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):

    lenA = len(A)
    lenB = len(B)
    result = 'A*B is Not Possible'

    if len(A[0]) == lenB:
        R_columns = len(B[0])
        result = [ [0 for column in range(R_columns)] for row in range(lenA)]

        for i in range(len(A)):
            for j in range(R_columns):
                for k in range(len(B)):
                    result[i][j] += A[i][k] * B[k][j]

    return result

A = [[1,3,4], [2,5,7], [5,9,6]]
B = [[1,0,0], [0,1,0], [0,0,1]]
C = [[1, 2], [3, 4]]
D = [[1, 2, 3, 4, 5], [5, 6, 7, 8, 9]]
E = [[1, 2], [3, 4]]
F = [[1, 4], [5, 6], [7, 8], [9, 6]]

print(matrix_mul(A, B), '\n')
print(matrix_mul(C, D), '\n')
print(matrix_mul(E, F))
```

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

[[11, 14, 17, 20, 23], [23, 30, 37, 44, 51]]

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

In [2]:

```
from random import uniform

def pick_a_number_from_list(A):

    sum_list = sum(A)

    cum_proba_list =[]
    cum_proba = 0

    for i in A:
        proba = i/sum_list
        cum_proba +=proba
        cum_proba_list.append(cum_proba)

    r = uniform(0,1)

    for index, value in enumerate(cum_proba_list):
        if r <= value:
            break

    return A[index]

def sampling_based_on_magnitued(A):
    sampled = []
    for i in range(1,100):
        number = pick_a_number_from_list(A)
        sampled.append(number)

    for i in sorted(A):
        print('number : {} , frequency : {}'.format(i, sampled.count(i)))

A = [0, 5, 27, 6, 13, 28, 100, 45, 10, 79]
# A = [2, 6, 1.2, 5.8, 20]

sampling_based_on_magnitued(A)
```

```
number : 0 , frequency : 0
number : 5 , frequency : 1
number : 6 , frequency : 2
number : 10 , frequency : 3
number : 13 , frequency : 4
number : 27 , frequency : 9
number : 28 , frequency : 12
number : 45 , frequency : 17
number : 79 , frequency : 22
number : 100 , frequency : 29
```

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 [3]:

```
import re

String = ['234', 'a2b3c4', 'abc', '#2a$#b%c%561#']

def replace_digits(Strings):

    for string in Strings:
        len_ = len(re.sub(r'\D','', string))
        h_string = '#' * len_
        print("'{}' equivalent is : {}".format(string, h_string))

replace_digits(String)
```

```
'234' equivalent is : ###
'a2b3c4' equivalent is : ###
'abc' equivalent is :
'#2a$#b%c%561#' equivalent is : ####
```

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','student8','student9','st

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 [4]:

```
def top_5_student(students, marks):

#     print('a.')
    n = len(marks)
    least_5 = []
    for i in range(n-1,4,-1):
        least_5.append((marks[i][1], marks[i][0]))
    return least_5

def least_5_student(students, marks):
#     print('b.')
    top_5 =[]
    for i in range(5):
        top_5.append((marks[i][1], marks[i][0]))
    return top_5

def student_within_25_and_75(students, marks):

    # Reference : https://youtu.be/3pjSpNbas14?list=PLZ2ps__7DhBZoOybiNj--teGePoNZNO2C&t=176

#     print('c.')
    n= len(marks)
    p_25 = .25
    p_75 = .75
    pr_25 = marks[int(n*p_25)][0]
    pr_75 = marks[int(n*p_75)][0]
    student_within_25_and_75 = []

    for i in range(n):
        if (marks[i][0]) >= pr_25 and (marks[i][0] < pr_75):
            student_within_25_and_75.append((marks[i][1], marks[i][0]))

    return student_within_25_and_75

def display_dash_board(students, marks):

    top_5_students = top_5_student(students, marks)

    least_5_students = least_5_student(students, marks)

    students_within_25_and_75 = student_within_25_and_75(students, marks)

    return top_5_students, least_5_students, students_within_25_and_75

Students=['student1','student2','student3','student4','student5','student6','student7','student8','student9','stu
Marks = [45, 78, 12, 14, 48, 43, 47, 98, 35, 80]
Marks = [35,38,47,58,61,66,68,68,70,79]
Marks = sorted(zip(Marks,Students))
```

```
top_5_students, least_5_students, students_within_25_and_75 = display_dash_board(Students, Marks)

print(top_5_students)
print(least_5_students)
print(students_within_25_and_75)
```

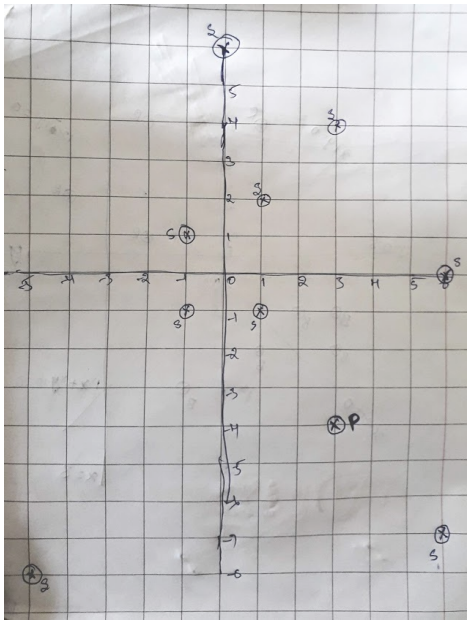
```
[('student10', 79), ('student9', 70), ('student8', 68), ('student7', 68), ('student6', 66)]
[('student1', 35), ('student2', 38), ('student3', 47), ('student4', 58), ('student5', 61)]
[('student3', 47), ('student4', 58), ('student5', 61), ('student6', 66)]
```

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 [5]:

```
import math

# # here S is list of tuples and P is a tuple ot len=2
def closest_points_to_p(S, P):

    closest_points_to_p = []
    cosine_dist_list = []

    for point in S:
        numerator = point[0] * P[0] + point[1] * P[1]
        denominator = math.sqrt(point[0]**2 + point[1]**2) * math.sqrt(P[0]**2 + P[1]**2)

        if denominator !=0:
            cosine_dist = math.acos(numerator/denominator)
            cosine_dist_list.append((cosine_dist, point))

    cosine_dist_list = sorted(cosine_dist_list)

    for i in range(5):
        closest_points_to_p.append(cosine_dist_list[i][1])

    return closest_points_to_p

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

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

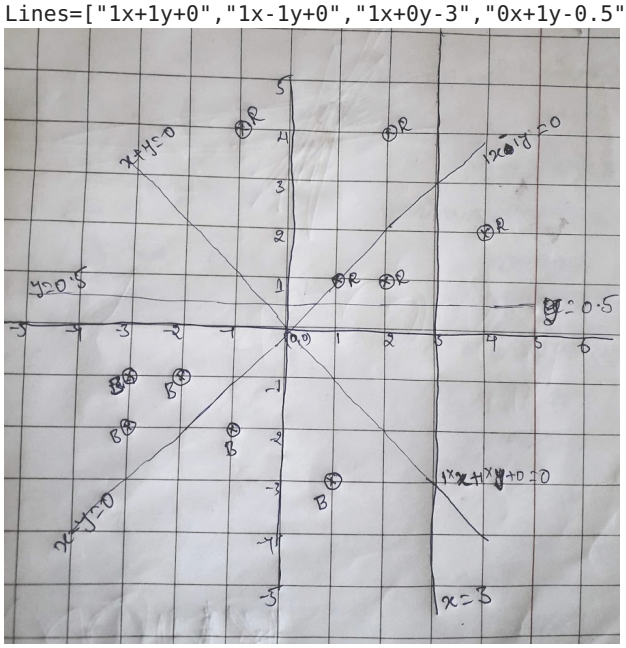
```
Red =[(R11,R12),(R21,R22),(R31,R32),(R41,R42),(R51,R52),..,(Rn1,Rn2)]
Blue=[(B11,B12),(B21,B22),(B31,B32),(B41,B42),(B51,B52),..,(Bm1,Bm2)]
```

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"]
```




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

In [6]:

```
import math

def i_am_the_one(red,blue,line):

    if eval(line.replace('x', f'*{red[0][0]}').replace('y', f'*{red[0][1]}')) >0:
        for red_point in red:
            if eval(line.replace('x','*%s' %red_point[0]).replace('y', '*%s' %red_point[1])) >0:
                pass
            else :
                return 'NO'
    else :
        return 'NO'

    if eval(line.replace('x', f'*{blue[0][0]}').replace('y', f'*{blue[0][1]}')) <0:
        for blue_point in blue:
            if eval(line.replace('x', f'*{blue_point[0]}').replace('y', f'*{blue_point[1]}')) <0:
                pass
            else :
                return 'NO'
    else :
        return 'NO'

    return 'YES'
```

```
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"]

for i in Lines:
    yes_or_no = i_am_the_one(Red, Blue, i)
    print(yes_or_no)
```

```
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 4
places

Ex 2: 40, _, _, _, 60 ==> (60+40)/5,(60+40)/5,(60+40)/5,(60+40)/5,(60+40)/5 ==> 20, 20, 20, 20,
20 i.e. the sum of (60+40) is distributed qually 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 right 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 [7]:

```
def curve_smoothing(string):

    slits = string.split(',')

    if slits[0].isnumeric() and slits[-1].isnumeric():
        replace_val = eval(f'{slits[0]}+{slits[-1]}') // len(slits)
        return [replace_val for i in range(len(slits))]

    if slits[0].isnumeric() and slits[-1] == '_':
        replace_val = int(slits[0]) // len(slits)
        return [replace_val for i in range(len(slits))]

    if slits[0] == '_' :
        cou, l_val, r_list = 0, 0, []
        for v in slits:
            cou += 1
            if v.isnumeric():
                if l_val:
                    l_val = (l_val + int(v)) // (cou+1)
                    r_list.extend([l_val] * (cou))
                else:
                    l_val =  int(v) // (cou)
                    r_list.extend([l_val] * (cou-1))
                cou = 0
        r_list.extend([l_val// (cou +1)] * (cou+1))
        return r_list

sequences = ['_,_,_,24', '40,_,_,_,60', '80,_,_,_,_', '_,_,30,_,_,_,50,_,_']
```

```
for seq in sequences:
    smoothed_values= curve_smoothing(seq)
    print(f'{seq:19} :: {smoothed_values}')
```

```
_,_,_,24            :: [6, 6, 6, 6]
40,_,_,_,60         :: [20, 20, 20, 20, 20]
80,_,_,_,_          :: [16, 16, 16, 16, 16]
_,_,30,_,_,_,50,_,_ :: [10, 10, 12, 12, 12, 12, 4, 4, 4]
```

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

In [8]:

```
def compute_conditional_probabilites(A):

    length_A = len(A)

    pairs = {}
    second = {'S1':0,'S2' :0,'S3' :0}

    for i in range(length_A):
        key = A[i][0]+A[i][1]
        pairs[key] = 0

    for i in range(length_A):
        key_pairs = A[i][0] + A[i][1]
        pairs[key_pairs] += 1

        key_s = A[i][1]
        second[key_s] +=1

    print('P(F=F1|S==S1)={F1S1}/{S1}, P(F=F1|S==S2)={F1S2}/{S2}, P(F=F1|S==S2)={F1S3}/{S3}'.format(
        F1S1=pairs.get('F1S1',0), S1=second['S1'],
        F1S2=pairs.get('F1S2',0), S2=second['S2'],
        F1S3=pairs.get('F1S3',0), S3=second['S3']))

    print('P(F=F2|S==S1)={F2S1}/{S1}, P(F=F2|S==S2)={F2S2}/{S2}, P(F=F2|S==S2)={F2S3}/{S3}'.format(
        F2S1=pairs.get('F2S1',0), S1=second['S1'],
        F2S2=pairs.get('F2S2',0), S2=second['S2'],
        F2S3=pairs.get('F2S3',0), S3=second['S3']))

    print('P(F=F3|S==S1)={F3S1}/{S1}, P(F=F1|S==S2)={F3S2}/{S2}, P(F=F3|S==S2)={F3S3}/{S3}'.format(
        F3S1=pairs.get('F3S1',0), S1=second['S1'],
        F3S2=pairs.get('F3S2',0), S2=second['S2'],
        F3S3=pairs.get('F3S3',0), S3=second['S3']))

    print('P(F=F4|S==S1)={F4S1}/{S1}, P(F=F1|S==S2)={F4S2}/{S2}, P(F=F4|S==S2)={F4S3}/{S3}'.format(
        F4S1=pairs.get('F4S1',0), S1=second['S1'],
        F4S2=pairs.get('F4S2',0), S2=second['S2'],
        F4S3=pairs.get('F4S3',0), S3=second['S3']))

    print('P(F=F5|S==S1)={F5S1}/{S1}, P(F=F1|S==S2)={F5S2}/{S2}, P(F=F5|S==S2)={F5S3}/{S3}'.format(
        F5S1=pairs.get('F5S1',0), S1=second['S1'],
        F5S2=pairs.get('F5S2',0), S2=second['S2'],
        F5S3=pairs.get('F5S3',0), S3=second['S3']))


A = [['F1','S1'],['F2','S2'],['F3','S3'],['F1','S2'],['F2','S3'],['F3','S2'],['F2','S1'],['F4','S1'],['F4','S3'],

compute_conditional_probabilites(A)
```

```
P(F=F1|S==S1)=1/4, P(F=F1|S==S2)=1/3, P(F=F1|S==S2)=0/3
```

```
P(F=F2|S==S1)=1/4, P(F=F2|S==S2)=1/3, P(F=F2|S==S2)=1/3
P(F=F3|S==S1)=0/4, P(F=F1|S==S2)=1/3, P(F=F3|S==S2)=1/3
P(F=F4|S==S1)=1/4, P(F=F1|S==S2)=0/3, P(F=F4|S==S2)=1/3
P(F=F5|S==S1)=1/4, P(F=F1|S==S2)=0/3, P(F=F5|S==S2)=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']
```

In [9]:

```
def string_features(S1, S2):

    split_s1 = set(S1.split())
    split_s2 = set(S2.split())

    a = len(split_s1.intersection(split_s2))
    b = list(split_s1 - split_s2)
    c = list(split_s2 - split_s1)

    return a, b, c

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

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

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
```

$\frac{-1}{8} \cdot ((1 \cdot log_{10}(0.4) + 0 \cdot log_{10}(0.6)) + (0 \cdot log_{10}(0.5) + 1 \cdot log_{10}(0.5)) + \ldots + (1 \cdot log_{10}(0.8) + 0 \cdot log_{10}(0.2)))$

In [10]:

```
from math import log

def compute_log_loss(A):

    length_A = len(A)
    loss = 0

    for value in A:
```

```
        loss += value[0]*log(value[1], 10) + (1-value[0])*log(1-value[1], 10)
    loss = (-loss/length_A)
    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]]
log_loss = compute_log_loss(A)
print(round(log_loss,7))
```

```
0.4243099
```