Day Objectives

- Maps
- Lambda
- Filter
- Use cases-File/ Data Encryption

```
### Map

Mapping  - Entity with a function

f : x^2

x : [1,10]

f(x)

f(1) -> 1
f(2) -> 4
.......

Relation

y = f(x)

 f:x^2
x      y
1      1
2      4
3      9
4      16
5      25
6      36
7      49
8      64
9      81
10     100

map( function, Iterable )


```

In [2]:
```
### Power Function

def PowerN(a,n):
#     return a**n
    r = 1
    for i in range(0,n):
        r*=a
    return r
PowerN(2,10)

def recursivepowerN(a,n):
    if n == 0:
        return 1
    else:
        return a*recursivepowerN(a,n-1)
recursivepowerN(2,10)
```

Out[2]: 1024

In [75]:
```

def Cube(n):
    return n ** 3

li = ['1','2','3','4','5','6']

li2 = list(map(int, li))

# list(map(float, li2))
tuple(map(float, li2))

numbers = [int(i) for i in li]

[Cube(i) for i in numbers]

```

Out[75]: [1, 8, 27, 64, 125, 216]

Filter

- Used to check the boolean True values

In [33]:
```
li = [1,2,'a','b','c',3]
def isDigit(c):
    c = str(c)
    if c.isdigit():
        return 0
    return 1

isDigit('a')

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

Out[33]:
```
['a', 'b', 'c']
```

In [45]:
```
####### Identify all Prime in a range

li = [1,2,3,4,5,6,7,8,9,10,11]
def is_Prime(n):
    c=0
    for i in range(1,n+1):
        if(n%i==0):
            c+=1
    if(c==2):
        return True
    else:
        return False
is_Prime(1)
list(map(is_Prime,li))
# list(filter(is_Prime,li))

```

Out[45]:
```
[False, True, True, False, True, False, True, False, False, False, True]
```

In [72]:
```
#### Another way to Prime range

def checkPrime(n):
    if n < 2:
        return False
    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(500,600)))

primeList2 = [i for i in range(lb,ub+1) if checkPrime(i)]

# Map
print(primeList)
print(primeList2)

```

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

Lambda

- is keyword in Python
- Greek word Anonymous Functions can be embedded into Lists Comprehension, Maps and Filters

In [98]:
```
a = lambda x : x ** 3
list(map(lambda x:x**3, [1,2,3,4,5,6]))

list(filter(lambda x:( x % 2 == 0), [1,2,3,4,5,6]))


```

Out[98]:
```
[2, 4, 6]
```

In [120]:
```
from random import randint

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)]

averageInternal = list(map(lambda x,y,z: (x+y+z)//2, internal1,internal2,int
failedMarks = list(filter(lambda x: x < 15, averageInternal ))
failedMarks
```

Out[120]:
```
[12, 14, 14]
```

Applying Functional Programming to the Marks Analysis Applications

In [31]:
```
## Generate Marks data
from random import randint

def generateMarks(n, lb, ub):
    filename = 'DataFiles/student.txt'
    with open(filename, 'w') as f:
        for i in range(n):
            marks = randint(lb,ub)
            f.write(str(marks)+'\n')
    return
generateMarks(10, 0, 100)
```

In [16]:
```
#### Marks Analysis
## Class Average, % of Passed,Failed and Distinction
## Frequency of Highest &Lowest Mark.
import re
def readMarkList(filepath):
    with open(filepath, 'r')as f:
        filedata =f.read().split()
    return list(map(int,filedata))

def ClassAverage(filepath):
    with open(filepath, 'r')as f:
        filedata = f.read().split()
        markslist = list(map(int, filedata))
    return sum(markslist)//len(markslist)

filepath='DataFiles/student.txt'
ClassAverage(filepath)
```

Out[16]: 51

In [24]:
```
### Percentage Failed
import re
def readMarkList(filepath):
    with open(filepath, 'r')as f:
        filedata =f.read().split()
    return list(map(int,filedata))

def PercentageFailed(filepath):
    markslist = readMarkList(filepath)
    failedcount = len(list(filter(lambda mark : mark < 40, markslist)))
    return (failedcount/len(markslist))*100
PercentageFailed(filepath)
```

Out[24]: 37.333333333333336

In [25]:
```
def PercentagePassed(filepath):
    return 100 - PercentageFailed(filepath)
PercentagePassed(filepath)

```

Out[25]: 62.666666666666664

In [26]:
```
def Percentageof_Distinction(filepath):
    markslist = readMarkList(filepath)
    distcount = len(list(filter(lambda mark:mark>70,markslist)))
    return (distcount/len(markslist))*100
Percentageof_Distinction(filepath)
```

Out[26]: 28.999999999999996

In [33]:
```
def Highestfrequency(filepath):
    markslist = readMarkList(filepath)
    return [markslist.count(max(markslist))]
Highestfrequency(filepath)
```

Out[33]: [1]

In [32]:
```
def Lowestfrequency(filepath):
    markslist = readMarkList(filepath)
    return [markslist.count(min(markslist))]
Lowestfrequency(filepath)
```

Out[32]: [1]

```
## Data Encryption

Key - Mapping of characters with replaced

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

0 4
1 5
2 6


```

In [40]:
```
### 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')
        return
GenerateKey(keypath)


```

In [43]:
```
### 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[43]:
```
{'0': '4',
 '1': '5',
 '2': '6',
 '3': '7',
 '4': '8',
 '5': '9',
 '6': '0',
 '7': '1',
 '8': '2',
 '9': '3'}
```

In [ ]:
```

```