In [ ]:
```
1.procedural style programming
2.functional style programming
3. Object oriented style of programming
```

In [ ]:
```
i=10   # var=value     #i-> var or namespace pointing to object of <class 'int '>   10-> object available in hea
```

In [1]:
```
id(10)
```

Out[1]:
```
140714377458888
```

In [2]:
```
id(7+3)
```

Out[2]:
```
140714377458888
```

In [3]:
```
a=10
b=5+5
print(id(a), id(b))
```

```
140714377458888 140714377458888
```

In [ ]:
```
var=10
|
__main__.var
```

In [ ]:
```
connect()--> def connect(..)   {
                 class ..:
                   def f1 # method
                    var   # properties
                        # members
            }
```

In [ ]:
```
var=10 # int
L=[1,2,3,4]  # list
####################
print(var,L,L[2])  # direct access
```

class cname: var=100 L=[1,2,3,4] # class attributes print(var,L)# Error

In [ ]:
```
#class, object, Inheritance
```

In [ ]:
```
class- blueprint of object
  +-------------------------------------+
  |                                     |
  |                                     |  bluprint sheet- class
  +-------------------------------------+
    |        |                          |
    |        |                          |
    B1      B2                          B3   Building - object
   sqft                                        ==> address(memory)
```

In [ ]:
```
class cname:      - type
        |
       not start with digit ; not allow space,special char
```

In [7]:
```
class student:
    sname=''
    sid=0
    sdept=''

# syntax:-    objectname=classname()

s1=student() # object creation
s1.sname='arun'
s1.sid=1001
s1.sdept='sales'

s1=student() # object creation
s2.sname='Sam'
s2.sid=1002
s2.sdept='sales'

s3=student() # object creation
s3.sname='anand'
s3.sid=1020
s3.sdept='Prod'

print("Name:{}\tUSN:{}\tDept:{}".format(s1.sname,s1.sid,s1.sdept))
print("Name:{}\tUSN:{}\tDept:{}".format(s2.sname,s2.sid,s2.sdept))
print("Name:{}\tUSN:{}\tDept:{}".format(s3.sname,s3.sid,s3.sdept))
```

```
Name:arun       USN:1001        Dept:sales
Name:Sam        USN:1002        Dept:sales
Name:anand      USN:1020        Dept:Prod
```

In [9]:
```
print("Name:{}\tUSN:{}\tDept:{}".format(s1.SNAME,s1.sid,s1.sdept)) # AttributeError
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[9], line 1
----> 1 print("Name:{}\tUSN:{}\tDept:{}".format(s1.SNAME,s1.sid,s1.sdept))

AttributeError: 'student' object has no attribute 'SNAME'
```

In [5]:
```
sname=''
sid=0

print(sname,sid)   # direct acces
```

```
 0
```

In [8]:
```
var=0
print(VAR) # NameError
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[8], line 2
      1 var=0
----> 2 print(VAR)

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

In [ ]:
```
function---> functioncall()
method-----> object.functioncall()---> methodcall
```

In [10]:
```
def f1(a):
    print(a,type(a))

f1(10)
f1(1.333)
f1('')
f1({})
f1(True)

def f2():
    print("hello")

f2()
f2("hello")# TypeError
```

```
10 <class 'int'>
1.333 <class 'float'>
 <class 'str'>
{} <class 'dict'>
True <class 'bool'>
hello
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[10], line 14
     11     print("hello")
     13 f2()
---> 14 f2("hello")

TypeError: f2() takes 0 positional arguments but 1 was given
```

In [18]:
```
class box:
    var=100
    def f2(self,*a1,**a2):
        print("hello",self)
        print(a1,a2)
        print(self.var)

# box-class
#var,f2--- attributes
obj=box()  # obj creation
obj.var   # 100
obj.f2()  # method call ==> object.function()---> function(object)  ->  obj.f2()---> f2(obj)
print(obj)
obj.f2(1,2,3,4,5,name="theeba",port=8080)

obj1=box()
obj1.f2()
```

```
hello <__main__.box object at 0x0000028558E09BE0>
() {}
<__main__.box object at 0x0000028558E09BE0>
hello <__main__.box object at 0x0000028558E09BE0>
(1, 2, 3, 4, 5) {'name': 'theeba', 'port': 8080}
hello <__main__.box object at 0x0000028558F807D0>
() {}
```

In [21]:

```
obj=box()
print(obj)

obj1=box()
print(obj1)

print(box)  # class hold small memory but other lang class holds no mem
```

```
<__main__.box object at 0x0000028559020290>
<__main__.box object at 0x0000028559938160>
<class '__main__.box'>
```

In [27]:

```
class box:
    var=100

boxobj=box()
print(boxobj.var)
```

```
100
```

In [28]:

```
# classname.attribute
print(box.var)
```

```
100
```

In [29]:

```
box.var=2000   # Modify classname.attribute
print(box.var)   # classname.attr
print(boxobj.var)   #obj.attr
```

```
2000
2000
```

In [34]:

```
boxobj1=box()
boxobj2=box()
boxobj3=box()
print(box.var)
print(boxobj1.var)
print(boxobj2.var)
print(boxobj3.var)

# change classname.var
box.var="data"

print(box.var)
boxobj1.var=100   # Object Initialization
print(boxobj1.var)
print(boxobj2.var)
print(boxobj3.var)

# change classname.var
box.var=True
print(boxobj1.var)
```

```
data
data
data
data
data
100
data
data
100
```

In [36]:

```
class box:
    __port=8080  # __variablename    (or)   def __methodname() - user defined private attributes

print(box.__port)
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[36], line 4
      1 class box:
      2     __port=8080  # __variablename    (or)   def __methodname() - user defined private attributes
----> 4 print(box.__port)

AttributeError: type object 'box' has no attribute '__port'
```

In [37]:

```
obj=box()
```

```
print(obj.__port)
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[37], line 2
      1 obj=box()
----> 2 print(obj.__port)

AttributeError: 'box' object has no attribute '__port'
```

In [38]:
```
class box:
    __port=8080
    def f1(self):
        print(self.__port)
    def f2(self,a1):
        self.__port = a1

obj=box()
obj.f1()
obj.f2(5000)
obj.f1()
```

```
8080
5000
```

In [40]:
```
class student:
    __sname=''
    __sid=0
    __sdept=''

    def f1(self,a1,a2,a3):
        self.__sname=a1
        self.__sid=a2
        self.__sdept=a3
        print("Initialization Done")

    def f2(self):
        print("Name:{}\tUSN:{}\tdept:{}".format(self.__sname,self.__sid,self.__sdept))

s1=student()
s1.f1("Arun",101,"Maths")   #   f1(s1,"Arun",101,"Maths")

s2=student()
s2.f1("Vijay",102,"Physics")
```

```
Initialization Done
Initialization Done
```

In [41]:
```
L=[]
help(list.append)
```

```
Help on method_descriptor:

append(self, object, /) unbound builtins.list method
    Append object to the end of the list.
```

In [ ]:
```
L.append("D1")   # append(L,"D1")
class str:
    ...
    def upper(self):
        ..
        return UPPERCSE

s="abc"
s.upper()---> upper(s)
```

In [ ]:
```
DB--> connection -> query
     // fails        -> we can't query

class DBI:
    def f1(self,dn..):
        connection
    def f2(self,..):
        query
    def f3(self..):
        ....

obj=DBI()
obj.f3()
obj.f2()
obj.f1() // No error - Programming
// Logically - Error
```

```
In [ ]: constructor
        ----------
        |
        special method => called during object creation   =>  def __init__()      Vs Package -> __init__.py -> this is 
                                                                --------------
                                                                    Method
```

```
In [43]: class box:
             def __init__(self):
                 print("Initialization")
             def f1(self):
                 print("f1 block")

         obj=box()   # object creation---> constructorcall
         obj.f1()  # methodcall f1
```

```
Initialization
f1 block
```

```
In [46]: class box:
             def __init__(self,a1,a2):
                 self.__sname=a1
                 self.__sip=a2
                 print("Initialization")

             def f2(self):
                 return self.__sname, self.__sip

         #S=box() # object creation
         S=box('Linux','10.20.30.40') # Object Creation and Initialization through constructor
         S.f2()
```

```
Initialization
Out[46]: ('Linux', '10.20.30.40')
```

```
In [47]: S1=box('Winx','192.168.1.2')
         S1.f2()
```

```
Initialization
Out[47]: ('Winx', '192.168.1.2')
```

```
In [ ]: Task
        =====
        Modify the student class with
         --->constructor for object initialization
         ---> display method to print values
         ---> dept_update method to change the dept value
```

```
In [51]: class student:
             __sname=''
             __sid=0
             __sdept=''

             def __init__(self,a1,a2,a3):
                 self.__sname=a1
                 self.__sid=a2
                 self.__sdept=a3
                 print("Initialization Done")

             def f2(self):
                 print("Name:{}\tUSN:{}\tdept:{}".format(self.__sname,self.__sid,self.__sdept))
             def f3(self,a1):
                 self.__sdept=a1
                 print("Updation Done")
         s1=student("Arun",101,"Maths")

         s2=student("Vijay",102,"Physics")

         s1.f2()
         s2.f2()

         s1.f3("Commerce")
         s1.f2()
         s2.f3("Science")
         s2.f2()
```

```
Initialization Done
Initialization Done
Name:Arun       USN:101 dept:Maths
Name:Vijay      USN:102 dept:Physics
Updation Done
Name:Arun       USN:101 dept:Commerce
Updation Done
Name:Vijay      USN:102 dept:Science
```

In [55]:
```
class box:
    __no=101
    dept='sales'
# print(box.__no)  # Error
print(box._box__no)  #   _classname__var  =>  __var
print(box.dept)
```

```
101
```

In [56]:
```
#class, object, method, constructors
# Inheritance
#------------
class A:
    service='apache2'
class B(A):  # inheritance
    port=8080

obj=B()
print(obj.port, obj.service)
```

```
8080 apache2
```

In [ ]:
```
file:AB.py              file: SAB.py              file: p1.py
----------               -----------              ------------
class Box:              import AB                  import SAB
    def f1(self)        class Fax(AB.Box):         class myclass(SAB.Fax):
    def f2(self)           def f3(self)                ....
                           def f4(self)            obj=myclass()
                                                    obj.f1()
                                                    obj.f2()
                                                    obj.f3()
                                                    obj.f4()


file : p2.py
------------
from SAB import Fax
class myclass(Fax)
      ...

obj=myclass()
```

In [57]:
```
class A:
    def f1(self):
        print("Class-A f1 block")
class B(A):
    def f2(self):
        print("Class-B f2 Block")

obj=B()
obj.f2()
obj.f1()
```

```
Class-B f2 Block
Class-A f1 block
```

In [58]:
```
class A:
    def f1(self):
        print("Class-A f1 block")
class B(A):
    def f1(self):
        print("Class-B f1 Block")

obj=B()
obj.f1() # Child's method defn   # method overriding - Runtime polymorphism
```

```
Class-B f1 Block
```

In [62]:
```
#Polymorphism- one in many form

print(len("python"))
print(len([12,3,34,9]))
print(len((1,2,3,4)))
```

```
6
4
4
```

```
In [66]: class A:
             def f1(self):
                 return 'f1-A'
         class B:
             def f1(self):
                 return 'f1-B'

         obj=[A(),B()]
         for var in obj:
             print(var.f1())  # 1 in many forms
```

```
f1-A
f1-B
```

```
In [69]: class A:
             def f1(self):
                 print("Class-A f1 block")
         class B(A):
             def f1(self):
                 print("Class-B f1 Block")
                 super().f1()   # call to Parent class Method
                 A.f1(self)

         obj=B()
         obj.f1() # Child's method defn   # method overriding - Runtime polymorphism
```

```
Class-B f1 Block
Class-A f1 block
Class-A f1 block
```

```
In [72]: class A:
             def f1(self,*a):
                 print("Class-A f1 block",a)
         class B(A):
             def f1(self):
                 print("Class-B f1 Block")
                 super().f1(10,20)   # call to Parent class Method
                 # A.f1(self) # call to Parent class Mthod- classname.methodname

         obj=B()
         obj.f1() # Child's method defn   # method overriding - Runtime polymorphism
```

```
Class-B f1 Block
Class-A f1 block (10, 20)
```

```
In [ ]: print(10+5)  #   int.__add__(10,5)
        print("s1"+"s2")   # str.__add__(s1,s2)
```

```
In [75]: help(object.__str__)
```

```
Help on wrapper_descriptor:

__str__(self, /) unbound builtins.object method
    Return str(self).
```

```
In [76]: help(str.__str__)
```

```
Help on wrapper_descriptor:

__str__(self, /) unbound builtins.str method
    Return str(self).
```

```
In [77]: '''
         class classname:
             attributes- methods & var
             '''

         dir(A)   # dir(classname)---> attributes
```

```
Out[77]: ['__class__',
          '__delattr__',
          '__dict__',
          '__dir__',
          '__doc__',
          '__eq__',
          '__firstlineno__',
          '__format__',
          '__ge__',
          '__getattribute__',
          '__getstate__',
          '__gt__',
          '__hash__',
          '__init__',
          '__init_subclass__',
          '__le__',
          '__lt__',
          '__module__',
          '__ne__',
          '__new__',
          '__reduce__',
          '__reduce_ex__',
          '__repr__',
          '__setattr__',
          '__sizeof__',
          '__static_attributes__',
          '__str__',
          '__subclasshook__',
          '__weakref__',
          'f1']
```

```
In [78]: obj=A()
         dir(obj)  # dir(object)--> Attributes of class
```

```
Out[78]: ['__class__',
          '__delattr__',
          '__dict__',
          '__dir__',
          '__doc__',
          '__eq__',
          '__firstlineno__',
          '__format__',
          '__ge__',
          '__getattribute__',
          '__getstate__',
          '__gt__',
          '__hash__',
          '__init__',
          '__init_subclass__',
          '__le__',
          '__lt__',
          '__module__',
          '__ne__',
          '__new__',
          '__reduce__',
          '__reduce_ex__',
          '__repr__',
          '__setattr__',
          '__sizeof__',
          '__static_attributes__',
          '__str__',
          '__subclasshook__',
          '__weakref__',
          'f1']
```

```
In [ ]: len([1,2,3]) ==> 3 --> __len__()
```

```
In [80]: len([1,2,3])
         dir([])
```

```
Out[80]: ['__add__',
          '__class__',
          '__class_getitem__',
          '__contains__',
          '__delattr__',
          '__delitem__',
          '__dir__',
          '__doc__',
          '__eq__',
          '__format__',
          '__ge__',
          '__getattribute__',
          '__getitem__',
          '__getstate__',
          '__gt__',
          '__hash__',
          '__iadd__',
          '__imul__',
          '__init__',
          '__init_subclass__',
          '__iter__',
          '__le__',
          '__len__',
          '__lt__',
          '__mul__',
          '__ne__',
          '__new__',
          '__reduce__',
          '__reduce_ex__',
          '__repr__',
          '__reversed__',
          '__rmul__',
          '__setattr__',
          '__setitem__',
          '__sizeof__',
          '__str__',
          '__subclasshook__',
          'append',
          'clear',
          'copy',
          'count',
          'extend',
          'index',
          'insert',
          'pop',
          'remove',
          'reverse',
          'sort']
```

```
In [81]: class box:
             var=100


         obj=box()
         len(obj)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[81], line 5
      2     var=100
      4 obj=box()
----> 5 len(obj)

TypeError: object of type 'box' has no len()
```

```
In [82]: dir(box)
```

```
Out[82]: ['__class__',
          '__delattr__',
          '__dict__',
          '__dir__',
          '__doc__',
          '__eq__',
          '__firstlineno__',
          '__format__',
          '__ge__',
          '__getattribute__',
          '__getstate__',
          '__gt__',
          '__hash__',
          '__init__',
          '__init_subclass__',
          '__le__',
          '__lt__',
          '__module__',
          '__ne__',
          '__new__',
          '__reduce__',
          '__reduce_ex__',
          '__repr__',
          '__setattr__',
          '__sizeof__',
          '__static_attributes__',
          '__str__',
          '__subclasshook__',
          '__weakref__',
          'var']
```

```
In [83]: class box:
             var=100
             def __len__(self):
                 return 10

         obj=box()
         len(obj) # return   # len()---> __len__()
```

```
Out[83]: 10
```

```
In [84]: a=10
         b=20
         a+b    __add__
```

```
Out[84]: 30
```

```
In [85]: class A:
             def __init__(self,a):
                 self.a=a
         obj=A("hello")
```

```
In [86]: str(obj)
```

```
Out[86]: '<__main__.A object at 0x0000028558E0A900>'
```

```
In [87]: dir(A)
```

Out[87]:
```
['__class__',
 '__delattr__',
 '__dict__',
 '__dir__',
 '__doc__',
 '__eq__',
 '__firstlineno__',
 '__format__',
 '__ge__',
 '__getattribute__',
 '__getstate__',
 '__gt__',
 '__hash__',
 '__init__',
 '__init_subclass__',
 '__le__',
 '__lt__',
 '__module__',
 '__ne__',
 '__new__',
 '__reduce__',
 '__reduce_ex__',
 '__repr__',
 '__setattr__',
 '__sizeof__',
 '__static_attributes__',
 '__str__',
 '__subclasshook__',
 '__weakref__']
```

In [88]:
```
class A:
    def __init__(self,a):
        self.a=a
    def __str__(self):
        return self.a

obj=A("hello")
```

In [89]:
```
str(obj)
```

Out[89]:
```
'hello'
```

In [90]:
```
dir(obj)
```

Out[90]:
```
['__class__',
 '__delattr__',
 '__dict__',
 '__dir__',
 '__doc__',
 '__eq__',
 '__firstlineno__',
 '__format__',
 '__ge__',
 '__getattribute__',
 '__getstate__',
 '__gt__',
 '__hash__',
 '__init__',
 '__init_subclass__',
 '__le__',
 '__lt__',
 '__module__',
 '__ne__',
 '__new__',
 '__reduce__',
 '__reduce_ex__',
 '__repr__',
 '__setattr__',
 '__sizeof__',
 '__static_attributes__',
 '__str__',
 '__subclasshook__',
 '__weakref__',
 'a']
```

In [91]:
```
class box:
    pass
obj=box()
```

In [92]:
```
def f1():
    pass
f1()
```

```
In [93]: callable(obj)  # bool-> False -> is not callable

Out[93]: False

In [94]: callable(f1)   # bool-> True -> can be called

Out[94]: True

In [96]: class box:
             def __call__(self):
                 return "Hello"
         obj=box()
         callable(obj)

Out[96]: True

In [97]: obj()

Out[97]: 'Hello'

In [98]: dir(box)

Out[98]: ['__call__',
          '__class__',
          '__delattr__',
          '__dict__',
          '__dir__',
          '__doc__',
          '__eq__',
          '__firstlineno__',
          '__format__',
          '__ge__',
          '__getattribute__',
          '__getstate__',
          '__gt__',
          '__hash__',
          '__init__',
          '__init_subclass__',
          '__le__',
          '__lt__',
          '__module__',
          '__ne__',
          '__new__',
          '__reduce__',
          '__reduce_ex__',
          '__repr__',
          '__setattr__',
          '__sizeof__',
          '__static_attributes__',
          '__str__',
          '__subclasshook__',
          '__weakref__']

In [99]: del(L) # --> __del__

In [100… class box:
             def __init__(self):
                 print("Object Initialization")
             def __del__(self):
                 print("Thankyou")
         obj=box()

       Object Initialization

In [101… del(obj)

       Thankyou

In [102… class box:
             __fname="saturn"
             __fno=6

In [103… box.__fname

       ---------------------------------------------------------------------------
       AttributeError                            Traceback (most recent call last)
       Cell In[103], line 1
       ----> 1 box.__fname

       AttributeError: type object 'box' has no attribute '__fname'

In [104… box.__dict__   # hold all class attributes including private attr in dict format
```

```
Out[104… mappingproxy({'__module__': '__main__',
                            '__firstlineno__': 1,
                            '_box__fname': 'saturn',
                            '_box__fno': 6,
                            '__static_attributes__': (),
                            '__dict__': <attribute '__dict__' of 'box' objects>,
                            '__weakref__': <attribute '__weakref__' of 'box' objects>,
                            '__doc__': None})
```

```
In [105… dir([])
```

```
Out[105… ['__add__',
           '__class__',
           '__class_getitem__',
           '__contains__',
           '__delattr__',
           '__delitem__',
           '__dir__',
           '__doc__',
           '__eq__',
           '__format__',
           '__ge__',
           '__getattribute__',
           '__getitem__',
           '__getstate__',
           '__gt__',
           '__hash__',
           '__iadd__',
           '__imul__',
           '__init__',
           '__init_subclass__',
           '__iter__',
           '__le__',
           '__len__',
           '__lt__',
           '__mul__',
           '__ne__',
           '__new__',
           '__reduce__',
           '__reduce_ex__',
           '__repr__',
           '__reversed__',
           '__rmul__',
           '__setattr__',
           '__setitem__',
           '__sizeof__',
           '__str__',
           '__subclasshook__',
           'append',
           'clear',
           'copy',
           'count',
           'extend',
           'index',
           'insert',
           'pop',
           'remove',
           'reverse',
           'sort']
```

class has __iter__() ----> object -> iterable object

```
In [ ]: Iterator--> an obj that produces value one at a time.  -> __next__()
```

```
In [107… L=[1,2,3,4]
         it= iter(L)
         for i in it:
             print(i)
```

```
1
2
3
4
```

```
In [109… it= iter(L)
         print(next(it))
         print(next(it))
```

```
1
2
```

```
In [114… class Count:
             def __init__(self,start):
                 self.no=start
```

```
    def __iter__(self):
        return self

    def __next__(self):
        if self.no <=0:
            raise StopIteration
        current = self.no
        self.no-= 1
        return current


obj=Count(5)   # iterable object
for i in obj:   # way 1: Automatically traverse through all elemnts in iterable obj
    print(i)
```

```
5
4
3
2
1
```

In [115…
```
obj=Count(3)     # way2: Manually traverse through all elemnts in iterable obj
print(next(obj))
print(next(obj))
print(next(obj))
```

```
3
2
1
```

In [111…
```
for i in range(5):
    print(i)
```

```
0
1
2
3
4
```

In [ ]:
```
Generator
---------
---> Generator is a special function that returns an iterator

Named Function                          VS                    Generator
===============                                                ===============
def f1(a):                                                       def f1(a):
    if a> 10:                                                       while a> 0:
        return True  # exit from fn          vs                           yield a   # pauses here & returns valu
    else:                                                                 a-=1
        return (1,4)
```

In [116…
```
def count(a):
    while a>0:
        yield a
        a-=1
```

In [118…
```
gen=count(5)   # generator--> returns iterator
gen
```

Out[118…
```
<generator object count at 0x0000028559784400>
```

In [119…
```
print(next(gen))   # Manual way to traverse iterable obj
print(next(gen))
print(next(gen))
print(next(gen))
print(next(gen))
```

```
5
4
3
2
1
```

In [120…
```
gen=count(3)
for i in gen:     # Automatically traverse through iterable obj elements
    print(i)
```

```
3
2
1
```

In [122…
```
def fx():
    yield "v1"
    yield "v2","v3"
    yield "v4",['D1','D2','D3']
    print("hello")
    print("hai")
    yield 10+3.14
    yield ("Hello")

obj=fx()
print(next(obj))
print(next(obj))
print(next(obj))
print(next(obj))
print(next(obj))
```

```
v1
('v2', 'v3')
('v4', ['D1', 'D2', 'D3'])
hello
hai
13.14
Hello
```

In [123…

```
obj1=fx()
for i in obj1:
    print(i)
```

```
v1
('v2', 'v3')
('v4', ['D1', 'D2', 'D3'])
hello
hai
13.14
Hello
```

In [ ]:

```
Decorator
=========
|----- metaprogramming - adding more features to your code
|-----  functionname(function)()    (or) @function
```

In [126…

```
def f1():
    def f2():
        def app1():
            print("appl-1")
        def app2():
            print("appln-2")
        app1()
        app2()
    return f2

f1()
```

Out[126…

```
<function __main__.f1.<locals>.f2()>
```

In [127…

```
f1()()
```

```
appl-1
appln-2
```

In [135…

```
def f1(a):          # decorator    a=fx
    def f2():           # Wrapper function
        def app1():
            print("appl-1")
        def app2():
            print("appln-2")
        app1()
        app2()
        a()
    return f2
```

In [136…

```
def fx():
    print("App-3 -Updated ")

r=f1(fx)
r()
```

```
appl-1
appln-2
App-3 -Updated
```

In [ ]:

```
def f1(a):         # Decorator
    def f2():       # Wrapper
        ...
        a()
    return f2
```

In [142…
```
def news(a):
    def fwrapper():
        a()
    return fwrapper
@news
def city1():
    print("City-1")

@news
def city2():
    print("City-2")

def city3():
    print("city-3")

def city4():
    print("city-4")
city1()    # news(city1)()
city2()
city3()   # direct call

r=news(city4)
r()
# or news(city4)()
```

```
City-1
City-2
city-3
city-4
```

In [ ]:
```
Home           AboutUs            News               ContactUs
                                 |----City1
                                 |----City2



@news
def city1():
    ...
@news
def city2():
    ...
city1()
city2()
```

In [ ]:
```
@classmethod
@staticmethod
```

In [143…
```
class cname:
    def method1(self):  # self-> obj invoked method1
        print("Object based method")
obj=cname()
obj.method1()
```

```
Object based method
```

In [144…
```
cname.method1()   # method1(cname)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[144], line 1
----> 1 cname.method1()

TypeError: cname.method1() missing 1 required positional argument: 'self'
```

In [145…
```
cname
```

Out[145…
```
__main__.cname
```

In [146…
```
obj
```

Out[146…
```
<__main__.cname at 0x28557f2fb60>
```

In [147…
```
class cname:
    @classmethod
    def f1(cls):    # classname
        print("This is class based method")

cname.f1()   # f1(cname)
```

```
This is class based method
```

In [ ]:
```
@staticmethod--> helper method inside class - 1st arg is not cls (or) self
@classmethod -> class based ---- 1st arg should be cls
```

```
@property -> object based ---- ist arg should be self
@staticmethod                            vs     @classmethod         vs        @property
def f1(a1):                                     def f1(cls):                   def f1(self):
    a1+=200                                         cls.var                         self.var
    return a1
```

In [ ]:
```
Thread
======
 - concurrent execution
 - light weight process - require separate memory

Global Intrepretor Lock - GIL --> disable in cpython - thread will not actually run in parallel.

Task---CPU intensive work
     |--IO intensive work
```

In [ ]:
```
#  Thread creation --> import threading--> Thread
# Thread Synchronization ->
```

In [ ]:
```
xml
===
xml.etree.ElementTree
```

In [148…
```
import xml.etree.ElementTree as et

xml_data="""<bookstore>
<book category="programming">
    <title> Basics Python </title>
    <author> Guido Von roosum </author>
</book>
<book category="ML">
    <title> Python with ML</title>
    <author> Smith </author>
</book>
</bookstore>"""
```

In [150…
```
r=et.fromstring(xml_data)
for book in r.findall("book"):
    category=book.get("category")   # attribute
    title=book.find("title").text
    author=book.find("author").text

    print(f"{category} - {title}   by {author}")
```

```
programming -  Basics Python    by  Guido Von roosum
ML -  Python with ML   by  Smith
```

In [ ]:
```
convert xml to dict--->  xmltodict module
>>> xml_data="""<bookstore>
... <book category="programming">
...     <title> Basics Python </title>
...     <author> Guido Von roosum </author>
... </book>
... <book category="ML">
...     <title> Python with ML</title>
...     <author> Smith </author>
... </book>
... </bookstore>"""
>>> type(xml_data)
<class 'str'>
>>>

>>> import xmltodict
>>> data=xmltodict.parse(xml_data)
>>> type(data)
<class 'dict'>
>>> data
{'bookstore': {'book': [{'@category': 'programming', 'title': 'Basics Python', 'author': 'Guido Von roosum'}, {
```