28th June 2019

Day Objectives:-

- Map
- Lambda
- Filter
- Uses cases- File /Data Encryption

Maps

- Mapping --> An Entity with a funciton
  - f : x^2
  - x : [1,10]
  - f(x)

y=f(x)

x ----------> y=(x^2)

1 .....> 2
2 .....> 4
3 .....> 9
4 .....> 16
5 .....> 25
6 .....> 36
7 .....> 49
8 .....> 64
9 .....> 81
10 .....> 100

map ===>> (function , Iterable)

In [1]:
```
def powerN1(a,n):
    return a**n
a=int(input())
n=int(input())
powerN(3,2)
```

Out[1]: 9

In [2]:
```
def powerN2(a,n):
    r=1
    for i in range(0,n):
        r*=a
    return r
a=int(input())
n=int(input())
powerN2(a,n)
```

```
1
0
```

Out[2]: 1

In [10]:
```
def powerN3_Recurssion(a,n):
    if n==0:
        return 1
    else:
        return a * powerN3_Recurssion(a,n-1)
a=int(input())
n=int(input())
powerN3_Recurssion(a,n)
```

```
2
4
```

Out[10]: 16

In [21]:
```
def cube(n):
    return n**3
li=[1,2,3,4,5,7,9,8]
set(map(cube, li))
```

Out[21]: {1, 8, 27, 64, 125, 343, 512, 729}

In [98]:
```
def cube(n):
    return n**3
li=['1','2','3','4','5','6']

li2=list(map(int,li))  # to convert string to int
li2

list(map(str,li2)) # to convert int to string

tuple(map(float,li2)) # map used to apply every element to iterable
```

Out[98]: (1.0, 2.0, 3.0, 4.0, 5.0, 6.0)

In [99]:
```
[str(i) for i in li]
[int(i) for i in li]
```

Out[99]:
```
[1, 2, 3, 4, 5, 6]
```

In [102]:
```
numbers=[int(i) for i in li]
[cube(i) for i in numbers]

[isprime(i) for i in numbers]
```

Out[102]:
```
[None, 1, 1, None, 1, None]
```

Filter

- Used to check the boolean values Fileter also used like map but boolean values

```
- f:x ---> {T,F}

y C x

x  ------ y
1  ------
2  ------ 2
3  ------ 3
4  ------
5  ------ 5
```

In [49]:
```
li=[1,2,3,'a','b','c',3]
def isDigit(c):
    c=str(c)
    if c.isdigit():
        return True
    else:
        return False
isDigit('a')
```

Out[49]:
```
False
```

In [57]:
```
li=[1,2,3,'a','b','c']
def isDigit(c):
    c=str(c)
    if c.isdigit():  # isdigit(i)
        return 0
    else:
        return 100
isDigit('a')

list(filter(isDigit,li))
```

Out[57]:
```
['a', 'b', 'c']
```

In [58]:
```
li=[1,2,3,'a','b','c']
def isDigit(c):
    c=str(c)
    if c.isdigit():  # isdigit(i)
        return 1
    else:
        return 0
isDigit('a')

list(filter(isDigit,li))
```

Out[58]:
```
[1, 2, 3]
```

In [62]:
```
li=[1,2,3,'a','b','c']
def isDigit(c):
    c=str(c)
    if c.isdigit():  # isdigit(i)
        return -1
    else:
        return 0
isDigit('a')

list(filter(isDigit,li))
```

Out[62]:
```
0
```

In [75]:
```
# Identify all primes in a range
li=[2,3,4,5,6,7,8]
def  isprime(n):
    count=0
    for i in range(1,n+1):
        if n%i==0:
            count=count+1
    if(count==2):
        return 1
#         print(i)
#n=int(input())
isprime(3)
list(filter(isprime,li))
```

Out[75]:
```
[2, 3, 5, 7]
```

In [94]:
```
def checkprime(n):
    for i in range(2,n//2+1):
        if n%i==0:
            return False
    return True

lb,ub=500,600
primeList=list(filter(checkprime,range(lb,ub)))
primeList


# map fails because it doesn't apply for checking conditions
#primeList=list(map(checkprime,range(lb,ub)))
#primeList
```

Out[94]:
```
[503, 509, 521, 523, 541, 547, 557, 563, 569, 571, 577, 587, 593, 599]
```

In [91]:
```
# List comprehension

primeList=[i for i in range(lb,ub+1) if checkprime(i)]
primeList
```

Out[91]:
```
[503, 509, 521, 523, 541, 547, 557, 563, 569, 571, 577, 587, 593, 599]
```

In [ ]:

Lambda

- Anonymous Functions (unknown lambda is inbuilt function
- Can be embedded into Lists comprehension,Maps,Filters
- keyword is different from inbuilt function....

In [121]:
```
a =lambda x :  x**3
a(3)
```

Out[121]: 27

In [135]:
```
a =lambda x :  x**3
list(map(lambda x: x**3,[1,2,3,4,5,6,7]))
```

Out[135]: [1, 8, 27, 64, 125, 216, 343]

In [136]:
```
a =lambda x :  x%2
list(filter(lambda x: x%2!=0,[1,2,3,4,5,6,7]))
```

Out[136]: [1, 3, 5, 7]

In [4]:
```
from random import randint
internal1=[randint(0,25) for i in  range(10)]
internal2=[randint(0,25) for i in range(10)]
internal2
```

Out[4]: [23, 17, 22, 5, 25, 22, 22, 7, 13, 20]

In [5]:
```
internal1
```

Out[5]: [9, 16, 4, 9, 17, 13, 20, 9, 21, 24]

In [6]:
```
averageInternal = list(map(lambda x,y:(x+y)/2,internal1,internal2))
averageInternal
```

Out[6]: [16.0, 16.5, 13.0, 7.0, 21.0, 17.5, 21.0, 8.0, 17.0, 22.0]

In [8]:
```
internal1=[randint(0,25) for i in  range(10)]
internal2=[randint(0,25) for i in range(10)]
internal3=[randint(0,25) for i in  range(10)]
avera3=list(map(lambda x,y,z:(x+y+z)//3,internal1,internal2,internal3))
avera3
```

Out[8]: [10, 10, 9, 7, 8, 17, 15, 12, 4, 14]

In [11]:
```
internal1=[randint(0,25) for i in  range(10)]
internal2=[randint(0,25) for i in range(10)]
internal3=[randint(0,25) for i in  range(10)]
avera3=list(map(lambda x,y,z:(x+y+z)//3,internal1,internal2,internal3))
print(avera3)
FailedMarks=list(filter(lambda i: i<10, avera3))
print(FailedMarks)
```

```
[12, 9, 17, 9, 17, 8, 11, 11, 14, 16]
[9, 9, 8]
```

In [ ]:
```
```

Applying Functional Programming to the Marks Analysis Applications

In [14]:
```
# Generate Marks data
from random import randint
def generateMarks(n,lb,ub):
    filename='D\marks.txt'
    with open (filename ,'w') as f:
        for i in range(n):
            marks=randint(lb,ub)
            f.write(str(marks)+'\n')
generateMarks(10,1,100)
```

In [60]:
```
# Marks Analysis
    # Class Average % passed ,Failed and Disticntion
    # Frequency of Highest and lowest Mark

import re
def ca(filepath):
    with open(filepath,'r') as f:
        filedata=f.read()
        markslist=re.split('\n',filedata)
        markslist=list(map(int,markslist))
    return sum(markslist)//len(markslist)
filepath='D\marks.txt'
ca(filepath)
```

Out[60]: 56

In [61]:
```
def readMarksList(filepath):
    with open(filepath,'r') as f:
        filedata=f.read().split()
    return list(map(int,filedata))
def classAverage(filepath):
    markslist=readMarksList(filepath)
    return sum(markslist)//len(markslist)
filepath='D\marks.txt'
classAverage(filepath)
```

Out[61]: 56

In [ ]:
```

```

In [62]:
```
def percentageFailed(filepath):
    markslist=readMarksList(filepath)
    failedcount = len(list(filter(lambda mark : mark < 40,markslist)))
    return (failedcount/len(markslist))*100
filepath='D\marks.txt'
percentageFailed(filepath)
```

Out[62]: 36.36363636363637

In [64]:
```
def PercentagePassed(filepath):
    return 100-percentageFailed(filepath)
PercentagePassed(filepath)
```

Out[64]: 63.63636363636363

In [63]:
```
def PercentageDistinction(filepath):
    markslist=readMarksList(filepath)
    discount=len(list(filter(lambda mark:mark>70,markslist )))
    return (discount/len(markslist))*100
PercentageDistinction(filepath)
```

Out[63]: 36.36363636363637

In [65]:
```
def HighestFrequency(filepath):
    markslist=readMarksList(filepath)
    return markslist.count(max(markslist))
HighestFrequency(filepath)
```

Out[65]: 1

In [67]:
```
def LowestFrequency(filepath):
    markslist=readMarksList(filepath)
    return markslist.count(min(markslist))
LowestFrequency(filepath)
```

Out[67]: 2

In [ ]:

Data Encryption

- Key - Mapping of data with replaced

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

In [ ]:

In [71]:
```
# Function to generate the key for encription

keypath='DataFiles\key.txt'
def generatekey():
    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')
    return
generatekey()
```

In [79]:
```
# 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)

#def encryptMarksData(datafile,keyfile):
    # construct a dictionary   for key data
```

Out[79]:
```
{'0': '4',
 '1': '5',
 '2': '6',
 '3': '7',
 '4': '8',
 '5': '9',
 '6': '0',
 '7': '1',
 '8': '2',
 '9': '3'}
```

In [ ]:
```

```