```
## Day Objective
- Map
- Lambda
- Filter
- use cases - file/Data Encryption
```

```
### Map

Mapping - Entity with Function

- f: x^2 + 3*x + 9

- x:[1,10]

- f(x)

- map(function,iterable)
```

In [6]:
```
def powerN(a,n):
    for i in range(n):
        print(a**i,end=" ")
    return
powerN(5,10)
```

```
1 5 25 125 625 3125 15625 78125 390625 1953125
```

In [37]:
```
def cube(n):
    return n**3
li=[1,2,3,4,5,6]
set((map(cube,li)))

```

Out[37]:
```
{1, 8, 27, 64, 125, 216}
```

In [44]:
```
numbers=[int(i) for i in li]
[cube(i) for i in numbers]
```

Out[44]:
```
[1, 8, 27, 64]
```

In [43]:
```
#li=["1","2","3","4"]
li1=list(range(1,10))
list(map(str,li1))
```

Out[43]:
```
['1', '2', '3', '4', '5', '6', '7', '8', '9']
```

Filter

- used to check Boolean values

In [18]:
```
li=[1,2,3,"a"]
def isDigit(c):
    c=str(c)
    if c.isdigit():

        return True
    return False
isDigit(li)
list(filter(isDigit,li))
```

Out[18]:
```
[1, 2, 3]
```

In [17]:
```
#li=[1,2,3,"a"]
li=[1,2,3,"a","b","v"]
def isDigit(c):
    c=str(c)
    if c.isdigit():

        return 0
    return 1
isDigit(li)
list(filter(isDigit,li))
```

Out[17]:
```
['a', 'b', 'v']
```

In [35]:
```
# Identify  the given number is prime or not
li=[1,2,3,4,5,6,7,8]
def prime(n):
    count=0
    for i in range(1,n+1):
        if n%i==0:
            count=count+1
    if count==2:
        return True
    return False

primelist1=list(filter(prime,range(lb,ub)))
print(primelist1)


```

```
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 7
3, 79, 83, 89, 97]
```

In [34]:
```
ub=100
lb=1
primelist=[i for i in range(lb,ub+1) if prime(i)]
primelist
```

Out[34]:
```
[2,
 3,
 5,
 7,
 11,
 13,
 17,
 19,
 23,
 29,
 31,
 37,
 41,
 43,
 47,
 53,
 59,
 61,
 67,
 71,
 73,
 79,
 83,
 89,
 97]
```

In [36]:
```
list(range(1,10))
```

Out[36]:
```
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Lambda

- Anonymous Functions
- can be embedded into Lists,Maps,Filters
- synatx: lambda arguments(s):expression

In [47]:
```
a = lambda x: x%2==0
print(a(2))
```

```
True
```

In [48]:
```
a = lambda x: x%2
print(a(2))
```

```
0
```

In [55]:
```
#lambda in map functions
list(map(lambda x:x**2 ,range(1,10)))

```

Out[55]: [1, 4, 9, 16, 25, 36, 49, 64, 81]

In [60]:
```
list(filter(lambda x:x%2,range(1,10)))
```

Out[60]: [1, 3, 5, 7, 9]

In [74]:
```
from random import randint

internal1=[randint(0,25) for i in range(10)]
internal2=[randint(0,25) for j in range(10)]
average=list(map(lambda x,y:(x+y)//2,internal1,internal2))
average
```

Out[74]: [8, 19, 16, 14, 16, 9, 8, 6, 13, 15]

In [71]:
```
passedstudents=list(filter(lambda passed:passed>10,average))
passedstudents
```

Out[71]: [15, 11, 15, 16, 11]

In [75]:
```
failedstudents=list(filter(lambda fail:fail<10,average))
failedstudents
```

Out[75]: [8, 9, 8, 6]

In [ ]:
```

```

Applying Programming to the Marks Analysis Application

In [81]:
```
#Generate Marks data
from random import randint
def generateMarks(n,lb,ub):
    filename = "DataFiles/Marks.txt"
    with open(filename,"w") as f:
        for i in range(n):
            marks=randint(lb,ub)
            f.write(str(marks)+"\n")
generateMarks(100,0,100)

```

In [82]:
```
# Marks Analysis
#Percentage of passed,failed and distinction
#Frequency of Highest and lowest marks

import re
def ReadMarkslist(filepath):
    with open(filepath,"r") as f:
        filedata=f.read()
    return(list(map(int,filedata.split())))
def classAverage(filepath):
    Markslist=ReadMarkslist(filepath)
    return sum(Markslist)//len(Markslist)

filepath = "DataFiles/Marks.txt"
classAverage(filepath)
```

Out[82]: 50

In [86]:
```
def Failedpercentage(filepath):
    Markslist=ReadMarkslist(filepath)
    k= list(filter(lambda marks:marks<35,Markslist))
    return (len(k)/len(Markslist) )*100
filepath = "DataFiles/Marks.txt"
Failedpercentage(filepath)
```

Out[86]: 28.999999999999996

In [87]:
```
def Distinction(filepath):
    Markslist=ReadMarkslist(filepath)
    k= list(filter(lambda marks:marks>70,Markslist))
    return (len(k)/len(Markslist) )*100
filepath = "DataFiles/Marks.txt"
Distinction(filepath)
```

Out[87]: 30.0

In [92]:
```
def highestfrequency(filepath):
    Markslist=ReadMarkslist(filepath)
    return Markslist.count(max(Markslist))

highestfrequency(filepath)
```

Out[92]: 2

In [ ]:
```

```

Data Encryption

- key- Mapping of characters with replaced
- 0-->4

- 1-->5
- 2-->6
- 3-->7
- 4-->8
- 5-->9
- 6-->0
- 7-->1
- 8-->2
- 9-->3

In [93]:
```
#Function to generate key for encryption
keypath="DataFiles/key.txt"
def generatekey(keypath):
    with open(keypath,"w") as f:
        for i in range(10):
            if i<6:
                f.write(str(i)+" "+str(i+4)+"\n")
            else:
                f.write(str(i)+" "+str(i-6)+"\n")

generatekey(keypath)
```

In [101]:
```
#Function to generate key for encryption
keypath="DataFiles/key.txt"
def generatekey(keypath):
    with open(keypath,"w") as f:
        for i in range(10):
            if i<5:
                f.write(str(i)+" "+str(i+5)+"\n")
            else:
                f.write(str(i)+" "+str(i+5)[-1:]+"\n")

generatekey(keypath)
```

In [106]:
```
# Function to encrypt a data file
keyfile="DataFiles/key.txt"
def dictionarykeyfile(keyfile):
    key={}
    with open(keyfile,"r") as f:
        for line in f:
            line=line.split()
            key[line[0]] = line[1]
    return key
dictionarykeyfile(keyfile)
```

Out[106]:
```
{'0': '5',
 '1': '6',
 '2': '7',
 '3': '8',
 '4': '9',
 '5': '0',
 '6': '1',
 '7': '2',
 '8': '3',
 '9': '4'}
```

In [105]:
```
a={}
k="12"
a[k[0]]=k[1]
print(a)
```

```
{'1': '2'}
```

In [ ]:
```

```