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 [16]:

```python
def matrix(b,c,d):
    e,f = list(),list()
    count = 0
    for i in range(c):
        for j in range(d):
            f.append(b[j+count])
        e.append(f)
        f = []
        count = (i+1)*d

    return e

def zero_matrix(c,d):
    e,f = list(),list()
    for i in range(c):
        for j in range(d):
            f.append(0)
        e.append(f)
        f = []

    return e

def matrix_multiplication(A,B):
    multiplication = zero_matrix(len(A),len(B[0]))
    if len(A[0]) == len(B):
        # iterating A by row
        for i in range(len(A)):
            # iterating B by column
            for j in range(len(B[0])):
                # iterating B by row
                for k in range(len(B)):
                    multiplication[i][j] += A[i][k] * B[k][j]
        print("the A*B is:")
        for i in range(len(multiplication)):
            print(multiplication[i])

    else:
        print("A*B =Not possible")
```

In [17]:

```
A = [1,2,3,4,5,6,7,8,9,1,2,3,4,5,6,5]
B = [1,2,3,4,5,6,7,8,9,1,2,3,4,5,6,5]

A = matrix(A,4,4)
B = matrix(B,4,4)

matrix_multiplication(A,B)
```

```
the A*B is:
[54, 37, 47, 49]
[130, 93, 119, 129]
[44, 41, 56, 65]
[103, 69, 89, 99]
```

In [18]:

```
A = [1,2,3,4,5,6,7,8]
B = [1,2,3,4,5,6,7,8]

A = matrix(A,2,4)
B = matrix(B,2,4)

matrix_multiplication(A,B)
```

```
A*B =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 [19]:

```
import random
def pick_a_number_from_list(A):
    A = sorted(A, reverse=True)
    Sum = 0
    l = len(A)
    for i in range(l):
        Sum+=A[i]
    collection = []
    for i in range(l):
        b=A[i]/Sum
        collection.append(b)
    sum = 0
    cumsum = []
    for i in collection:
      sum = sum + i
      cumsum.append(sum)

    sample_value = random.uniform(0.0,1.0)
    r = sample_value
    result = 0
    for i in range(l):
        if r < cumsum[i]:
            result = A[i]
            break

    return result

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

```
def sampling_based_on_magnitued():

    A = [0,5,27,6,13,28,100,45,10,79]
    n = len(A)
    number = pick_a_number_from_list(A)

    return number
```

In [20]:

```
li = []
for i in range(100):
    a = sampling_based_on_magnitued()
    li.append(a)

counts = []
for i in A:
    count = 0
    for j in li:
        if i == j:
            count += 1
    counts.append(count)

freq = dict(zip(A, counts))
print(f'number of times each element occurs out of 100 times: {freq}')
```

```
number of times each element occurs out of 100 times: {0: 0, 5: 1, 27: 8, 6: 1, 13: 3, 28: 7, 100: 31, 4
5: 17, 10: 2, 79: 30}
```

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 [21]:

```
string = '#2a$#b%c%561#' # input string
def replace_digits(String):
    string_after_operation = ''
    for i in string:
        if i.isdigit():      # Check the character type
            string_after_operation += '#'  # add '#' if it is a digit else don't do anything

    return string_after_operation

replace_digits(string)
```

Out[21]:

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

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','studer
```

```
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 [48]:

```
students=['student1','student2','student3','student4','student5','student6','student7','student8',\
          'student9','student10']

marks = [45, 78, 12, 14, 48, 43, 47, 98, 35, 80]
def display_dash_board(students, marks):
    mark = marks

    least_5 = sorted(marks, reverse=False)[:5]

    top_5 = sorted(marks, reverse=True)[:5]

    students_top = []
    for i in top_5:
        students_top.append(students[mark.index(i)])
    # using zip() to map values
    top_5_students = zip(students_top, top_5)

    students_least = []
    for i in least_5:
        students_least.append(students[mark.index(i)])

    least_5_students = zip(students_least, least_5)

    dic = dict()
    maxs = max(marks)
    mins = min(marks)
    diff = maxs - mins
    pre_25 =diff*0.25
    pre_75 = diff*0.75
    for i, j in enumerate(marks):
        if j >=pre_25 and j <=pre_75:
            dic[students[i]] = marks[i]

    students_within_25_and_75 = sorted(dic.items(), key = lambda d:(d[1], d[0]))


    return top_5_students, least_5_students, students_within_25_and_75

top_5_students, least_5_students, students_within_25_and_75 = display_dash_board(students, marks)
```

In [49]:

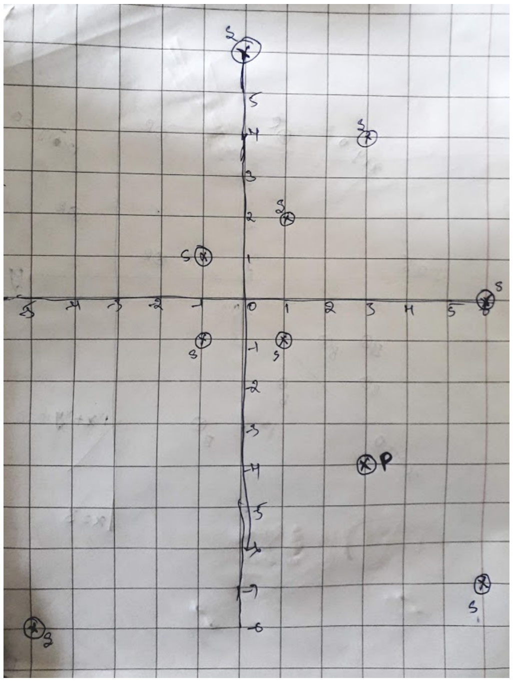
```
print("students who has got top 5 ranks:\n")
for i , j in top_5_students:
    print ("%s   %d" %(i, j))
print("\n")

print("students who has got least 5 ranks:\n")
for i , j in least_5_students:
    print ("%s  %d" %(i, j))
print("\n")

print("students got marks between >25th percentile <75th percentile, in the increasing order of marks:\n")
for i , j in students_within_25_and_75:
    print ("%s  %d" %(i, j))
print("\n")
```

```
students who has got top 5 ranks:

student8   98
student10   80
student2   78
student5   48
student7   47


students who has got least 5 ranks:

student3  12
student4  14
student9  35
student6  43
student1  45


students got marks between >25th percentile <75th percentile, in the increasing order of marks:

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 [24]:

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

import math as m

def closest_points_to_p(S, P):
    Cosine_distance = []
    for i in S:
        a = (i[0]*P[0]+i[1]*P[1])
        b = m.sqrt(i[0]*i[0]+i[1]*i[1])
        c = m.sqrt(P[0]*P[0]+P[1]*P[1])

        Cosine_distance.append(m.acos((a/(b*c))))


    Cosine_distance_sorted = sorted(Cosine_distance, reverse=False)[:5]

    closest_points_from_p = []
    for i in Cosine_distance_sorted:
        closest_points_from_p.append(S[Cosine_distance.index(i)])

    return closest_points_from_p


closest_points_to_p(S, P)
```

Out[24]:

```
[(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

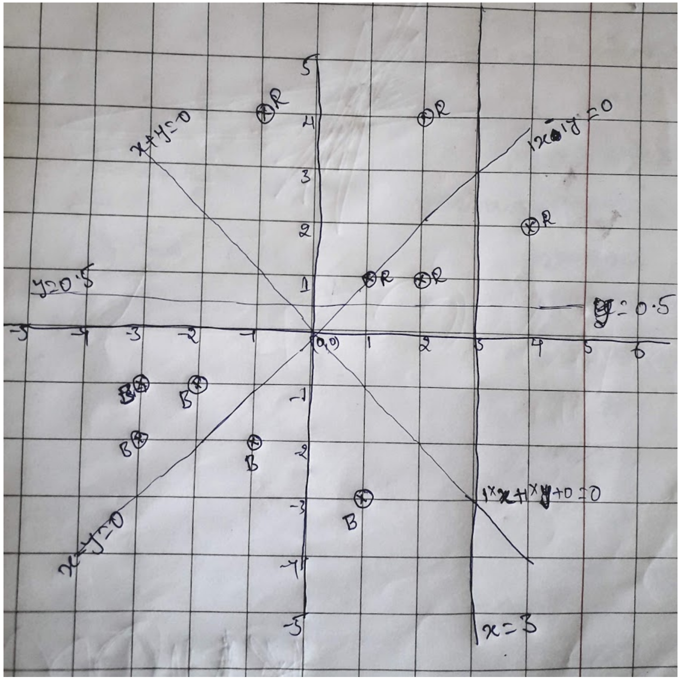
```
Red =[(R11,R12),(R21,R22),(R31,R32),(R41,R42),(R51,R52),..,(Rn1,Rn2)]
Blue=[(B11,B12),(B21,B22),(B31,B32),(B41,B42),(B51,B52),..,(Bm1,Bm2)]
```

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 [25]: import math
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,i):
    i = i.replace('x','*j[0]').replace('y','*j[1]')
    c1,c2,c3,c4 = 0,0,0,0
    box = []
    for j in red:
        distance = eval(i)
        box.append(distance)


    for k in box:
        if k < 0:
            c1 += 1
        else:
            c2 += 1

    box1 = []
    for j in blue:
        distance = eval(i)
        box1.append(distance)

    for k in box1:
        if k < 0:
            c3 += 1
        else:
            c4 += 1

    if (c1 == len(red) or c2 == len(red)) and (c3 == len(blue) or c4 == len(blue)):
        print("YES")

    else:
        print("NO")
```

In [26]:

```
for i in lines:
    i_am_the_one(red, blue, i)
```

```
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 [27]:

```
def curve_smoothing(string):
    line = S.split(',')
    a = [0 for i in range(len(line))]
    box = []
    for i , j in enumerate(line):
        if j.isnumeric():
            a[i] = int(j)

    left ,right = a[0] , 0
    left_index, right_index = 0 , 0
    z = []
    for i, j in enumerate(a[1:]):
        if j > 0:
            right_index , right = i , j
            smooth_num = (left+right)/(right_index-left_index+2)
            a = [smooth_num for i in range(left_index,right_index+2)]
            z.extend(a)
            z.pop()
            left ,left_index = smooth_num , i+1
            right_index, right = 0 , 0
    smooth_num = (left+right)/(len(line)-left_index)
    a = [smooth_num for i in range(left_index,len(line))]
    z.extend(a)

    return z
```

In [28]:

```
S=  "_,_,30,_,_,_,50,_,_"
smoothed_values= curve_smoothing(S)
print(smoothed_values)
```

```
[10.0, 10.0, 12.0, 12.0, 12.0, 12.0, 4.0, 4.0, 4.0]
```

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 [29]:

```
#https://stackoverflow.com/questions/2669059/how-to-sort-alpha-numeric-set-in-python
import re
def sorted_nicely( l ):
    """ Sort the given iterable in the way that humans expect."""
    convert = lambda text: int(text) if text.isdigit() else text
    alphanum_key = lambda key: [ convert(c) for c in re.split('([0-9]+)', key) ]
    return sorted(l, key = alphanum_key)
```

In [30]:

```
def compute_conditional_probabilites(A):
    line1 = []
    line2 = []
    for i in A:
        line1.append(i[0])
        line2.append(i[1])

    line1 = sorted_nicely(list(set(line1)))
    line2 = sorted_nicely(list(set(line2)))

    s = [0 for i in range(len(line2))]

    for i , j in enumerate(line2):
        for k in A:
            if j == k[1]:
                s[i] += 1

    count = 0
    for m,a in enumerate(line1):
        print(f'{m+1}.', end = ' ')
        for i,b in enumerate(line2):
            for c in A:
                if [a,b] == c:
                    count += 1
            if i < len(line2)-1:
                print(f'P(F={a}|S=={b})={count}/{s[i]},', end = ' ')
            else:
                print(f'P(F={a}|S=={b})={count}/{s[i]}', end = ' ')
            count = 0
        print('\n')
```

In [31]:

```
A = [['F1','S1'],['F2','S2'],['F3','S3'],['F1','S2'],['F2','S3'],
     ['F3','S2'],['F2','S1'],['F4','S1'],['F4','S3'],['F5','S1']]
compute_conditional_probabilites(A)
```

```
1. P(F=F1|S==S1)=1/4, P(F=F1|S==S2)=1/3, P(F=F1|S==S3)=0/3

2. P(F=F2|S==S1)=1/4, P(F=F2|S==S2)=1/3, P(F=F2|S==S3)=1/3

3. P(F=F3|S==S1)=0/4, P(F=F3|S==S2)=1/3, P(F=F3|S==S3)=1/3

4. P(F=F4|S==S1)=1/4, P(F=F4|S==S2)=0/3, P(F=F4|S==S3)=1/3

5. P(F=F5|S==S1)=1/4, P(F=F5|S==S2)=0/3, P(F=F5|S==S3)=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 [32]:

```
S1= "the first column F will contain only 5 unique values"
S2= "the second column S will contain only 3 unique values"

def string_features(S1, S2):
    s1 = list(set(S1.split(' ')))
    s2 = list(set(S2.split(' ')))
    count = 0
    words1 = []
    words2 = []
    for i in s1:
        for j in s2:
            if i == j:
                count +=1
            if j not in s1:
                if j not in words2:
                    words2.append(j)
        if i not in s2:
            words1.append(i)

    return count, words1,  words2
```

```
a,b,c = string_features(S1, S2)
print(a)
print(b)
print(c)
```

```
7
['first', '5', 'F']
['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
```

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

In [33]:

```
from math import log

def compute_log_loss(A):
    n = len(A)
    total_loss =  0
    for i in A:
        total_loss+=(i[0]*log(i[1],10)+((1-i[0])*log(1-i[1],10)))

    avg_log_loss =-1 * total_loss/n
    return avg_log_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, 5))
```

```
0.42431
```

In [ ]: