**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**

**14None**

**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**

**Int():**

* To represent integer values without decimal values.

Ex: a=10

>>> type(a)

<class 'int'>

Based on integer representation

1.decimal form

2.bainary form

3.octal form

4.hex decimal form

**Decimal Form** :

* Base is 10 that is 0 to 9

Ex:

a=10

>>> type(a)

<class 'int'>>

**Binary form:**

* Base is 2 i.e 0 and 1

1. In python number start with 0b (or)0B

Ex: 1

>>> a=0b1010

>>> a

10

>>> type(a)

<class 'int'>

Ex2 : c=0b1111

1. >>> c
2. 15
3. >>> type(c)
4. <class 'int'>

**Octal Form :**

* octal form means is ‘8’that is 0To 7
* in python number start with 0o(or) 0o

Ex:1 w=0o1111

>>> w

585

>>> type(w)

<class 'int'>

Ex: y=0o123

>>> y

83

>>> type(y)

<class 'int'>

**Hex Decimal Form :**-

* Base is ‘16’ 0To 9 ato f (or)atof
* In python number start with ‘0x (or)0X

Ex:1 :- e=0x123af

>>> e

74671

>>> type(e)

<class 'int'>

Ex:2 f=0X123ef

>>> f

74735

>>> type(f)

<class 'int'>

Ex:3 r=123re

**SyntaxError: invalid syntax**

**>>>it is out of the limit that Is base is ‘16’a10 of 0To9**

**float():**-

Ex:1 f=12.456

>>> f

12.456

>>> type(f)

<class 'float'>

**ID()**:-

* To know the address of variable .

Ex:1f=12.456

>>> f

12.456

>>> type(f)

<class 'float'>

>>> id(f)

2263526416496

**Base Conversion** :-

* To converd the information into particular base

**Bin():**

Ex: bin(10) # Decimal

'0b1010'

>>> bin(0o123) #Octal

'0b1010011'

>>> bin(0x123ef) #Hex decimal

'0b10010001111101111'

>>> bin(12.45) # float

Traceback (most recent call last):

File "<pyshell#3>", line 1, in <module>

bin(12.45)

TypeError: 'float' object cannot be interpreted as an integer

* **In binary int ,oct,hex,decimal,conversion possible .**

**Oct()**:-

Ex :-

>> oct(10) #decimal

'0o12'

>>> oct(0b1111) #bin

'0o17'

>>> oct(0x12ef) # hex decimal

'0o11357'

>>> oct(12.12) #float

Traceback (most recent call last):

File "<pyshell#3>", line 1, in <module>

>> oct(12.12)

TypeError: 'float' object cannot be interpreted as an integer

* **In octal conversion decimal ,binary ,hex decimal is possible.**

**Hex()**:-

Ex :-

>> hex(10) #Decimal

'0xa'

>>> hex(0b1111) #bin

'0xf'

>>> hex(0o123) #Octal

'0x53'

* **In hex decimal conversion decimal ,octal , binary is possible .**

**Float ()**:-

Ex:-

>> f=12.12

>>> f

12.12

>>> f=0b1010.11 #bin

SyntaxError: invalid syntax

>>> f=0o123.12 # octal

SyntaxError: invalid syntax

>>> f=0x123ef #hex

>>> f

74735

* **In float conversion binary ,octal ,hex ,decimal is not possible decimal form is only is possible .**

Note:- **exponential form (or) scientific notetation E(or)e that means 10 to the power of value .**

Ex :-

>> f=10e2 **#10 \*102**

>>> f

1000.0

>>> d=2e3 **#2\*103**

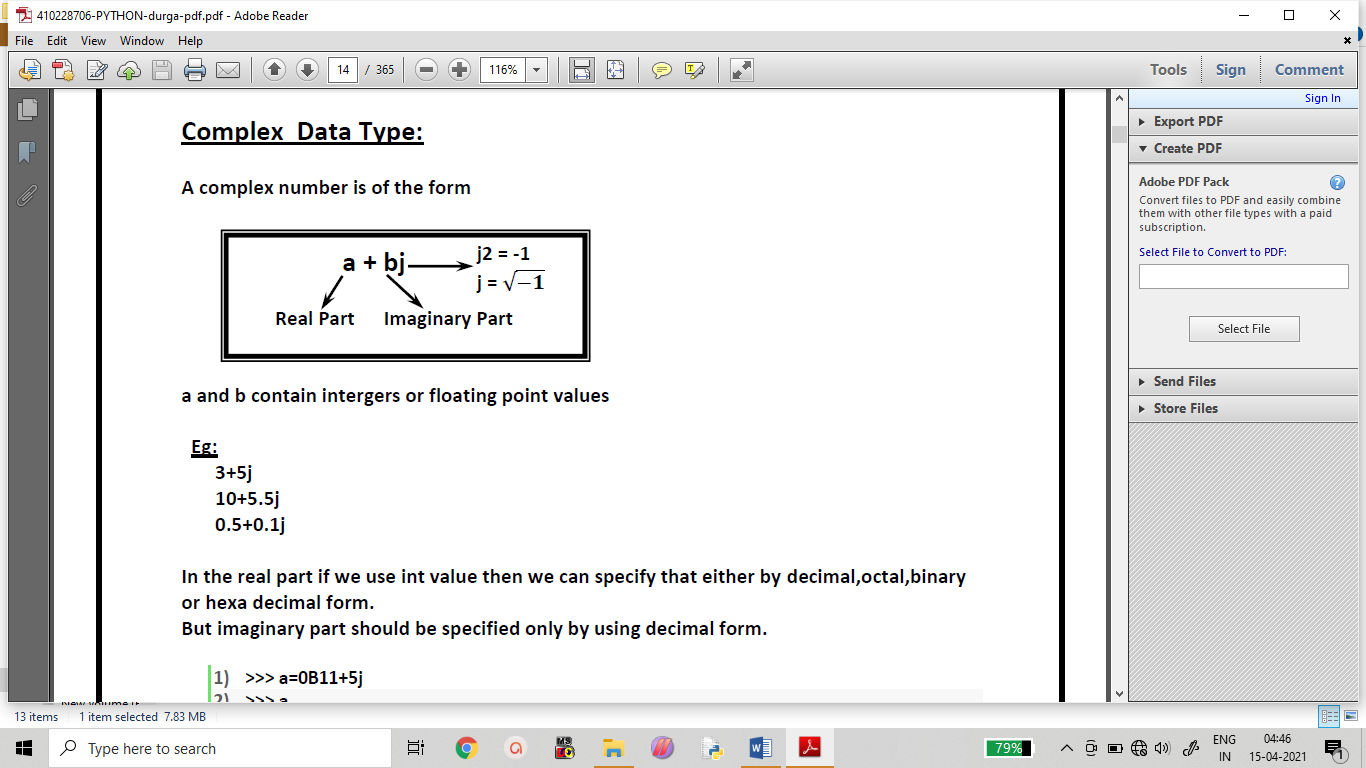
>>> d

2000.0

>>> w=10e4

>>> w

100000.0

**Complex ()**:-

* a+ bj
* (a is a )Real part &( bj is a )imaginary part.

Ex:-

>>f=10+34j

>>> f

(10+34j)

>>> type(f)

<class 'complex'>

**Retrieve value** :-

f=10+34j

Real Part

* f.real

>> 10.0

**Imaginary part**

* f.imag

>>34.0

**conversion** :-

* **Real part accept any kind of values but imaginary part accept any decimal values** .

**Real part** :-

Ex :-

>>d=0b1010+12j #bin

>>> d

(10+12j)

>>> d=0o123+12j #octal

>>> d

(83+12j)

>>> d=0x12ef+12j #hex

>>> d

(4847+12j)

>>> d=12+67j # decimal

>>> d

(12+67j)

**Imaginary Part** :-

* **In imaginary part decimal ,float values (or )possible in imaginary part binary ,octal ,hex decimal are not possible.**

Ex :-

>> a=12+0b1010j #bin

SyntaxError: invalid syntax

>>> a=12+0o123j #octal

SyntaxError: invalid syntax

>>> a=12+0x12efj #hex

SyntaxError: invalid syntax

>>> a=12+12.45j #float

>>> a

(12+12.45j)

>>> a=12+65j #decimal

>>> a

(12+65j)

**Bool ()**:-

* True means ‘1’
* False means ‘0’

Ex :-

>> a=True

>>> a

True

>>> type(a)

<class 'bool'>

>>> b=False

>>> b

False

>>> type(b)

<class 'bool'>

**TRUTH TABLE** :-

Ex :-

True+ True

>> 2

>>> True+ False

>>1

>>> False +False

>>0

Note:- T is capital in True.

F is capital in False .

Ex :-

a=10

>>> b=20

>>> a<b

True

>>> a>b

False

**String()**:-

* **String can represent in ‘ ‘ and “ “ and ‘’’ ‘’’**

Ex :-

>> s="sekhar"

>>> s

'sekhar'

>>> type(s)

<class 'str'>

>>> d='sekhar'

>>> d

'sekhar'

>>> type(d)

<class 'str'>

>>> e='''sekhar'''

>>> e

'sekhar'

>>> type(e)

<class 'str'>

**Slicing operation ():**-

* “[]” operated is called slice operator
* python can fallow ‘0’ based index method index can be positive numbers \_left to right direction.
* Index can be negative numbers \_right to left direction.
* s[began :end] end is optional .

Ex :-

>> s="sekhar"

>>> s

'sekhar'

>>> s[0]

's'

>>> s[-1]

'r'

>>> s[-4]

'k'

>>> s[:] the default index values ‘0’ position &end is index (-1) value ‘sekhar ‘

'sekhar'

>>> s[:3]

'sek'

S[-4:-1]

‘kha’

S[100]

Error string index is out of range .

S[-1:-2]

‘ ‘

* **if any ‘two’ destination negative index values it return ‘EMPTY’ parentheses** .

Ex :-

s="book"

>>> s

'book'

>>> s\*5

'bookbookbookbookbook'

>>>

**len()**:-

* **To fined the particular string length .**

Ex:-

>>> e="mythri ojas institute"

>>> e

'mythri ojas institute'

>>> len(e)

21

>>> len("sekhar")

8

>>> len("hai pandu how are you ")

22

**int**

**Float**

**Complex => Fundamental data types .**

**Bool**

**Str**

**Type Conversion** :-

* **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()**

**2. float()**

**3. complex()**

**4. bool()**

**5. str()**

**int ()**:-

* **To converd particular data into ‘int’data type.**

Ex:-

>>int(10)

10

>>> int(12.45)

12

>>> int(0b111)

7

>>> int(0o123)

83

>>> int(0x12aef)

76527

>>> int(True)

1

>>> int(False)

**Decimal**

**Oct =======>conversion is possible in INT**

**Hex decimal**

**Binary**

**Bool**

**Complex**

**Str =>conversion is not possible in INT**

0

>>> int("10")

10

>>> int("34")

34

>>> int(1+2j)

Traceback (most recent call last):

File "<pyshell#10>", line 1, in <module>

int(1+2j)

TypeError: can't convert complex to int

**float():-**

Ex :-

>>float(10)

10.0

>>> float(0b111)

7.0

>>> float(0o123)

83.0

>>> float(0x123af)

74671.0

>>> float(True)

1.0

>>> float(False)

0.0

>>> float(1+12j)

**FLOAT DATA TYPE**

* **Decimal**
* **Oct ==>To conversion is possible in float data type.**
* **Hex**
* **Bin**
* **Bool**

**complex**

**str =====>Not possible in Float**

Traceback (most recent call last):

File "<pyshell#6>", line 1, in <module>

float(1+12j)

TypeError: can't convert complex to float

>>> float("sekhar")

Traceback (most recent call last):

File "<pyshell#7>", line 1, in <module>

float("sekhar")

ValueError: could not convert string to float: 'sekhar'

**complex():-**

**Method:-1**

>> complex(10)

(10+0j)

>>> complex(10.5)

(10.5+0j)

>>> complex(True)

(1+0j)

>>> complex(False)

0j

>>> complex(0b111)

(7+0j)

>>> complex(0o123)

(83+0j)

>>> complex(0x12aef)

(76527+0j)

>>> complex("sekhar")

Traceback (most recent call last):

File "<pyshell#7>", line 1, in <module>

complex("sekhar")

ValueError: complex() arg is a malformed string

**Method:-2**

Syntax :- complex (x, y) (or) (x+ y j)

Real part & imaginary part

Ex :-

>> complex(10,20)

(10+20j)

>>> complex(10.3,12,5)

Traceback (most recent call last):

File "<pyshell#1>", line 1, in <module>

complex(10.3,12,5)

TypeError: complex() takes at most 2 arguments (3 given)

>>> complex(10.3,12.5)

(10.3+12.5j)

>>> complex(True ,False)

**COMPLEX DATA TYPE**

**1) decimal**

**2) float**

**3)oct**

**3)hex decimal => to conversion is possible in complex data type**

**4)bin**

**5)bool**

**str ==>not possible complex data type**

(1+0j)

>>> complex("12","13")

Traceback (most recent call last):

File "<pyshell#4>", line 1, in <module>

complex("12","13")

TypeError: complex() can't take second arg if first is a string

>>> complex(0o123)

(83+0j)

>>> complex(0x12e,0x12e)

(302+302j)

**Bool():-**

* **non ‘0’ zero is True \_1**
* **‘0’ is False \_0**

**Bool to decimal**:-

Ex:-

>>bool (0)

False

>>> bool (1)

True

>>> bool (10)

True

>>> bool (-10)

True

**Bool To Float** :-

* In float form every decimal point ‘0’ means False
* In float form every decimal point non zero ‘0’ means True.

Ex:-

>>bool(12.12)

True

>>> bool(0.0)

False

>>> bool(0.01)

True

>>> bool(0.2)

True

>>> bool(0.1)

True

>>> bool(0.000)

False

\*bool to complex :-

Truth table :-

Example :-

bool(1+0j)

True

>>> bool(0+1j)

True

>>> bool(10+20j)

True

**Bool To String ()**:-

* if argument is EMPTY string it treat as false in other all cases it is true
* Space is treated as ‘1’character .

Ex:-

>>> bool(" ")

True

>>> bool("")

False

>>> bool(' ')

True

>>> bool('hi')

True

>>> bool("sekhar")

True

* bool to string as arguments .

**string ()**:-

* It convert any values to string .

Ex :-

>>> str(10)

'10'

>>> str(12.34)

'12.34'

>>> str(12+34j)

'(12+34j)'

>>> str(True)

'True'

>>> str(False)

'False'

>>> str(0b1111)

'15'

>>> str(0o123)

'83'

>>> str(0xa12)

'2578'

>>>

**IMMUTABLE (VS) FUNDAMENTAL DATA TYPE .**

* **All Fundamental Data types are immutable. i.e once we creates an object, we cannot perform any changes in that object. If we are trying to change then with those changes a new object will be created. This non-changeable behaviour is called immutability.**
* **To prevent this immutability concept is required. According to this once creates an object we are not allowed to change content. If we are trying to change with those changes a new object will be created.**

Examples :-

>> v1="hyd"

>>> v2="hyd"

>>> v3="hyd"

>>> v4="world"

>>> id(v1)

1698383456368

>>> id(v2)

1698383456368

>>> id(v3)

1698383456368

>>> id(v4)

1698383456240

>>> v5="world"

>>> id(v5)

1698383456240

=>memory utilization performance is their .

Note:- immutable means changeable .

**BYTES DATA TYPE** :-

* Group of value and we can use it.
* It is immutable .

**Conclusion 1:**

**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.**

**Conclusion 2:**

**Once we creates bytes data type value, we cannot change its values, otherwise we will get Type Error.**

Ex:-

>>> x=[10,30,50]

>>> b=bytes(x)

>>> type(b)

<class 'bytes'>

>>> b[-1]

50

>>> b[1]

10

>>> b[1]=100

Traceback (most recent call last):

File "<pyshell#7>", line 1, in <module>

b[1]=100

TypeError: 'bytes' object does not support item assignment

* **Dose not support item assignment .**

Ex: >>> x=[10,20,256]

>>> for b in x:

print(x)

output:

10

20

256

Ex:-

>>> x=[10,20,255]

>>> b=bytes(x)

>>> for x in b:print(x)

Output:

10

20

255

**Byte Array** :-

* **It hold group values the mutable.**
* **In the range ‘0’ to 256 Only**

Ex:-

>>> x=[10,20,255]

>>> b=bytes(x)

>>> for x in b:print(x)

Output:

10

20

255

* **Reason: byte array it hold group values these immutable in the range ‘0’ to 255**

Ex:-

>>> x=[10,20,30,40]

>>> b=bytearray(x)

>>> type(b)

<class 'bytearray'>

>>x=[10,20,30,40]

>>> for x in b:

print(x)

10

20

30

40

>>> b[-1]

40

>>> b[0]

10

=>b[0]=100# ten is replacead with 100 so that assignment possible .

Ex:-

For x in b:

print(x)

output:

100

20

30

40

>>b[1]=200

>>b[1]

>>200

X=[10,20,256]

b=bytearray (x)

>>Error :because bytearray range is 0 to 256 only

**Note : No one are used bytes and bytearray** .

List Data Type :-

* **It contained group of values list of values represent in a[ ].**

**List():-**

1. **Order allowed**
2. **Duplicate cation allowed**
3. **Heterogeneous allowed**
4. **Growable value allowed---> incase and decrease ‘or’ remove**
5. **Values enclosed with in the []**

Ex :-

>>> l=[]

>>> type(l)

<class 'list'>

>>> l.append(10)

>>> l.append(20)

>>> l.append(30)

>>> print(l)

>>> print(l)

>> [10, 20, 30]

>>> l.append(20)

>>> print(l)

[10, 20, 30, 20]

**Note:-Hetrogenious object means different types of object allowed .**

Ex:-

>>> l.append("sekhar")

>>> l.append(12.34)

>>> print(l)

>> **[10, 20, 30, 20, 'sekhar', 12.34]**

>>> l.append(None)

>>> print(l)

>>**[10, 20, 30, 20, 'sekhar', 12.34, None]**

>>> l[0]

10

>>> **l[0:5]**

[10, 20, 30, 20, 'sekhar']

>>> l[-1]

>>> print(1-1)

1

[10,20,20,’sekhar’,12,34,None]

1.remove(-1)

1

[10,20,’sekhar’ 12.34]

**tuple ()**:-

* **A tuple contain group of values .**
* **List and tuple are the same but list is mutable tuple is immutable .**
* **Tuple can mention ‘()’ parentheses once we create tuple we canot’ modify the data range**
* **Range not there like ‘0’ to 256 it is only bytes and bytearray .**

Ex:-

>>> t=(10,20,30,40,60)

>>> type(t)

<class 'tuple'>

>>> t[0]

10

>>> t[-1]

60

>>> **t[0:4]**

(10, 20, 30, 40)

>>> t[0]

10

* **tuple object does not support item assignment tuple is immutable data type .**

Ex:-

>>> t=(10,'sekhar','sekhar',10,20)

>>> print(t)

(10, 'sekhar', 'sekhar', 10, 20)

* **Duplication is available it fallowing oder**

>>> t1=t\*2

>>> t1

(10, 'sekhar', 'sekhar', 10, 20, 10, 'sekhar', 'sekhar', 10, 20)

**Range ()**:-

* Represent the sequence of values it is always immutable range is data type and also function .
* Range is always can mainly data type used .

**Form:-1 range (end )**

* **It represent the values ‘0’ to 1 end .**

Ex :-

>>> range(10)

range(0, 10)

>>> type(range(10))

<class 'range'>

>>> r=range(10)

>>> type(r)

<class 'range'>

>>> for i in r:print(i)

0

1

2

3

4

5

6

7

8

9

>>> r[0]

0

>>> r[0:3]

>>range(0, 3)

>>> r[0]=100

**>>Error**

**Range does not support item assignment** .

**Form :2**

Ex:-

r= range(10,20)

Range (10,20) {10 is **starting** and 20 is **end-1**”}

Ex :-

>>> r=range(10,20)

>>> for i in r:print(i)

10

11

12

13

14

15

16

17

18

19

**Form :-3**

Range (10,20,2) {**start and end-1 ,step every time increment by 2 steps** .}

Ex:-

>>> r=range(20,40,2)

>>> for i in r:print(i)

20

22

24

26

28

30

32

34

36

38

>>> r=range(20,40,3)

>>> for i in r:print(i)

20

23

26

29

32

35

38

>>> range(10.5,12.5)

Traceback (most recent call last):

File "<pyshell#8>", line 1, in <module>

range(10.5,12.5)

TypeError: 'float' object cannot be interpreted as an integer

>>> error not support in float values ,only with support integer values .

**Note:- all data types immutable**

* **Byte**
* **List**
* **Tuple**
* **Range**
* **bytearray ------->Mutable .**

**SET():-**

* **It can represent the information {}curly brackets**
* **Incersation order not allowed .**
* **Duplication not allowed .**
* **heterogeneous objects allowed .**
* **Index value not value .**
* **Slicing not applicable ,not possible .**
* **It is a mutable .**

Ex:-

>>> s={10,20,30,10,20,30}

>>> type(s)

<class 'set'>

>>> print(s)

{10, 20, 30}

>>> s[0]

Traceback (most recent call last):

File "<pyshell#4>", line 1, in <module>

s[0]TypeError: 'set' object is not subscriptable

* **Indexing slicing switch type of terminology not applicable set because set internally order is not there** .

Ex:--

>>s.add (“mytri”)

>>print(s)

>>{10,’mytri’,20,30}

>>s.remove(30)

>>s

>>{10,”mytri”}

**Note :- append is useful only list data type not set data type** .

Ex:-

>>> s={'hi', 'good' ,'mornig'}

>>> print(s)

>>{'hi’,’good’,’mornig'}

* **oder in not impartent in set data type**.

**frozenset():-**

* **Group of values does not change any values we can usein frozenset.**
* **It is exactly same as set data type**
* **It is immutable**
* **Indexing not applicable**
* **Heterogeneous is allowed**
* **Order is not important .**
* **Duplication not allowed** .

#set data type

Ex:-

>>> s={20,10,'hello'}

>>> fs=frozenset(s)

>>> type(fs)

>> <class 'frozenset'>

>>> fs

>>frozenset{'hello', 10, 20}

>>> fs[0]

Traceback (most recent call last):

File "<pyshell#9>", line 1, in <module>

>> fs[0]

TypeError: 'frozenset' object is not subscriptable

>>> fs.add(13)

Traceback (most recent call last):

File "<pyshell#10>", line 1, in <module>

>>> fs.remove(10)

Traceback (most recent call last):

File "<pyshell#11>", line 1, in <module>

* **Add And Remove Not Possible In Frozenset Data Type**.

**Dict()**:-

* **Group Of Objects As a Key Value**

Ex:-

>> d={100:'sekhar',200:'ram',300:'venu'}

>>> type(d)

>> <class 'dict'>

>>> print(d)

**{100: 'sekhar', 200: 'ram', 300: 'venu'}**

* **If the empty curly {} it is like a dict data type not a set data type .**

Ex:- >>> d1={}

>>> type(d1)

<class 'dict'>

>>> d1[10]='sekhar'

>>> d1

>>{10: 'sekhar'}

>>> d1[20]="hari"

>>> d1

{10: 'sekhar', 20: 'hari'}

>>> d1[20]="sekhar"

>>> d1

{10: 'sekhar', 20: 'sekhar'}

**NOTE:-**

* **Dict is a mutable data type.**
* **BYTES AND BYTEARRAY TO REPRESENT BINARY DATA TYPE LIKE IMAGES VIDEO ,FILES AND AUDIO FILES**

**None()**:-

* None means nothing (or) no values is associated.
* it is just like null values but python does not support null values concepts .

Ex:-

>>> a=None

>>> type(a)

**<class 'NoneType'>**

**Escape sequence character**:-

**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**

**Ex:**

**>>> s="welcome\n to\n python class"**

**>>> print(s)**

**welcome**

**to**

**python class**

**>>> \t horigental:-**

**>>> s="welcome\t to\t python class"**

**>>> print(s)**

**welcome to python class**

**>>> \v vertical:-**

**>>> s="welcome\v to\v python class"**

**>>> print(s)**

**welcome  
 to  
 python class**

**>>> \f form feed- like page down:**

**>>> s="welcome\f python class"**

**>>> print(s)**

**welcome python class**

**>>> \'-back slash with single quotes:-**

**>>> s="welcome\'python"**

**>>> print(s)**

**welcome ‘python**

**>>> \\- back double slash with double quotes:-**

**>>> s="welcome\\python class"**

**>>> print(s)**

**welcome\python class**

**Note:-**

**DATA TYPE SYMBOL**

**BYTES[] IMMUTABLE**

**BYTEARAY[] MUTABLE**

**LIST[] MUTABLE**

**TUPLE[ ] IMMUTABLE**

**RANGE[ ] IMMUTABLE**

**SET{ } MUTABLE**

**FROZENSET { } IMMUTABLE**

**DICT{ } MUTABLE**

