Python Introduction

- Python is a general-purpose high-level programming language.
- Python was developed by Guido Van Rossum in 1989 while working at National Research Institute at Netherlands. But officially Python was made available to public in 1991. The official Date of Birth for Python is: Feb 20th 1991.
- Python is recommended as first programming language for beginners.

Python Hello Word program:

```
(base) C:\Users\gauta>python
Python 3.9.7 (default, Sep 16 2021, 16:59:28) [MSC v.1916 64 bit (AMD64)] :: Anaconda, Inc. on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> print('Hello World!')
Hello World!
>>>
```

The name Python was selected from the TV Show "**The Complete Monty Python's Circus**", which was broadcasted in BBC from 1969 to 1974.

Guido developed the Python language, taking almost all programming functionality from different languages like

- Functional Programming concepts from C
- Object Oriented Programming concepts from C++
- Scripting Features from Perl and Shell Script
- Modular Programming concepts from Modula-3

Usage of Python:

- Using in developing Desktop Applications
- Using in developing web Applications
- Using in developing database Applications
- Using in Network Programming
- Using in developing games
- Using in Data Analysis Applications
- Using in Machine Learning
- Using in developing Artificial Intelligence Applications
- Using in IOT(Internet of Thing)

Tip:
Google and YouTube use Python coding, NASA and Newark Stock Exchange Applications developed in Python, Microsoft, IBM, Yahoo using Python.

Python Features:

- **Simple and easy to learn:** Python is a simple programming language. When we read a Python program, it can appear that we are reading statements in English. The syntaxes are very simple. Compared to other languages, we can write programs with much less lines and greater legibility and simplicity, using python we can reduce the development and cost of the project.
- **Freeware and Open Source:** Everyone can use Python software without any licence and it is freeware, its source code is open we can customize based on our requirement.
- **Platform Independent:** Once a program is written in Python, it can be run on any platform without having to rewrite it. Internally, PVM is responsible for converting it into a machine understandable form.
- **Portability:** Python programs are portable. i.e we can migrate from one platform to another platform very easily.
- **Dynamically Typed:** In Python we are not forced to declare the type of the variables. Whenever we are assigning the value, based on the value, the type will be assigned automatically. Therefore, Python is considered as a dynamically typed language.
- **Both Procedure Oriented and Object Oriented:** Python language supports both Procedure oriented (like C, pascal etc) and object oriented (like C++, Java) features
- **Interpreted:** We are not required to explicitly compile Python programs. Internal Python interpreter will take care of the compilation. If the compilation fails, the interpreter generates syntax errors. Once the build is successful, then PVM (Python Virtual Machine) is responsible for execution.
- **Extensible:** We can use other language programs in Python. The main advantages of this approach are:
    - Use already existing legacy non-Python code
    - Improve performance of the application
- **Embedded:** We can use Python programs in any other language programs
- **Extensive Library:** Python has a rich inbuilt library, like NumPy, Pandas, Sklearn, Tkinter.

Tips:
Performance wise not up to the mark because its interpreted language. Not fit for mobile Applications.

Python Identifiers & Variable

Identifiers: A name in Python program is called identifier, It can be class name or function name or module name or variable name.
Variable: A variable is a name given to a memory location, that is used to hold a value.

Both an identifier and a variable are the names allotted by users to a particular entity in a program. The identifier is only used to identify an entity uniquely in a program at the time of execution whereas, a variable is a name given to a memory location, that is used to hold a value

Rules to define identifiers in Python:

- Alphabet Symbols (Either Upper case OR Lower case)
- If Identifier is start with Underscore (_) then it indicates it is private.
- Identifier should not start with Digits.
- Identifiers are case sensitive.
- cannot use reserved words as identifiers
- no length limit for Python identifiers. But not recommended to use too lengthy identifiers.
- Doller ($) Symbol is not allowed in Python.

Tips:
If identifier starts with _ symbol then it indicates that it is private, If identifier starts with __(two underscore symbols) indicating that strongly private identifier. If the identifier starts and ends with two underscore symbols then the identifier is language defined special name, which is also known as magic methods.

Reserved Words:

Python language has some words are reserved to represent some meaning or functionality. Such type of words is called Reserved words.
There are 36 reserved words available in Python.

“'False', 'None', 'True', '__peg_parser__', 'and', 'as', 'assert', 'async', 'await', 'break', 'class', 'continue', 'def', 'del', 'elif', 'else', 'except', 'finally', 'for', 'from', 'global', 'if', 'import', 'in', 'is', 'lambda', 'nonlocal', 'not', 'or', 'pass', 'raise', 'return', 'try', 'while', 'with', 'yield'”

```
(base) C:\Users\gauta>python
Python 3.9.7 (default, Sep 16 2021, 16:59:28) [MSC v.1916 64 bit (AMD64)] :: Anaconda, Inc. on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import keyword
>>> keyword.kwlist
['False', 'None', 'True', '__peg_parser__', 'and', 'as', 'assert', 'async', 'await', 'break', 'class', 'continue', 'def'
, 'del', 'elif', 'else', 'except', 'finally', 'for', 'from', 'global', 'if', 'import', 'in', 'is', 'lambda', 'nonlocal',
 'not', 'or', 'pass', 'raise', 'return', 'try', 'while', 'with', 'yield']
>>>
```

Data Types in python:

Data Type represent the type of data present inside a variable. it’s not required to specify the type explicitly in python language. Based on value provided, the type will be assigned automatically because Python is Dynamically Typed Language.

Python contains the following inbuilt data types:

- Int
- Float
- Complex
- Bool

- Str
- Bytes
- Bytearray
- Range
- List
- Tuple
- Set
- frozen set
- Dict

Tip: Python contains several inbuilt functions

type(): To check the type of variable
id(): To get address of object
print(): to print the value

Tip: In Python everything is object

Number System in Python:

- Decimal Number System
- Binary Number System
- Octal Number System
- Hexa decimal Number System

Decimal form(base-10):
Default number system in Python
The allowed digits are: 0 to 9
Eg: a =10

Binary form (Base-2):
The allowed digits are: 0 & 1
Value should be prefixed with 0b or 0B
Eg: a = 0B1111
a =0B123
a=b111

Octal Form (Base-8):
The allowed digits are: 0 to 7
Value should be prefixed with 0o or 0O.
Eg: a=0o123
a=0o786

Hexa Decimal Form (Base-16):
The allowed digits are: 0 to 9, a-f (both lower and upper cases are allowed)
Value should be prefixed with 0x or 0X
Eg:
a =0XFFFF

a=0XBBBB
a =0XBbbb

Tip: Being a programmer we can specify literal values in decimal, binary, octal and hexa decimal forms. But PVM (python Virtual Machine) will always provide values only in decimal form.
Eg:

```
decimal=10
octal=0o10
hexa=0X10
binary=0B10
print(decimal)
print(octal)
print(hexa)
print(binary)
```

```
10
8
16
2
```

Base Conversions :

Python provide the following in-built functions for base conversions
bin():
you can use bin() to convert from any base to binary
Eg:

```
print(bin(15))
print(bin(0o11))
print(bin(0X10))
```

```
0b1111
0b1001
0b10000
```

oct():
you can use oct() to convert from any base to octal
Eg:

```
print(oct(10))
print(oct(0B1111))
print(oct(0X123))
```

```
0o12
0o17
0o443
```

hex():
you can use hex() to convert from any base to hexa decimal
Eg:

```
print(hex(100))
print(hex(0B111111))
print(hex(0o12345))
```

```
0x64
0x3f
0x14e5
```

Float data type:

you can use float data type to represent floating point values (decimal values), and can also represent floating point values by using exponential form (scientific notation)
Eg:

```
f=1.234
type(f)
```

```
float
```

Tip: you can represent int values in decimal, binary, octal and hexa decimal forms. But we can represent float values only by using decimal form.

Complex Data Type:

A complex number is in the form of a+bj , a and b contain integers or floating point values
Eg:
3+5j
10+5.5j
0.5+0.1j
In the real part if we use int value then we can specify that either by decimal, octal, binary or hexa decimal form, but imaginary part should be specified only by using decimal form.

```
a=0B11+5j
print(a)
```

```
(3+5j)
```

```
b=3+0B11j
print(b)
```

```
  Input In [13]
    b=3+0B11j
           ^
SyntaxError: invalid syntax
```

Bool data type:

you can use this data type to represent boolean values, the only allowed values for this data type are:
True and False
Internally Python represents True as 1 and False as 0
b=True
type(b) =>bool

Eg:

```
]: a=10
   b=20
   c=a<b
   print(c)
   print(True+True)
   print(True-False)
```

```
True
2
1
```

Str type:

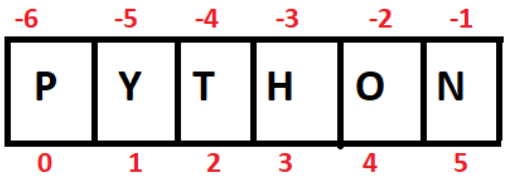
str represents String data type.
A String is a sequence of characters enclosed within single quotes or double quotes.
s1='Python_Class'
s1="Python_Class"
By using single quotes or double quotes we cannot represent multi line string literals.
s1="Python_Class"

For this requirement we should go for triple single quotes (''') or triple double quotes (""")
s1='''Python_Class'''
s1="""Python_Class"""

you can embed one string in another string
'''This “Python class " organised by verzeo'''

Slicing of Strings:

Slice means a piece [ ] operator is called slice operator, which can be used to retrieve parts of String.
Python Strings follows zero based index.
The index can be either +ve or -ve.
+ve index means forward direction from Left to Right
-ve index means backward direction from Right to Left



```
]: s="PythonClass"
   print(s[0])
   print(s[1])
   print(s[-1])
   print(s[1:40])
   print(s[1:])
   print(s[:4])
   print(s[:])
   print(s*3)
   print(len(s))
```

```
P
y
s
ythonClass
ythonClass
Pyth
PythonClass
PythonClassPythonClassPythonClass
11
```

Bytes Data Type:

Bytes data type represents a group of byte numbers just like an array.

```
>>> x=[10,20,30,40]
>>> b=bytes(x)
>>> type(b)
<class 'bytes'>
>>> print(b[0])
10
>>> for i in b:print(i)
...
10
20
30
40
>>>
```

Tip1 : The only allowed values for byte data type are 0 to 256. By mistake if we are trying to provide any other values then we will get value error.

Tip2 : Once we creates bytes data type value, we cannot change its values, otherwise we will get TypeError.

```
>>> x=[10,20,30,40]
>>> b=bytes(x)
>>> type(b)
<class 'bytes'>
>>> print(b[0])
10
>>> for i in b:print(i)
...
10
20
30
40
>>> b[0]=100
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'bytes' object does not support item assignment
>>>
```

Bytearray Data type:

Bytearray is exactly same as bytes data type except that its elements can be modified.

```
>>> x=[10,20,30,40]
>>> b=bytearray(x)
>>> for i in b:print(i)
...
10
20
30
40
>>> b[0]=100
>>> for i in b:print(i)
...
100
20
30
40
>>>
```

```
>>> x=[10,256]
>>> b=bytearray(x)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: byte must be in range(0, 256)
>>>
```

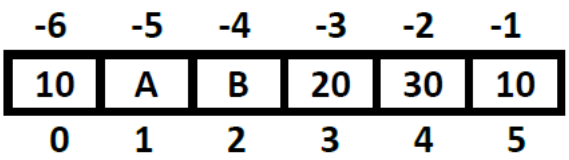
List :

If you want to represent a group of values as a single entity where insertion order required to preserve and duplicates are allowed then we should go for list.

- Insertion order is preserved
- Heterogeneous objects are allowed
- Duplicates are allowed
- Growable in nature
- values should be enclosed within square brackets.

List is dynamic because based on our requirement you can increase the size and decrease the size, in List the elements will be placed within square brackets and with comma separator, you can differentiate duplicate elements by using index and we can preserve insertion order by using index. Hence index will play very important role. Python supports both positive and negative indexes. +ve index means from left to right, whereas negative index means right to left.

List: [10,"A","B",20, 30, 10]



List objects are mutable .i.e. we can change the content.

Accessing elements of List:

you can access elements of the list either by using index or by using slice operator(:)

1. By using index:

List follows zero based index. i.e. index of first element is zero. list supports both +ve and -ve indexes +ve index meant for Left to Right -ve index meant for Right to Left.

2. By using slice operator:

Syntax: list2= list1[start: stop: step]

start ==>it indicates the index where slice has to start default value is 0

stop ===>It indicates the index where slice has to end default value is max allowed index of list i.e. length of the list

step ==>increment value default value is 1

```
>>> list=[10,11.5,'Gautam',True,11]
>>> print(list)
[10, 11.5, 'Gautam', True, 11]
>>> list[0]
10
>>> list[-1]
11
>>> list[1:3]
[11.5, 'Gautam']
>>> list[0]=1000
>>> for i in list:print(i)
...
1000
11.5
Gautam
True
11
>>>
```

Tip: list is growable in nature. i.e. based on our requirement we can increase or decrease the size and list is an ordered, mutable, heterogenous collection of elements is nothing but list, where duplicates also allowed.

```
>>> list.append("Gautam_Kumar")
>>> list
[1000, 11.5, 'Gautam', True, 11, 'Gautam_Kumar']
>>> list.remove(11)
>>> list
[1000, 11.5, 'Gautam', True, 'Gautam_Kumar']
>>> list1=list*2
>>> list1
[1000, 11.5, 'Gautam', True, 'Gautam_Kumar', 1000, 11.5, 'Gautam', True, 'Gautam_Kumar']
>>>
```

Tuple:

Tuple data type is exactly same as list data type except that it is immutable, you can't

change values. Tuple elements can be represented within parenthesis.

```
>>> tup=(10,20,30)
>>> type(tup)
<class 'tuple'>
>>> tup[0]=100
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
>>> tup.append("Gautam")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'tuple' object has no attribute 'append'
>>> tup.remove(10)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'tuple' object has no attribute 'remove'
>>>
```

Tip: Tuple is the read only version of list.

Range:

Range Data Type represents a sequence of numbers. The elements present in range Data type are not modifiable. i.e., range Data type is immutable.

Generate numbers from 0 to 9:

```
>>> range(10)
range(0, 10)
>>> r=range(10)
>>> for i in r:print(i)
...
0
1
2
3
4
5
6
7
8
9
>>>
```

Generate numbers from 20 to 30

```
>>> r=range(20,30)
>>> for i in r:print(i)
...
20
21
22
23
24
25
26
27
28
29
>>>
```

Generate numbers with 2 increment value

```
>>> r=range(30,40,2)
>>> for i in r:print(i)
...
30
32
34
36
38
>>>
```

Set :

If we want to represent a group of values without duplicates where order is not important then we should go for set Data Type.

- Insertion order is not preserved
- Duplicates are not allowed
- Heterogeneous objects are allowed
- Index concept is not applicable
- It's mutable collection
- Growable in nature

```
: set1={100,0,10,200,10.6,'Gautam'}
print(set1)
set1.add(60)
print(set1)
set1.remove(100)
print(set1)
```

```
{0, 10.6, 100, 200, 10, 'Gautam'}
{0, 10.6, 100, 200, 10, 'Gautam', 60}
{0, 10.6, 200, 10, 'Gautam', 60}
```

Frozen set:

It's exactly same as set except that it is immutable. Hence, we cannot use add or remove functions.

```
: s={10,20,30,40}
  fs=frozenset(s)
  print(type(fs))
  print(fs.add(70))
```

```
<class 'frozenset'>

---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Input In [2], in <cell line: 4>()
      2 fs=frozenset(s)
      3 print(type(fs))
----> 4 print(fs.add(70))

AttributeError: 'frozenset' object has no attribute 'add'
```

```
: print(fs.remove(10))
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Input In [4], in <cell line: 1>()
----> 1 print(fs.remove(10))

AttributeError: 'frozenset' object has no attribute 'remove'
```

Dict:

If you want to represent a group of values as key-value pairs then we should go for dict data type, Duplicate keys are not allowed but values can be duplicated. If we are trying to insert an entry with duplicate key then old value will be replaced with new value. dict is mutable and the order won't be preserved.

```
]: dict1={101:'durga',102:'ravi',103:'shiva'}
   print(dict1)
   dict1[101]='sunny'
   print(dict1)
   dict1['a']='apple'
   dict1['b']='banana'
   print(dict1)
```

```
{101: 'durga', 102: 'ravi', 103: 'shiva'}
{101: 'sunny', 102: 'ravi', 103: 'shiva'}
{101: 'sunny', 102: 'ravi', 103: 'shiva', 'a': 'apple', 'b': 'banana'}
```

Operators

Operator is a symbol that performs certain operations, Python provides the following set of operators

- Arithmetic Operators
- Relational Operators or Comparison Operators
- Logical operators
- Bitwise operators
- Assignment operators
- Special operators

Arithmetic Operators:

- + ==>Addition
- - ==>Subtraction
- * ==>Multiplication
- / ==>Division operator
- % ===>Modulo operator
- // ==>Floor Division operator
- ** ==>Exponent operator or power operator

```
]: a=20
   b=2
   print('a+b=',a+b)
   print('a-b=',a-b)
   print('a*b=',a*b)
   print('a/b=',a/b)
   print('a//b=',a//b)
   print('a%b=',a%b)
   print('a**b=',a**b)
```

```
a+b= 22
a-b= 18
a*b= 40
a/b= 10.0
a//b= 10
a%b= 0
a**b= 400
```

Tip: / operator always performs floating point arithmetic. Hence it will always return float value. But Floor division (//) can perform both floating point and integral arithmetic. If arguments are int type then result is int type. If at least one argument is float type, then result is float type.

Tip: you can use +, * operators for str type also, If you want to use + operator for str type then compulsory both arguments should be str type only otherwise we will get error.

Relational Operators:

>,>=,<,<=

```
: a=10
  b=20
  print("a > b is ",a>b)
  print("a >= b is ",a>=b)
  print("a < b is ",a<b)
  print("a <= b is ",a<=b)
```

```
a > b is  False
a >= b is  False
a < b is  True
a <= b is  True
```

Equality operators:

== , !=

```
print(10==20)
print(10!= 20)
print(10==True)
print(False==False)
print("Gautam"=="Gautam")
print(10=="Gautam")
```

```
False
True
False
True
True
False
```

Logical Operators:

and, or, not
you can apply for all types.

```
print("Gautam" and "Gautam Kumar")
print("" and "Gautam")
print("Gautam" and "")
print("" or "Gautam")
print("Gautam" or "")
print(not "")
print(not "Gautam")
```

```
Gautam Kumar


Gautam
Gautam
True
False
```

Bitwise Operators:

you can apply these operators bitwise. These operators are applicable only for int and Boolean types.
& ==> If both bits are 1 then only result is 1 otherwise result is 0
| ==> If atleast one bit is 1 then result is 1 otherwise result is 0
^ ==>If bits are different then only result is 1 otherwise result is 0
~ ==>bitwise complement operator
1==>0 & 0==>1
<< ==>Bitwise Left shift
>> ==>Bitwise Right Shift

Tip: The most significant bit acts as sign bit. 0 value represents +ve number where as 1 represents -ve value, positive numbers will be represented directly in the memory whereas -ve numbers will be represented indirectly in 2's complement form.

Assignment Operators:

The following is the list of all possible compound assignment operators in Python
+=
-=
*=
/=
%=
//=
**=
&=
|=
^=

```
4]: x=10
    x+=20
    print(x)
    x=10
    x&=5
    print(x)
```

```
30
0
```

Special operators:

Python defines the following 2 special operators

- Identity Operators
- Membership Operators

Identity Operators

you can use identity operators for address comparison, 2 identity operators are available **is and is not**

```
]: a=10
   b=10
   print(a is b)
   x=True
   y=True
   print( x is y)
```

```
True
True
```

```
]: list1=["one","two","three"]
   list2=["one","two","three"]
   print(id(list1))
   print(id(list2))
   print(list1 is list2)
   print(list1 is not list2)
   print(list1 == list2)
```

```
2585602723520
2585602723648
False
True
True
```

Membership operators:

you can use Membership operators to check whether the given object present in the given collection. (It may be String, List, Set, Tuple or Dict)

in ==>Returns True if the given object presents in the specified Collection.
not in ==>Returns True if the given object not present in the specified Collection.

```
.8]: x="hello learning Python is very easy!!!"
     print('hello' in x)
     print('Python' in x)
     print(' programming' not in x)
     print('Language' in x)
```

```
True
True
True
False
```

Input And Output Statements

Reading dynamic input from the keyboard:
In Python 3 the following functions are available to read dynamic input from the keyboard.
input()

```
[21]: x=input("Enter Value : ")
      type(x)
```

```
Enter Value : 20.6
[21]: str
```

```
[22]: x=input("Enter Value : ")
      type(x)
```

```
Enter Value : Gautam
[22]: str
```

```
[23]: x=input("Enter Value : ")
      type(x)
```

```
Enter Value : 11
[23]: str
```

input () function can be used to read data directly in our required format. You are not required to perform type casting.

Tip: Python 3 input () function considered every input value is treated as str type only.
Output Statements:
you can use **print ()** function to display output.
print () without any argument, just it prints new line character

```
print("String")
print("Hello World")
print("Hello \n World")
print("Hello\tWorld")
print(10*"Hello")
print("Hello"*10)
print("Hello"+"World")
```

```
String
Hello World
Hello
 World
Hello   World
HelloHelloHelloHelloHelloHelloHelloHelloHelloHello
HelloHelloHelloHelloHelloHelloHelloHelloHelloHello
HelloWorld
```

```
#print() with variable number of arguments:
a,b,c=10,20,30
print("The Values are :",a,b,c)
```

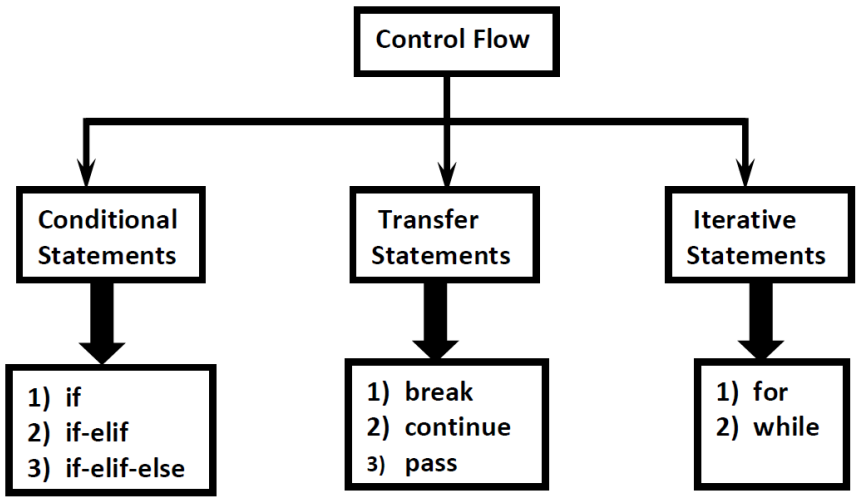
```
The Values are : 10 20 30
```

```
#print(String,variable list):
#You can use print() statement with String and any number of arguments.
str1="Gautam"
a=28
str2="Sql"
str3="Python"
print("Hello",str1,"Your Age is",a)
print("You are teaching",str2,"and",str3)
```

```
Hello Gautam Your Age is 28
You are teaching Sql and Python
```

Flow Control

Flow control describes the order in which statements will be executed at runtime.



I. Conditional Statements

1) if

if condition :
    statement
or
if condition :
    statement-1
    statement-2
    statement-3

If condition is true then statements will be executed.

```
: name=input("Enter Name:")
  if name=="Gautam" :
      print("Hello Gautam Good Morning")
      print("How are you!!!")
```

```
Enter Name:Gautam
Hello Gautam Good Morning
How are you!!!
```

2) if-else:

if condition :
    Action-1
else :
    Action-2

if condition is true then Action-1 will be executed otherwise Action-2 will be executed.

```
]: name=input("Enter Name:")
   if name=="Gautam" :
       print("Hello Gautam Good Morning")
   else:
       print("Hello Guest Good Moring")
       print("How are you!!!")
```

```
Enter Name:Gautam
Hello Gautam Good Morning
```

```
]: name=input("Enter Name:")
   if name=="Gautam" :
       print("Hello Gautam Good Morning")
   else:
       print("Hello Guest Good Moring")
       print("How are you!!!")
```

```
Enter Name:kumar
Hello Guest Good Moring
How are you!!!
```

3) if-elif-else:

Syntax:

```
if condition1:
        Action-1
elif condition2:
        Action-2
elif condition3:
        Action-3
elif condition4:
        Action-4
...
else:
        Default Action
```

Based condition the corresponding action will be executed.

```
]: brand=input("Enter Your Favourite Brand:")
if brand=="Reebok":
    print("Only 10% discount")
elif brand=="Addidas":
    print("Only 25% discount")
elif brand=="Nike":
    print("Buy one get Free One")
else :
    print("please look for other brands")
```

```
Enter Your Favourite Brand:Reebok
Only 10% discount
```

```
: brand=input("Enter Your Favourite Brand:")
if brand=="Reebok":
    print("Only 10% discount")
elif brand=="Addidas":
    print("Only 25% discount")
elif brand=="Nike":
    print("Buy one get Free One")
else :
    print("please look for other brands")
```

```
Enter Your Favourite Brand:Addidas
Only 25% discount
```

```
brand=input("Enter Your Favourite Brand:")
if brand=="Reebok":
    print("Only 10% discount")
elif brand=="Addidas":
    print("Only 25% discount")
elif brand=="Nike":
    print("Buy one get Free One")
else :
    print("please look for other brands")
```

```
Enter Your Favourite Brand:ducati
please look for other brands
```

Tip: There is no switch statement in Python.

II. Iterative Statements

If you want to execute a group of statements multiple times then we should go for Iterative statements.

Python supports 2 types of iterative statements.

1. for loop
2. while loop

1) for loop:

If you want to execute some action for every element present in some sequence(it may be string or collection)then we should go for for loop.

Syntax:

```
for x in sequence :
        body
```

where sequence can be string or any collection. Body will be executed for every element present in the sequence.

```
sub="Python"
for x in sub :
    print(x)
```

```
P
y
t
h
o
n
```

```
for x in range(11) :
    print(x)
```

```
0
1
2
3
4
5
6
7
8
9
10
```

```
for x in range(21) :
    if (x%2!=0):
        print(x)
```

```
1
3
5
7
9
11
13
15
17
19
```

Nested Loops:

Sometimes we can take a loop inside another loop, which are also known as nested loops.

```
for i in range(2):
    for j in range(2):
        print("i=",i," j=",j)
```

```
i= 0  j= 0
i= 0  j= 1
i= 1  j= 0
i= 1  j= 1
```

2) while loop:

If you want to execute a group of statements iteratively until some condition becomes false,then we should go for while loop.

Syntax:

while condition:

    body

```
]: #To display the sum of first n numbers
n=int(input("Enter number:"))
sum=0
i=1
while i<=n:
    sum=sum+i
    i=i+1
    print("The sum of first",n,"numbers is :",sum)
```

```
Enter number:6
The sum of first 6 numbers is : 1
The sum of first 6 numbers is : 3
The sum of first 6 numbers is : 6
The sum of first 6 numbers is : 10
The sum of first 6 numbers is : 15
The sum of first 6 numbers is : 21
```

III. Transfer Statements

1) break:

you can use break statement inside loops to break loop execution based on some condition.

```
: for i in range(10):
    if i==7:
        print("plz break")
        break
    print(i)
```

```
0
1
2
3
4
5
6
plz break
```

2) continue:

you can use continue statement to skip current iteration and continue next iteration.

```
]: for i in range(10):
    if i%2==0:
        continue
    print(i)
```

```
1
3
5
7
9
```

```
3]: numbers=[10,20,0,5,0,30]
for n in numbers:
    if n==0:
        print("Hey how we can divide with zero..just skipping")
        continue
    print("100/{} = {}".format(n,100/n))
```

```
100/10 = 10.0
100/20 = 5.0
Hey how we can divide with zero..just skipping
100/5 = 20.0
Hey how we can divide with zero..just skipping
100/30 = 3.3333333333333335
```

3) pass statement:

pass is a keyword in Python. In our programming syntactically if block is required which won't do anything then you can define that empty block with pass keyword.

pass

- It is an empty statement
- It is null statement
- It won't do anything

```
: for i in range(20):
      if i%9==0:
          print(i)
      else:
          pass
```

```
0
9
18
```

Tip: Sometimes in the parent class we have to declare a function with empty body and child class responsible to provide proper implementation. Such type of empty body we can define by using pass keyword.

del:

del is a keyword in Python, after using a variable, it is highly recommended to delete that variable if it is no longer required, so that the corresponding object is eligible for Garbage Collection.

you can delete variable by using del keyword.

```
]: x=10
   print(x)
   del x
   print(x)
```

```
10

---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Input In [51], in <cell line: 4>()
      2 print(x)
      3 del x
----> 4 print(x)

NameError: name 'x' is not defined
```