Python Part – 1

Introduction

- Python is a general purpose high level programming language.
- Python was developed by **Guido Van Rossam 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.

Eg1: To print Helloworld:

In C:

```c
#include<stdio.h>
void main()
{
print("Hello world");
}
```

In Python:

```python
print("Hello World")
```

Eg2: To print the sum of 2 numbers

In C:

```c
#include <stdio.h>

void main()
{
int a,b;
a =10;
b=20;
printf("The Sum:%d",(a+b));
}
```

In Python:

```python
a=10
b=20
print("The Sum:",(a+b))
```

The name Python was selected from the TV Show **"The Complete Monty Python's Circus"**, which was broad casted in BBC from 1969 to 1974. Guido developed Python language by taking almost all programming features from0different languages

1. Functional Programming Features from C
2. Object Oriented Programming Features from C++
3. Scripting Language Features from Perl and Shell Script
4. Modular Programming Features from Modula-3

Most of syntax in Python Derived from C and ABC languages.
Where we can use Python: - We can use everywhere. The most common important application areas are:

1. For developing Desktop Applications
2. For developing web Applications
3. For developing database Applications
4. For Network Programming
5. For developing games
6. For Data Analysis Applications
7. For Machine Learning
8. For developing Artificial Intelligence Applications
9. For IOT etc...

Note: Internally Google and YouTube use Python coding NASA and Network Stock Exchange Applications developed by Python. Top Software companies like Google, Microsoft, IBM, Yahoo using Python.

Features of Python:

1. Simple and easy to learn:

Python is a simple programming language. When we read Python program, we can feel like reading english statements. The syntaxes are very simple and only 30+ keywords are available.

When compared with other languages, we can write programs with very less number of lines.
Hence more simplicity. We can reduce development and cost of the project.

2. Freeware and Open Source:

We can use Python software without any licence and it is freeware. Its source code is open,
So that we can we can customize based on our requirement.

Eg: Jython is customized version of Python to work with Java Applications.

3. High Level Programming language:

Python is high level programming language and hence it is programmer friendly language.
Being a programmer we are not required to concentrate low level activities like memory management and security etc...

4. Platform Independent:

Once we write a Python program, it can run on any platform without rewriting once again.
Internally PVM is responsible to convert into machine understandable form.

5. Portability:

Python programs are portable. i.e. we can migrate from one platform to another platform very easily.
Python programs will provide same results on any paltform.

6. Interpreted:

We are not required to compile Python programs explicitly. Internally Python interpreter will take care that compilation. If compilation fails interpreter raised syntax errors. Once compilation success then PVM (Python Virtual Machine) is responsible to execute.

7. Dynamically Typed:

In Python we are not required to declare type for variables. Whenever we are assigning the value, based on value, type will be allocated automatically. Hence Python is considered as dynamically typed language. But Java etc. are Statically Typed Languages b'z we have to provide type at the beginning only.

This dynamic typing nature will provide more flexibility to the programmer.

8. Both Procedure Oriented and Object Oriented:

Python language supports both procedure oriented (like C, Pascal etc.) and object oriented (like C++, Java) features.

Hence we can get benefits of both like security and reusability etc.

9. Interpreted:

We are not required to compile Python programs explicitly. Internally Python interpreter will take care that compilation. If compilation fails interpreter raised syntax errors. Once compilation success then PVM (Python Virtual Machine) is responsible to execute.

10. Extensible:

We can use other language programs in Python. The main advantages of this approach are:

1. We can use already existing legacy non-Python code.
2. We can improve performance of the application.

11. Embedded:

We can use Python programs in any other language programs.
i.e. we can embed Python programs anywhere.

12. Extensive Library:

Python has a rich inbuilt library.
Being a programmer we can use this library directly and we are not responsible to0implement the functionality.

Limitations of Python:

1. Performance wise not up to the mark b'z it is interpreted language.
2. Not using for mobile Applications

```
Python Versions:

Python 1.0V introduced in Jan 1994
Python 2.0V introduced in October 2000
Python 3.0V introduced in December 2008
Note: Python 3 won't provide backward compatibility to Python2
i.e. there is no guarantee that Python2 programs will run in Python3.

Current versions
Python 3.6.1 Python 2.7.13
```

Identifiers

A name in Python program is called identifier.0It can be class name or function name or module name or variable name.

a = 10

Rules to define identifiers in Python:

1. The only allowed characters in Python are-

- alphabet symbols (either lower case or upper case)
- digits (0 to 9)
- underscore symbol (_)

By mistake if we are using any other symbol like $ then we will get syntax error.

- cash = 10 √
- ca$h = 20 *

2. Identifier should not starts with digit

- 123total *
- total123 √

3. Identifiers are case sensitive. Of course Python language is case sensitive language.

- Total = 10
- TOTAL = 999
- print(total) **#10**
- print(TOTAL) **#999**

Rules:-

1. Alphabet Symbols (Either Upper case OR Lower case)
2. Identifier should not start with Digits.
3. Identifiers are case sensitive.
4. We cannot use reserved words as identifiers Eg: def=10 *
5. There is no length limit for Python identifiers. But not recommended to use too lengthy identifiers.
6. Dollor ($) Symbol is not allowed in Python.

Q. Which of the following are valid Python identifiers?

- 123total *
- total123 √
- java2share √
- ca$h *
- *abc_abc* √
- def *
- if *

Note:

1. If identifier starts with _ symbol then it indicates that it is private
2. If identifier starts with __(two underscore symbols) indicating that strongly private identifier.
3. If the identifier starts and ends with two underscore symbols then the identifier is0language defined special name, which is also known as magic methods.

Eg: __add__

Reserved Words:

In Python some words are reserved to represent some meaning or functionality. Such type of words are called reserved words.

Note:

1. All Reserved words in Python contain only alphabet symbols.
2. Except the following 3 reserved words, all contain only lower case alphabet symbols.

- True
- False
- None

Eg:

- a= true *
- a=True √

There are 33 reserved words available in Python.

In [1]:

```
help('keywords')
```

```
Here is a list of the Python keywords.  Enter any keyword to get more help

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

Summary of Datatypes in Python3

Datatype	Description	Is Immutable	Example

Int	We can use to represent the whole/integral numbers	Immutable	a=10
			type(a)
			<class 'int'>

Float	We can use to represent the decimal/floating point numbers	Immutable	b=10.5
			type(b)
			<class 'float'>

Complex	We can use to represent the complex numbers	Immutable	c=10+5j type(c) <class 'complex'> c.real 10.0 c.imag 5.0

Bool	We can use to represent the logical values(Only allowed values are True and False)	Immutable	flag=True
			flag=False
			type(flag)
			<class 'bool'>

Str	To represent sequence of Characters	Immutable	s='nikhil'
			type(s)
			<class 'str'>
			s="nikhil"
			s='''prakriti
			yadav'''
			type(s)
			<class 'str'>

bytes	To represent a sequence of byte values from 0-255	Immutable	list=[1,2,3,4]
			b=bytes(list)
			type(b)
			<class 'bytes'>

bytearray	To represent a sequence of byte values from 0-255	Mutable	list=[10,20,30]
			ba=bytearray(list)
			type(ba)
			<class 'bytearray'>

range	To represent a range of values	Immutable	r=range(10)
			r1=range(0,10)
			r2=range(0,10,2)

list	To represent an ordered collection of objects	Mutable	l=[10,11,12,13,14,15]
			type(l)
			<class 'list'>

tuple	To represent an ordered collections of objects	Immutable	t=(1,2,3,4,5)
			type(t)
			<class 'tuple'>

set	To represent an unordered collection of unique objects	Mutable	s={1,2,3,4,5,6}
			type(s)
			<class 'set'>

Forezenset	To represent an unordered collection of unique objects	Immutable	s={11,2,3,'nikhil',100,'prakriti'}
			fs=frozenset(s)
			type(fs)
			<class 'frozenset'>

Dict	To represent a group of key value pairs	Mutable	d={101:'prakriti',102:'nikhil',103:'happy'}
			type(d)
			<class 'dict'>

Data Types:

Data Type represent the type of data present inside a variable. In Python we are not required to specify the type explicitly. Based on value provided, the type will be assigned automatically.
Hence Python is Dynamically Typed Language.

Python contains the following inbuilt data types.

1. int
2. float
3. complex
4. bool
5. str
6. bytes
7. bytearray
8. range
9. list
10. tuple
11. set
12. frozenset
13. dict
14. None

Note: Python contains several inbuilt functions

1. type() - to check the type of variable
2. id() - to get address of object
3. print() - to print the value

In Python everything is object.

1) Int Data Type:

We can use int data type to represent whole numbers (integral values)

Eg:

- a=10
  type(a) **#int**

Note: In Python2 we have long data type to represent very large integral values. But in Python3 there is no long type explicitly and we can represent long values also by using int type only.

We can represent int values in the following ways

1. Decimal form
2. Binary form
3. Octal form
4. Hexa decimal form

Note: Being a programmer we can specify literal values in decimal, binary, octal and hexa decimal forms. But PVM will always provide values only in decimal form.

```
1) a=10
2) b=0o10
3) c=0X10
4) d=0B10
5) print(a) 10
6) print(c) 16
8) print(d) 2
```

2) Float Data Type:

We can use float data type to represent floating point values (decimal values)

Eg:
```
f=1.234
type(f) float
```

We can also represent floating point values by using exponential form (scientific notation)

Eg:
```
f=1.2e3
print(f) 1200.0
```

instead of 'e' we can use 'E'

The main advantage of exponential form is we can represent big values in less memory.

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

In [2]:

```
# Example
f = 11.5
type(f)
```

Out[2]:

```
float
```

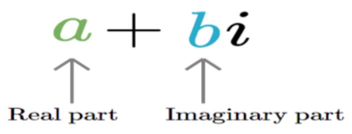
In [3]:

```
f = 1.2E5
print(f)
```

```
120000.0
```

3) Complex Data Type:

A complex number is of the form a and b contain integers or floating point values



Complex numbers are the numbers that are expressed in the form of a+ib where, a,b are real numbers and 'i' is an imaginary number called “iota”. The value of i = (√-1).

For example, 2+3i is a complex number, where 2 is a real number (Re) and 3i is an imaginary number (Im).

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.

Note: Complex data type has some inbuilt attributes to retrieve the real part and0imaginary part

We can use complex type generally in scientific Applications and electrical engineering0Applications.

In [4]:

```
# Example
c = 5+3j
type(c)
```

Out[4]:

```
complex
```

4) Bool Data Type:

We 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

Eg:

```
b=True
type(b) =>bool
```

Eg:

```
a=10
b=20
c=a<b
```

```
print(c)==>True
True+True==>2
True-False==>1
```

In [5]:

```
a=10
b=20
c=a<b
print(c)
```

```
True0
```

5) Str Type:

Str represents String data type.

A String is a sequence of characters enclosed within single quotes or double quotes.

```
s1='nikhil'

s1="nikhil"
```

By using single quotes or double quotes we cannot represent multi line string literals.

```
s1="nikhil yadav"
```

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

```
      s1='''nikhil
          yadav'''

s1="""nikhil

    yadav"""
```

We can also use triple quotes to use single quote or double quote in our String.

```
''' This is " character'''
' This i " Character '
```

We can embed one string in another string

```
'''This "Python class very helpful" for C students'''
```

In [6]:

```
# example
name = 'nikhil'
type(name)
```

Out[6]:

```
str
```

Slicing of Strings:

Slice means a piece[ ] operator is called slice operator, which can be used to retrieve parts of String.

In 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
```

```
A string in python can be sliced for getting a part of the string.

- Consider the following string:-

    ------>012345
   name = 'nikhil'
        -6-5-4-3-2-1<------
==> length = 6

The index is a string starts from 0 to (length -1) in python.
We use the following syntax:

  name = prakriti

  sl ==> name[index_start : ind_end : step
  value] sl[0:3] returns "pra"
  sl[1:3] returns "ra"

- NEGATIVE INDEXING
  Negative indices can also be used as shown in the figure above -1 corresponds to the
  (length -1) index, -2 to (length -2).

- Slicing with skip value
  we can provide a skip value as a part of our slice like this:-

  name = "prakriti"
  name[1:8:2] ---> 'rkii'

Note:

1. In Python the following data types are considered as Fundamental Data types

- int
- float
- complex
- bool
- str

2. In Python, we can represent char values also by using str type and explicitly char
type is not available.
```

In [7]:

```
# Slicing
name = "nikhil"
name[1:8:2]
```

Out[7]:

```
'ihl'
```

In [8]:

```
# Indexing
name = 'prakriti'
name[3]
```

Out[8]:

```
'k'
```

Type Casting

We can convert one type value to another type. This conversion is called Typecasting or Type coersion.

The following are various inbuilt functions for type casting.

1. int() - We can convert from any type to int except complex type.
2. float() - We can convert any type value to float type except complex type.
3. complex()
4. bool()
5. str()

Fundamental Data Types v/s Immutability:

All Fundamental Data types are immutable. i.e. once we creates an object, we cannot0perform any changes in that object. If we are trying to change then with those changes a new object will be created.
This non-chargeable behaviour is called immutability.

6) Bytes Data Type:

Bytes data types represent a group of byte numbers just like an array.

In [9]:

```
x = [10,20,30,40]
b = bytes(x)
type(b)
print(b[0])
```

Out[9]:

```
bytes
10
```

In [11]:

```
 for i in b :
     print(i)
```

```
10
20
30
40
```

Conclusion 1:

The only allowed values for byte data type are 0 to 256. By mistake if we are trying to0provide any other values then we will get value error.

Conclusion 2:

Once we create bytes data type value, we cannot change its values; otherwise we will get TypeError.

```
# 'bytes' object does not support item assignment

x=[10,20,30,40]
b=bytes(x)
b[0]=100
```

7) Bytearray Data Type:

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

In [12]:

```
# example
x=[10,20,30,40]
b = bytearray(x)
for i in b :
    print(i)
```

```
10
20
30
40
```

In [13]:

```
b[0]=100
for i in b:
    print(i)
```

```
100
20
30
40
```

8) Range Data Type:

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.

```
Form-1: range(10)
generate numbers from 0 to 9
Eg:
r=range(10)
for i in r : print(i) 0 to 9

Form-2: range(10,20)
generate numbers from 10 to 19
Eg:
r = range(10,20)
for i in r : print(i) 10 to 19

Form-3: range(10,20,2)
2 means increment value
Eg:
r = range(10,20,2)
for i in r : print(i) 10,12,14,16,18

We can access elements present in the range Data Type by using index.
r=range(10,20)
r[0]==>10
r[15]==>IndexError: range object index out of range

We cannot modify the values of range data type
Eg:
r[0]=100
TypeError: 'range' object does not support item assignment

We can create a list of values with range data type
Eg:
l = list(range(10))
l
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

In [14]:

```
# example
range(5)
for i in range(5):
    print(i)
```

```
0
1
2
3
4
```

9) List data Type:

If we 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 data type.

1. insertion order is preserved
2. heterogeneous objects are allowed
3. duplicates are allowed
4. growable in nature
5. values should be enclosed within square brackets.

In [15]:

```
# example
list=[10,10.5,'nikhil',True,10]
print(list)
```

```
[10, 10.5, 'nikhil', True, 10]
```

In [16]:

```
list=[10,20,30,40]
list[2]
```

Out[16]:

```
30
```

List is growable in nature. i.e. based on our requirement we can increase or decrease the0size.

Note: An ordered, mutable, heterogeneous collection of elements is nothing but list, where duplicates also allowed.

In [17]:

```
list=[10,20,30]
list.append("nikhil")
list
```

Out[17]:

```
[10, 20, 30, 'nikhil']
```

In [18]:

```
[10, 30, 'nikhil']
list2=list*2
list2
```

Out[18]:

```
[10, 20, 30, 'nikhil', 10, 20, 30, 'nikhil']
```

10) Tuple Data Type:

Tuple data type is exactly same as list data type except that it is immutable. i.e we cannot change values.

Tuple elements can be represented within parenthesis.

In [20]:

```
# example
t = (10,20,30,40)
type(t)
```

Out[20]:

```
tuple
```

```
1) t[0]=100
2) TypeError: 'tuple' object does not support item assignment


3) >>> t.append("nikhil")
4) AttributeError: 'tuple' object has no attribute 'append'


5) >>> t.remove(10)
6) AttributeError: 'tuple' object has no attribute 'remove'

Note: tuple is the read only version of list
```

11) Set Data Type:

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

1. insertion order is not preserved
2. duplicates are not allowed
3. heterogeneous objects are allowed
4. index concept is not applicable
5. It is mutable collection
6. Growable in nature

```
Eg:
1) s={100,0,10,200,10,'nikhil'}
2) s # {0, 100, 'nikhil', 200, 10}
3) s[0] ==>TypeError: 'set' object does not support indexing
4)
5) set is growable in nature, based on our requirement we can increase or decrease the
size.
6)
7) >>> s.add(60)
8) >>> s
9) {0, 100, 'nikhil', 200, 10, 60}
10) >>> s.remove(100)
11) >>> s
12) {0, 'nikhil', 200, 10, 60}
```

In [21]:

```
# example
s={100,0,100,200,10,'nikhil',10,15}
print(s)
```

```
{0, 100, 200, 10, 15, 'nikhil'}
```

12) Frozenset Data Type:

It is exactly same as set except that it is immutable.0Hence we cannot use add or remove functions.

```
s={10,20,30,40}
fs=frozenset(s)
type(fs)
<class 'frozenset'>
fs
frozenset({40, 10, 20, 30})
for i in fs:
print(i)
 40
10
20
30

fs.add(70)
AttributeError: 'frozenset' object has no attribute 'add'
fs.remove(10)
AttributeError: 'frozenset' object has no attribute 'remove'
```

In [22]:

```
# erxample
s={10,20,30,40}
fs=frozenset(s)
type(s)
```

Out[22]:

```
set
```

13) Dict Data Type:

If we want to represent a group of values as key-value pairs then we should go for dict0data type.

Eg:
```
    d={101:'nikhil',102:'ravi',103:'shiva'}
```

Duplicate keys are not allowed but values can be duplicated. If we are trying to insert an0entry with duplicate key then old value will be replaced with new value.

Note: dict is mutable and the order wont be preserved.

Note:

1. In general we can use bytes and bytearray data types to represent binary information like images, video files etc.
2. In Python2 long data type is available. But in Python3 it is not available and we can represent long values also by using int type only.
3. In Python there is no char data type. Hence we can represent char values also by using str type.

In [23]:

```
# example
d={101:'nikhil',102:'ravi',103:'shiva'}
d[101]='sunny'
d
```

Out[23]:

```
{101: 'sunny', 102: 'ravi', 103: 'shiva'}
```

In [24]:

```
# We can create empty dictionary as follows
d={ }

#We can add key-value pairs as follows

d['a']='apple'
d['b']='banana'
print(d)
```

```
{'a': 'apple', 'b': 'banana'}
```

14) None Data Type:

None means nothing or No value associated.
If the value is not available, then to handle such type of cases None introduced.

It is something like null value in Java.

Eg:
```
    def m1():
     a=100
    print(m1())
     None
```

Escape Characters:

In String literals we can use escape characters to associate a special meaning.

The following are various important escape characters in Python

```
1) \n ==> New Line
2) \t ===> Horizontal tab
3) \r ==> Carriage Return
4) \b ===> Back space
5) \f ===> Form Feed
6) \v ==> Vertical tab
7) \' ===> Single quote
8) \" ===> Double quote
9) \\ ===> back slash symbol
```

In [25]:

```
# example
s = 'nikhil\nyadav'
print(s)
```

```
nikhil
yadav
```

In [26]:

```
s = 'prakriti\tyadav'
print(s)
```

```
prakriti        yadav
```

Constants:

Constants concept is not applicable in Python.
But it is convention to use only uppercase characters if we don't want to change value.

MAX_VALUE=10

It is just convention but we can change the value.

Operators:

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

1. Arithmetic operators
2. Relational operators or Comparison operators
3. Logical operators
4. Bitwise operators
5. Assignment operators
6. Special operators

```
Python divides the operators in the following groups:
-Arithmetic operators ==>  +,-,*,/ etc..
-Assignment operators ==>  ==,-=,+= etc.
-Comparison operators ==>  ==,<,>=,>,!= etc.
-Logical operators ==>  and,or,not
```

1. Arithmetic Operators:

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

In [27]:

```
# example
a=10
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 = 12
a-b = 8
a*b = 20
a/b = 5.0
a//b = 5
a%b = 0
a**b = 100
```

```
Eg:

10/2 ==> 5.0
10//2 ==> 5
10.0/2 ===> 5.0
10.0//2 ===> 5.0
```

<u>Note:</u> 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.

```
Note:

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

1) >>> "nikhil"+10
2) TypeError: must be str, not int
3) >>> "nikhil"+"10"
4) 'nikhil10'

If we use * operator for str type then compulsory one argument should be int and other
argument should be str type.

2*"nikhil"
"nikhil"*2

2.5*"nikhil" ==> TypeError: can't multiply sequence by non-int of type 'float'
"nikhil"*"nikhil" ==> TypeError: can't multiply sequence by non-int of type 'str'

+ ====> String concatenation operator
* ===> String multiplication operator

Note: For any number x, x/0 and x%0 always raises "ZeroDivisionError"

10/0
10.0/0
```

2. Relational Operators:

In [28]:

```
# >,>=,<,<=

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:

In [29]:

```
# ==,!=
# We can apply these operators for any type even for incompatible types also.

10==20
```

Out[29]:

```
False
```

In [30]:

```
10!= 20
```

Out[30]:

```
True
```

In [31]:

```
False==False
```

Out[31]:

```
True
```

In [32]:

```
10=="nikhil"
```

Out[32]:

```
False
```

3. Logical Operators:

and, or ,not

We can apply for all types.

For boolean types behaviour:

```
and ==> If both arguments are True then only result is True0
or ====> If atleast one arugemnt is True then result is
True0 not ==> complement

True and False ==> False
True or False ===> True
not False ==> True
```

For non-boolean type's behaviour:

```
0 means False
non-zero means True
empty string is always treated as False
```

4. Bitwise Operators:

We can apply these operators bitwise.

These operators are applicable only for int and boolean types.

By mistake if we are trying to apply for any other type then we will get Error.

&,|,^,~,<<,>>

```
print(4&5) ==>valid
print(10.5 & 5.6) ==>
 TypeError: unsupported operand type(s) for &: 'float' and 'float'
print(True & True) ==>valid
```

In [33]:

```
print(4&5)
```

```
40
```

```
& ==> 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

print(4&5) ==>4
print(4|5) ==>5
print(4^5) ==>1
```

5. Assignment Operators:

We can use assignment operator to assign value to the variable.

Eg:

- x=100We can combine assignment operator with some other operator to form compound assignment operator.

Eg: x+=10 ====> x = x+10

The following is the list of all possible compound assignment operators in Python.

```
    +=      |     **=
    -=      |     &=
    *=      |     |=
    /=      |     |=
    %=      |     ^=
    //=     |     >>=
```

In [34]:

```
# example
x=10
x+=20
print(x)

x&=5
print(x)
```

```
30
4
```

6. Special operators:

Python defines the following 2 special operators

1. Membership operators
2. Identity operators

a. Membership operators:

We 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 present in the specified Collection
not in --> Returns True if the given object not present in the specified
collection.
```

```
# example
x="hello learning Python is very easy!!!"
print('h' in x)
print('d' in x)
print('d' not in x)
print('Python' in x)
```

In [37]:

```
list1=["sunny","bunny","chinny","pinny"]
print("sunny" in list1)
print("tunny" in list1)
print("tunny" not in list1)
```

```
True
False
True
```

b. Identity Operators

We can use identity operators for address comparison.

2 identity operators are available

1. is
2. is not

- r1 is r2 returns True if both r1 and r2 are pointing to the same object
- r1 is not r2 returns True if both r1 and r2 are not pointing to the same object

In [35]:

```
# example
a=10 b=10
print(a is b)
x=True y=True
print(x is y)
```

```
True
True
```

In [36]:

```
# Note: We can use is operator for address comparison whereas == operator for content comp
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)
```

```
2049112833536
2049112833344
False
True
True
```

Ternary Operator:

Syntax:

x = firstValue if condition else secondValue

If condition is True then firstValue will be considered else secondValue will be considered.

In [38]:

```
# Eg 1:
a,b=10,20
x=30 if a<b else 40
print(x)
```

```
30
```

In [39]:

```
# Eg 2: Read two numbers from the keyboard and print minimum value
a=int(input("Enter First Number:"))
b=int(input("Enter Second Number:"))
if a<b:
    print("Minimum Value:",a)
else:
    print("Minimum Value:",b)
```

```
Enter First Number:120
Enter Second Number:90
Minimum Value: 90
```

Operator Precedence:

If multiple operators present then which operator will be evaluated first is decided by operator precedence. Eg:-

- print(3+10*2) --> 23
- print((3+10)*2) --> 26

The following list describes operator precedence in Python.

```
() --> Parenthesis
** --> exponential operator
~,- --> Bitwise complement operator,unary minus operator
*,/,%,// --> multiplication,division,modulo,floor division
+,- --> addition,subtraction
<<,>> --> Left and Right Shift
& --> bitwise And
& ^ --> Bitwise X-OR
& --> Bitwise OR0
>,>=,<,<=, ==, != --> Relational or Comparison operators
=,+=,-=,*= --> Assignment operators is ,
is not --> Identity Operators
in , not in --> Membership operators
not --> Logical not
and --> Logical and
or --> Logical or
```

In [40]:

```
#example
a=30
b=20
c=10
d=5
print((a+b)*c/d)
print((a+b)*(c/d))
print(a+(b*c)/d)
```

```
100.0
100.0
70.0
```

Python Roadmap

Step: 1 Learn the Basics - Syntax, Variables, Data Types

Step: 2 Conditionals, Loops, Functions, Built-in-Functions

Step: 3 Data Structures – Strings, Lists, Tuples, Sets, Dictionaries

Step: 4 OOP- Classes, Inheritance, Objects

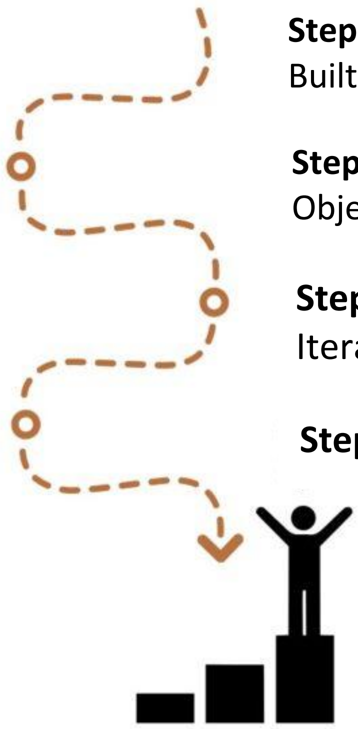
Step: 5 Advance Topics 1- RegEx, Decorators, Lambada

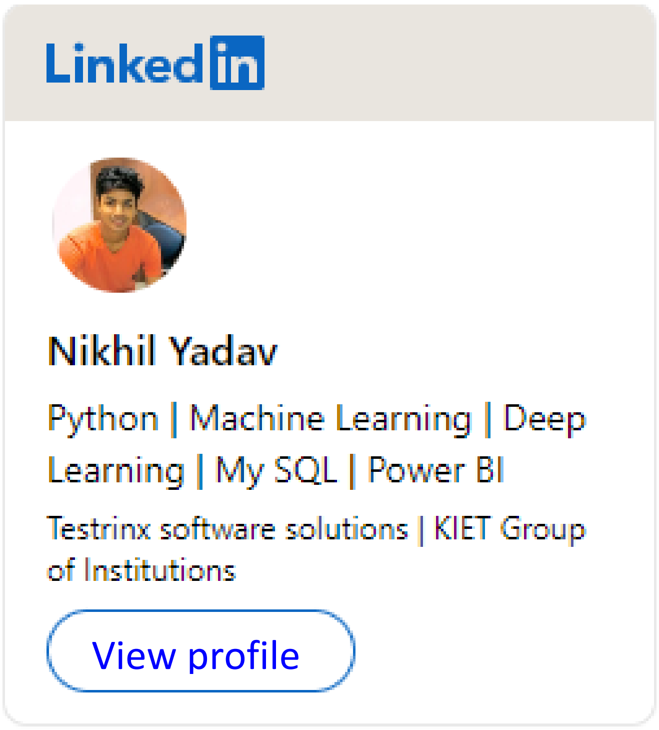
Step: 6 Advance Topics 2 - Modules, Iterators

Step: 7 Learn Python Libraries

Step: 8 Learn Version Control Systems

Step: 9 Build Python Apps




Linkedin

Nikhil Yadav

Python | Machine Learning | Deep Learning | My SQL | Power BI

Testrinx software solutions | KIET Group of Institutions

View profile