DATA TYPES:

1. INT :

- Int means integers or number
- There is effectively no limit to how long an integer value can be.

In [1]:

```
a= 56
print(id(a))
print(type(a))
```

```
140709926743552
<class 'int'>
```

2. float :

- float values are specified with a decimal point

In [11]:

```
a= 56.6
print(id(a))
print(type(a))
```

```
2180711311920
<class 'float'>
```

BASE CONVERSION :

- Binary : `0` OR `1` , `0b or 0b --> bin()`
- Octadecimal : `0` to `7` , `0o or 0O --> oct()`
- Hexadecimal : `0` to `9` and `a` , `b` , `c` , `d` , `e` , `f` , `0x or 0X --> hex()`

In [3]:

```
a= 10
b1 = bin(a)
print(b1)
print(type(b1))
```

```
0b1010
<class 'str'>
```

In [4]:

```
a= 89
b2 = oct(a)
print(b2)
print(type(b2))
```

```
0o131
<class 'str'>
```

In [5]:

```
a= 25889
b3 = oct(a)
print(b3)
print(type(b3))
```

```
0o62441
<class 'str'>
```

Binary to octadecimal,hexadecimal,decimal

In [6]:

```
a= 0b1010
a_oct = oct(a)
print(a_oct)
a_hex =hex(a)
print(a_hex)
a_dec = int(a)
print(a_dec)
```

```
0o12
0xa
10
```

Octadecimal to Binary, Hexadecimal, Decimal

In [8]:

```

a= 0o131
a_dec = int(a)
print(a_dec)
a_bin = bin(a)
print(a_bin)
a_hex =hex(a)
print(a_hex)
```

```
89
0b1011001
0x59
```

Hexadecimal to Binary, Octadecimal, Decimal

In [10]:

```
a= 0o62441
a_dec = int(a)
print(a_dec)
a_bin = bin(a)
print(a_bin)
a_oct =oct(a)
print(a_oct)
```

```
25889
0b110010100100001
0o62441
```

3. bool :

- bool means boolean value .
- It supports only `True` and `False` or `1` and `0`

In [12]:

```
a= True
print(id(a))
print(type(a))
```

```
140709926459216
<class 'bool'>
```

In [13]:

```
a = False
print(id(a))
print(type(a))
```

```
140709926459248
<class 'bool'>
```

In [14]:

```
True + True # True = 1. So, 1+1 =2
```

Out[14]:

```
2
```

In [15]:

```
False+True  # True = 1 and Flase = 0. So, 0+1 = 1
```

Out[15]:

```
1
```

In [16]:

```
False + False
```

Out[16]:

```
0
```

In [17]:

```
a = 20
b =30
a>b
```

Out[17]:

```
False
```

In [18]:

```
a<b
```

Out[18]:

```
True
```

In [19]:

```
a==b
```

Out[19]:

```
False
```

In [20]:

```
a!=b
```

Out[20]:

```
True
```

4. String :

- It is a sequence of character within the `quote (' ' or " ")` .
- It is immutable.

In [21]:

```
a= 'Aditya'
```

In [22]:

```
print(id(a))
print(type(a))
```

```
2180711763376
<class 'str'>
```

In [23]:

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

```
A
d
i
t
y
a
```

In [24]:

```
a[0]
```

Out[24]:

```
'A'
```

In [25]:

```
print(a[-1])
print(a[-3])
print(a[:3])
print(a[4:])
```

```
a
t
Adi
ya
```

In [26]:

```
print(a[1:5])
```

```
dity
```

In [27]:

```
print(a[::2])
```

```
Aiy
```

In [28]:

```
b='Ranjan'
print(b[::-1]) # reverse of a string
```

```
najnaR
```

In [29]:

```
c= 'adityakumar'
print(c[3:6:2])
```

```
ta
```

In [30]:

```
print(c[3:6:2]) # start at 't' ,end at 'a',step is 2-1 =1 , t_a ===> ta
print(c[2::3]) # start at 'i' ,end at 'r',step is 3-1 =2 , i__a__m ===> iam
```

```
ta
iam
```

In [31]:

```
print(c[-6:-2]) # start at 'a' ,end at 'm',step is -2+1 = -1 alternate
```

```
akum
```

In [32]:

```
d = "Hello, World!"
print(d[:-2])# start at 'H' and end at 'd'
print(d[-3:-2]) # start at 'l' and end at 'l'
print(d[-8:-1]) # start at ',' and end at 'd'
```

```
Hello, Worl
l
, World
```

Type Casting or Type conversion

- conversion of one data type into other data type is called `Type Casting` or `Type Conversion` .
- int()
- float()
- str()
- bool()
- complex()

1. int()

In [34]:

```
a= 16
print(type(a))
```

```
<class 'int'>
```

In [35]:

```
# int to str
a_str = str(a)
print(a_str)
print(type(a_str))

# int to float
a_float = float(a)
print(a_float)
print(type(a_float))

# int to bool
a_bool = bool(a)
print(a_bool)
print(type(a_bool))

# int to complex
a_complex = complex(a)
print(a_complex)
print(type(a_complex))
```

```
16
<class 'str'>
16.0
<class 'float'>
True
<class 'bool'>
(16+0j)
<class 'complex'>
```

2. float()

In [36]:

```
b= 15.6
print(type(b))



# float to str
b_str = str(b)
print(b_str)
print(type(b_str))

# float to bool
b_bool = bool(b)
print(b_bool)
print(type(a_bool))

# float to complex
b_complex = complex(b)
print(b_complex)
print(type(b_complex))


# float to int
b_int = int(b)
print(b_int)
print(type(b_int))
```

```
<class 'float'>
15.6
<class 'str'>
True
<class 'bool'>
(15.6+0j)
<class 'complex'>
15
<class 'int'>
```

3. str()

- Str() can be convertable into `bool` only

In [37]:

```
c= 'aditya'
print(type(c))
```

```
<class 'str'>
```

In [38]:

```
# str to int
c1 = int(c)
print(c1)
print(type(c1))
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-38-bda89cb36b34> in <module>
      1 # str to int
----> 2 c1 = int(c)
      3 print(c1)
      4 print(type(c1))

ValueError: invalid literal for int() with base 10: 'aditya'
```

In [39]:

```
# str to bool
c2 = bool(c)
print(c2)
print(type(c2))
```

```
True
<class 'bool'>
```

In [40]:

```
# str to complex
c3 = complex(c)
print(c3)
print(type(c3))
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-40-da0a3fbca79d> in <module>
      1 # str to complex
----> 2 c3 = complex(c)
      3 print(c3)
      4 print(type(c3))

ValueError: complex() arg is a malformed string
```

4. bool()

In [42]:

```
d = True
type(d)
```

Out[42]:

```
bool
```

In [44]:

```
d1 =str(d)
type(d1)
print(d1)
```

```
True
```

In [46]:

```
d2 = complex(d)
type(d2)
print(d2)
```

```
(1+0j)
```

In [47]:

```
d3 = int(d)
type(d3)
d3
```

Out[47]:

```
1
```

In [48]:

```
d4 = float(d)
type(d4)
d4
```

Out[48]:

```
1.0
```

5. Complex()

- complex() cannot be convertable to `int` and `float`

In [49]:

```
e = 2+3j
type(e)
```

Out[49]:

```
complex
```

In [50]:

```
e1 = str(e)
type(e1)
e1
```

Out[50]:

```
'(2+3j)'
```

In [51]:

```
e2 = int(e)
type(e2)
e2
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-51-7494fec3ac7a> in <module>
----> 1 e2 = int(e)
      2 type(e2)
      3 e2

TypeError: can't convert complex to int
```

In [52]:

```
e3 =float(e)
type(e3)
e3
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-52-eb9d30772418> in <module>
----> 1 e3 =float(e)
      2 type(e3)
      3 e3

TypeError: can't convert complex to float
```

In [53]:

```
e4 =bool(e)
type(e4)
e4
```

Out[53]:

```
True
```

In [54]:

```
a=10
print(id(a))
b=10
print(id(b))
c=10
print(id(c))
d=10
print(id(d))
e=10
print(id(e))
# we have same id for every variable because we store value in memory not the variable
```

```
140709926742080
140709926742080
140709926742080
140709926742080
140709926742080
```

5. List --> [ ] :

1. List is the collection of `different object` in a single entity
2. It contains any data structure

Important Points

1. Insertion order preserved.
2. Duplicate objects are allowed.
3. Heterogenous objects are allowed.
4. It is mutable.
5. mutable means it can be changed
6. value at any index can be changed
7. Slicing and neagtive indexing can be done on list
8. Denoted by `[ ]`

In [55]:

```
l =[1,'a',10.5,True,1+10j]
print(type(l))
print(l)
```

```
<class 'list'>
[1, 'a', 10.5, True, (1+10j)]
```

In [56]:

```
l[0]
```

Out[56]:

```
1
```

In [57]:

```
l[-1]
```

Out[57]:

```
(1+10j)
```

In [99]:

```
l1 =[10,10.5,'aditya',10+20j,False]
l1
```

Out[99]:

```
[10, 10.5, 'aditya', (10+20j), False]
```

In [59]:

```
print(l1[-2])
print(l1[1])
```

```
(10+20j)
10.5
```

In [60]:

```
print(l1[::-1])
```

```
[False, (10+20j), 'aditya', 10.5, 10]
```

In [100]:

```
l2 =[10,10,20,20,'as','as',True,True]
l2
```

Out[100]:

```
[10, 10, 20, 20, 'as', 'as', True, True]
```

In [62]:

```
#Slicing
print(l1[1:4])
print(l2[2:6])
```

```
[10.5, 'aditya', (10+20j)]
[20, 20, 'as', 'as']
```

In [101]:

```
# Adding value/element in list l1 i.e mutable
l1.append(50)
l1
```

Out[101]:

```
[10, 10.5, 'aditya', (10+20j), False, 50]
```

Methods supported by list data type in python:

In [102]:

```
# copy() : Copy the list into other list
l2= l1.copy()
print(l2)
```

```
[10, 10.5, 'aditya', (10+20j), False, 50]
```

In [103]:

```
# clear() : Clear the whole list and returns empty list
l2.clear()
print(l2)
```

```
[]
```

In [104]:

```
# reverse(): reverse the list
print("Original list : ",l1)
l1.reverse()
print("Reversed list :", l1)
```

```
Original list :  [10, 10.5, 'aditya', (10+20j), False, 50]
Reversed list : [50, False, (10+20j), 'aditya', 10.5, 10]
```

In [91]:

```
# extend() : The extend() method adds the specified list elements (or any iterable) to the
f = ['apple', 'banana', 'cherry']
print("list f :",f)
c = ['Ford', 'BMW', 'Volvo']
print("list c :",c)
f.extend(c)

print("Extended List : ",f)
```

```
list f : ['apple', 'banana', 'cherry']
list c : ['Ford', 'BMW', 'Volvo']
Extended List :  ['apple', 'banana', 'cherry', 'Ford', 'BMW', 'Volvo']
```

In [108]:

```
# sort() : sort the element in the list
num = [5,6,10,8,4,5,12,4]
print("Before sort: ",num)
num.sort()
print("After sorting: ",num)
```

```
Before sort:  [5, 6, 10, 8, 4, 5, 12, 4]
After sorting:  [4, 4, 5, 5, 6, 8, 10, 12]
```

In [111]:

```
# count(): Returns the number of elements with the specified value
num.count(5)

# pop() : pop the last elements from the list
num.pop()
print(num)
```

```
[4, 4, 5, 5, 6]
```

In [112]:

```
# index(): Returns the index of the element
fruits = ['apple', 'banana', 'cherry']

x = fruits.index("cherry")
print(x)
```

```
2
```

In [113]:

```
# remove() :  The remove() method removes the first occurrence of the element with the spec
fruits = ['apple','cherry', 'banana', 'cherry','banana']

x = fruits.remove("cherry")
print(fruits)
```

```
['apple', 'banana', 'cherry', 'banana']
```

6. Tuple --> ( ) :

1. It is also the collection of `different object` in a single entity
2. It contains any data structure
3. It is same as List.

Important Points

1. Insertion order preserved.
2. Duplicate objects are allowed.
3. Heterogenous objects are allowed.
4. It is immutable.
5. Immutable means it cannot be changed onces declared
6. Value at any index cannot be changed
7. Slicing and negative indexing can be done on list
8. Declared using ( )
9. If tuple contain only one element then it will act as int :
    - a= (10)
    - type(a) --> int

In [114]:

```
t = (1,2,10.5,True,1+10j)
print(type(t))
t
```

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

Out[114]:

```
(1, 2, 10.5, True, (1+10j))
```

In [115]:

```
t1 = (11,11,12,10.5,False,10j)
print(t1)
```

```
(11, 11, 12, 10.5, False, 10j)
```

In [116]:

```
t1[-2]
```

Out[116]:

```
False
```

In [117]:

```
t1[2]
```

Out[117]:

```
12
```

In [118]:

```
t1[2:4]
```

Out[118]:

```
(12, 10.5)
```

In [121]:

```
t2 =(10,10,10,20,30,50)
print(t2)
print(t2.count(10))
```

```
(10, 10, 10, 20, 30, 50)
3
```

In [120]:

```
t2.append(50)
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-120-b181a5d236f1> in <module>
----> 1 t2.append(50)

AttributeError: 'tuple' object has no attribute 'append'
```

Tuple methods:

- `count()`: Returns the number of times a specified value occurs in a tuple
- `index()`: Searches the tuple for a specified value and returns the position of where it was found

7. Set --> { } :

1. Set is a important data type in Python.

Important Points

1. Set contains unique value
2. It does not preserved insertion order.
3. Declared by { }
4. It allows hetrogeneous Objects
5. It is mutable

In [122]:

```
s = {10, 10,20 ,30}  # Always give unique values
type(s)
print(s)
```

```
{10, 20, 30}
```

In [123]:

```
s1 = {10,10.5,10.20j,"aadi",True}
print(s1)
```

```
{True, 10, 10.5, 'aadi', 10.2j}
```

In [124]:

```
s1.add(20)
s1
```

Out[124]:

```
{10, 10.2j, 10.5, 20, True, 'aadi'}
```

In [125]:

```
# s1.clear()

s1
```

Out[125]:

```
{10, 10.2j, 10.5, 20, True, 'aadi'}
```

Set methods:

- `add()` : Adds an element to the set
- `clear()`: Removes all the elements from the set
- `copy()`: Returns a copy of the set
- `difference()`: Returns a set containing the difference between two or more sets
- `difference_update()`: Removes the items in this set that are also included in another, specified set
- `discard()`: Remove the specified item
- `intersection()`: Returns a set, that is the intersection of two other sets
- `intersection_update()`: Removes the items in this set that are not present in other, specified set(s)
- `isdisjoint()`: Returns whether two sets have a intersection or not
- `issubset()`: Returns whether another set contains this set or not
- `issuperset()`: Returns whether this set contains another set or not
- `pop()`: Removes an element from the set
- `remove()`: Removes the specified element
- `symmetric_difference()`: Returns a set with the symmetric differences of two sets
- `symmetric_difference_update()`: inserts the symmetric differences from this set and another
- `union()`: Return a set containing the union of sets
- `update()`: Update the set with the union of this set and others

8. Frozonset --> { }

- set and frozonset are same but frozenset is immutable

Important Points

1. Set contains unique value
2. It does not preserved insertion order.
3. Declared by { }
4. It allows hetrogeneous Objects
5. It is immutable

In [127]:

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

Out[127]:

```
frozenset
```

In [128]:

```
fs.add(20)
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-128-4eba47f9f4b6> in <module>
----> 1 fs.add(20)

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

In [129]:

```
fs[0]
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-129-7056c4be9cfe> in <module>
----> 1 fs[0]

TypeError: 'frozenset' object is not subscriptable
```

In [130]:

```
s1 = {10,10.5,10,10+20j,"aadi",True}
fs1 =frozenset(s1)
fs1
```

Out[130]:

```
frozenset({(10+20j), 10, 10.5, True, 'aadi'})
```

In [131]:

```
s2 = {0,30,40}
fs2 = frozenset(s2)
fs2
```

Out[131]:

```
frozenset({0, 30, 40})
```

In [132]:

```
fs1.union(fs2)
```

Out[132]:

```
frozenset({(10+20j), 0, 10, 10.5, 30, 40, True, 'aadi'})
```

9. Dictionary --> {Key: value}:

- It is data type which store value in the form of Key -value pairs.
- Key must be unique but value can be duplicate
- Key is immutable while Values are mutable
- So, overall dict is mutable.
- Heterogeneous objects are allowed for both key and value.
- Declared by {'key': 'value' }

In [133]:

```
d = {1:4,2:5,6:8,9:27}
print(type(d))
print(id(d))
print(d)
```

```
<class 'dict'>
2180711142016
{1: 4, 2: 5, 6: 8, 9: 27}
```

In [134]:

```
d1 ={'aditya':10,"ranjan":50,'amar':41,'sonu':89}
d1
```

Out[134]:

```
{'aditya': 10, 'ranjan': 50, 'amar': 41, 'sonu': 89}
```

10. range:

- it represent sequence of numbers.
- denoted by range()

In [136]:

```
a= range(10)
a
```

Out[136]:

```
range(0, 10)
```

In [137]:

```
type(a)
```

Out[137]:

```
range
```

In [138]:

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

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

In [139]:

```
c=[]
for i in range(11):
    c.append(i)
print(c)
```

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

In [140]:

```
# range (initial_position,end_position)
b= range(10,24)
print(b[-1])

print(b[5])
```

```
23
15
```

In [141]:

```
for i in range(60,50,-1):
    print(i)
```

```
60
59
58
57
56
55
54
53
52
51
```

Bytes

In [142]:

```
b1 = [10,20,30,40]
b1
```

Out[142]:

```
[10, 20, 30, 40]
```

In [145]:

```
b = bytes(b1)
print(type(b))
print(b)
```

```
<class 'bytes'>
b'\n\x14\x1e('
```

In [146]:

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

```
10
20
30
40
```

In [148]:

```
x= [20,30,40,255,257,256]
x1 = bytes(x)
x1
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-148-29ee299bade8> in <module>
      1 x= [20,30,40,255,257,256]
----> 2 x1 = bytes(x)
      3 x1

ValueError: bytes must be in range(0, 256)
```

In [149]:

```
x2= [20,30,40,255]
x3 = bytes(x2)
x3
```

Out[149]:

```
b'\x14\x1e(\xff'
```

bytearray()

- bytes are immutable and bytearray are mutable

In [150]:

```
x= [20,30,40,255]
x1 = bytearray(x)
x1
```

Out[150]:

```
bytearray(b'\x14\x1e(\xff')
```

In [151]:

```
for i in x1:
    print(i)
```

```
20
30
40
255
```

None

- It means nothing i.e. no value is associated with it.
- if value is not available then none is here.
- None is simialr to NULL value in Java

Operator

1 .Arthemetic operator:

`+` addition
`-` Subtraction
`*` Multiplication
`/` Division
`//` Floor division
`**` Exponential or power

In [152]:

```
a= 10
b= 3
print(f'{a}+{b} =',a+b)
print(f'{a}*{b} =',a*b)
print(f'{a}/{b} =',a/b)
print(f'{a}//{b} =',a//b)
print(f'{a}**{b} =',a**b)
print(f'{a}-{b} =',a-b)
```

```
10+3 = 13
10*3 = 30
10/3 = 3.3333333333333335
10//3 = 3
10**3 = 1000
10-3 = 7
```

In [153]:

```
a= 10.5
b= 1.6
print(f'{a}+{b} =',a+b)
print(f'{a}*{b} =',a*b)
print(f'{a}/{b} =',a/b)
print(f'{a}//{b} =',a//b)
print(f'{a}**{b} =',a**b)
print(f'{a}-{b} =',a-b)
```

```
10.5+1.6 = 12.1
10.5*1.6 = 16.8
10.5/1.6 = 6.5625
10.5//1.6 = 6.0
10.5**1.6 = 43.04303423791882
10.5-1.6 = 8.9
```

NOTE:

- If our inputs are integers then output of floor division(//) will always returns integer.
- On the other hand, if our inputs are float then floor division and division will always returns float value. i.e. output of data type of division doesnot depends on the data type of inputs.
- Floor Division : we always get immediate lower as an output.

In [154]:

```
# arthemetic opeartor * is only accepatable
s= 'aditi'
t = 5
print(f'{s}+{t} =',s+t)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-154-3bb31576e06c> in <module>
      2 s= 'aditi'
      3 t = 5
----> 4 print(f'{s}+{t} =',s+t)

TypeError: can only concatenate str (not "int") to str
```

In [155]:

```
print(f'{s}*{t} =',s*t)
```

```
aditi*5 = aditiaditiaditiaditiaditi
```

In [156]:

```
print(f'{s}/{t} =',s/t)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-156-86c51538c9e8> in <module>
----> 1 print(f'{s}/{t} =',s/t)

TypeError: unsupported operand type(s) for /: 'str' and 'int'
```

In [157]:

```
print(f'{s}//{t} =',s//t)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-157-e036f0eab873> in <module>
----> 1 print(f'{s}//{t} =',s//t)

TypeError: unsupported operand type(s) for //: 'str' and 'int'
```

In [158]:

```
a= 'Naman'
b= 'Thakur'
print(f'{a}+{b} =',a+b)
```

```
Naman+Thakur = NamanThakur
```

In [159]:

```
print(f'{a}*{b} =',a*b)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-159-82396a07337f> in <module>
----> 1 print(f'{a}*{b} =',a*b)

TypeError: can't multiply sequence by non-int of type 'str'
```

In [160]:

```
print(f'{a}/{b} =',a/b)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-160-751ab265d39f> in <module>
----> 1 print(f'{a}/{b} =',a/b)

TypeError: unsupported operand type(s) for /: 'str' and 'str'
```

In [161]:

```
print(f'{a}**{b} =',a**b)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-161-ad46becd8a32> in <module>
----> 1 print(f'{a}**{b} =',a**b)

TypeError: unsupported operand type(s) for ** or pow(): 'str' and 'str'
```

In [162]:

```
a1 = 2
b1 =0
print(a1/b1)
```

```
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
<ipython-input-162-0c82463d86d7> in <module>
      1 a1 = 2
      2 b1 =0
----> 3 print(a1/b1)

ZeroDivisionError: division by zero
```

Assignment Operator

- To assign any value to variable

In [163]:

```
x= 10
```

In [164]:

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

```
20
```

In [165]:

```
# shift operator
print(10>>2)
print(2<<4)
```

```
2
32
```

Special Oprator

1. Identity operator

- identity operator is a operator made for address or reference comparison

In [167]:

```
a= 10
b=10
```

In [168]:

```
print(id(a))
print(id(b))
```

```
140709926742080
140709926742080
```

In [169]:

```
print(a is b) # address is same so true
```

```
True
```

In [170]:

```
print(a!=b)
```

```
False
```

In [171]:

```
c= 20
print(id(c))
```

```
140709926742400
```

In [173]:

```
a1=[10,20,30]
b1 =[10,20,30]
print (id(a1))
print(id(b1))
```

```
2180744908096
2180710441792
```

In [174]:

```
print(a1 is b1)
```

```
False
```

2. Membership function:

- In membership operator we are comparing elements

In [175]:

```
a= [20,60,2,50]
print(2 in a)
```

```
True
```

In [176]:

```
print(200 in a)
```

```
False
```

input statement

- The input() function allows user input.

In [178]:

```
a= input('Enter a value:')
```

```
Enter a value:5
```

In [179]:

```
type (a)
```

Out[179]:

```
str
```

In [180]:

```
a1= int(a)
print(type(a1))
a2=float(a)
print(type(a2))
```

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

In [181]:

```
a=  eval(input('Enter a value'))
```

```
Enter a value1.5
```

In [182]:

```
type(a)
```

Out[182]:

```
float
```

In [183]:

```
b=  eval(input('Enter a value: '))
```

```
Enter a value: 51
```

In [185]:

```
type(b)
```

Out[185]:

```
int
```

In [187]:

```
c= eval(input('Enter a value: '))
type(c)
```

```
Enter a value: 'aditya'
```

Out[187]:

```
str
```

In [189]:

```
c1= eval(input('Enter a value: '))
```

```
Enter a value: {1:'aditya',2:'sonu','c':'cat'}
```

In [190]:

```
type(c1)
```

Out[190]:

```
dict
```

MATH MODULE

- MATH MODEULE IS A GROUP OF PACKAGES
- EVERY PACKAGES SEVERAL MODULE OR SUB PACKAGES ALSO
- FORM IMPLEMENTING MATH MODULE FIRST IMPORT MATH

In [191]:

```
import math
```

In [192]:

```
print(math.sqrt(144))
```

```
12.0
```

In [193]:

```
math.sin(45)
```

Out[193]:

```
0.8509035245341184
```

In [194]:

```
print(math.pi)
```

```
3.141592653589793
```

In [195]:

```
print(math.ceil(10.6))
print(math.floor(10.6))
```

```
11
10
```

Output Statement

- We use print() statement to get output

In [196]:

```
print('Hello world')
print('Hello\tworld')
print('hello\nworld')
```

```
Hello world
Hello	world
hello
world
```

In [197]:

```
print('hello','world')
```

```
hello world
```

In [198]:

```
print('Hello' ,end ='_')
```

```
Hello_
```

In [199]:

```
print ('Hello ', end ='Aditya :')
print('Good Morning')
```

```
Hello Aditya :Good Morning
```

Formatted string

1. %i -- Use for int()
2. %d -- Use for int()
3. %f -- Use for float()
4. %s -- Use for str()

In [201]:

```
a,b,c =1,2,3
print('Value of a is :' ,a)
print('Value of a is :' ,b)
print('Value of a is :' ,c)
```

```
Value of a is : 1
Value of a is : 2
Value of a is : 3
```

In [202]:

```
print('Value of a is :' ,a,'Value of a is :' ,b,'Value of a is :' ,c)

print('Value of a is %i and Value of b is %i and Value of c is %d'%(a,b,c))
print('Value of a is %i and Value of b is %i and Value of c is %d'%(b,a,c))
```

```
Value of a is : 1 Value of a is : 2 Value of a is : 3
Value of a is 1 and Value of b is 2 and Value of c is 3
Value of a is 2 and Value of b is 1 and Value of c is 3
```

In [203]:

```
s= 'aditya'
f=10.2
print('Value of s is %s and value of f is %f'%(s,f))
```

```
Value of s is aditya and value of f is 10.200000
```

In [204]:

```

l=[10,20,56,48,76,59,'aditya']
print('List is %s'%(l))
print(type(l))
```

```
List is [10, 20, 56, 48, 76, 59, 'aditya']
<class 'list'>
```

In [205]:

```
# code for greater of two number

x= input("Enter the value of x :")
y= input ("Enter the value of y :")
if x>y:
    print('x is greater than y')
else:
    print('y is greater than x')
```

```
Enter the value of x :5
Enter the value of y :6
y is greater than x
```

Control Statement

- There are three types:
    1. Conditional statement
    2. Iterative Statement
    3. Transfer Statement

Note :

- In Python there is no concept of do-while statement. There is only concept of while loop and for loop.

In [208]:

```
name = input('Enter your name')
if name == 'Aditya':
    print(f'Hello Welcome to Python World: {name}')
else:
    print('You answer is Incorrect')
```

```
Enter your nameAditya
Hello Welcome to Python World: Aditya
```

In [209]:

```
# code for greater of three number
x= int(input("Enter the value of x :"))
y= int(input ("Enter the value of y :"))
z= int(input ("Enter the value of z :"))
if x>y and x>z:
    print('x is greater than y and z')
elif y>x and y>z :
    print('y is greater than x and z')
else:
    print('z is greater')
```

```
Enter the value of x :52
Enter the value of y :12
Enter the value of z :8
x is greater than y and z
```

1. code for input is even number

In [210]:

```
x= int(input("Enter the value of x :"))
if x%2 ==0:
    print('x is an even number')
else:
    print('x is an odd number')
```

```
Enter the value of x :50
x is an even number
```

2. Area of Circle

In [211]:

```
import math
r= int(input("Enter the value of radius:"))
area_of_circle = (math.pi )* r** 2
print ("Area of circle is :",area_of_circle )
```

```
Enter the value of radius:5
Area of circle is : 78.53981633974483
```

3. Area of Triangle

In [212]:

```
a1= eval(input("Enter value of side a1:"))
b1= eval(input("Enter value of side b1:"))
c1= eval(input("Enter value of side c1:"))
s= (a1+b1+c1)/2
print(s)
area_triangle = (s*(s-a1)*(s-b1)*(s-c1))**0.5
print("Area of Triangle is %f :"%area_triangle)
```

```
Enter value of side a1:5
Enter value of side b1:6
Enter value of side c1:7
9.0
Area of Triangle is 14.696938 :
```

4. Swap two number

In [215]:

```
a= eval(input("Enter  no 'a' : " ))
print('Value of "a" before swap: ',a)
b= eval(input("Enter no 'b':"))
print('Value of "b" before swap: ',b)
#  1st method:
# a,b = b,a # Using purely Python

#  2nd method: Uisng third variable
c=a
a=b
b=c

#  3rd method:

# a= a+b
# b= a-b
# a= a-b

print("Result: ")
print('Value of "a" after swap: ',a)
print('Value of "b" after swap: ',b)
```

```
Enter  no 'a' : 45
Value of "a" before swap:  45
Enter no 'b':79
Value of "b" before swap:  79
Result:
Value of "a" after swap:  79
Value of "b" after swap:  45
```

5. To find square root of a number

In [216]:

```
import math
x= eval(input("Enter value of x :"))
square_root = math.sqrt(x)
print(square_root)
```

```
Enter value of x :225
15.0
```

6. Code to find roots of quadratic equation

In [218]:

```
import math
import cmath # for complex number
a= eval(input("Enter value of a :"))
b= eval(input("Enter value of b :"))
c= eval(input("Enter value of c :"))
d = (b**2 -4*a*c)
print(d)
if d>0 and d==0:
    x1= (-b+math.sqrt(d))/(2*a)
    print('First Roots of quadratic equation is :',x1)
    x2 = (-b - math.sqrt(d)) / (2 * a)
    print('Second Roots of quadratic equation is :',x2)
else:
    x1 = (-b + cmath.sqrt(d)) / (2 * a)
    print('First Roots of quadratic equation is :', x1)
    x2 = (-b - cmath.sqrt(d)) / (2 * a)
    print('Second Roots of quadratic equation is :', x2)
```

```
Enter value of a :5
Enter value of b :45
Enter value of c :47
1085
First Roots of quadratic equation is : (-1.2060661815998792+0j)
Second Roots of quadratic equation is : (-7.79393381840012+0j)
```

In [219]:

```
# WAP TO CHECK LEAP YEAR
year= int(input("ENTER YOUR YEAR :"))
if year%4 == 0:
    print(f'{year} is a leap year')
else:
    print(f'{year} is not a leap year')
```

```
ENTER YOUR YEAR :2020
2020 is a leap year
```

In [220]:

```
# WAP TO CHECK THE NUMBER IS POSITIVE, NEGATIVE OR ZERO NUMBER
number= eval(input("Enter Your number :"))
if number>0:
    print(f'{number} is a positive number')
elif number<0:
    print(f'{number} is a negative number')
else:
    print(f'{number} is a zero')
```

```
Enter Your number :50
50 is a positive number
```

In [221]:

```
# WAP TO CHECK Number is a  PRIME NUMBER
number= int(input("Enter Your number :"))
count =0
for i in range(1,number+1):
    if number%i == 0:
        count += 1
if (count == 2):
    print(f'{number} is a prime number')
else:
    print(f'{number} is not a prime number')
```

```
Enter Your number :89
89 is a prime number
```

In [222]:

```
# WAP  for factorial OF A NUMBER
# Using for-loop

number= int(input("Enter Your number :"))
fact =1
if number ==0:
    print(f'Factorial of {number}! is ',fact)
elif number ==1:
    print(f'Factorial of {number}! is ',number)
elif number <0:
    print('Factorial of neagtive number does not exixts.')
else:
    for i in range(1,number+1):
        fact =fact*i
    print(f'Factorial of a {number}! is :',fact)
```

```
Enter Your number :5
Factorial of a 5! is : 120
```

While loop

- While loop is an iterative statement and this will execute as long as a condition is true

In [223]:

```
# using while loop
number= int(input("Enter Your number :"))
fact =1
if number ==0:
    print(f'Factorial of {number}! is ',fact)
elif number ==1:
    print(f'Factorial of {number}! is ',number)
elif number <0:
    print('Factorial of neagtive number does not exixts.')
else:
    while number>1:
        fact = number*fact
        number -= 1
    print(f'Factorial of a {number}! is :', fact)
```

```
Enter Your number :8
Factorial of a 1! is : 40320
```

In [224]:

```
# addition of 10 number using while loop
n= int(input("Enter n :"))
i=1
sum =0
while i<=n:
    sum = sum + i
    i +=1
print('sum of n number is ',sum)
```

```
Enter n :5
sum of n number is  15
```

NESTED LOOP

In [225]:

```
for i in range(3):
    for j in range(3):
        print('hello')
```

```
hello
hello
hello
hello
hello
hello
hello
hello
hello
```

In [227]:

```
# Pattern  1:
#
# * * * * *
# * * * * *
# * * * * *
# * * * * *
# Code
# print('* * * *\n* * * *\n* * * *\n* * * *')
n= int(input('Enter the number of star: '))
for j in range(n):
    print('* '*n)
print('\n')
```

```
Enter the number of star: 5
* * * * *
* * * * *
* * * * *
* * * * *
* * * * *
```

In [228]:

```
# Pattern  2:
# 1 1 1 1 1
# 2 2 2 2 2
# 3 3 3 3 3
# 4 4 4 4 4
# 5 5 5 5 5

# By me
# n= int(input('Enter the number: '))
# for j in range(1,n+1):
#     print(f'{j} '*n)
# print('\n')
# # or

# By SIR:
n=int(input("enter the number"))
for i in range(1,n+1):
    print((str(i))*n)
```

```
enter the number5
11111
22222
33333
44444
55555
```

In [229]:

```
# Pattern  3:
# a a a a a
# b b b b b
# c c c c c
# d d d d d
# e e e e e

n=int(input("enter the number"))
for i in range(1,n+1):
    print((chr(96+i)+' ')*n)
```

```
enter the number4
a a a a
b b b b
c c c c
d d d d
```

In [230]:

```
# Pattern  4:
# *
# * *
# * * *
# * * * *
# * * * * *
# code :
# print('*    \n* *  \n* * * \n' )

n=int(input("enter the number "))
for i in range(n+1):
    print('* '*i)
```

```
enter the number 5

*
* *
* * *
* * * *
* * * * *
```

In [231]:

```
# Pattern  5:
# 1
# 2 2
# 3 3 3
# 4 4 4 4
# 5 5 5 5 5

rows = int(input("enter the rows"))
for num in range(1,rows+1):
    for i in range(num):
        print(num, end=" ") # print number
        # line after each row to display pattern correctly
    print(" ")
```

```
enter the rows5
1
2 2
3 3 3
4 4 4 4
5 5 5 5 5
```

In [234]:

```
# Pattern 6:
#  1 2 3 4 5
#  1 2 3 4
#  1 2 3
#  1 2
#  1
n = int(input("enter the number: "))
for i in range(n+1,0,-1):
    for j in range(1, i):
        print(" "+f'{j}',end='')
    print(' '*(n-i))
```

```
enter the number: 6
 1 2 3 4 5 6
 1 2 3 4 5
 1 2 3 4
 1 2 3
 1 2
 1
```

In [235]:

```
# Pattern 7
# 12345
#   1234
#    123
#     12
#      1
for i in range(1,n+1):
    print(" "*(i-1),end=' ')
    for j in range(1,n+2-i):
        print(f"{j}",end="")
    # print(" ", end='')
    print()
```

```
123456
 12345
  1234
   123
    12
     1
```

Functions in Python

- It is a block of code
- WRITE ONE TIME USE multiple times
- def keyword is used to define function

Types of function:

1. `In-built function:` The function which is already defined by Python. Ex: id(),type(),length(),sum(),eval(),print()..etc
2. `User-defined function:`
3. The function which is created by Users according to user requirements.
4. In this we can use parameter or argument also.

Argument :

- Information passed on the function depend on the requirement.
- Argument are values of the parameter.

`Parameter:`

- Parameters are specified after the function name i.e inside the parenthesis '( )' of the function.We can as many parameters.
- Parameter are separated with a comma(,).

Four types of parameter:

- Positional parameters:
- Keyword parameters:
- Default parameter:
- Variable length parameter:

Note:

- There is slightly or logical difference between argument and parameter
- Parameter is a variable which is declared inside the function
- Arguments are the value passed by the user while calling the function

In [238]:

```
# `Example:`
def wish(name):    # name is parameter
    print("Good Morning: ",name)
# wish()
wish('Aditya Kumar') # Aditya kumar is a argument
```

```
Good Morning:  Aditya Kumar
```

1. Positional parameters: argument passed to a function at correct position

In [239]:

```
# `Example:`
def wish(name):    # name is parameter
    print("Good Morning: ",name)
wish()
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-239-cf89d37d89f0> in <module>
      2 def wish(name):    # name is parameter
      3     print("Good Morning: ",name)
----> 4 wish()

TypeError: wish() missing 1 required positional argument: 'name'
```

2. Keyword arguments :

- Related to function call.
- Once we use keyword argument in a function call,the caller or user identify the arguments by the parameter name

In [240]:

```
# Example:
def wish(name):    # name is parameter
    print("Good Morning: ",name)
# wish()
wish(name ='Aditya Kumar') # Aditya kumar is a argument
```

```
Good Morning:  Aditya Kumar
```

In [241]:

```
def logicoutput(arg1, *arg2):
    print('output is',arg1)

    print('output 2 is: ', arg2)
logicoutput(10)
logicoutput(10,20,'asas','sas')
```

```
output is 10
output 2 is:  ()
output is 10
output 2 is:  (20, 'asas', 'sas')
```

Variable :

- It is an object which is created by the user onces it required
- all variable in the program may not be accesssable at all the time
- Variable depends on where we declared a variable

Types of Variable:

1. `Local Variable:` A variable which is in the limitation of accessing is known as local variable.

2. `Global Variable:` A variable which we can accessed entire the program is known as global variable.

In [242]:

```
total =0 # global variable
def total(arg1,arg2):
    total_inner =arg1+arg2
    print('Local variable output: ',total_inner)
    return total_inner
total(30,50)
print('Global variable : ',total)
print('Global variable : ',total_inner)
```

```
Local variable output:  80
Global variable :  <function total at 0x000001FBBE764670>

---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-242-0b822897e7ba> in <module>
      6 total(30,50)
      7 print('Global variable : ',total)
----> 8 print('Global variable : ',total_inner)

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

Python Module:

- A module allows us to logically organised our python code
- we can say that module is a file consists of python code
- A file containing a set of functions which we want to include in our application

In [243]:

```
import factorial
factorial.fact(5)
```

Out[243]:

```
120
```

In [244]:

```
# Rename a module name
import factorial  as fac
fac.fact(8)
```

Out[244]:

```
40320
```

File input and output

- in python a file takes place different different operation
- `1. OPEN A FILE :` To Open a file, We will use `open()` function.In the oprn() command we use `r` for read a file and `w` for write in the file
- `2. READ /WRITE A FILE:` For read a file we use read() function amd for write a file we use write() function
- `3. APPEND A FILE` : we use mode 'a+' to append a file into current file.
- `4. CLOSE THE FILE:` for closing a file we use close() function

In [262]:

```
f=open("file1.txt", "r")
contents = f.read()
print(contents)
```

```
This is line 1

This is line 2

This is line 3

This is line 4

This is line 5

This is line 6

This is line 7

This is line 8

This is line 9

This is line 10
```

In [253]:

```
f=open("file1.txt")
if f.mode == 'r':
    contents = f.read()
    print(contents)
else:
    print("Your mode is not read")
```

```
Open the file in Read mode.
This is Aditya kumar
HELLO EVERYONE !!!
HOW are YOU!!
```

In [268]:

```
f= open("file1.txt","a+")
for i in range(3):
    f.write("Append line %d\r\n" % (i+1))
# f.close()
f= open("file1.txt")
contents = f.read()
print(contents)
f.close()
```

```
This is line 1

This is line 2

This is line 3

This is line 4

This is line 5

This is line 6

This is line 7

This is line 8

This is line 9

This is line 10
```

rename() method:

In the python we will remain our existing file name with other file name.

Import os

os is module to play witht the files in the computer

In [270]:

```
import os
os.rename('file.txt','edubridge.txt')
```

remove() method:

In the python we will remove our existing file by usning remove() method by supply ing the name of the file to be deleted as arguments.

In [272]:

```
os.remove('edubridge.txt')
```

Directory in python

- It is nothing but the locations or the folder where we are working.

1. getcwd() method :

By this method we are finding out directory location where we are currently working. get means getting , cwd means current working directory

2. mkdir() method :

It is used for creating new folder in the current existing directory. mk means make , dir means directory

3. rmdir() method :

It is used for removing or delete the directory which is passed as an arguments. rm means remove , dir means directory

In [273]:

```
os.getcwd()
```

Out[273]:

```
'C:\\Users\\ADITYA KUMAR\\EdubridgeDA'
```

In [274]:

```
os.mkdir('test')
```

In [275]:

```
os.rmdir('test')
```

In [277]:

```
import sys # import system
def how():
    try:
        f= open('file21.txt','r')
    except IOError as e :
        print(e)
        print (sys.exec_prefix)

how()
```

```
[Errno 2] No such file or directory: 'file21.txt'
C:\Users\ADITYA KUMAR\anaconda3
```

- First Install all python libraries on your system, then we will import the module
- Use command `!pip install module_name` to install any module

In [278]:

```
import math
# x= eval(input("enter a "))
math.sqrt(-20)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-278-13a1d6d5f185> in <module>
      1 import math
      2 # x= eval(input("enter a "))
----> 3 math.sqrt(-20)

ValueError: math domain error
```

Error and Exception Handling:

- Error is the outcome which we will get after the execution of the code. but the outcome is not exceptable according to our reqirement.
- `Exception:` Anexception is an error that happen during execution of a program. When the error occurs python generate an exception that can be handled which avoids our program to crash

why we use Exception:

- Exception are convenient in many ways for handling errors and special condition in a program.
- When we think that we have a code which can produce an error then we can use Exceptional Handling.

Types of Exception Error:

1. `io Error:` io Error is input-output Error. This error raised only when if the file cannot be opeed.
2. `import Error:` import Error can be possible only when if we are unable to import module. i.e. if python cannot find imported module.
3. `Value Error:` it is raised when a built-in operation or function receives an argument that has the right type but in inappropriate value.
4. `keyboard interrupt Error:` This type of error raised when the user/coder hits the interrupted key.

5. `eof Error:` Eof atand for End of File Error. It raised when one of the built-in function hits an EOF condition without reading any data.

In [279]:

```
import math

x = int(input('Please enter a positive number:\n'))

try:
    print(f'Square Root of {x} is {math.sqrt(x)}')
except ValueError as ve:
    print(f'You entered {x}, which is not a positive number.')
```

```
Please enter a positive number:
52
Square Root of 52 is 7.211102550927978
```

In [280]:

```
#Keyboard error: it is caused by the user once try to interrupt the executed code
#import signal
#import sys
import time
try:
    while True:
        print("Heavy task!")
        time.sleep(4)
except KeyboardInterrupt:
    print("KeyboardInterrupt has been caught.")
```

```
Heavy task!
Heavy task!
Heavy task!
Heavy task!
Heavy task!
Heavy task!
Heavy task!
Heavy task!
Heavy task!
Heavy task!
Heavy task!
Heavy task!
Heavy task!
Heavy task!
Heavy task!
KeyboardInterrupt has been caught.
```

In [283]:

```
# End of file : it is one of the exception handling errors and it is raised in senarios suc
# of the input function in python

#EOFError program
#try and except blocks are used to catch the exception
# try:
#     while True:
# #input is assigned to a string variable check
#         check = input('The input provided by the user is being read which is:')
# #the data assigned to the string variable is read
#     print ('READ:', check)
# #EOFError exception is caught and the appropriate message is displayed
# except EOFError as x:
#     print (x)
```

In [284]:

```
#try and except blocks are used to catch the exception
try:
    data = input ("Do you want to continue?: ")
except EOFError:
    print ("Error: No input or End Of File is reached!")
data = ""
print (data)
```

```
Do you want to continue?: 54
```

OOPS(Object Oriented Programming):

- Python is object oriented programming language. Major principle of OOPS:

1. `Object :`

- It is an real world entity.
- It is a collection of data ,variable and method.
- It has state and behaviour it many be anything either physical or logcal.
- Every things in python is an object and almost every object has attributes and methods.

2. `Class :`

- It is the blueprint of the object,like a structure or prototype.
- it is defined as a collections of object
- It is the Logical entity that some has specific attributes and method.

`For example:` If we have employee class then eid ,ename ,salary,age are the attribute or method

In [285]:

```
class employee:
    empcount =0
    def __init__(self,name,salary):
        self.name = name
        self.salary = salary
        employee.empcount += 1
    def display(self):
        print('Total number of employee is : ',employee.empcount)

    def displayemployee(self):
        print('Employee Name is: ',self.name)
        print('Salaey of Employee is: ',self.salary)
s1 = employee('Aditya',5000)
s2 = employee('Ranjan Sinha',6000)

s1.displayemployee()
s2.displayemployee()
s1.display()
```

```
Employee Name is:  Aditya
Salaey of Employee is:  5000
Employee Name is:  Ranjan Sinha
Salaey of Employee is:  6000
Total number of employee is :  2
```

`__init__()`: It is a special method of the Python which called as class instructor or initialization method that Python calls when we create a new instance of this class.

`self` : It represent the instance o the class .By using self keyword we can access the attributes and method of the class in Python.

In [286]:

```
class student:
    def __init__(self,rollno,name):
        self.rollno=rollno
        self.name=name
    def displaystudent(self):
        print("rollno:", self.rollno, "name",self.name )
emp1=student(121,"A")
emp2=student(122,"B")
emp3=student(123,"c")
emp1.displaystudent()
emp2.displaystudent()
emp3.displaystudent()
```

```
rollno: 121 name A
rollno: 122 name B
rollno: 123 name c
```

Built-in class attributes:

- Every Python class keep following built-in atributes and they can be accessed using . opearators like any other attributes
- __dict__ : This is the dictionary contains the classes name space.
- __doc__ : This is the class document string or none if undefined.
- __name__ : This is for class name.
- __module__ : Module name in which the class is defined.
- __bases__ : A possible empty tuple containing the base classes in the order of there occurence in base class list .

In [287]:

```
print(student.__dict__)
```

```
{'__module__': '__main__', '__init__': <function student.__init__ at 0x00000
1FBBF222C10>, 'displaystudent': <function student.displaystudent at 0x000001
FBBF222B80>, '__dict__': <attribute '__dict__' of 'student' objects>, '__wea
kref__': <attribute '__weakref__' of 'student' objects>, '__doc__': None}
```

In [288]:

```
print ("Employee.__doc__:", employee.__doc__)
print ("Employee.__name__:", student.__name__)
print ("Employee.__module__:", employee.__module__)
print ("Employee.__bases__:", employee.__bases__)
print ("Employee.__dict__:", employee.__dict__)
```

```
Employee.__doc__: None
Employee.__name__: student
Employee.__module__: __main__
Employee.__bases__: (<class 'object'>,)
Employee.__dict__: {'__module__': '__main__', 'empcount': 2, '__init__': <fu
nction employee.__init__ at 0x000001FBBC869C10>, 'display': <function employ
ee.display at 0x000001FBBF222040>, 'displayemployee': <function employee.dis
playemployee at 0x000001FBBF2220D0>, '__dict__': <attribute '__dict__' of 'e
mployee' objects>, '__weakref__': <attribute '__weakref__' of 'employee' obj
ects>, '__doc__': None}
```

In [289]:

```
print ("student.__doc__:", employee.__doc__)
print ("student.__name__:", student.__name__)
print ("student.__module__:", employee.__module__)
print ("student.__bases__:", employee.__bases__)
print ("student.__dict__:", employee.__dict__)
```

```
student.__doc__: None
student.__name__: student
student.__module__: __main__
student.__bases__: (<class 'object'>,)
student.__dict__: {'__module__': '__main__', 'empcount': 2, '__init__': <fun
ction employee.__init__ at 0x000001FBBC869C10>, 'display': <function employe
e.display at 0x000001FBBF222040>, 'displayemployee': <function employee.disp
layemployee at 0x000001FBBF2220D0>, '__dict__': <attribute '__dict__' of 'em
ployee' objects>, '__weakref__': <attribute '__weakref__' of 'employee' obje
cts>, '__doc__': None}
```

3. Garbage Collection :

- Python deletes unneeded object automatically to free the memory space.The process by which Python periodically reclaims block of memory that is no longer are in used is known as `Garbage Collection` .
- In simple words, garbage collection means distroying objects.
- Python garbage collector runs during program execution.

In [290]:

```
a = 10 # Creation of object
b = a  # increasing reference count of 10
c = [b] # increasing reference count of 10

del(a) # deleting  'a' object ,decreasing reference count of 10
b = 500
c[0] = 90
```

In [291]:

```
class point:
    def __init__(self,x=0,y=0):
        self.x=x
        self.y=y
    def __del__(self):
        class_name=self.__class__.__name__
        print (class_name,"destoryed")

p1=point()
p2=p1
p3=p2
print( id(p1), id (p2), id (p3))
del (p1)
del (p2)
del (p3)
```

```
2180751205232 2180751205232 2180751205232
point destoryed
```

Difference between function and method

- Function , Free (Free means not belong to an object or class)
- Method , Member (A member of an object or class)
- method is related to objects but not functions
- functions that are created inside the class are called method

4. Constructor :

- It is a special type of method or a function which is used to initialize the instance member of the Class.

Creating a constructor:

- A constructor is a class function or method with double underscore.
- The name of constructor is always the same as **init**().
- while creating an constructor can accept a arguments if necessary.
- When we create class Python automatically create a default constructor that doesnot do anything.

- Every class must have a constructor even if it simply realize on the default construtor.

Types of constructor:

1. Paramaterized constructor :

- we will pass parameter in this constructor

2. Non Paramaterized constructor :

- we will not pass parameter in this constructor

In [292]:

```
# parmerterized constructor

class student:
    def __init__(self, name):
        print("parmerterized constructor")
        self.name=name
    def show (self):
        print("Hello !!", self.name)

student=student('Ranjan')
student.show()
```

```
parmerterized constructor
Hello !! Ranjan
```

In [293]:

```
# non_parmertrized constructor

class student:
    def __init__(self):
        print("non_parmertrized constructor")
    def show (self, name):
        print("Hello !!", name)

student=student()
student.show('Ranjan')
```

```
non_parmertrized constructor
Hello !! Ranjan
```

5. Inheritance :

- Inheritance is a feature of `OOPs` .
- It specifies that one object aquire all the properties and behaviour of the parent object.
- By using inheritance, we can define a new class with a liitle change or no change to the existing class.
- The new class is known as `derived class` or `child class` .
- The existing class is known as `base class` or `parent class`

In [294]:

```
class animal:     # Parent class
    def eat(self):
        print("Animals eat ")

class dog(animal):   # Child class, We are inheriting 'eat' property of animal by dog class
    def bark(self):
        print("Dog is barking")

d = dog()
d.eat()
d.bark()
```

```
Animals eat
Dog is barking
```

Types of inheritance:

1. Single inheritance
2. Multiple inheritance
3. Multi- level Inheritance
4. Hierarichal Inheritance
5. Hybrid inheritance

1. Single inheritance :

- Single inheritance is the simplest type of inheritance in which it enable derived class to inherit properties from a single parent class.

In [295]:

```
# example of single inheritance
class parent:
    def fun1(self):
        print("This is the parent class")

class child(parent):
    def fun2(self):
        print("This is the Child class")
o = child()
o.fun1()
o.fun2()
```

```
This is the parent class
This is the Child class
```

2. Multiple inheritance :

- When a class can be derived from more than one base class is known as multiple inheritance.

In [297]:

```
class mother:
    mother_name = ''
    def mother(self):
        print(self.mother_name)

class father:
    father_name =''
    def father(self):
        print(self.father_name)

class child(mother,father):  # child inherit both mother and father class
    def parent(self):
        print("Father name is : ",self.father_name)
        print("Mother  name is :",self.mother_name)

c = child()
c.father_name = 'Your Fathers Name'
c.mother_name = 'Yours Mother Name'
c.parent()
```

```
Father name is :  Your Fathers Name
Mother  name is : Yours Mother Name
```

3. Multilevel inheritance :

- In multilevel inheritance, there are many intermediate or levelled class will be present.
- `Example:` grandparent, parents and grandson is the best example of multilevel inheritance

In [298]:

```
# multi level inhertance
class animal:
    def eat(self):
        print("animal eats")
class dog(animal):
    def bark(self):
        print ('Dog is barking')
class pug(dog):
    def weep(self):
        print('pug is weeping')
d=pug()
d.eat()
d.bark()
d.weep()
```

```
animal eats
Dog is barking
pug is weeping
```

4. Hierarichal Inheritance :

- In hierarichal inehritance there will be more than one child or derived class of the parent class.
- Example: sibling with parents

In [299]:

```
# Hierarichal inhertance
# Base class
class Parent:
    def func1(self):
        print("This function is in parent class.")

# Derived class1
class Child1(Parent):
    def func2(self):
        print("This function is in child 1.")

# Derivied class2
class Child2(Parent):
    def func3(self):
        print("This function is in child 2.")

# Driver's code
object1 = Child1()
object2 = Child2()
object1.func1()
object1.func2()
object2.func1()
object2.func3()
```

```
This function is in parent class.
This function is in child 1.
This function is in parent class.
This function is in child 2.
```

5. Hybrid inheritance:

- Hybrid inheritance consists of multiple type of inheritance i.e combination of diffeent different inheritance

In [300]:

```
class School:
    def func1(self):
        print("This function is in school.")

class Student1(School):
    def func2(self):
        print("This function is in student 1. ")

class Student2(School):
    def func3(self):
        print("This function is in student 2.")

class Student3(Student1,School):
    def func4(self):
        print("This function is in student 3.")

# Driver code
object = Student3()
object.func1()
object.func2()
```

```
This function is in school.
This function is in student 1.
```

Polymorphism:

- `poly` means 'Many' + `morphism` means 'forms or shape'
- it is defined that one task can be performed in different ways

In [1]:

```
print(len('ashwini'))
print(len([1,2,5.,6,78]))
```

```
7
5
```

In [3]:

```
def add(a,b,c=0):
    return a+b+c
print(add(2,3))
print(add(5,1,2))
print(add(5,65,2))
```

```
5
8
72
```

In [5]:

```
# polymorphism
class hindi:
    def analytics(self):
        print("Aditya")
class french:
    def analytics(self):
        print("Kumar")
def intro(language):
    language.analytics()

f=hindi()
a=french()

intro(f)
intro(a)
```

```
Aditya
Kumar
```

Encapsulation :

- It is one of the fundamental concepts of OOPS.
- It describe the idea of wrapping data and methods that work on data within one unit.
- It is used to restrict access to method and variables.
- Code and data are wrapped and make one single unit.

In [6]:

```
class car:
    maxspeed=0
    name=''
    def __init__(self):
        self.maxspeed=250
        self.name="marcedise"
    def drive(self):
        print("driving with maximum speed"+ str(self.maxspeed))

r=car()
r.drive()
r.maxspeed=500
r.drive()
```

```
driving with maximum speed250
driving with maximum speed500
```

Data Abstraction :

- Data abstraction and encapsulation are synonyms.
- It is used to hide internal details and show only required functionalities.
- Abstract something means to give names to things, so that the names captures basic idea of what afunction or a whole program does

- only showing the require things but not showing the internal functionalities how it works.
- `Data abstraction = Data encapsulation + Data hiding`

- `Real life Example :` Car contain many things like engine ,battery,silencer,etc but we use only handle ,gyers, break .All other things are hided from the users.

In [7]:

```
#Data Abstraction
class roboot:
    def __init__(self, name=None):
        self.name=name
    def say_hi(self):
        if self.name:
            print("Hello !!"+ self.name)
        else:
            print("Hii !!")

    def set_name(self, name):
        self.name=name
    def get_name(self):
        return self.name
x=roboot()
x.set_name("Ranjan")
x.say_hi()
y=roboot()
y.set_name(x.get_name())
print(y.get_name())
```

```
Hello !!Ranjan
Ranjan
```

Multi threading:

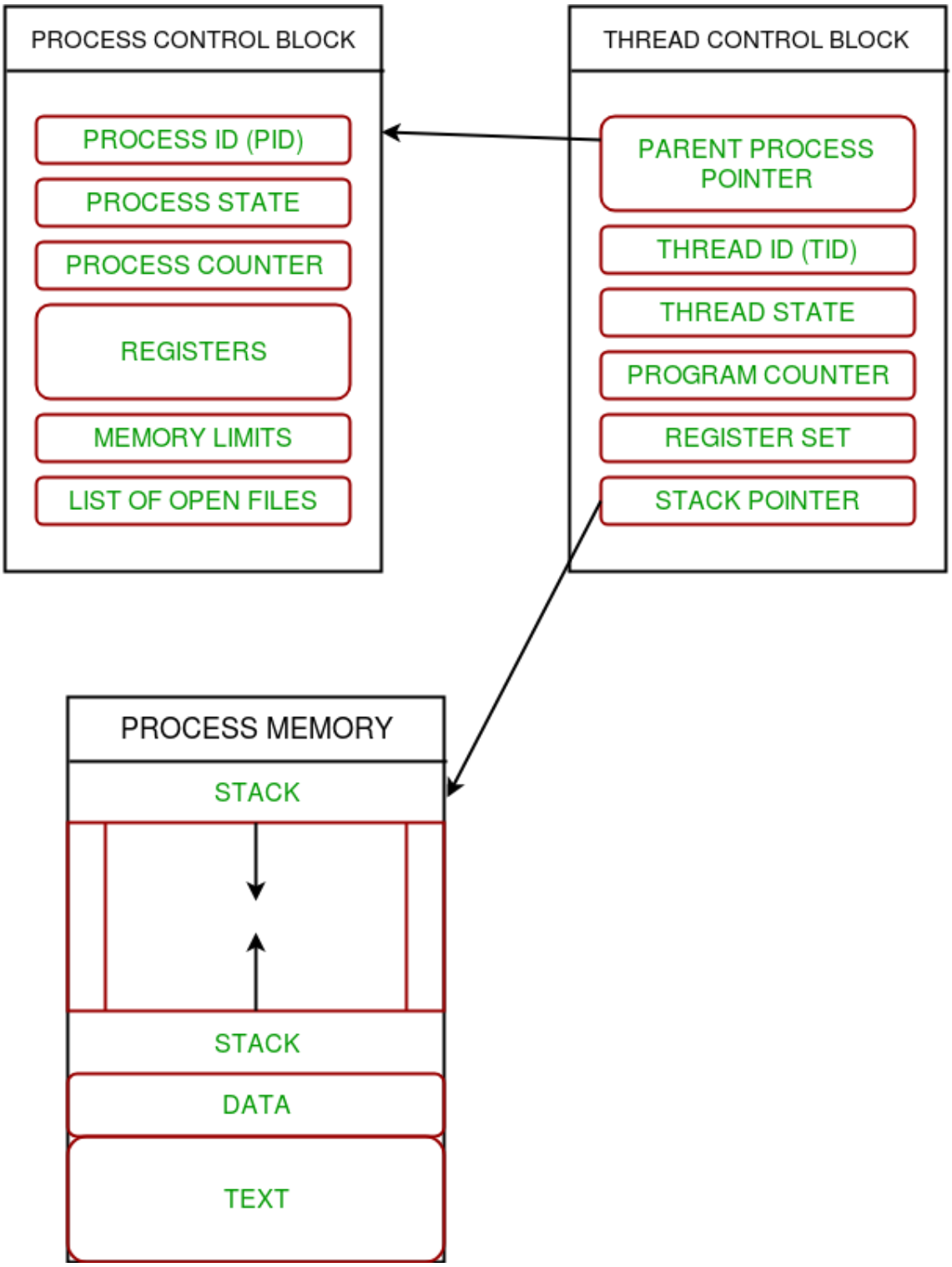
- `Thread:`

- It is an entity within a process can be scheduled for execution.
- Also it is the Smallest unit of processing that can be performed in an OS.
- In simple word, a thread is a sequence of instructions within a program that can be executed independently of the other code
- Subset of a process.
- A thread contains all the information in a `Thread Control Block` .

In [14]:

```
from IPython.display import Image
Image(filename='multithreading-python-11.png')
```

Out[14]:



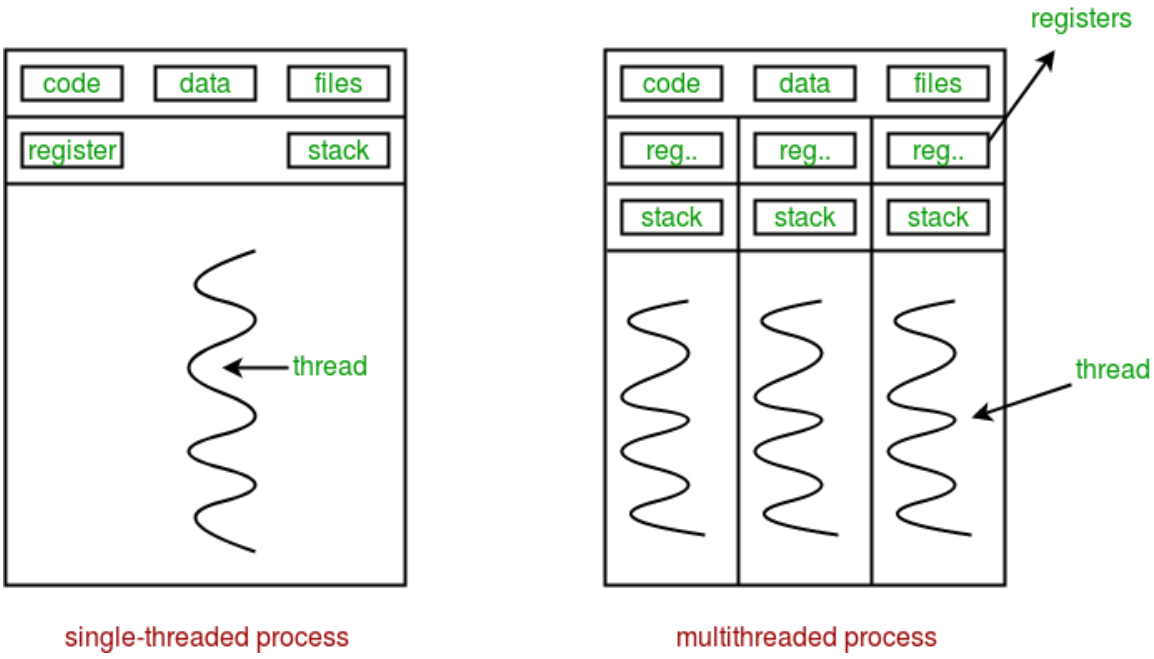
Multi-thread can exist within one process where:

1. Each thread conatins it's own registerset and local variable.
2. All threads of a process share global variables and the program code

In [13]:

```
from IPython.display import Image
Image(filename='multithreading-python-21.png')
# multithreading-python-11.png
```

Out[13]:



single-threaded process multithreaded process

In [15]:

```
# multithreading
import threading
def print_cube(num):
    """
    function to print cube of given number
    """
    print("Cube: {}".format(num*num*num))
def print_square(num):
    """
    function to print square of given number
    """
    print("Square: {}".format(num*num))

if __name__=="__main__":
    t1=threading.Thread(target=print_square, args=(10,))
    t2=threading.Thread(target=print_cube, args=(10,))

    t1.start()
    t2.start()

    #wait untill the thread complete
    t1.join()
    t2.join()

    print("Done")
```

```
Square: 100
Cube: 1000
Done
```

In [ ]:

```
## Assignment:
#   Application of thread classes: run() ,start() ,join() ,isAlive(), getName(),setName()
```

Method Overloading:

- Method ovreloading nothing but a polymorphism at the time of compile our code
- In this, more than one method ofthr same class share the same method name having different signature
- method overloading is used to add more behaviour of the method

Note:

- Python does not supports method overloading.

In [16]:

```
def add(a,b):
    return a+b
def add(a,b,c):
    return a+b+c
add(2,3)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-16-5e28ee79f065> in <module>
      3 def add(a,b,c):
      4     return a+b+c
----> 5 add(2,3)

TypeError: add() missing 1 required positional argument: 'c'
```

In [17]:

```
# First product method.
# Takes two argument and print their
# product
def product(a, b):
    p = a * b
    print(p)

# Second product method
# Takes three argument and print their
# product
def product(a, b, c):
    p = a * b*c
    print(p)

# Uncommenting the below line shows an error
# product(4, 5) gives error

# This line will call the second product method
product(4, 5, 5)
```

```
100
```

Method Overriding:

- Method overriding is an example of runtime polymorphism.
- In this, Specific implementation of the method that is already provided by parent the class is provided by the child class
- Inheritance always reqiured between parent and child class in the method overriding

In [18]:

```
class A:
    def sayhi(self):
        print("I am in class A")
class B(A):
    def sayhi(self):
        print("I am in class B")
b= B()
b.sayhi()
```

```
I am in class B
```

In [21]:

```
class A:
    def fun1(self):
        print('feature_1 of class A')

    def fun2(self):
        print('feature_2 of class A')

# Modified function that is
# already exist in class A
class B(A):
    def fun1(self):
        print('Modified feature_1 of class A by class B')
    def fun3(self):
        print('feature_3 of class B')

# Create instance
obj = B()
# Call the override function
obj.fun1()
```

```
Modified feature_1 of class A by class B
```

In [ ]:

```

```