In [ ]:
```
# import re
# re.search('pattern','inputstring')
#
# re.compile('pattern') --> re.compile(r'pattern')
#
# Log=r'C:\\DIR\\Dir\\'
```

In [1]:
```
class Box:                                    class Box:
    var=100                                        var=100
                                                   @classmethod
print(Box.var)                                     def f1(cls):
Box.var=230                                              print(cls.var) # 100
Box.name='admin'                                         cls.var=230
print(Box.var)                                           cls.name='admin'
print(Box.name)                               Box.f1()
```

```
100
230
admin
```

In [ ]:
```
# decorator
# -----------
#  |__meta programming
#
# python -->App1
#      |
#    function() ->App2
#     |
#   function()  ->App3
def function1(arg):
    def function2(): Wrappercode
        args()
        ....
        ....
    return function2()

r=function1()
r()
```

In [11]:
```
def f1():
    def f2():
        def f3():
            print("App1")
        def f4():
            print("App2")
        def f5():
            print("App3")
        f3()
        f4()
        f5()
    return f2

#f1()()
r=f1()
r()
```

```
App1
App2
App3
```

In [14]:
```
def f1(a):
    def f2():
        def f3():
            print("App1")
        def f4():
            print("App2")
        def f5():
            print("App3")
        f3()
        a()
        f4()
        f5()

    return f2

def f6():
    print("UpdatedApp")

r=f1(f6)
r()
```

```
App1
UpdatedApp
App2
App3
```

In [22]:
```
def f1(a):
    def f2():
        def f3():
            print("App1")
        def f4():
            print("App2")
        def f5():
            print("App3")
        f3()
        f4()
        f5()
        a()
    return f2

@f1
def f6():
    print("Updated App")

f6()

@f1
def f7():
    print("F7 block")
f7()
```

```
App1
App2
App3
Updated App
App1
App2
App3
F7 block
```

In [18]:
```
def f1(a):
    def f2():
        a()
    return f2

@f1
def fx():
    print("Fx operation")

@f1
def fy():
    print("Fy operation")

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

fz()
fx()
```

```
Fz operation
Fx operation
```

In [13]:
```
class Box:
    def f1(self,a1,a2):
        self.v1=a1
        self.v2=a2
obj=Box()
obj.f1(10,20) # obj.f1() ->f1(obj)

class Box:
    var=100
    @classmethod
    def f2(cls):
        print("This is classmethod")
        print(cls) # __main__.Box
        print(cls.var) # Box.var
        cls.var=125 # we can modifiy existing class variable
        cls.name='admin' # wen can create newclass variable


Box.f2() # f2(Box)
print(Box.var) # 125
print(Box.name) # admin
```

```
This is classmethod
<class '__main__.Box'>
100
125
admin
```

In [19]:
```
class Box:
    fname="p1.log"
print(Box.fname) # p1.log
Box.fname="Test.log"
print(Box.fname) # Test.log

obj=Box()
print(obj.fname) # Test.log
obj.fname="p1.py"
print(obj.fname) # p1.py
Box.fname="/var/log/repo.log"
print(obj.fname) # p1.py

obj1=Box()
print(obj1.fname) # /var/log/repo.log
```

```
p1.log
Test.log
Test.log
p1.py
p1.py
/var/log/repo.log
```

In [ ]:
```
>>> class Box:
...     fname="p1.log"
...     @classmethod
...     def f1(cls):
...             print(cls.fname)
...             cls.fname="Test.log"
...             print(cls.fname)
...
>>> obj=Box()
>>> obj.fname
'p1.log'
>>> Box.f1()
p1.log
Test.log
>>> obj.fname
'Test.log'
>>> obj.fname="p1.py"
>>> obj.fname
'p1.py'
>>> Box.fname="/var/log/repo.log"
>>>
>>> Box.f1()
/var/log/repo.log
Test.log
>>> obj.fname
'p1.py'
>>> obj1=Box()
>>> obj1.fname
'Test.log'
>>>
```

In [ ]:
```
class Enrollment:
    name=''
    dept=''
    def f1(self,a1,a2):
        self.name=a1
        self.dept=a2
    def f2(self):
        print("Name:{}\tDept:{}".format(self.name,self.dept))
    @classmethod
    def f3(cls):
        cls.place=''
        cls.bgroup=''
    def f4(self,a1,a2):
        self.place=a1
        self.bgroup=a2
    def f5(self):
        print("{} Updated details:-".format(self.name))
        print("NAME:{}\t DEPT:{}\n".format(self.name,self.dept))
        print("PLACE:{}\t Bgroup:{}\n".format(self.place,self.bgroup))


e1=Enrollment()
e1.f1("Arun","sales")
e2=Enrollment()
e2.f1("vijay","prod")
e3=Enrollment()
e3.f1("Anu","HR")
e1.f2()
e2.f2()
e3.f2()
#e1.f5() # Error
Enrollment.f3()  ### Classmethod
e1.f4("City1","A+Ve")
e2.f4("City2","AB+")
e3.f4("City3","O-v")
e1.f5()
e2.f5()
e3.f5()

e4=Enrollment()
e4.f1("Kumar","Admin")
e4.f2()
e4.f4("City4","AB-ve")
e4.f5()
```

In [23]:
```
class Enrollment:
    __name=''
    __dept=''
    def f1(self,a1,a2):
        self.__name=a1
        self.__dept=a2
    def f2(self):
        print("Name:{}\tDept:{}".format(self.__name,self.__dept))
    @classmethod
    def f3(cls):
        cls.__place=''
        cls.__bgroup=''
    def f4(self,a1,a2):
        self.__place=a1
        self.__bgroup=a2
    def f5(self):
        print("{} Updated details:-".format(self.__name))
        print("NAME:{}\t DEPT:{}\n".format(self.__name,self.__dept))
        print("PLACE:{}\t Bgroup:{}\n".format(self.__place,self.__bgroup))


e1=Enrollment()
e1.f1("Arun","sales")
e2=Enrollment()
e2.f1("vijay","prod")
e3=Enrollment()
e3.f1("Anu","HR")
e1.f2()
e2.f2()
e3.f2()
#e1.f5() # Error
Enrollment.f3()  ### Classmethod
e1.f4("City1","A+Ve")
e2.f4("City2","AB+")
e3.f4("City3","O-v")
e1.f5()
e2.f5()
e3.f5()

e4=Enrollment()
e4.f1("Kumar","Admin")
e4.f2()
e4.f4("City4","AB-ve")
e4.f5()
```

```
Name:Arun       Dept:sales
Name:vijay      Dept:prod
Name:Anu        Dept:HR
Arun Updated details:-
NAME:Arun        DEPT:sales

PLACE:City1      Bgroup:A+Ve

vijay Updated details:-
NAME:vijay       DEPT:prod
```

```
PLACE:City2      Bgroup:AB+

Anu Updated details:-
NAME:Anu         DEPT:HR

PLACE:City3      Bgroup:O-v

Name:Kumar      Dept:Admin
Kumar Updated details:-
NAME:Kumar       DEPT:Admin

PLACE:City4      Bgroup:AB-ve
```

In [24]:

```
class Box:
    __port=123
    def f1(self):
        print("Instance method")
        print(self.__port)
    @classmethod
    def f2(cls):
        print("This is class method")
        print(cls.__port)
    @staticmethod
    def f3():
        print("Staticmethod")
        # common task

# Box.f2() # f2(Box)
Box.f3() # classname.f3()
obj=Box()
obj.f3() # classinstance.f3()
```

```
Staticmethod
Staticmethod
```

In [ ]:

```
# class Fs:
#     def f1(self,fs...):
#            # instancemethod
#         self.f2()
#     @staticmethod
#     def f2():
#           os.system("df -Th")


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

obj2=Fs()
obj2.f1("ext4")

obj3=Fs()
obj3.f1("btrfs")
```

In [31]:
```
# in C pointer -> reference(address) -->de-reference(value)
s='abcd' # s |a|b|c|d|0x1234
           # 0x 0y 0a 0b

iter(s)

# de-reference
#  -----------
#     |__ manual ->next(address) .. STOPIteation
#     |__ automatic -->for loop -> for var in iterator:

r=iter(s)
print(r)
print(next(r))
print(next(r))
print(next(r))
print(next(r))
# print(next(r)) # Error
```

```
<str_iterator object at 0x0000000004E64A00>
a
b
c
d
```

In [32]:
```
r=iter(s)
for var in r:
    print(var)
```

```
a
b
c
d
```

In [35]:
```
# function returns iterator(address) - called ->generator
#                    yield value - returns address of value

def f1():
    return 10  # exit from function block
    print("Hello")

print(type(f1))
print(type(f1()))
```

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

In [38]:
```
def f2():
    yield 10
    print("Hello")
    yield 20+30
    print("Test")
    yield "D1","D2"
    yield "D1",["F1","F2"],["F3","F4"]
print(type(f2))
print(type(f2()))
```

```
<class 'function'>
<class 'generator'>
```

In [43]:
```
def f2():
    yield 10
    print("Hello")
    yield 20+30
    print("Test")
    yield "D1","D2"
    yield "D1",["F1","F2"],["F3","F4"]

r=f2()
print(next(r))
print(next(r))
print(next(r))
print(next(r))
print(next(r))
```

```
10
Hello
50
Test
('D1', 'D2')
('D1', ['F1', 'F2'], ['F3', 'F4'])

---------------------------------------------------------------------------
StopIteration                             Traceback (most recent call last)
<ipython-input-43-09a7ef57d009> in <module>
     12 print(next(r))
     13 print(next(r))
---> 14 print(next(r))

StopIteration:
```

In [44]:
```
for var in f2():
    print(var)
```

```
10
Hello
50
Test
('D1', 'D2')
('D1', ['F1', 'F2'], ['F3', 'F4'])
```

In [46]:
```
class Box:
    def f1(self):
        yield "Data1"
        yield "Data2"
    @classmethod
    def f2(cls):
        yield "D1","D2","D3"

obj=Box()
#print(obj.f1())
for var in obj.f1():
    print(var)
```

```
Data1
Data2
```

In [47]:
```
for var in Box.f2():
    print(var)
```

```
('D1', 'D2', 'D3')
```

In [ ]:
```
# lambda - unnamed function
#   |
# lambda args:expression
#
# lambda - function call arguments with return value
#
# def f1():<==named function
```

In [49]:
```
def f1(a1,a2):
    return a1+a2
f1(10,20)
```

Out[49]:
```
30
```

In [51]:
```
# lambda args:expression
f2=lambda a1,a2:a1+a2
f2(10,20)
```

Out[51]:
```
30
```

In [53]:
```
f3=lambda a1,a2:a1>a2
f3(1000,200)
```

Out[53]:
```
True
```

In [54]:
```
def fx(a):
    return a+100

f4=lambda a1:fx(a1)
f4(10)
```

Out[54]:
```
110
```

In [55]:
```
f5=lambda a:a.upper()
f5("abc")
```

Out[55]:
```
'ABC'
```

In [56]:
```
L=list()
for var in range(1,6):
    r=var+100
    L.append(r)
L
```

Out[56]:
```
[101, 102, 103, 104, 105]
```

In [57]:
```
# [value for var in iterable]
#         -----(1)-->----
# --<-(2)--

[var+100 for var in range(1,6)]
```

Out[57]:
```
[101, 102, 103, 104, 105]
```

In [58]:
```
L=list()
for var in [10,20,30,40,50,60]:
    if(var>30):
        L.append(var+100)
    else:
        L.append(var+500)

L
```

Out[58]:
```
[510, 520, 530, 140, 150, 160]
```

In [59]:
```
[var+100 if var>30 else var+500 for var in [10,20,30,40,50,60]]
```

Out[59]:
```
[510, 520, 530, 140, 150, 160]
```

In [ ]:
```
# map() filter() reduce()

# map(function,collection) ->[]
# filter(function,collection) ->[]
# reduce(function,collection)->Single
```

In [60]:
```
L=list()
def fx(a):
    return a+100

for var in [10,20,30,40,50]:
    r=fx(var)
    L.append(r)
print(L)
```

```
[110, 120, 130, 140, 150]
```

In [61]:
```
# map(function,collection)

list(map(fx,[10,20,30,40,50]))
```

Out[61]:
```
[110, 120, 130, 140, 150]
```

In [62]:
```
list(map(lambda a:a+100,[10,20,30,40,50]))
```

Out[62]:
```
[110, 120, 130, 140, 150]
```

In [63]:
```
list(map(lambda a:a.upper(),open("D:\\emp.csv")))
```

Out[63]:
```
['RAM,SALES,PUNE,1000\n',
 'ASHI,PROD,BGLORE,2345\n',
 'XEROX,SALES,CHENNAI,45900\n',
 'YAHOO,PROD,PUNE,32450\n',
 'ANU,HR,HYD,4560\n',
 'BIJU,PROD,BGLORE,4567\n',
 'VIJAY,HR,CHENNAI,3453\n',
 'THEEB,SALES,HYD,5678\n',
 'NITHIN,PROD,PUNE,1236']
```

In [65]:
```
import pprint
d={"CSV":list(map(lambda a:a.upper(),open("D:\\emp.csv")))} # 1 to many
pprint.pprint(d)
```

```
{'CSV': ['RAM,SALES,PUNE,1000\n',
         'ASHI,PROD,BGLORE,2345\n',
         'XEROX,SALES,CHENNAI,45900\n',
         'YAHOO,PROD,PUNE,32450\n',
         'ANU,HR,HYD,4560\n',
         'BIJU,PROD,BGLORE,4567\n',
         'VIJAY,HR,CHENNAI,3453\n',
         'THEEB,SALES,HYD,5678\n',
         'NITHIN,PROD,PUNE,1236']}
```

In [64]:
```
Files=[]
def f(a):
    return a.upper()

for var in open("D:\\emp.csv"):
        r=f(var)
        Files.append(r)
Files
```

Out[64]:
```
['RAM,SALES,PUNE,1000\n',
 'ASHI,PROD,BGLORE,2345\n',
 'XEROX,SALES,CHENNAI,45900\n',
 'YAHOO,PROD,PUNE,32450\n',
 'ANU,HR,HYD,4560\n',
 'BIJU,PROD,BGLORE,4567\n',
 'VIJAY,HR,CHENNAI,3453\n',
 'THEEB,SALES,HYD,5678\n',
 'NITHIN,PROD,PUNE,1236']
```

In [66]:
```
# map(function,collection)
list(map(lambda a:a+100,[100,200,300,400,500]))
```

Out[66]:
```
[200, 300, 400, 500, 600]
```

In [68]:
```
list(map(lambda a:a>50,[34,56,75,120,400,300,210,120]))
```

Out[68]:
```
[False, True, True, True, True, True, True, True]
```

In [69]:
```
L=list()
def f1(a):
    if(a>50):
        return True
    else:
        return False

for var in [34,56,75,120,400,300,210,120]:
    rv=f1(var)
    L.append(rv)
L
```

Out[69]:
```
[False, True, True, True, True, True, True, True]
```

In [70]:
```
list(filter(lambda a:a>50,[34,56,75,120,400,300,210,120]))
```

Out[70]:
```
[56, 75, 120, 400, 300, 210, 120]
```

In [71]:
```
list(filter(lambda a:a in "python",["java","html","python","perl","python3","pyth
```

Out[71]:
```
['python', 'python']
```

In [72]:
```
L=[10,20,30,40,50]
s=0
for var in L:
    s=s+var
print(s)
```
```
150
```

In [74]:
```
import functools
functools.reduce(lambda a1,a2:a1+a2,L)
```
Out[74]:
```
150
```

In [75]:
```
from functools import reduce
if(reduce(lambda a,b:a+b,L)>100):
    print("Yes")
else:
    print("No")
```
```
Yes
```

In [79]:
```
import re
#for v in open("D:\\emp.csv"):
 #   print(re.split(",",v)[-1])
```

In [81]:
```
(lambda a1,a2:int(a1)+int(a2),[re.split(",",v)[-1] for v in open("D:\\emp.csv")])
```
Out[81]:
```
101189
```

In [ ]:
```
reduce(lambda a1,a2:int(a1)+int(a2),[re.split(",",v)[-1] for v in open("D:\\emp.c
```

In [83]:
```
list(filter(lambda a1:int(a1)>5000,[re.split(",",v)[-1] for v in open("D:\\emp.cs
```
Out[83]:
```
['45900\n', '32450\n', '5678\n']
```

In [84]:
```
reduce(lambda a,b:a+b,['t','e','s','t','c','o','d','e'])
```
Out[84]:
```
'testcode'
```

In [85]:
```
"".join(['t','e','s','t','c','o','d','e'])
```
Out[85]:
```
'testcode'
```

In [ ]:
```
Exception Handling

Errors
1.Syntax Error - not following python rules - python won't start execution
2.Logical Error - following python rules - LogicalError ->exit state
Exception
-----------
|__ Signal - program(process) - Exit state

try             try:
                   code block # monitoring block
                except ExceptionNAme as obj:
                    Handle the Exception
                else:
                    There is no Exception
except          finally:
else                 Always running Block
finally
```

In [87]:
```
try:
    print(VAR)
except NameError as eobj:
    print("Exception occured")
    print(eobj)
else:
    print("else block")
finally:
    print("Thank you")
```

```
Exception occured
name 'VAR' is not defined
Thank you
```

In [88]:
```
try:
    VAR=10
except NameError as eobj:
    print("Exception occured")
    print(eobj)
else:
    print("else block")
    print(VAR+100)
finally:
    print("Thank you")
```

```
else block
110
Thank you
```

In [89]:
```
try:
    F=open("invalidfile")
except FileNotFoundError as eobj:
    print("Exception is occured")
    print(eobj)
```

```
Exception is occured
[Errno 2] No such file or directory: 'invalidfile'
```

In [ ]:
```
try:
    F=open("invalidfile")
except FileNotFoundError as eobj:
    print("Exception is occured")
    print(eobj)
else:
    for var in F:
        print(var.strip())
```

In [91]:
```
try:
    Va=10
    print(VA)
except Exception as eobj:
    print(eobj)
```

```
name 'VA' is not defined
```

In [96]:
```
try:
    n=int(input("Enter n value:"))
    if(n>10):
        raise ValueError("n above 10 ")
except Exception as eobj:
    print(eobj)
else:
    print("n value:{}".format(n))
```

```
Enter n value:15
n above 10
```

In [103]:
```
def f1():
    class Box:
        def method1(self):
            self.name='root'
            print(self.name)
    obj=Box()
    return obj
f1()
```

Out[103]:
```
<__main__.f1.<locals>.Box at 0x7fba700>
```

In [99]:
```
obj=f1()
obj.method1()
```

```
root
```

In [ ]:
```
# python (DS+function) ---------->-------------  DB(SQL)
s='select *from table;'       [DBI]              select *from table;
#                                                ...
#    ()[]<iterator>         ------<--------------
```

In [ ]:
```
>>> import sqlite3
>>> sqlite3.connect("test1.db")
<sqlite3.Connection object at 0x002B1CA0>
>>> type(sqlite3)
<class 'module'>
>>>
>>> type(sqlite3.connect)
<class 'builtin_function_or_method'>
>>>
>>> dbh=sqlite3.connect("test1.db")
>>> sth=dbh.cursor()
>>> sth.execute("create table emp(ID int,Name text);")
<sqlite3.Cursor object at 0x0027AC60>
>>> sth.execute("insert into emp(ID,Name)values(101,'arun')")
<sqlite3.Cursor object at 0x0027AC60>
>>> sth.execute("insert into emp(ID,Name)values(234,'vijay')")
<sqlite3.Cursor object at 0x0027AC60>
>>> sth.execute("insert into emp(ID,Name)values(343,'anu')")
<sqlite3.Cursor object at 0x0027AC60>
>>> uid=359
>>> uname='kumar'
>>>
>>> sth.execute("insert into emp(ID,Name)values(?,?),(uid,uname)")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
sqlite3.OperationalError: no such column: uid
>>> sth.execute("insert into emp(ID,Name)values(?,?),((uid,uname))")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
sqlite3.OperationalError: no such column: uid
>>>
>>> sth.execute("insert into emp(ID,Name)values(?,?)",(uid,uname))
<sqlite3.Cursor object at 0x0027AC60>
>>>
>>> sth.execute("select *from emp")
<sqlite3.Cursor object at 0x0027AC60>
>>> sth.fetchone()
(101, 'arun')
>>> sth.fetchone()
(234, 'vijay')
>>> sth.fetchone()
(343, 'anu')
>>> sth.fetchone()
(359, 'kumar')
>>> sth.execute("select *from emp")
<sqlite3.Cursor object at 0x0027AC60>
>>> sth.fetch_all()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'sqlite3.Cursor' object has no attribute 'fetch_all'
>>>
>>> sth.fetchall()
[(101, 'arun'), (234, 'vijay'), (343, 'anu'), (359, 'kumar')]
>>>
>>> sth.execute("select *from emp")
<sqlite3.Cursor object at 0x0027AC60>
```

```
>>>
>>> for var in sth:
...     print(var)
...
(101, 'arun')
(234, 'vijay')
(343, 'anu')
(359, 'kumar')
>>> for var in  sth.execute("select *from emp"):
...     print("{}\t{}".format(var[0],var[1]))
...
101     arun
234     vijay
343     anu
359     kumar
>>> with open("emp.db","w") as WH:
...     for var in  sth.execute("select *from emp"):
...             WH.write("{}\t{}\n".format(var[0],var[1]))
...
9
10
8
10
>>> with open("emp.db") as FH:
...     print(FH.read())
...
101     arun
234     vijay
343     anu
359     kumar

>>>
```

In [ ]:

```

mkdir project
cd project

python -m venv venv

E:\project>venv\Scripts\activate

(venv) E:\project>python -m pip install django


$ python -m django startproject myproject

python manage.py makemigrations

python manage.py migrate

python managet.py createsuperuser

```

In [ ]:
```
1st - Scalar - number,str,bytes,bool,None
conditional + regx
loops + FileHandling

2nd Functions
function call
call with args
return vs global
return vs yield (generator)
decorator
lambda

3rd module + package

4th oops - class + object + method

5th case studies = DB - refer dbmodule doc - python.org/pypi
                                                       /docs/lib_reference
                 (ex:DB)      |
                              |
                           oops - classname,methods -return type
                          procedure code - ds+functions
```