Python: without numpy or sklearn

Q1: Given two matrices please print the product of those two matrices

```
Ex 1: A   = [[1 3 4]
             [2 5 7]
             [5 9 6]]
      B   = [[1 0 0]
             [0 1 0]
             [0 0 1]]
      A*B = [[1 3 4]
             [2 5 7]
             [5 9 6]]


Ex 2: A   = [[1 2]
             [3 4]]
      B   = [[1 2 3 4 5]
             [5 6 7 8 9]]
      A*B = [[11 14 17 20 23]
             [23 30 36 42 51]]

Ex 3: A   = [[1 2]
             [3 4]]
      B   = [[1 4]
             [5 6]
             [7 8]
             [9 6]]
      A*B =Not possible
```

In [1]:

```
def matrix_mul(A,B):

    """
    This function returns the product of given two matrices
    """

    # matrix multiplication is not possible when no.of columns in 1st matrix != no.of rows in 2nd matrix
    if len(A[0])!= len(B):
        print("Matrix multiplication not possible")

    else :
        # Create the result matrix and fill it with zeros
        result = []
        result = [[0 for j in range (len(B[0]))] for j in range (len(A))]

        #for matrix multiplication
        for i in range(len(A)):
            for j in range(len(B[0])):
                for k in range(len(B)):
                    result[i][j] += A[i][k]* B[k][j]
        return(result)
```

In [2]:

```
A = [[1,3,4],
     [2,5,7],
     [5,9,6]]

B = [[1,0,0],
    [0,1,0],
    [0,0,1]]

matrix_mul(A,B)
```

Out[2]:

```
[[1, 3, 4], [2, 5, 7], [5, 9, 6]]
```

In [3]:

```
X= [[1,2],
     [3,4]]
Y= [[1,4],
     [5,6],
     [7,8],
     [9,6]]

matrix_mul(X,Y)
```

```
Matrix multiplication not possible
```

Q2: Select a number randomly with probability proportional to its magnitude from the given array of n elements

consider an experiment, selecting an element from the list A randomly with probability proportional to its magnitude. assume we are doing the same experiment for 100 times with replacement, in each experiment you will print a number that is selected randomly from A.

```
Ex 1: A = [0 5 27 6 13 28 100 45 10 79]
let f(x) denote the number of times x getting selected in 100 experimen
ts.
f(100) > f(79) > f(45) > f(28) > f(27) > f(13) > f(10) > f(6) > f(5) >
f(0)
```

In [4]:

```
from random import uniform

def pick_a_number_from_list(A):

    #sum
    sum =0;
    for i in A:
        sum += i

    #cumulative sum
    x=0
    lst=[]
    for i in A:
        lst.append(x+i/sum)
        x=x+i/sum

    #selecting random number
    n = uniform(0,1)
    for i in range(0,len(lst)):
        if n< lst[i]:
            return A[i]

def sampling_based_on_magnitued():

    for i in range(1,100):
        number = pick_a_number_from_list(A)
        print(number)


A = [0,5,27,6,13,28,100,45,10,79]
sampling_based_on_magnitued()
```

```
79
27
100
79
```

```
28
79
79
45
10
45
6
100
100
45
28
28
100
100
100
27
79
27
79
100
79
100
100
79
45
27
28
79
79
79
45
10
100
100
100
100
100
13
45
28
27
79
10
28
100
79
100
6
100
28
100
79
79
```

```
100
79
100
5
100
100
79
100
100
79
79
100
28
13
45
6
100
10
45
79
79
100
100
79
100
5
100
28
79
100
28
79
27
100
79
100
79
27
10
79
27
79
```

Q3: Replace the digits in the string with

consider a string that will have digits in that, we need to remove all the not digits and replace the digits with #

```
Ex 1: A = 234              Output: ###
Ex 2: A = a2b3c4           Output: ###
Ex 3: A = abc              Output:   (empty string)
Ex 5: A = #2a$#b%c%561#    Output: ####
```

In [5]:

```
import re

def replace_digits(String):
    """
    This functions returns modified string which is after replacing the digits in the string with #
    """
    modified_string = '#'* len(re.sub(r'\D','',String))
    return modified_string

print(replace_digits('a2b3c4'))
print(replace_digits('234'))
print(replace_digits('#2a$#b%c%561#'))
print(replace_digits('abc'))
```

```
###
###
####
```

Q4: Students marks dashboard

consider the marks list of class students given two lists
Students =
['student1','student2','student3','student4','student5','student6','student7','student8','student9','student1
Marks = [45, 78, 12, 14, 48, 43, 45, 98, 35, 80]
from the above two lists the Student[0] got Marks[0], Student[1] got Marks[1] and so on

your task is to print the name of students **a. Who got top 5 ranks, in the descending order of marks**
b. Who got least 5 ranks, in the increasing order of marks
d. Who got marks between >25th percentile <75th percentile, in the increasing order of marks

```
Ex 1:
Students=['student1','student2','student3','student4','student5','stude
nt6','student7','student8','student9','student10']
Marks = [45, 78, 12, 14, 48, 43, 47, 98, 35, 80]
a.
student8  98
student10 80
student2  78
student5  48
student7  47
b.
student3 12
student4 14
student9 35
student6 43
student1 45
c.
student9 35
student6 43
student1 45
student7 47
student5 48
```

In [6]:

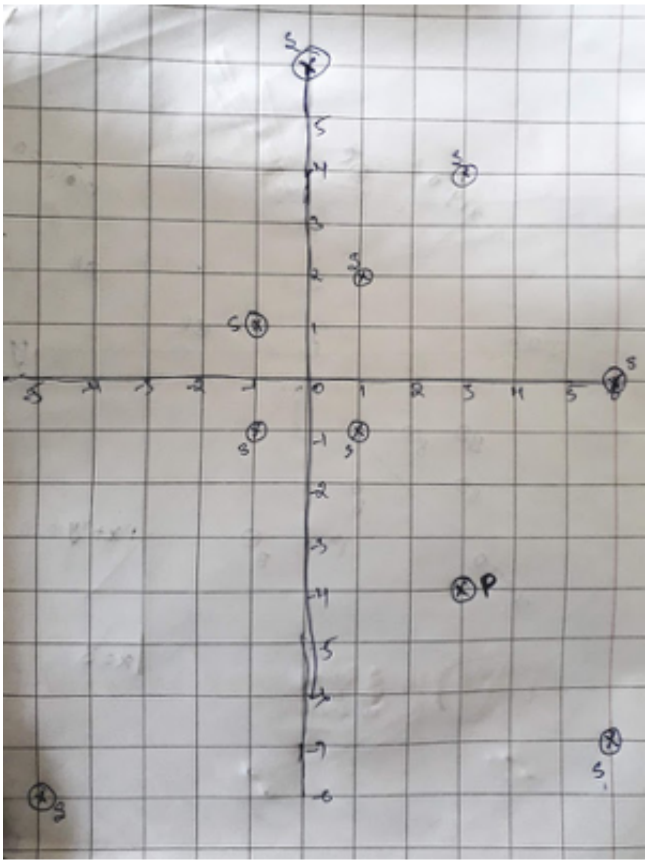
```
def display_dash_board(students, marks):

    mark_list= dict(zip(Students,Marks))
    top_5_students = sorted(mark_list.items(), key = lambda x: x[1]
, reverse = True)
    least_5_students = sorted(mark_list.items(), key = lambda x: x[
1])
    students_within_25_and_75 = dict(least_5_students)

    # computing top 5 students
    print('top_5_students :')
    for k,v in top_5_students[:5]:
        print(k,v)

    # computing least 5 students
    print('\nleast_5_students :')
    for k,v in least_5_students [:5]:
        print(k,v)

    # computing students_within_25_and_75
    print('\nstudents_within_25_and_75:')
    for k, v in students_within_25_and_75.items():
        pct = (v * 100.0)/100
        if 25< pct <75:
            print(k,v)

Students=['student1','student2','student3','student4','student5','s
tudent6','student7','student8','student9','student10']
Marks = [45, 78, 12, 14, 48, 43, 47, 98, 35, 80]
display_dash_board(Students, Marks)
```

```
top_5_students :
student8 98
student10 80
student2 78
student5 48
student7 47

least_5_students :
student3 12
student4 14
student9 35
student6 43
student1 45

students_within_25_and_75:
student9 35
student6 43
student1 45
student7 47
student5 48
```

Q5: Find the closest points

consider you have given n data points in the form of list of tuples like S=[(x1,y1),(x2,y2),(x3,y3),(x4,y4),(x5,y5),..,(xn,yn)] and a point P=(p,q)
your task is to find 5 closest points(based on cosine distance) in S from P
cosine distance between two points (x,y) and (p,q) is defind as $cos^{-1}(\frac{(x\cdot p+y\cdot q)}{\sqrt{(x^2+y^2)}\cdot\sqrt{(p^2+q^2)}})$

```
Ex:

S= [(1,2),(3,4),(-1,1),(6,-7),(0, 6),(-5,-8),(-1,-1)(6,0),(1,-1)]
P= (3,-4)
```



```
Output:
(6,-7)
(1,-1)
(6,0)
(-5,-8)
(-1,-1)
```

In [7]:

```
import math

def closest_points_to_p(S, P):

    p = P[0]
    q = P[1]
    cosine_dist = []

    #cosine distance between two points (x,y) and (p,q)
    for x,y in S:
        dist = math.acos((x*p+ y*q)/(math.sqrt(x**2 + y**2) * math.
sqrt(p**2 + q**2)))
        cosine_dist.append(dist)

    #top 5 closest points
    print("closest points : ")
    closest =sorted(zip(S,cosine_dist), key = lambda i:i[1])
    for i in closest[:5]:
        print(i[0])

S= [(1,2),(3,4),(-1,1),(6,-7),(0, 6),(-5,-8),(-1,-1),(6,0),(1,-1)]
P= (3,-4)
closest_points_to_p(S, P)
```

```
closest points :
(6, -7)
(1, -1)
(6, 0)
(-5, -8)
(-1, -1)
```

Q6: Find Which line separates oranges and apples

consider you have given two set of data points in the form of list of tuples like

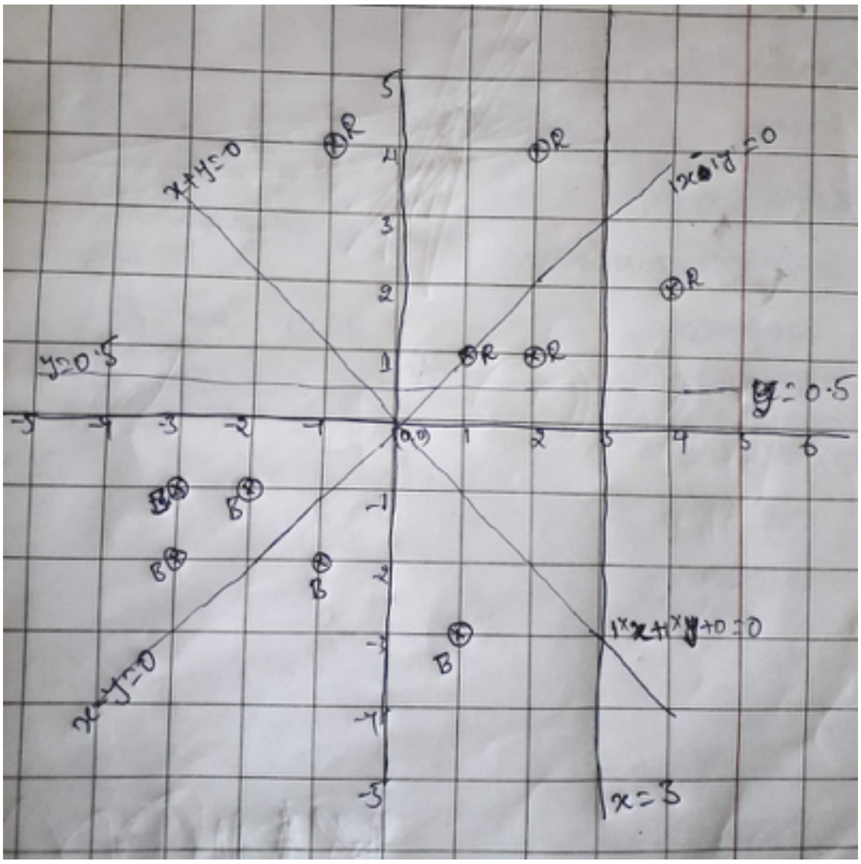
```
Red =[(R11,R12),(R21,R22),(R31,R32),(R41,R42),(R51,R52),..,(Rn1,Rn2)]
Blue=[(B11,B12),(B21,B22),(B31,B32),(B41,B42),(B51,B52),..,(Bm1,Bm2)]
```

and set of line equations(in the string formate, i.e list of strings)

```
Lines = [a1x+b1y+c1,a2x+b2y+c2,a3x+b3y+c3,a4x+b4y+c4,..,K lines]
Note: you need to string parsing here and get the coefficients of x,y a
nd intercept
```

your task is to for each line that is given print "YES"/"NO", you will print yes, if all the red points are one side of the line and blue points are other side of the line, otherwise no

```
Ex:
Red= [(1,1),(2,1),(4,2),(2,4), (-1,4)]
Blue= [(-2,-1),(-1,-2),(-3,-2),(-3,-1),(1,-3)]
Lines=["1x+1y+0","1x-1y+0","1x+0y-3","0x+1y-0.5"]
```



```
Output:
YES
NO
NO
YES
```

In [8]:

```
import re

def i_am_the_one(red,blue,line):
    """
     This function finds Which line separates red and blue
    """
    R =[]
    B =[]
    Red_points=[]
    Blue_points=[]

    #to get the coefficients of x,y and intercept
    for line in Lines:
        a, b, c = [float(x.strip()) for x in re.split('[xy]', line)
]

        m=[]
        for x,y in Red:
            eqn = a*x + b*y + c
            m.append(eqn)
        R.append(m)


        n=[]
        for x,y in Blue:
             eqn = a*x + b*y + c
             n.append(eqn)
        B.append(n)

    #returns True when the red points are one side of the line
    Red_points= [all(j > 0 for j in i) for i in R]

    #returns True when the blue points are other side of the line
    Blue_points= [all(j<0 for j in i) for i in B]

    for i in range(len(Lines)):
        if(Red_points[i]==True and Blue_points[i]==True):
            print("Yes")
        else:
            print("No")

Red= [(1,1),(2,1),(4,2),(2,4), (-1,4)]
Blue= [(-2,-1),(-1,-2),(-3,-2),(-3,-1),(1,-3)]
Lines=["1x+1y+0","1x-1y+0","1x+0y-3","0x+1y-0.5"]

i_am_the_one(Red,Blue,Lines)
```

```
Yes
No
No
Yes
```

Q7: Filling the missing values in the specified formate

You will be given a string with digits and '_'(missing value) symbols you have to replace the '_' symbols as explained

```
Ex 1: _, _, _, 24 ==> 24/4, 24/4, 24/4, 24/4 i.e we. have distributed t
he 24 equally to all 4 places

Ex 2: 40, _, _, _, 60 ==> (60+40)/5,(60+40)/5,(60+40)/5,(60+40)/5,(60+4
0)/5 ==> 20, 20, 20, 20, 20 i.e. the sum of (60+40) is distributed qual
ly to all 5 places

Ex 3: 80, _, _, _, _  ==> 80/5,80/5,80/5,80/5,80/5 ==> 16, 16, 16, 16,
16 i.e. the 80 is distributed qually to all 5 missing values that are r
ight to it

Ex 4: _, _, 30, _, _, _, 50, _, _
==> we will fill the missing values from left to right
    a. first we will distribute the 30 to left two missing values (10,
10, 10, _, _, _, 50, _, _)
    b. now distribute the sum (10+50) missing values in between (10, 10
, 12, 12, 12, 12, 12, _, _)
    c. now we will distribute 12 to right side missing values (10, 10,
12, 12, 12, 12, 4, 4, 4)
```

for a given string with comma seprate values, which will have both missing values numbers like ex: "_, _, x, _, _, _" you need fill the missing values Q: your program reads a string like ex: "_, _, x, _, _, _" and returns the filled sequence Ex:

```
Input1: "_,_,_,24"
Output1: 6,6,6,6

Input2: "40,_,_,_,60"
Output2: 20,20,20,20,20

Input3: "80,_,_,_,_"
Output3: 16,16,16,16,16

Input4: "_,_,30,_,_,_,50,_,_"
Output4: 10,10,12,12,12,12,4,4,4
```

In [9]:

```
def curve_smoothing(string):

    s = string.split(",")

    #indices having number
    num_index = [i for i, num in enumerate(s) if num != '_']

    # string starts with blank
    if num_index[0] != 0:
        num_index = [-1] + num_index

    # string ends with blank
    if num_index[-1] != len(s)-1:
        num_index = num_index + [-1]

    z = zip(num_index [:-1],num_index [1:])

    for (x, y) in z:

        if x == -1:
            k = float(s[y])/(y+1)
            for i in range(x+1,y+1):
                s[i] = int(k)

        elif y == -1:
            k = float(s[x])/(len(s)-x)
            for i in range(x, len(s)):
                s[i] = int(k)

        else:
            k = (float(s[x])+float(s[y]))/(y-x+1)
            for i in range(x,y+1):
                 s[i] = int(k)

    print(s)

strings = ["_,_,_,24",
           "40,_,_,_,60",
           "80,_,_,_,_",
           "_,_,30,_,_,_,50,_,_"]

for string in strings:
    curve_smoothing(string)
```

```
[6, 6, 6, 6]
[20, 20, 20, 20, 20]
[16, 16, 16, 16, 16]
[10, 10, 12, 12, 12, 12, 4, 4, 4]
```

Q8:compute_conditional_probabilites

You will be given a list of lists, each sublist will be of length 2 i.e. [[x,y],[p,q],[l,m]..[r,s]] consider its like a martrix of n rows and two columns

1. the first column F will contain only 5 uniques values (F1, F2, F3, F4, F5)
2. the second column S will contain only 3 uniques values (S1, S2, S3)

```
your task is to find
a. Probability of P(F=F1|S==S1), P(F=F1|S==S2), P(F=F1|S==S3)
b. Probability of P(F=F2|S==S1), P(F=F2|S==S2), P(F=F2|S==S3)
c. Probability of P(F=F3|S==S1), P(F=F3|S==S2), P(F=F3|S==S3)
d. Probability of P(F=F4|S==S1), P(F=F4|S==S2), P(F=F4|S==S3)
e. Probability of P(F=F5|S==S1), P(F=F5|S==S2), P(F=F5|S==S3)
```

Ex:

```
[[F1,S1],[F2,S2],[F3,S3],[F1,S2],[F2,S3],[F3,S2],[F2,S1],[F4,S1],[F4,S3
],[F5,S1]]

a. P(F=F1|S==S1)=1/4, P(F=F1|S==S2)=1/3, P(F=F1|S==S3)=0/3
b. P(F=F2|S==S1)=1/4, P(F=F2|S==S2)=1/3, P(F=F2|S==S3)=1/3
c. P(F=F3|S==S1)=0/4, P(F=F3|S==S2)=1/3, P(F=F3|S==S3)=1/3
d. P(F=F4|S==S1)=1/4, P(F=F4|S==S2)=0/3, P(F=F4|S==S3)=1/3
e. P(F=F5|S==S1)=1/4, P(F=F5|S==S2)=0/3, P(F=F5|S==S3)=0/3
```

In [10]:

```
lst1 = []
lst2 = []
def compute_conditional_probabilites(A):
    for i in range(len(A)):
        k = A[i][0]+A[i][1]
        lst1.append(k)
        lst2.append(A[i][1])


A = [['F1','S1'],['F2','S2'],['F3','S3'],['F1','S2'],['F2','S3'],['F3','S2'],['F2','S1'],['F4','S1'],['F4','S3'],['F5','S1']]
X =['F1','F2','F3','F4','F5']*3
Y =['S1','S2','S3']*5

compute_conditional_probabilites(A)

for i in range(len(X)):
    print('P(F={}|S=={})= {}/{} = {}'.format(X[i],Y[i],lst1.count(X[i]+Y[i]),lst2.count(Y[i]),float(lst1.count(X[i]+Y[i])/lst2.count(Y[i]))))
```

```
P(F=F1|S==S1)= 1/4 = 0.25
P(F=F2|S==S2)= 1/3 = 0.3333333333333333
P(F=F3|S==S3)= 1/3 = 0.3333333333333333
P(F=F4|S==S1)= 1/4 = 0.25
P(F=F5|S==S2)= 0/3 = 0.0
P(F=F1|S==S3)= 0/3 = 0.0
P(F=F2|S==S1)= 1/4 = 0.25
P(F=F3|S==S2)= 1/3 = 0.3333333333333333
P(F=F4|S==S3)= 1/3 = 0.3333333333333333
P(F=F5|S==S1)= 1/4 = 0.25
P(F=F1|S==S2)= 1/3 = 0.3333333333333333
P(F=F2|S==S3)= 1/3 = 0.3333333333333333
P(F=F3|S==S1)= 0/4 = 0.0
P(F=F4|S==S2)= 0/3 = 0.0
P(F=F5|S==S3)= 0/3 = 0.0
```

Q9: Given two sentances S1, S2

You will be given two sentances S1, S2 your task is to find

```
a. Number of common words between S1, S2
b. Words in S1 but not in S2
c. Words in S2 but not in S1
```

Ex:

```
S1= "the first column F will contain only 5 uniques values"
S2= "the second column S will contain only 3 uniques values"
Output:
a. 7
b. ['first','F','5']
c. ['second','S','3']
```

In [11]:

```
def string_features(S1, S2):
    s1 = set(S1.split())
    s2 = set(S2.split())

    a = len(list(s1&s2))
    print('Number of common words between S1, S2 =',a)

    b =list(s1-s2)
    print('Words in S1 but not in S2 :',b)

    c =list(s2-s1)
    print('Words in S2 but not in S1 :',c)

S1= "the first column F will contain only 5 uniques values"
S2= "the second column S will contain only 3 uniques values"
string_features(S1, S2)
```

```
Number of common words between S1, S2 = 7
Words in S1 but not in S2 : ['F', '5', 'first']
Words in S2 but not in S1 : ['3', 'second', 'S']
```

Q10: Given two sentances S1, S2

You will be given a list of lists, each sublist will be of length 2 i.e. [[x,y],[p,q],[l,m]..[r,s]] consider its like a martrix of n rows and two columns

a. the first column Y will contain interger values
b. the second column Y_{score} will be having float values
Your task is to find the value of
$f(Y, Y_{score}) = -1 * \frac{1}{n}\Sigma_{foreachY,Y_{score} pair}(Ylog10(Y_{score}) + (1 - Y)log10(1 - Y_{score}))$ here n is the number of rows in the matrix

```
Ex:
[[1, 0.4], [0, 0.5], [0, 0.9], [0, 0.3], [0, 0.6], [1, 0.1], [1, 0.9],
[1, 0.8]]
output:
0.4243099
```

$$\frac{-1}{8} \cdot ((1 \cdot log_{10}(0.4) + 0 \cdot log_{10}(0.6)) + (0 \cdot log_{10}(0.5) + 1 \cdot log_{10}(0.5)) + \ldots + (1 \cdot log_{10}(0.8) + 0 \cdot lo$$

In [12]:

```
import math

def compute_log_loss(A):

    sum =0
    n = len(A)

    for x,y in A:
        sum += (x*math.log10(y))+((1-x)*math.log10(1-y))

    loss = (-1)*((1/n)*sum)
    return loss


A = [[1, 0.4], [0, 0.5], [0, 0.9], [0, 0.3], [0, 0.6], [1, 0.1], [1
, 0.9], [1, 0.8]]
compute_log_loss(A)
```

Out[12]:

```
0.42430993457031635
```