Date:28June2019

Day Objectives:

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

Map

Mapping - Entity with Functions

f:x^2
x: [1,10]

f(x) ->

f(1) -> 1
f(2) -> 4
...
..
y=f(x)

x	y
1	1
2	4
3	9
4	16
5	
6	
7	
8	
9	
10	

map(function,Iterable)

In [10]:

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

def recursivePowerN(a,n):
    if n==0:
        return 1
    else:
        return a*recursivePowerN(a,n-1)
recursivePowerN(2,6)
```

Out[10]:

```
64
```

In [20]:

```
def cube(n):
    return n**3

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

set(map(cube,li))
#set(map(cube,[123]))


```

Out[20]:

```
{1, 8, 27, 64, 125, 216}
```

In [76]:

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

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

list(map(str,li2)) # to convert back into string  #['1','2','3','4'.'5']

tuple(map(float,li2))  #(1.0,2.0,3.0,4.0,5.0)

[int(i) for i in li]

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

[cube(i) for i in numbers]


```

Out[76]:

```
[1, 8, 27, 64, 125]
```

Filter

Used to check boolean values

Y C X f:x--> {T,F}

x y 1 2 2 3 3 4 5 5 6 7 7 8 9 10

In [49]:

```python
li=[1,2,'a','b','c',3]

def isDigit(c):
    c=str(c)
    if c.isdigit():
        return 1
    return 0

isDigit(2)

list(filter(isDigit,li))

#filter(isdigi)
```

Out[49]:

```
[1, 2, 3]
```

In [77]:

```python
# Identify all primes in a given range

def prime(n):
    if n<2:
        return False
    for i in range(2,n//2+1):
        if n%i==0:
            return False
    return True

lb,ub=1,50

PrimeList=list(filter(prime,range(lb,ub)))

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

# Map fails because it doesn't apply for checking conditions

print(PrimeList)
print(primeList2)
```

```
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47]
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47]
```

In [ ]:

```python

```

Lambda

Anonymous Functions can be embedded into List Comprehensions, Maps, Filters

In [98]:

```
a = lambda x: x**3

print(list(map(lambda x: x**3,[1,2,3,4,5,6,7,8])))

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

```

```
[1, 8, 27, 64, 125, 216, 343, 512]
[2, 4, 6]
```

In [114]:

```
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 x1,x2,x3: (x1+x2+x3)//3,internal1,internal
print(averageInternal)

failedmarks = list(filter(lambda x: x<10, averageInternal))
failedmarks
```

```
[6, 10, 13, 7, 13, 9, 12, 15, 12, 14]
```

Out[114]:

```
[6, 7, 9]
```

In [ ]:

```

```

Applying Functional Programming to the Marks Analysis Application

In [4]:

```
# 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')
    return
generateMarks(100,0,100)


```

In [5]:

```python
# Marks Analysis
# Class average, % of passed, Failed and Distinction
# Frequency of Highest & Lowest Marks
import re

def readMarksList(filepath):
    with open(filepath,'r') as f:
        filedata=f.read().split()
    return list(map(int,filedata))
filepath='DataFiles/marks.txt'
readMarksList(filepath)

def classAverage(filepath):
    markslist=readMarksList(filepath)
    return sum(markslist)//len(markslist)

filepath='DataFiles/marks.txt'

#readMarksList(filepath)
classAverage(filepath)

```

Out[5]:

```
49
```

In [6]:

```python
def percentageFailed(filepath):
    markslist = readMarksList(filepath)
    failedcount = len(list(filter(lambda x : x<40 , markslist )))

    return (failedcount/len(markslist)*100)
percentageFailed(filepath)
```

Out[6]:

```
35.0
```

In [8]:

```python
def percentagePassed(filepath):
    return 100 - percentageFailed(filepath)
percentagePassed(filepath)

```

Out[8]:

```
65.0
```

In [9]:

```python
def distinction(filepath):
    markslist=readMarksList(filepath)
    distinctioncount = len(list(map(lambda x: x>=75, markslist)))

    return (distinctioncount/len(markslist)*100)
distinction(filepath)
```

Out[9]:

```
100.0
```

In [10]:

```python
def HighestFrequency(filepath):
    markslist=readMarksList(filepath)
    return markslist.count(max(markslist))
HighestFrequency(filepath)
```

Out[10]:

```
2
```

In [11]:

```python
def LowestFrequency(filepath):
    markslist=readMarksList(filepath)
    return markslist.count(min(markslist))
LowestFrequency(filepath)
```

Out[11]:

```
3
```

In [ ]:

```python

```

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 [15]:

```python
# 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 [17]:

```
# 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[17]:

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

In [ ]:

```

```