In [2]:
```
class Product1:
    pname='ITEM-A'
    pcost=1000

class Product2:
    ptax=0.13
    pvendor='ABC'

obj1=Product1()
print(obj1.pname)
print(obj1.pcost)
obj2=Product2()
print(obj2.ptax)
print(obj2.pvendor)
```

```
ITEM-A
1000
0.13
ABC
```

In [7]:
```
# class childname(Parent_classname):  vs  def function_name(argument):
class Product2(Product1): # inheritance
    ptax=0.13
    pvendor='ABC'

obj=Product2()
print(obj.pname)
print(obj.pcost)
print(obj.pcost*obj.ptax)
```

```
ITEM-A
1000
130.0
```

In [5]:
```
class Box:          #   class Box(object):
    var=100         #       var=100
obj=Box()           #
obj.var
```

Out[5]:
```
100
```

In [ ]:
```
# File: A.py
# ---------------
# class Enrollment:
#      def f1(self):
#      def f2(self):
#      def f3(self):

# File:B.py
# -----------
# import A
# class Student(A.Enrollment):
#      def f4(self):
#

# File:p1.py
# -------------
# import B
# obj=B.Student()
# obj.f1()
# obj.f2()
# obj.f3()
# obj.f4()
```

In [11]:
```
class P1:
    def f1(self):
        print("==>F1 block from P1 class")
class P2(P1):
    def f1(self):
        print("F1 block from P2 class")
        P1.f1(self) # calling parent method
        super(P2,self).f1() #  calling parent method

obj=P2()
obj.f1()
```

```
F1 block from P2 class
==>F1 block from P1 class
==>F1 block from P1 class
```

In [16]:
```
class P1:
    def f1(self):
        print("P1-class")
class P2(P1):
    def f1(self):
        print("P2-class")
        #super(P2,self).f1()
class P3(P2):
    def f1(self):
        print("P3-class")
        super(P3,self).f1()
        #P1.f1(self) # calling P1 class method

obj=P3()
obj.f1()
```

```
P3-class
P2-class
```

In [19]:
```
a=10
isinstance(a,int)
L=[10,20]
T=(100,200)
isinstance(L,tuple)
```

Out[19]:
```
False
```

In [21]:
```
class A:                                    # [  Class A]     [ Class B]
    def f1(self):                           #  ----------------------
        print("Class-A")                    #       |_____________|
class B:                                    #              |
    pass                                    #           [Class C]
class C(A,B): # multiple inheritance
    pass

obj=C()
#obj.f1()
print(C.mro())
```

```
[<class '__main__.C'>, <class '__main__.A'>, <class '__main__.B'>, <class 'obje
ct'>]
```

In [23]:
```
class A:                                    # [  Class A]     [ Class B]
    def f1(self):                           #  ----------------------
        print("Class-A")                    #       |_____________|
class B:                                    #              |
    def f1(self):
        print("Class-B")                    #           [Class C]
class C(A,B): # multiple inheritance
    pass

obj=C()
obj.f1()
obj.f1()
print(C.mro())
```

```
Class-A
Class-A
[<class '__main__.C'>, <class '__main__.A'>, <class '__main__.B'>, <class 'obje
ct'>]
```

In [24]:
```
class A:                                    # [  Class A]     [ Class B]
    #def f1(self):                          #  ----------------------
        #print("Class-A")
    pass                                    #       |_____________|
class B:                                    #              |
    def f1(self):
        print("Class-B")                    #           [Class C]
class C(A,B): # multiple inheritance
    pass

obj=C()
obj.f1()
obj.f1()
print(C.mro())
```

```
Class-B
Class-B
[<class '__main__.C'>, <class '__main__.A'>, <class '__main__.B'>, <class 'obje
ct'>]
```

In [25]:
```
class A:
    def f1(self):
        print('class-A')
class B:
    def f1(self):
        print('Class-B')

class C(A,B):
    def f1(self):
        print('class-C')

class D(C,B):
    pass
obj=D()
D.mro()
```

Out[25]:
```
[__main__.D, __main__.C, __main__.A, __main__.B, object]
```

In [ ]:
```
# Decorator - function design - metaprogramming
#   |___classmethod,staticmethod
#             |         |
#          decorator methods ->  @functionname
```

In [28]:
```
# app1 app2 app3
def f1(): # decoarator function
    def f2(): # wrapper function
        def f3():
            print("App-1")
        def f4():
            print("App-2")
        def f5():
            print("App-3")
        f3()
        f4()
        f5()
    return f2
rv=f1()
rv()
```

```
App-1
App-2
App-3
```

In [29]:
```
def f1(a1): # decoarator function
    def f2(): # wrapper function
        def f3():
            print("App-1")
        def f4():
            print("App-2")
        def f5():
            print("App-3")
        f3()
        f4()
        f5()
        a1()
    return f2


def f6():
    print("App-4")

rv=f1(f6)
rv()
```

```
App-1
App-2
App-3
App-4
```

In [30]:
```
def f1(a1): # decoarator function
    def f2(): # wrapper function
        def f3():
            print("App-1")
        def f4():
            print("App-2")
        def f5():
            print("App-3")
        f3()
        f4()
        f5()
        a1()
    return f2

@f1
def f6():
    print("App-4")
f6()
```

```
App-1
App-2
App-3
App-4
```

In [43]:
```
def f1(a1):
    def f2():
        a1()
    return f2

@f1
def fx():
    print("Fx-operation")
@f1
def fy():
    print("Fy-Operation")

@f1
def fz():
    print("Fz-operation")

fx() # rv=f1(fx) ->rv()
fz() # rv=f1(fy) ->rv()
```

```
Fx-operation
Fz-operation
```

In [ ]:
```
class Box:
    def f1(self):
        print("Instancemethod")
obj=Box()
obj.f1() # f1(obj)
```

In [34]:
```
class Box:
    @classmethod
    def f1(cls):
        print("classmethod")
        print(cls) # __main__.Box

Box.f1() # f1(Box)
obj=Box()
obj.f1() # f1(<classname>)
```

```
classmethod
<class '__main__.Box'>
classmethod
<class '__main__.Box'>
```

In [37]:
```
class Fsinfo:
    server='default'
Fsinfo.server='Unix'
Fsinfo.port=21323
print(Fsinfo.server,Fsinfo.port)
```

```
Unix 21323
```

In [39]:
```
class Fsinfo:
    server='default'
    @classmethod
    def f1(cls,a1):
        print(cls.server)
        cls.server=a1

print(Fsinfo.server)
Fsinfo.f1("unix") # f1(Fsinfo,"unix")
print(Fsinfo.server)

obj=Fsinfo()
obj.server
```

```
default
default
unix
```

Out[39]:
```
'unix'
```

In [ ]:

```
>>> class Box:
...     var=100
...
>>> obj=Box()
>>> obj.var
100
>>> obj.var='Data1' #<<<
>>>
>>> obj.var
'Data1'
>>> del(Box.var)
>>> obj.var
'Data1'
>>> a=10
>>> b=20
>>> c=30
>>> fname='p1.log'
>>>
>>> class Box:
...     print(fname)
...
p1.log
>>> class Box:
...     def f1(self):
...             print(fname)
...
>>> obj=Box()
>>> obj.f1()
p1.log
>>> class Box:
...     global v1
...     v1=234
...     def f1(self):
...             print(v1)
...
>>> obj=Box()
>>> obj.f1()
234
>>> print(v1)
234
>>> obj.v1
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'Box' object has no attribute 'v1'
>>> Box.v1
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: type object 'Box' has no attribute 'v1'
>>> v1
234
>>> a=10
>>>
>>> a=int(10)
>>> b=int()
>>> b
0
```

```
>>> f=1.45
>>>
>>> f=float(1.45)
>>> f
1.45
>>> f=float()
>>> f
0.0
>>> s=str()
>>> s
''
>>> s=str("Hello")
>>> s
'Hello'
>>> obj=str("Hello")
>>> obj.upper()
'HELLO'
>>> obj.title()
'Hello'
>>> # class str:
>>> #   def __init__(self,a=''):
>>> #       ..
>>> #   def upper(self):
>>> #        ...
>>> #   def title(self):
>>> #        ...
>>>
>>> help(str.upper)
Help on method_descriptor:

upper(self, /)
    Return a copy of the string converted to uppercase.

>>> s='abc'
>>> s.upper() # in oops -> upper(s)
'ABC'
>>> L=list()
>>> L.append('d1')
>>> L.insert(1,'d2')
>>> # class list:
>>> #   def __init__(self,a=[]):
>>> #   def append(self,a1):
>>> #   def insert(self,index,a1):
>>>
>>> obj=list()
>>> obj.append("D1")
>>> obj.insert(1,"D2")
>>>
>>> obj
['D1', 'D2']
>>>
>>> # obj.append("D1")-->append(self,Value) ==>append(obj,"D1")
>>> # obj.insert(1,"D2") -->insert(self,index,Value) -> insert(obj,1,"D2")
>>> help(list.append)
Help on method_descriptor:

append(self, object, /)
```

```
    Append object to the end of the list.

>>> help(list.insert)
Help on method_descriptor:

insert(self, index, object, /)
    Insert object before index.

>>> d={}
>>> s={1,2}
>>> type(s)
<class 'set'>
>>> type(d)
<class 'dict'>
>>>
>>> d=dict()
>>>
>>> s={}
>>> type(s)
<class 'dict'>
>>> type({})
<class 'dict'>
>>>
>>> s=set()
>>> type(s)
<class 'set'>
>>> len(s)
0
>>> # ''/"" b'' b"" [] () {}
>>> type('')
<class 'str'>
>>> type(b'')
<class 'bytes'>
>>> type([])
<class 'list'>
>>> type(())
<class 'tuple'>
>>> type({})
<class 'dict'>
>>>
>>>
>>> class Box:
...     pass
...
>>> obj=Box()
>>> obj
<__main__.Box object at 0x00BCB410>
>>>
>>>
>>>
>>> class Box:
...     def __init__(self,a=0):
...             self.var=a
...     def method1(self):
...             return self.var,"STDERR"
...
>>> obj=Box("date")
```

```
>>> obj.method1()
('date', 'STDERR')
>>> obj.method1()
('date', 'STDERR')
>>>
>>>
>>> import socket
>>> obj=socket.socket()
>>>
>>> from socket import socket
>>>
>>> # from module import member
>>> #
>>>
>>> import bs4
>>> obj=bs4.BeautifulSoup("<html><b>sample</b></html>")
>>>
>>> from bs4 import BeautifulSoup
>>> obj=BeautifulSoup("<html><b>sample</b></html>")
>>>
>>> import cgi
>>> obj=cgi.FieldStorage()
>>> obj
FieldStorage(None, None, [])
>>> import sqlite3
>>>
>>> obj=sqlite3.connect("t.db")
>>> obj
<sqlite3.Connection object at 0x02BFACA0>
>>> import re
>>> re.search("pattern","afd pattern")
<re.Match object; span=(4, 11), match='pattern'>
>>> type(sqlite3.connect)
<class 'builtin_function_or_method'>
>>> help(super)
Help on class super in module builtins:

class super(object)
 |  super() -> same as super(__class__, <first argument>)
 |  super(type) -> unbound super object
 |  super(type, obj) -> bound super object; requires isinstance(obj, type)

>>> import Emp
>>> Emp.Enrollment()
<Emp.Enrollment object at 0x02B73E30>
>>>
>>> e1=Emp.Enrollment()
>>> e1.f1("Arun","1st JAN")
>>> e1.f2()
('Arun', '1st JAN')
>>> e1.f5()
Arun Working city and Bgroup details:-
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "D:\Emp.py", line 20, in f5
    print("City:{}\tBgroup:{}".format(self.PL
AttributeError: 'Enrollment' object has no at
```

```
>>>
>>> Emp.Enrollment.f3()
>>> e1.f4()
Enter a city name:City1
Enter a blood group:A+
>>> e1.f5()
Arun Working city and Bgroup details:-
City:City1      Bgroup:A+
>>>
>>> e2=Emp.Enrollment()
>>> e2.f1("Vijay","2ndFeb")
>>> e2.f2()
('Vijay', '2ndFeb')
>>> e2.f5()
Vijay Working city and Bgroup details:-
City:   Bgroup:
>>> e2.f4()
Enter a city name:City2
Enter a blood group:AB+
>>> e2.f5()
Vijay Working city and Bgroup details:-
City:City2      Bgroup:AB+
>>> from Emp import Enrollment
>>> e3=Enrollment()
>>> e3.f1("anu","3rdmarch")
>>> e3.f2()
('anu', '3rdmarch')
>>> e3.f5()
anu Working city and Bgroup details:-
City:   Bgroup:
>>> e3.f4()
Enter a city name:City3
Enter a blood group:AB-
>>> e3.f5()
anu Working city and Bgroup details:-
City:City3      Bgroup:AB-
>>>


>>> class Fsinfo:
...     server='default'
...
>>> Fsinfo.server
'default'
>>> Fsinfo.server='Unix'
>>> Fsinfo.server
'Unix'
>>> obj=Fsinfo()
>>> obj.server
'Unix'
>>> obj.server='10.20.30.40'
>>> obj.server
'10.20.30.40'
>>> objA=Fsinfo()
>>> objA.server
'Unix'
>>> objA.server='12.34.4.55'
```

```
>>> objA.server
'12.34.4.55'
>>> obj.server
'10.20.30.40'
>>> Fsinfo.server
'Unix'
>>>
>>> obj1=Fsinfo()
>>> obj1.server
'Unix'
>>> Fsinfo.server='Linux'
>>> obj1.server
'Linux'
>>> Fsinfo.server='aix'
>>> obj1.server
'aix'
>>> Fsinfo.server='Sunos'
>>> obj1.server
'Sunos'
>>> obj1.server='10.20.33.44'
>>> Fsinfo.server='minix'
>>> obj1.server
'10.20.33.44'
```

```
file:ab.py
-------------
class Box:
   pid=0
   def __init__(self):
       ..
   def method1(self):
       ..
   def method2(self):
       ..

def function1():
   ...
def function2():
   ...
port=4555
-----------------------
```

```
file:p1.py
    ------------
import ab

    obj=ab.Box()

    obj.method1()

    obj.method2()
obj.pid ->0

    ab.function1()

    ab.function2()
ab.pid ->Error
    ab.port ------->4555
            -----------------------
```

```
import subprocess
subprocess.check_output()
subprocess.call()
subprocess.function()
obj=subprocess.Popen()
obj.communicate()
obj.pid
```

In [44]:
```
class Box:
    __var=100
    def f1(self):
        print("Im instancemethod:{}".format(self.__var))
    @classmethod
    def f2(cls):
        print("Im classmethod:{}".format(cls.__var))
    @staticmethod
    def f3():
        print("This is staticmethod")
Box.f3()
obj=Box()
obj.f3()
```

```
This is staticmethod
This is staticmethod
```

In [47]:
```
class Fsinfo:
    def f1(self,a1):
        self.fstype=a1
        self.f3() # calling static method
    @classmethod
    def f2(cls):
        cls.findex=0
    @staticmethod
    def f3():
        print("Mounted Filesystem details:-")
        # os.system("df -Th")

obj1=Fsinfo()
obj1.f1("xfs")

obj2=Fsinfo()
obj2.f1("vfat")

obj3=Fsinfo()
obj3.f1("nfs4")

Fsinfo.f2()
obj1.findex=1234
obj1.f1("xfs")
```

```
Mounted Filesystem details:-
Mounted Filesystem details:-
Mounted Filesystem details:-
Mounted Filesystem details:-
```

In [55]:
```
class Outer:
    def f1(self):
        self.var1=10
        objI=self.Inner()
        print("Outer class")
        objI.f2() # calling nested classmethod
    class Inner:
        def f2(self):
            print("Nested class")

obj=Outer()
obj.f1()
#obj.f2() ->Error
objA=Outer.Inner()
objA.f2()
#objA.f1() ->Error
```

```
Outer class
Nested class
Nested class
```

In [61]:
```
# iterator ->Address
# ------------
s='abcde'  # s | a  |  b  |  c  |  e  |0x12345
           #     -     -     -     -
# 0x12345 (Reference-iterator)
# -------
#  de-reference (value)
#  ----------------------
#         |___1.manual next(iterator) ..... StopIterator
#         |___2.automatic -> for loop
#

add=iter(s)
print(next(add))
print(next(add))
print(next(add))
print(next(add))
print(next(add))
#print(next(add)) ->StopIteration
```

```
a
b
c
d
e
```

In [62]:
```
add=iter(s)
for var in add:
    print(var)
```

```
a
b
c
d
e
```

In [63]:
```
# print(s[0])
# step 1: python find the address of s[0]  - reference(iterator)
#                            -----------------
#                             step 2: open an address(de-reference)
```

In [68]:
```

F=open("D:\\emp.csv")
#next(F)
#next(F)
for var in F:
    print(var.strip())
```

```
ram,sales,pune,1000
ashi,prod,bglore,2345
xerox,sales,chennai,45900
yahoo,prod,pune,32450
anu,HR,hyd,4560
biju,prod,bglore,4567
vijay,hr,chennai,3453
theeb,sales,hyd,5678
nithin,prod,pune,1236
```

In [ ]:
```
>>> for var in os.popen("ps -f"):
...     print(var.strip())
...
UID        PID  PPID  C STIME TTY          TIME CMD
apelix    4097  4090  0 10:14 pts/0    00:00:00 bash
apelix    4188  4097  0 10:17 pts/0    00:00:00 python
apelix    4455  4188  0 10:51 pts/0    00:00:00 [ps] <defunct>
apelix    4458  4188  0 10:52 pts/0    00:00:00 [grep] <defunct>
apelix    5560  4188  0 14:38 pts/0    00:00:00 [sh] <defunct>
apelix    5566  4188  0 14:38 pts/0    00:00:00 sh -c ps -f
apelix    5567  5566  0 14:38 pts/0    00:00:00 ps -f
>>>
>>> for var in open("/home/apelix/emp.csv"):
...     print(var.strip())
...
ram,sales,pune,1000
ashi,prod,bglore,2345
xerox,sales,chennai,45900
yahoo,prod,pune,32450
anu,HR,hyd,4560
biju,prod,bglore,4567
vijay,hr,chennai,3453
theeb,sales,hyd,5678
nithin,prod,pune,1236
```

In [ ]:
```
Generator
----------
->function ->return an iterator(Address) ->called Generator
             ============================
def f1():              yield  value
    return 10
      |__ return a value not an address(iterator)
      |__ always we can place at the end of the definition
      |__ exit from functionblock
      |__ we can't use more than one return
```

In [69]:
```
def f1():
    return 10

print(type(f1))
print(type(f1()))
print("-"*15)
def f2():
    yield 10

print(type(f2))
print(type(f2()))
```

```
<class 'function'>
<class 'int'>
---------------
<class 'function'>
<class 'generator'>
```

In [76]:
```
def f1():
    v=10
    yield v+100
    v=20
    yield v
    print("Hello")
    yield 10,20,30
    yield "D1","D2",["D3","D4"]
    yield 10+20+30
add=f1()
print(next(add))
print(next(add))
print(next(add))
print(next(add))
print(next(add))
print(next(add))
```

```
110
20
Hello
(10, 20, 30)
('D1', 'D2', ['D3', 'D4'])
60

---------------------------------------------------------------------------
StopIteration                             Traceback (most recent call last)
<ipython-input-76-b1fe760aaef0> in <module>
     14 print(next(add))
     15 print(next(add))
---> 16 print(next(add))

StopIteration:
```

In [77]:
```
add=f1()
for var in add:
    print(var)
```

```
110
20
Hello
(10, 20, 30)
('D1', 'D2', ['D3', 'D4'])
60
```

In [78]:
```
for var in f1():
    print(var)
```

```
110
20
Hello
(10, 20, 30)
('D1', 'D2', ['D3', 'D4'])
60
```

In [81]:
```
class Box:
    def f1(self):
        yield 100
    @classmethod
    def f2(cls):
        yield 200
obj=Box()
obj.f1()
Box.f2()
for var in obj.f1():
    print(var)
for var in Box.f2():
    print(var)
```

```
100
200
```

In [82]:
```
def f1():
    yield "D1",["Dir1","Dir2","Dir3"],["F1","F2"]
for var in f1():
    print(var)
```

```
('D1', ['Dir1', 'Dir2', 'Dir3'], ['F1', 'F2'])
```

In [85]:
```
def f1(a1,a2):
    return a1+a2
def f2(a1,a2):
    yield a1+a2

print(f1(10,20))
print(next(f2(100,200)))
```

```
30
300
```

In [90]:
```
def fib(n):
    c1,c2=0,1
    count=0
    while(count<n):
        yield c1
        c3=c1+c2
        c1=c2
        c2=c3
        count+=1


add=fib(6)
for var in add:
    print(var)
```

```
0
1
1
2
3
5
```

In [94]:
```
def f1(a):
    return a+100

def f2(n):
    for var in range(n):
        yield f1(var)

add=f2(15)
for var in add:
    print(var)
```

```
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
```