In [ ]:
```
#True or False
1+2==3
b.False--> False
C. None of the above
d Empty Output
```

In [9]:
```
1+2==3
```

Out[9]:
```
True
```

In [ ]:
```
3.1434567890
```

Variable Assignment Object Storage and Mutability

In [1]:
```
#Variables:A Python variable is a symbolic name that is a reference or pointer to an object.
#Variable Assignment :
Variable = object_name
```

In [ ]:
```

In python each and everything is an object --> Behavior and properties
Examples of object --> Mobile phones, human being , animals etc
```

In [11]:
```
x=10.5
print(type(x))
```

```
<class 'float'>
```

In [2]:
```
#How are Objects Stored in Memory in Python
In python each and everything is an object so each and every object of python is stored in private
heap memory
```

In [ ]:
```
x=10.5 #1object will created
type(x)
```

In [13]:
```
x=[10,20,30,40,50]
print(type(x))
```

```
<class 'list'>
```

In [ ]:
```
x=[10,20,30,40,50]
```

In [19]:
```
x=(10,)
print(type(x))
```

```
<class 'tuple'>
```

In [ ]:
```
x=[10,20,"Hello world","two"]
```

In [17]:
```
(10+20+30)
```

Out[17]:
```
60
```

In [16]:
```

```

In [3]:
```
#Mutability vs Immutability
Mutability: elements got changed
Immutability : element not change
```

Note: All Standard datatypes are immutable i.e once we create an object we cannot change or perform any operation on that object. if we will try to perform any change then because of that change a new object will be created and the variable will be pointed that new object.

In [21]:
```
x=10
print(id(x))
x=x+1
print(id(x))
#Note: All standard datatypes are immutable . Int , float string , boolean , complex.
```

```
2074191620688
2074191620720
```

In [23]:
```
#float
x=10.0
print(id(x))
x=x+1
print(id(x))
```

```
2074273230800
2074273588048
```

In [25]:
```
#String
x="String"
print(id(x))
x=x+"string2"
print(id(x))
x
```

```
2074230105520
2074273623088
```

Out[25]:
```
'Stringstring2'
```

In [26]:
```
#Complex
x=1+2j
print(id(x))
x=x+5j
print(id(x))
x
```

```
2074273587664
2074273588560
```

Out[26]:
```
(3+7j)
```

In [27]:
```
#boolean
x=True
print(id(x))
x=x+True
print(id(x))
x
```

```
140707447658600
2074191620432
```

Out[27]:
```
2
```

Note: In python if we want create a new object. then PVM will not directly creat that object first it will check weather the content of that new object is already present in the memeory or not if the content is already presnet then only new refernce will again point to the old one.

This thing is applicable for all standard datatype except complex number.

In [33]:
```
x=10
y=1
x is y  #Compare the address of two variables or object
```

Out[33]:
```
False
```

In [35]:
```
x=10+2j
y=10+2j
print(id(x))
print(id(y))
```

```
2074273589200
2074273588592
```

In [55]:
```
#Conversions And Typecasting Meaning
#Typecasting : it will always create a new object with the given datatype with new memeory address.
x=10.4
x=int(x)
id(x)==id(x)
```

Out[55]:
```
True
```

In [40]:
```
#conversion of list into tuple is possible
x=[10,20,30,40,50]
x=tuple(x)
id(x)==id(x)
```

Out[40]:
```
True
```

In [42]:
```
#conversion of list into set is possible
x=[10,20,30,40,50]
x=set(x)
id(x)==id(y)
```

Out[42]:
```
False
```

In [ ]:
```
#conversion of list into set is possible
List to dictionary is not possible
```

In [ ]:
```
key:value
```

In [5]:
```
#In which of the follwoing indexing is not important?
set  --> unordered collection of datta that means index is not important
tuple--> orderd collection of data
dictionary  -->unordered collection of datta that means index is not important
string --> ordered indexing is important
list--> ordered
```

In [ ]:
```
list to conversion --> it is possible
tuple to list--> it is possible
```

In [43]:
```
x=(10,20,30,40)
x=list(x)
x
```

Out[43]:
```
[10, 20, 30, 40]
```

In [ ]:
```
differnece between is and ==
```

In [46]:
```
#== compares the objects based on the given value
# is compared the address the object
```

Out[46]:
```
True
```

In [54]:
```
a=[10,20,30]
b=a
c=a[:]
a is c
```

Out[54]:
```
False
```

In [ ]:
```

Datatypes which we can use as  a key in dictionary:
    tuple--> as a key
    float--> as  a key
    string --> as a key
    int --> as a key
    boolean --> as a key
cannot use as a key
    list---> we cannot use list as  a key ->
    set --> cannot as a list  -->
Dictionary keys are unique.
```

In [ ]:
```
Hashing --> dictionary key are hashable and hashable object are not mutable object
```

In [56]:
```
x=([10,20,30,40,50])
x[0]=200
x
```

Out[56]:
```
[200, 20, 30, 40, 50]
```

In [ ]:
```

```