code that is intended to do a particular task in the whole Python script. Python used the keyword ‘def’ to define a function.

Syntax:

```
def function-name(arguments): #function body
```

```
# Python program to illustrate
# functions
def hello():
    print("hello")
    print("hello again")
hello()

# calling function
hello()
```

Output:

hello hello again hello hello again

- any program starts from a ‘main’ function…lets create a main function like in many other programming languages.

```
# Python program to illustrate  function with main
def getInteger():
    result = int(input("Enter integer: "))
    return result
def Main():
    print("Started")
    # calling the getInteger
    output = getInteger()
    print(output)
# now we are required to tell Python  for 'Main' Fn existence
if __name__=="__main__":
    Main()
```

Output:

Started Enter integer: 5

Iteration (Looping)

As the name suggests it calls repeating things again and again. We will use the most popular ‘for’ loop here.

```
# Python program to illustrate
# a simple for loop

for step in range(5):
    print(step)
```

Output:

0 1 2 3 4

- Modules
- Python has a very rich module library that has several functions to do many tasks.
- ‘import’ keyword is used to import a particular module into your python code. For instance consider the following program.

```
# Python program to illustrate math module
import math
def Main():
    num = -85
    # fabs is used to get the absolute
    # value of a decimal
    num = math.fabs(num)
    print(num)
if __name__=="__main__":
    Main()
```

Output:

85.0

Statements

- Instructions written in the source code for execution are called statements. There are different types of statements in the Python programming language like Assignment statement, Conditional statement, Looping statements etc. These all help the user to get the required output. For example, n = 50 is an assignment statement.
- **Multi-Line Statements:** Statements in Python can be extended to one or more lines using parentheses (), braces {}, square brackets [], semi-colon (;), continuation character slash (\). When the programmer needs to do long calculations and cannot fit his statements into one line, one can make use of these characters.

Example :

- Declared using Continuation Character (\):

```
s = 1 + 2 + 3 + \ 4 + 5 + 6 + \ 7 + 8 + 9
```

- Declared using parentheses () :

```
n = (1 * 2 * 3 + 7 + 8 + 9)
```

Declared using square brackets [] :

```
footballer = ['MESSI',
              'NEYMAR',
              'SUAREZ']
```

Declared using braces {} :

```
x = {1 + 2 + 3 + 4 + 5 +
     6 + 7 + 8 + 9}
```

Declared using semicolons(;) :

flag = 2; ropes = 3; pole = 4

- **Indentation**
- A block is a combination of all these statements. Block can be regarded as the grouping of statements for a specific purpose. Most of the programming languages like C, C++, Java use braces { } to define a block of code.
- One of the distinctive features of Python is its use of indentation to highlight the blocks of code. Whitespace is used for indentation in Python. All statements with the same distance to the right belong to the same block of code. If a block has to be more deeply nested, it is simply indented further to the right.

```
# Python program showing indentation

site = 'gfg'

if site == 'gfg':
    print('Logging on to geeksforgeeks...')
else:
    print('retype the URL.')
print('All set !')
```

Output:

Logging on to geeksforgeeks... All set !

```
j = 1
while(j<= 5):
    print(j)
    j = j + 1
```

Output:

1 2 3 4 5

- To indicate a block of code in Python, you must indent each line of the block by the same whitespace.
- The two lines of code in the while loop are both indented.

- **Comments**
- Python developers often make use of the comment system as, without the use of it, things can get real confusing, real fast.
- Comments are the useful information that the developers provide to make the reader understand the source code. It explains the logic or a part of it used in the code.
- Comments are usually helpful to someone maintaining or enhancing your code when you are no longer around to answer questions about it.
- These are often cited as a useful programming convention that does not take part in the output of the program but improves the readability of the whole program. There are two types of comment in Python:

- **Single line comments :** Python single line comment starts with hashtag symbol with no white spaces (#) and lasts till the end of the line.
- If the comment exceeds one line then put a hashtag on the next line and continue the comment.
- Python’s single line comments are proved useful for supplying short explanations for variables, function declarations, and expressions.

Code 1:

```
# This is a comment
# Print “GeeksforGeeks !” to console
print("GeeksforGeeks")
```

Code 2:

```
a, b = 1, 3 # Declaring two integers
sum = a + b # adding two integers
print(sum) # displaying the output
```

Multi-line string as comment :

- Python multi-line comment is a piece of text enclosed in a delimiter (""") on each end of the comment.
- Again there should be no white space between delimiter (""").
- They are useful when the comment text does not fit into one line; therefore needs to span across lines.
- Multi-line comments or paragraphs serve as documentation for others reading your code.

Code 1:

```
"""
This would be a multiline comment in Python that
spans several lines and describes geeksforgeeks.
A Computer Science portal for geeks. It contains
well written, well thought
and well-explained computer science
and programming articles,
quizzes and more.
...
"""
print("GeeksForGeeks")
```

Code 2:

```
'''This article on geeksforgeeks gives you a
perfect example of
multi-line comments'''

print("GeeksForGeeks")
```

```
# Python 3 code to demonstrate variable
  assignment  upon condition using One liner if-
  else


# initialising variable using Conditional Operator
# a = 20 > 10 ? 1 : 0 is not possible in Python
# Instead there is one liner if-else
a = 1 if 20 > 10 else 0


# printing value of a
print ("The value of a is: " + str(a))
```

Output: The value of a is: 1

- **input ( ) :** This function first takes the input from the user and then evaluates the expression, which means Python automatically identifies whether user entered a string or a number or list. If the input provided is not correct then either syntax error or exception is raised by python. For example –

```
# Python program showing
# a use of input()
val = input("Enter your value: ")
print(val)
```

Taking input in Python

- Developers often have a need to interact with users, either to get data or to provide some sort of result.
- Most programs today use a dialog box as a way of asking the user to provide some type of input. While Python provides us with two inbuilt functions to read the input from the keyboard.

- **input (prompt)**
- **raw_input (prompt)**

- **How the input function works in Python :**
- When input() function executes program flow will be stopped until the user has given an input.
- The text or message display on the output screen to ask a user to enter input value is optional i.e. the prompt, will be printed on the screen is optional.
- Whatever you enter as input, input function convert it into a string.
- if you enter an integer value still input() function convert it into a string. You need to explicitly convert it into an integer in your code using typecasting.

```
# Program to check input
# type in Python
num = input ("Enter number :")
print(num)
name1 = input("Enter name : ")
print(name1)

# Printing type of input value
print ("type of number", type(num))
print ("type of name", type(name1))
```

Output:

Enter number: 12

12

Enter name : jack

jack

type of number <class ‘str’>

type of name <class ‘str’>

- **raw_input ( ) :** This function works in older version (like Python 2.x). This function takes exactly what is typed from the keyboard, convert it to string and then return it to the variable in which we want to store. For example –

```
# Python program showing
# a use of raw_input()

g = raw_input("Enter your name : ")
print g
```

Enter your name : jack

jack

```
# input
input1 = input()

# output
print(input1)
```

```
# input
num1 = int(input())
num2 = int(input())

# printing the sum in integer
print(num1 + num2)
```

```
# input
num1 = float(input())
num2 = float(input())

# printing the sum in float
print(num1 + num2)
```

```
# input
string = str(input())


# output
print(string)
```

Taking multiple inputs from user in Python

- In Python user can take multiple values or inputs in one line by two methods
  - Using split() method
  - Using List comprehension
- **Using split() method :**
- This function helps in getting a multiple inputs from user. It breaks the given input by the specified separator. If a separator is not provided then any white space is a separator. Generally, user use a split() method to split a Python string but one can use it in taking multiple input.

- **Syntax :**
- input().split(separator, maxsplit)

```
# Python program showing how to
# multiple input using split
# taking two inputs at a time
x, y = input("Enter a two value: ").split()
print("Number of boys: ", x)
print("Number of girls: ", y)
print()
```

```
# taking three inputs at a time
x, y, z = input("Enter a three value: ").split()
print("Total number of students: ", x)
print("Number of boys is : ", y)
print("Number of girls is : ", z)
print()
```

```
# taking two inputs at a time
a, b = input("Enter a two value: ").split()
print("First number is {} and second number is }".format(a, b))
print()
```

```
# taking multiple inputs at a time
# and type casting using list() function
x = list(map(int, input("Enter a multiple value: ").split()))
print("List of students: ", x)
```

```
Enter a two value: 5 10
Number of boys:  5
Number of girls:  10

Enter a three value: 30 10 20
Total number of students:  30
Number of boys is :  10
Number of girls is :  20

Enter a four value: 20 30
First number is 20 and second number is 30

Enter a multiple value: 20 30 10 22 23 26
List of students:  [20, 30, 10, 22, 23, 26]
```

- **Using List comprehension :**

- List comprehension is an elegant way to define and create list in Python.
- We can create lists just like mathematical statements in one line only.
- It is also used in getting multiple inputs from a user.

```
# Python program showing
# how to take multiple input
# using List comprehension

# taking two input at a time
x, y = [int(x) for x in input("Enter two value: ").split()]
print("First Number is: ", x)
print("Second Number is: ", y)
print()
```

```
# taking three input at a time
x, y, z = [int(x) for x in input("Enter three value: ").split()]
print("First Number is: ", x)
print("Second Number is: ", y)
print("Third Number is: ", z)
print()
```

```
# taking two inputs at a time
x, y = [int(x) for x in input("Enter two value: ").split()]
print("First number is {} and second number is
    {}".format(x, y))
print()


# taking multiple inputs at a time
x = [int(x) for x in input("Enter multiple value: ").split()]
print("Number of list is: ", x)
```

```
Enter two value: 2 5
First Number is:  2
Second Number is:  5

Enter three value: 2 4 5
First Number is:  2
Second Number is:  4
Third Number is:  5

Enter two value: 2 10
First number is 2 and second number is 10

Enter multiple value: 1 2 3 4 5
Number of list is:  [1, 2, 3, 4, 5]
>>>
```

- The above examples take input separated by spaces.
- In case we wish to take input separated by comma (,), we can use following:

```
# taking multiple inputs at a time separated by comma
x = [int(x) for x in input("Enter multiple value: ").split(",")]
print("Number of list is: ", x)
```

- **Python | Output using print() function**
- The simplest way to produce output is using the print() function where you can pass zero or more expressions separated by commas.
- This function converts the expressions you pass into a string before writing to the screen.

1. String Literals

- String literals in python's print statement are primarily used to format or design how a specific string appears when printed using the print() function.
- **\n :** This string literal is used to add a new blank line while printing a statement.
- **"" :** An empty quote ("") is used to print an empty line.

Code:

```
print ("GeeksforGeeks \n is best for DSA Content.")
```

Output:

GeeksforGeeks is best for DSA Content.

2. end= ” ” statement

- The end keyword is used to specify the content that is to be printed at the end of the execution of the print() function.
- By default, it is set to “\n”, which leads to the change of line after the execution of print() statement.

```
# This line will automatically add a new line before the
# next print statement
print ("GeeksForGeeks is the best platform for DSA
    content")

# This print() function ends with "**" as set in the end
    argumennt.
print ("GeeksForGeeks is the best platform for DSA
    content", end= "**")
```

Output:

```
GeeksForGeeks is the best platform for DSA content
GeeksForGeeks is the best platform for DSA content**
```

- **Separator**
- The print() function can accept any number of positional arguments. These arguments can be separated from each other using a **“,” separator**.
- These are primarily used for formatting multiple statements in a single print() function.

Example:

```
b = "for"
 print("Geeks", b , "Geeks")
```

Output:

Geeks for Geeks

- **Python Operators**
- Operators in general are used to perform operations on values and variables in Python.
- These are standard symbols used for the purpose of logical and arithmetic operations.

- **Arithmetic operators:** Arithmetic operators are used to perform mathematical operations like addition, subtraction, multiplication and division.

+ Addition: adds two operandsx + y

- Subtraction:subtracts two operands x- y

* Multiplication:multiplies two operands x * y

/ Division (float): divides the first operand by the second x / y

// Division (floor): divides the first operand by the second x // y

% Modulus: returns the remainder when first operand is divided by the second x % y

** Power : Returns first raised to power second

x ** y

```
# Examples of Arithmetic Operator
a = 9
b = 4
# Addition of numbers
add = a + b
# Subtraction of numbers
sub = a - b
# Multiplication of number
mul = a * b
# Division(float) of number
div1 = a / b
# Division(floor) of number
div2 = a // b
# Modulo of both number
mod = a % b
# Power
p = a ** b
```

```
# print results
print(add)
print(sub)
print(mul)
print(div1)
print(div2)
print(mod)
print(p)
```

Output:

13 5 36 2.25 2 1 6561

Relational Operators:

- Relational operators compares the values.
- It either returns **True** or **False** according to the condition.

- Greater than: True if left operand is greater than the right x > y
- <Less than: True if left operand is less than the right x < y
- ==Equal to: True if both operands are equal x == y
- !=Not equal to - True if operands are not equal

  x != y
- >=Greater than or equal to: True if left operand is greater than or equal to the right x >= y
- <=Less than or equal to: True if left operand is less than or equal to the right x <= y

```
# Examples of Relational Operators
a = 13
b = 33
# a > b is False
print(a > b)
# a < b is True
print(a < b)
# a == b is False
print(a == b)
# a != b is True
print(a != b)
# a >= b is False
print(a >= b)
# a <= b is True
print(a <= b)
```

Output:

False

True

False

True

False

True

3. Logical operators: Logical operators perform **Logical AND**, **Logical OR** and **Logical NOT** operations.

and Logical AND: True if both the operands are true x and y

or Logical OR: True if either of the operands is true x or y

not Logical NOT: True if operand is false not x

```
# Examples of Logical Operator
a = True
b = False

# Print a and b is False
print(a and b)

# Print a or b is True
print(a or b)

# Print not a is False
print(not a)
```

- Output:
- False True False

4. Bitwise operators: Bitwise operators acts on bits and performs bit by bit operation.

& Bitwise AND x & y

| Bitwise OR x | y

~ Bitwise NOT ~x

^ Bitwise XOR x ^ y

>> Bitwise right shift x>>

<< Bitwise left shift x<<

```
# Examples of Bitwise operators
a = 10
b = 4
# Print bitwise AND operation
print(a & b)
# Print bitwise OR operation
print(a | b)
# Print bitwise NOT operation
print(~a)
# print bitwise XOR operation
print(a ^ b)
# print bitwise right shift operation
print(a >> 2)
# print bitwise left shift operation
print(a << 2)
```

- Output:

0

14

-11

14

2

40

5. **Assignment operators:** Assignment operators are used to assign values to the variables.

= Assign value of right side of expression to left side operand
x = y + z

+=Add AND: Add right side operand with left side operand and then assign to left operand a+=b a=a+b

-=Subtract AND: Subtract right operand from left operand and then assign to left operand a-=b a=a-b

=Multiply AND: Multiply right operand with left operand and then assign to left operand a=b a=a*b

/=Divide AND: Divide left operand with right operand and then assign to left operand a/=b a=a/b

%=Modulus AND: Takes modulus using left and right operands and assign result to left operand a%=b a=a%b

//=Divide(floor) AND: Divide left operand with right operand and then assign the value(floor) to left operand
a//=b a=a//b

=Exponent AND: Calculate exponent(raise power) value using operands and assign value to left operand a=b a=a**b

&=Performs Bitwise AND on operands and assign value to left operand a&=b a=a&b

|=Performs Bitwise OR on operands and assign value to left operand a|=b a=a|b

^=Performs Bitwise xOR on operands and assign value to left operand a^=b a=a^b

>>=Performs Bitwise right shift on operands and assign value to left operand a>>=b a=a>>b

<<=Performs Bitwise left shift on operands and assign value to left operand a <<= b a= a << b

6. Special operators:

- There are some special type of operators like-
  Identity operators-
  is and **is not** are the identity operators both are used to check if two values are located on the same part of the memory.
- Two variables that are equal does not imply that they are identical.

is True if the operands are identical

is not True if the operands are not identical

```
# Examples of Identity operators
a1 = 3
b1 = 3
a2 = 'GeeksforGeeks'
b2 = 'GeeksforGeeks'
a3 = [1,2,3]
b3 = [1,2,3]
print(a1 is not b1)
print(a2 is b2)
# Output is False, since lists are mutable.
print(a3 is b3)
```

Output:

False True False

- **Membership operators**-
  in and **not in** are the membership operators; used to test whether a value or variable is in a sequence.

in True if value is found in the sequence

not in True if value is not found in the sequence

```
# Examples of Membership operator
x = 'Geeks for Geeks'
y = {3:'a',4:'b'}
print('G' in x)
print('geeks' not in x)
print('Geeks' not in x)
print(3 in y)
print('b' in y)
```

Output:

True True False True False