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
```

```
A=[[1,2],[3,4],[5,6]]
B=[[1,2],[3,4]]
def matrix_mul(A, B):
   sum=0
   c=[]
```

```
    for x in range(0,len(A)):
       c.append([0 for x in range (0,len(B[0]))])

    for i in range (0,len(A)):
       for k in range (0,len(B[0])):
         for j in range (0,len(B)):
           sum=sum+(A[i][j])*(B[j][k])
           c[i][k]=sum
         sum=0
    return c
matrix_mul(A, B)
```

```
[[7, 10], [15, 22], [23, 34]]
```

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 experiments.
f(100) > f(79) > f(45) > f(28) > f(27) > f(13) > f(10) > f(6) > f(5) > f(0)
```

```
import random
A=[1,5.6,3.2,5.8]
def pick_a_number_from_list(A):
    S=0
    Anormalized=[]

    for i in range(0,len(A)):
       S=S+A[i]

    for i in range(0,len(A)):
      Anormalized.append(A[i]/S)
    cum_sum=[Anormalized[0]]

    for i in range(1,len(Anormalized)):
      cum_sum.append(cum_sum[i-1]+Anormalized[i])

    r=random.uniform(0,1)

    for k in range(0,len(cum_sum)):
        if r<cum_sum[k]:
          return (A[k])
```

```
def sampling_based_on_magnitued():
  for i in range (0,100):
    number=pick_a_number_from_list(A)
    print(number)


sampling_based_on_magnitued()
```

```
5.6
5.6
5.6
3.2
5.6
5.8
5.6
5.8
5.6
5.8
3.2
3.2
1
5.8
3.2
5.6
5.8
3.2
5.6
5.6
5.8
5.6
5.6
1
5.8
3.2
5.6
5.6
5.8
3.2
3.2
5.6
5.8
1
5.8
5.6
5.8
5.8
5.6
5.8
5.8
5.6
5.8
5.8
5.6
5.8
5.8
5.6
```

```
5.6
5.8
5.6
5.8
1
1
5.8
5.6
5.6
5.8
5.6
```

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: ####
```

```
import re
def replace_digits(String):
  s=[]
  for match in re.finditer(r'\d',String):
    s.append(match.start())

  return ('#'*len(s))

replace_digits('#2a$#b%c%561#')
```

```
'####'
```

Q4: Students marks dashboard

consider the marks list of class students given two lists
Students = ['student1','student2','student3','student4','student5','student6','student7','student8','student9','student10']
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','student6','student7','stu
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
```

```
Students=['student1','student2','student3','student4','student5','student6','student7','stude
Marks = [45, 78, 12, 14, 48, 43, 47, 98, 35, 80]
def display_dash_board(Students, Marks):
  d={i:j for i,j in zip(Students,Marks)}
  #sort by ascending order of marks list
  Marks.sort()
  # list of bottom 5 marks based on sorted list
  bottom_marks=Marks[0:5]
  top_marks=Marks[-1:-6:-1]
  # calculate index for 25th and 75th percentile
  # sorted list of n elements n*(percentile/100)
  percentile_marks=Marks[(int(len(Marks)/4)):(int((3/4)*len(Marks)))]
  print('Last5_students')
  for i in range(0,len(bottom_marks)):
    for name,values in d.items():
      if values==bottom_marks[i]:
          print(name,values )
  print('Top5_students')
  for i in range(0,len(top_marks)):
```

```
    for i in range(0,len(top_marks)):
      for name,values in d.items():
        if values==top_marks[i]:
            print(name,values )
    print('students between 25 and 75 percentile')
    for i in range(0,len(percentile_marks)):
      for name,values in d.items():
        if values==percentile_marks[i]:
            print(name,values )

display_dash_board(Students,Marks)
#top_5_students, least_5_students, students_within_25_and_75 = display_dash_board(students, m
#print(# those values)
```

```
Last5_students
student3 12
student4 14
student9 35
student6 43
student1 45
Top5_students
student8 98
student10 80
student2 78
student5 48
student7 47
students between 25 and 75 percentile
student9 35
student6 43
student1 45
student7 47
student5 48
```

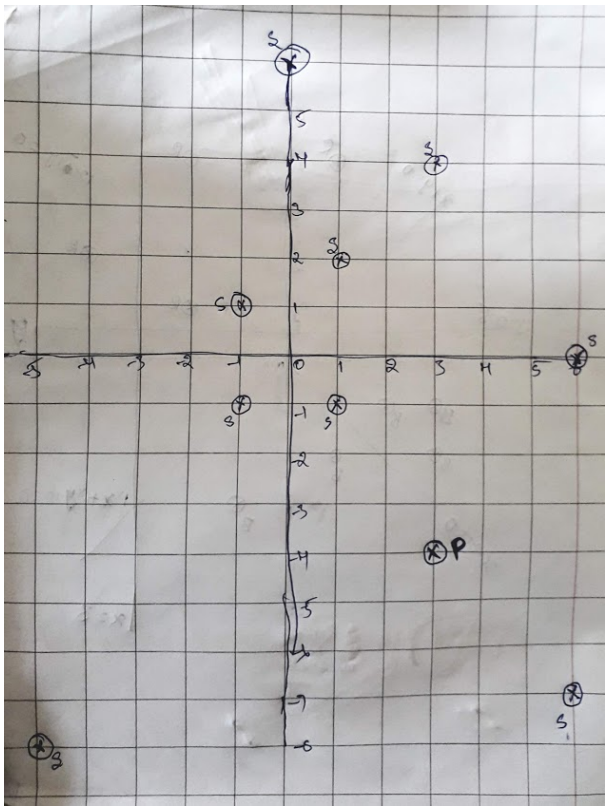
Q5: Find the closest points

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 $cos^{-1}\left(\frac{(x\cdot p+y\cdot q)}{\sqrt{(x^2+y^2)}\cdot\sqrt{(p^2+q^2)}}\right)$

```
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)

```
import math
S= [(1,2),(3,4),(-1,1),(6,-7),(0, 6),(-5,-8),(-1,-1),(6,0),(1,-1)]
p= (3,-4)
def closest_points_to_p(S,p):
  cosine_dist=[]
  for i in range(0,len(S)):
    x=S[i]
    sumx1=0
    for i,j in zip(x,p):
      sumx1+=i*j
    a=math.sqrt(((x[0]**2)+(x[1]**2)))
    b=math.sqrt(((p[0]**2)+(p[1]**2)))
    n=sumx1/(a*b)
    cosine_dist.append (math.acos(n))
  newdict={value:index for index,value in enumerate(cosine_dist)}
  cosine_dist.sort()
  points=[]
  for k in cosine_dist[0:5]:
    points.append(S[newdict[k]])
```

```
    return points


points = closest_points_to_p(S, p)
print(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)]
```

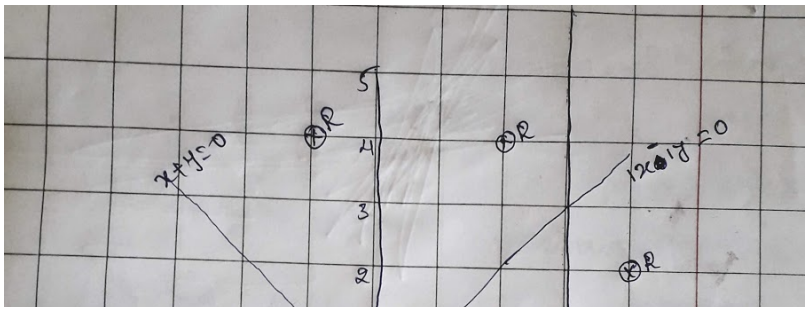
and set of line equations(in the string formate, i.e list of strings)

```
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 and 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"]
```




```
import math
import re
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"]

def i_am_the_one(red,blue,lines):
  coe_f=[]
  coef_list=[]
  for line in lines:
      pattern=r'[^xy]+'
      for match in re.findall(pattern,line):
        coe_f.append(match)
      coef_list.append(coe_f)
      coe_f=[]
  print(coef_list)
  red_values=[]
  blue_values=[]
  for i,j,k in coef_list:
    for m,n in red:
      red_value=(float(i)*float(m))+(float(j)*float(n))+float(k)
      if red_value>0:
        red_values.append('positive')
      else:
        red_values.append('negative')
    for a,b in blue:
      blue_value=(float(i)*float(a))+(float(j)*float(b))+float(k)
      if blue_value>0:
        blue_values.append('positive')
      else:
        blue_values.append('negative')
    red_value_set=set(red_values)
    red_values=[]
    blue_value_set=set(blue_values)
    blue_values=[]
    if len(red_value_set)==1 and len(blue_value_set)==1:
      if (red_value_set)!=(blue_value_set):
        print('yes')
    else:
      print('no')
```

```
i_am_the_one(red,blue,lines)

    [['1', '+1', '+0'], ['1', '-1', '+0'], ['1', '+0', '-3'], ['0', '+1', '-0.5']]
    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 the 24 equally to all

Ex 2: 40, _, _, _, 60 ==> (60+40)/5,(60+40)/5,(60+40)/5,(60+40)/5,(60+40)/5 ==> 20, 20, 20,

Ex 3: 80, _, _, _, _  ==> 80/5,80/5,80/5,80/5,80/5 ==> 16, 16, 16, 16, 16 i.e. the 80 is di

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
```

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
```

```
def filler(L,m,n):
  if m==-1:
    for k in range(0,n+1):
      L[k]=float(L[n])/(n+1)
      print(L[k])
  elif n==-1:
    fill_value=(float(L[m]))/(len(L)-m)
    for k in range(m,len(L)):
       L[k]=fill_value
  else:
    fill_value=(float(L[m])+float(L[n]))/(n-m+1)
    for k in range(m,n+1):
      L[k]=fill_value

  return L

def curve_smoothing(mis):
  L=mis.replace("","").split(',')
  blank=[]
  digits=[]
  for i in range(0,len(L)):
    if L[i]=='_':
     blank.append(i)
    else:
     digits.append(i)

  if digits[0]!=0:
    digits=[-1]+digits

  if digits[-1]!=len(L)-1:
    digits.append(-1)
  print(L)
  for (m,n) in zip(digits[:-1],digits[1:]):
    print(m,n)
    print(filler(L,m,n))

mis="_,_,30,_,_,_,50,_,_"
curve_smoothing(mis)

## Logical idea chosen/learnt used from stack overflow answers after understanding the idea
```

Q8: Filling the missing values in the specified formate

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

```
A = [['F1','S1'],['F2','S2'],['F3','S3'],['F1','S2'],['F2','S3'],['F3','S2'],['F2','S1'],['F4
# you can free to change all these codes/structure
def compute_conditional_probabilites(A):
  counts=0
  countf=0
  S=['S1','S2','S3']
  F=['F1','F2','F3','F4','F5']
  for s in S:
    for f in F:
      for i,j in A:
        if j==s:
          counts+=1
          if i==f:
            countf+=1
      print(countf,counts)
      countf=0
      counts=0


compute_conditional_probabilites(A)
```

```
1 4
1 4
0 4
1 4
1 4
1 3
1 3
1 3
0 3
0 3
0 3
```

```
1 3
1 3
1 3
0 3
```

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']
```

```
def string_features(S1, S2):
  sent_1=S1.split(' ')
  sent_2=S2.split(' ')
  count=0
  popie=[]
  popie2=[]
  for word in sent_1:
    if word in sent_2:
      count=count+1
  for i in range(0,len(sent_1)):
    if sent_1[i] not in sent_2:
         popie.append(sent_1[i])

  for i in range(0,len(sent_2)):
       if sent_2[i] not in sent_1:
         popie2.append(sent_2[i])
  return count,popie,popie2
```

```
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)
```

```
(7, ['first', 'F', '5'], ['second', 'S', '3'])
```

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
```

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

```
import math
def compute_log_loss(A):
  sum1=0
  sum2=0
  n=len(A)
  for i,j in A:
    sum1+=(i*(math.log(j,10)))
    sum2+=((1-i)*(math.log((1-j),10)))
  return (-1*(sum1+sum2)/n)

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]]
loss = compute_log_loss(A)
print(loss)
```

```
0.4243099345703163
```

✓ 0s completed at 1:25 PM