Day Objectives

25th June 2019

- Hacker earth exam problems solved

Marks Application ¶

Function for counting the frequency of digits which are in given string

In [ ]:

In [22]:
```
def uniDa(allnumbers):
    unque=[]
    for n in allnumbers:
        if n not in unque:
            unque.append(n)
    return unque

def digitfre(s):
    allnumbers=[]
    for i in s:
        if i.isdigit():
            allnumbers.append(i)
    un=uniDa(allnumbers)
    for i in range(0,10):
        if str(i) not in un:
            print(0,end=" ")
        else:
            count=allnumbers.count(str(i))
            print(count,end=" ")

digitfre("09876543211adlrfkvm3")
```

```
1 2 1 2 1 1 1 1 1 1
```

In [20]:
```
def model2(s):
    for i in range(0,10):
        count=s.count(str(i))
        print(count,end=" ")
s=input()
model2(s)
```

```
09876543211adlrfkvm3
1 2 1 2 1 1 1 1 1 1
```

In [1]:
```
#contacts Application
    # Add,Search,List,Modify Delete Contacts

# Find and Replace Application
    # Count the total number of occurances of a word
    # If word is existing
    # Replace all occurances of word with another word


# Marks Analysis Application
    # Generate marks file-
    # Input: Marks text file - each line contacts marks of students
    # Generates a report with with the following information
        # Class Average class(filepath)
        # % Of Students passed        all are same filepath
        # % of Students failed
        # % of students with Distinction
        # Frequency of highest marks
        # Frequency of lowest markss

        # common function that calls all the 6 sub functions generateReport(filep
```

In [ ]:
```

```

In [17]:
```
#Function to add contact to contacts text file if doestn't only add
from Packages.validators import phoneNumberValidator as pnv ,emailValidator as e

import re
def addContact(name,phone,email):

    # store data as name,phone,email in the contacts file
    filename='Data\contacts.txt'
    if not checkContactExists(name):
        if pnv(phone) and env(email):
            with open (filename,'a') as f:
                line = name + ',' + str(phone) + ',' + email + '\n'
                f.write(line)
            print(name,'added to contacts file')
        else:
            print("Invalid Phone Number")
            print("Invalid Email")
            return
    else:
        print(name,'already exists')
    return
def checkContactExists(name):
    filename='Data\contacts.txt'
    with open (filename,'r') as f:
        filedata=f.read()
        pattern=name + ','
        return re.search(pattern,filedata)
name=input()
phone=input()
email=input()
addContact(name,phone,email)
```

```
ammma
9440772640
ammma123@gmail.com
ammma added to contacts file
```

In [37]:
```
def searchContact(name):
        with open (filename, 'r') as f:
            filedata=f.read()
            if name in filedata:
                print("%s exists"%name)
            else:
                print("%s doesnot exists"%name)
filename='Data Files\contacts.txt'
name=input()
searchContact(name)
```

```
dsjciofherfjrefjper
dsjciofherfjrefjper doesnot exists
```

In [ ]:
```

```

In [1]:
```
# Generation of marks Function
from random import randint

def GenerateMarks(n,lb,ub):
    filepath='Data\marks.txt'
    with open(filepath,'w') as f:
        for i in range(0,n):
            r=randint(lb,ub)
            f.write(str(r)+'\n')
    print(n,"Marks stored/Generated in file Successfully")

n=int(input())
lb=int(input())
ub=int(input())
GenerateMarks(n,lb,ub)
```

```
50
1
100
50 Marks stored/Generated in file Successfully
```

In [2]:
```
# Class Average ---Class Average(filepath)
# Sum of total students marks/total students count

def classAverage(filepath):
    sum=0
    count=0
    with open (filepath,'r') as f:
        for i in f:
            sum=sum+int(i)
            count=count+1
    return sum/count
filepath='Data\marks.txt'
classAverage(filepath)
```

Out[2]: 47.06

In [3]:
```
# Function to find the percentage of passed students

def PassedPercentage(filepath):
    count=0
    tc=0
    with open(filepath,'r') as f:
        for i in f:
            tc=tc+1
            if (int(i)>=35):
                count=count+1
        return ((count/tc)*100)
filepath='Data\marks.txt'
PassedPercentage(filepath)
```

Out[3]: 57.99999999999999

In [6]:
```
# Function to find the percentage of passed students

def FailedPercentage(filepath):
    count=0
    tc=0
    with open(filepath,'r') as f:
        for i in f:
            tc=tc+1
            if (int(i)<35):
                count=count+1
        return ((count/tc)*100)
filepath='Data\marks.txt'
FailedPercentage(filepath)
```

Out[6]: 42.0

In [5]:
```
# Function to find the % of DistinctionPercentage(filepath)
# Total Distinction =(count/total students)*100


def DistinctionPercentage(filepath):
    count=0
    tc=0
    with open(filepath,'r') as f:
        for i in f:
            tc=tc+1
            if (int(i)>=75):
                count=count+1
        return ((count/tc)*100)
filepath='Data\marks.txt'
DistinctionPercentage(filepath)
```

Out[5]: 20.0

In [7]:
```
# Frequencty of Highest mark --FrequencyHighest (filepath)
def frequencyHighest(filepath):
    with open(filepath,'r') as f:
        li=f.read().split()
        li=list(map(int,li))
        print(max(li))
        return li.count(max(li))
filepath='Data\marks.txt'
frequencyHighest(filepath)
```

```
98
```

Out[7]:
```
3
```

In [8]:
```
# Frequency of Lowest Marks ---FrequencyLowest(filepath)
def LowestFrequency(filepath):
    with open(filepath,'r') as f:
        li=f.read().split()
        li=list(map(int,li))
        print(min(li))
        return li.count(min(li))
filepath='Data\marks.txt'
LowestFrequency(filepath)
```

```
1
```

Out[8]:
```
1
```

In [ ]:
```

```

Marks Application (Marks Report)

In [9]:
```
def MarksGenerationReport(filepath):
    while True:
        n=int(input("Choose option:\n 1.Generation Of Marks:\n 2.Class Average:\n
        st=int(input("Enter No of Students marks"))
        GenerateMarks(st,1,100)
        if(n==1):
            print(GenerateMarks(50,1,100))
            st=int(input("enter marks"))
        elif(n==2):
            print(classAverage(filepath))
        elif(n==3):
            print(PassedPercentage(filepath))
        elif(n==4):
            print(FailedPercentage(filepath))
        elif(n==5):
            print(DistinctionPercentage(filepath))
        elif(n==6):
            print(frequencyHighest(filepath))
        elif(n==7):
            print(LowestFrequency(filepath))
        else:
            break
MarksGenerationReport('Data\marks.txt')
```

```
Choose option:
 1.Generation Of Marks:
 2.Class Average:
 3.Percentage of fail:
 4.Percentage of Pass:
 5.Percentage of Distinction:
 6.FrequencyHighest:
 7.Frequency of Lowest
:2
Enter No of Students marks30
30 Marks stored/Generated in file Successfully
49.8
Choose option:
 1.Generation Of Marks:
 2.Class Average:
 3.Percentage of fail:
 4.Percentage of Pass:
 5.Percentage of Distinction:
 6.FrequencyHighest:
 7.Frequency of Lowest
:3
Enter No of Students marks40
40 Marks stored/Generated in file Successfully
65.0
Choose option:
 1.Generation Of Marks:
 2.Class Average:
 3.Percentage of fail:
 4.Percentage of Pass:
 5.Percentage of Distinction:
 6.FrequencyHighest:
 7.Frequency of Lowest
```

```
:50
Enter No of Students marks60
60 Marks stored/Generated in file Successfully
```

Type *Markdown* and LaTeX: α^2

Hacker earth exam questions

In [52]:
```
# Function to check the two strings are anagrams or not
# abc cba----> True
# aabbcc ---->ccbbaaa --->False

def  ana(s1,s2):
    if(len(s1)!=len(s2)):
        return  False
    if(sorted(s1)==sorted(s2)):
        return True
    else:
        return False
s1=input()
s2=input()
ana(s1,s2)
```

```
abc
nbd
```

Out[52]:
```
False
```

In [71]:
```
def chardeletionsAnagrams(s1,s2):
    uncommon=[]
    for i in s1:
        if i not in s2:
            uncommon.append(i)
    for i in s2:
        if i not in s1:
            uncommon.append(i)
    count=len(uncommon)
    freqs1={}
    freqs2={}
    us1=[]
    us2=[]
    for i in s1:
        if i not in uncommon and i not in us1:
            freqs1[i]=s1.count(i)
            us1.append(i)
    print(freqs1)
    for i in s2:
        if i not in uncommon and i not in us2:
            freqs2[i]=s2.count(i)
            us2.append(i)
    print(freqs2)
    for key in freqs1.keys():
        count+= abs(freqs1[key]-freqs2[key])
        return count
s1=input()
s2=input()
chardeletionsAnagrams(s1,s2)
```

```
abcde
cdjfjdfe
{'c': 1, 'd': 1, 'e': 1}
{'c': 1, 'd': 2, 'e': 1}
```

Out[71]: 6

In [ ]:

In [69]:
```
n=list(map(int,input().split()))
sum=0
count=0
for i in n:
    sum=sum+i
    count=count+1
print(sum//count)
```

```
1000 123456
62228
```

In [94]:
```
# Function to find the character having the kth largest frequency
# Largest nuber in a list
# Second Largest number in a list
# Kth largest number in a list

# Element with highest frequency
# Second Highest frequency
# Kth Highest frequency

def largestFrequency(N,K):
    # build the frequency dictionary for all unique characters
    unique=[]
    freq={}
    for i in N:
        if i not in freq.keys() :
            freq[i]=N.count(i)
    # Extract unique frequencies in descending
    values=sorted(freq.values(),reverse=True)
    uniquevalues=list(set(values))
    uniquevalues=sorted(uniquevalues,reverse=True)
    # Identify the kth largest frequency
    if K<=len(uniquevalues):
        kvalue=uniquevalues[K-1]
    else:
        return -1
    # Get all elements with kth largest frequecy
    li=[]
    for item in  freq.items():
        if item[1]==kvalue:
            li.append(item[0])
    # Minimum of Kth Largest frequency
    return min(li)
largestFrequency('aabcdcc',3)
```

Out[94]:
```
'b'
```

In [ ]:
```

```