In [1]:
```
# write lambda function to add two values
add=lambda a,b:a+b
```

In [4]:
```
# write one lambda fun to add 10 in some variable
add=lambda a : a+10
add(20)
```

Out[4]:
```
30
```

In [5]:
```
#filter() test each element of a sequence true or not based on some cond
list1=list(range(100,200))
list1
```

Out[5]:
```
[100,
 101,
 102,
 103,
 104,
 105,
 106,
 107,
 108,
 109,
 110,
 111,
 112,
 113,
 114,
 115,
 116,
 117,
 118,
 119,
 120,
 121,
 122,
 123,
 124,
 125,
 126,
 127,
 128,
 129,
 130,
 131,
 132,
 133,
 134,
 135,
 136,
 137,
 138,
 139,
 140,
 141,
 142,
 143,
 144,
 145,
```

```
147,
148,
149,
150,
151,
152,
153,
154,
155,
156,
157,
158,
159,
160,
161,
162,
163,
164,
165,
166,
167,
168,
169,
170,
171,
172,
173,
174,
175,
176,
177,
178,
179,
180,
181,
182,
183,
184,
185,
186,
187,
188,
189,
190,
191,
192,
193,
194,
195,
196,
197,
198,
199]
```

In [6]:

```
# filter the list based on cond where the element is even
# filter(fun,seq)
list2=list(filter(lambda num:num%2==0,list1))
list2
```

Out[6]:

```
[100,
 102,
 104,
 106,
```

```
110,
112,
114,
116,
118,
120,
122,
124,
126,
128,
130,
132,
134,
136,
138,
140,
142,
144,
146,
148,
150,
152,
154,
156,
158,
160,
162,
164,
166,
168,
170,
172,
174,
176,
178,
180,
182,
184,
186,
188,
190,
192,
194,
196,
198]
```

In [7]:

```
# filter the list based on cond where the element is even
# filter(fun,seq)
list2=list(filter(lambda num:num%2==0,list1))
list2
```

Out[7]:

```
[100,
 102,
 104,
 106,
 108,
 110,
 112,
 114,
 116,
 118,
 120,
 122,
```

```
126,
128,
130,
132,
134,
136,
138,
140,
142,
144,
146,
148,
150,
152,
154,
156,
158,
160,
162,
164,
166,
168,
170,
172,
174,
176,
178,
180,
182,
184,
186,
188,
190,
192,
194,
196,
198]
```

In [8]:

```
#write a filter function to filter elements where elements graeter than 3
list1=[5,6,8,1,2]
list2=list(filter(lambda num:num>3,list1))
list2
```

Out[8]:

```
[5, 6, 8]
```

In [10]:

```
#map function returns a new sequence after applying the given function on each element
list1=list(range(100,200))
list2=list(map(lambda x:x-50,list1))
list2
```

Out[10]:

```
[50,
 51,
 52,
 53,
 54,
 55,
 56,
 57,
 58,
 59,
 60,
 61
```

62,
63,
64,
65,
66,
67,
68,
69,
70,
71,
72,
73,
74,
75,
76,
77,
78,
79,
80,
81,
82,
83,
84,
85,
86,
87,
88,
89,
90,
91,
92,
93,
94,
95,
96,
97,
98,
99,
100,
101,
102,
103,
104,
105,
106,
107,
108,
109,
110,
111,
112,
113,
114,
115,
116,
117,
118,
119,
120,
121,
122,
123,
124,
125,

```
 126,
 127,
 128,
 129,
 130,
 131,
 132,
 133,
 134,
 135,
 136,
 137,
 138,
 139,
 140,
 141,
 142,
 143,
 144,
 145,
 146,
 147,
 148,
 149]
```

In [11]:

```
list1=list(range(100,200))
list2=list(map(lambda x:x*10,list1))
list2
```

Out[11]:

```
[1000,
 1010,
 1020,
 1030,
 1040,
 1050,
 1060,
 1070,
 1080,
 1090,
 1100,
 1110,
 1120,
 1130,
 1140,
 1150,
 1160,
 1170,
 1180,
 1190,
 1200,
 1210,
 1220,
 1230,
 1240,
 1250,
 1260,
 1270,
 1280,
 1290,
 1300,
 1310,
 1320,
 1330,
```

```
1350,
1360,
1370,
1380,
1390,
1400,
1410,
1420,
1430,
1440,
1450,
1460,
1470,
1480,
1490,
1500,
1510,
1520,
1530,
1540,
1550,
1560,
1570,
1580,
1590,
1600,
1610,
1620,
1630,
1640,
1650,
1660,
1670,
1680,
1690,
1700,
1710,
1720,
1730,
1740,
1750,
1760,
1770,
1780,
1790,
1800,
1810,
1820,
1830,
1840,
1850,
1860,
1870,
1880,
1890,
1900,
1910,
1920,
1930,
1940,
1950,
1960,
1970,
```

```
1980,
1990]
```

In [12]:
```
str1="hello09"
# filter fun that will extract number if number is present
list1=list(filter(lambda x:True if x in "0123456789" else False,str1))
list1
```

Out[12]:
```
['0', '9']
```

In [13]:
```
#OOP's
#creat a class & its object
class bird:
    pass
b1=bird()
b1
```

Out[13]:
```
<__main__.bird at 0x1c0074f28b0>
```

In [14]:
```
class bird:
   print("Hello this is bird class")
b1=bird()
print(b1)
```

```
Hello this is bird class
<__main__.bird object at 0x000001C007447EB0>
```

In [15]:
```
class bird:
   shape=35
   color="green"
b1=bird()
print(b1.shape)
print(b1.color)
```

```
35
green
```

In [16]:
```
class bird:
   shape=35
   color="green"
b1=bird()
print(b1.shape)
b1.shape=50
print(b1.shape)
```

```
35
50
```

In [17]:
```
class bird:
   shape=35
   color="green"
b1=bird()
print(b1.shape)
b1.shape=50
print(b1.shape)
b2=bird()
print(b2.shape)
```

```
35
50
35
```

In [20]:

```
# create a animal class, define some attribute, create object of class & access attribute
class animal:
   shape=4
   color="golden"
b1=animal()
print(b1.shape)
b1.shape=78
print(b1.shape)
print(b1.color)
```

```
4
78
golden
```

In [21]:

```
class animal:
   shape=4
   color="golden"
b1=animal()
print(b1.shape)
b1.shape=78
print(b1.shape)
print(b1.color)
b1.weight=76 # we can creat extra atribute outside of class if we want
print(b1.weight)
```

```
4
78
golden
76
```

In [22]:

```
# example method inside the class
class square:
    side=14

    def desc(self):
        print("this is square class")
sq1=square()
print(sq1.side)
sq1.desc() # whenever an object calls a function by default name of object is passed as ar
sq2=square()
sq2.desc()
```

```
14
this is square class
this is square class
```

In [23]:

```
# write a program to calculate perimeter
class square:

    def peri(self,side):
        return side*4
sq1=square()
sq1.peri(4)
```

Out[23]:

```
16
```

Out[24]:
```
356
```

In [30]:
```
#create a class circle , write a function inside class to calculate area of circle , creat
class circle:
    def area (self,radius):
        return 3.14*radius*radius
are1=circle()
are1.area(5)
```

Out[30]:
```
78.5
```

In [31]:
```
are2=circle()
are2.area(8)
```

Out[31]:
```
200.96
```

In [32]:
```
class car:
    pass
honda=car()
tata=car()
honda.model='city'
honda.price=1000000
tata.model='nexon'
tata.price=6000000
honda.price
```

Out[32]:
```
1000000
```

In [33]:
```
# create a constructor
class car:
    def __init__(self,m,p):
        self.modelname=m # honda.modelname=city
        self.price=p

honda=car('city',100000)
print(honda.modelname)
print(honda.price)
tata=car('nexon',560000)
print(tata.modelname)
print(tata.price)
# constructor is called automaticaly
```

```
city
100000
nexon
560000
```

In [34]:
```
class car:
    def __init__(self,m,p):
        self.modelname=m # honda.modelname=city
        self.price=p

    def display(self):
        print(self.modelname,self.price)

honda=car('city',100000)
#print(honda.modelname)
```

```
#print(honda.price)
honda.display()
```

```
city 100000
```

In [35]:

```
#create fun double- double the price
class car:
    def __init__(self,m,p):
        self.modelname=m # honda.modelname=city
        self.price=p

    def display(self):
        print(self.modelname,2*self.price)

honda=car('city',100000)
honda.display()
```

```
city 200000
```

In [42]:

```
# create employee class , create a constructor which takes id name salary, also write disp
class employee:
    def __init__(self,x,y,z):
        self.id=x
        self.salary=y
        self.name=z
    def display(self):
        print(self.id,self.salary,self.name)
Detail=employee("1356",500000,"Laxmi")
Detail.display()
```

```
1356 500000 Laxmi
```

In [43]:

```
# Inheritance----parent child class
class person:
    def display1(self):
        print("This is super class")

class employee(person):
    def display2(self):
        print("This is sub class")
emp1=employee()
emp1.display1()
emp1.display2()
```

```
This is super class
This is sub class
```

In [47]:

```
# parent class has constructor
class person:
     def __init__(self,n,s,d):
        self.name=n
        self.salary=s
        self.desgination=d

     def display1(self):
        print("This is super class",self.name,self.salary)

class employee(person):
        def display2(self):
            print("This is sub class",self.desgination)
emp1=employee('John',10000,'manager')
```

```
emp1.display1()
emp1.display2()
```

```
This is super class John 10000
This is sub class manager
```

In [48]:

```
# if both class have constructor then which constructor will call
class person:
     def __init__(self,n,s):
        self.name=n
        self.salary=s


class employee(person):
     def __init__(self,e_name,e_salary,e_desig):
        self.designation=e_desig
        person.__init__(self,e_name,e_salary)

emp1=employee('John',10000,'manager')
print(emp1.name,emp1.salary,emp1.designation)

#here we are telling n = e_name and s=e_salary
```

```
John 10000 manager
```

In [50]:

```
# Multilevel inheritance
class shape():
    def m(self):
        print("This is shape")
class rect(shape):
    def m1(self):
        print("This is rect")
class square(rect):
    def m2(self):
        print("This is square")
s=square()
s.m2()
s.m1()
s.m()
```

```
This is square
This is rect
This is shape
```

In [51]:

```
# multiple inheritance
class shape:
    def m(self):
        print("This is shape")
class rect():
    def m1(self):
        print("This is rect")
class square(rect,shape):
    def m2(self):
        print("This is square")
s=square()
s.m2()
s.m1()
s.m()
```

```
This is square
This is rect
This is shape
```

In [52]:

```
class main:
    var1=None #public
    _var2=None #protected
    __var3=None #privte

    def __init__(self,var1,var2,var3):
        self.var1=var1
        self._var2=var2
        self.__var3=var3

    def displaypublic(self):
        print(self.var1)

    def _displayprotected(self):
        print(self._var2)

    def __displayprivate(self):
        print(self.__var3)

class sub(main):
    def __init__(self,var1,var2,var3):
        main.__init__(self,var1,var2,var3)

    def accessprotected(self):
        self.__displayprotected()

obj=sub("John",123,50000)
print(obj.var1)
print(obj._var2)
print(obj.__var3)
```

```
John
123
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_20496/3580397236.py in <module>
     28 print(obj.var1)
     29 print(obj._var2)
---> 30 print(obj.__var3)

AttributeError: 'sub' object has no attribute '__var3'
```

In [53]:

```
class main:
    var1=None #public
    _var2=None #protected
    __var3=None #privte

    def __init__(self,var1,var2,var3):
        self.var1=var1
        self._var2=var2
        self.__var3=var3

    def displaypublic(self):
        print(self.var1)

    def _displayprotected(self):
        print(self._var2)

    def __displayprivate(self):
        print(self.__var3)

    def accessprivate(self):
```

```
        self.__displayprivate()

class sub(main):
    def __init__(self,var1,var2,var3):
        main.__init__(self,var1,var2,var3)

    def accessprotected(self):
        self.__displayprotected()

obj=sub("John",123,50000)
print(obj.var1)
print(obj._var2)
#print(obj.__var3)
#obj.displaypublic()
#obj._displayprotected()
obj.accessprivate()
main1=main("John",123,50000)
#print(main.__var3)
#main1.accessprivate()
```

```
John
123
50000
```

In [ ]: