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]
             [18 24 30 36 42]]

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

In [1]:

```
# taking input and printing second matrix
def input_matix_for_mul():
    '''This function takes input form the user and prints the matrix'''
    mat_Rows=int(input("Enter the number of rows of the matrix:  "))
    mat_Columns=int(input("Enter the number of columns of the matrix:  "))
    mat=[]
    for i in range (0,mat_Rows):
        mat.append([])
    for i in range (0,mat_Rows):
        for j in range(0,mat_Columns):
            mat[i].append(j)
            mat[i][j]=0
            print("Element in row:--> " ,i+1, " and column:--> ",j+1)
            mat[i][j]=int(input())
    return mat
#funtion for matrix multiplication
def matrix_mul(matrix_1,matrix_2):
    '''This funtion takes two matrices an argument and perform its multiplict

    if len(matrix_1[0])!=len(matrix_2):
        print("Multiplication not possible")
    else:
        print("Multiplication of the two matrices are : ")
        Res=[]    # creating an empty list to store result
        for i in range(len(matrix_1)):
            Res.append([])
        for i in range(len(matrix_1)):
            for j in range(len(matrix_2[0])):
                Res[i].append(j)
                Res[i][j]=0


        for p in range(len(matrix_1)):
            for q in range (len(matrix_2[0])):
                for r in range (len(matrix_2)):
                    Res[p][q]+=matrix_1[p][r]*matrix_2[r][q]
        for i in Res:
            print(i)


print("--------------------- First matrix---------------------")
matrix_1=input_matix_for_mul() #calling input_matix_for_mul fun to store firs
print(matrix_1)
print("****************************************")

print("--------------------- Second matrix---------------------")
matrix_2=input_matix_for_mul() ##calling input_matix_for_mul fun to store sec
print(matrix_2)
print("****************************************")

print("Result : -- ")
matrix_mul(matrix_1,matrix_2)  # getting the result

```

```
--------------------- First matrix---------------------
Enter the number of rows of the matrix:  2
```

```
Enter the number of columns of the matrix:  2
Element in row:-->  1  and column:-->  1
1
Element in row:-->  1  and column:-->  2
8
Element in row:-->  2  and column:-->  1
7
Element in row:-->  2  and column:-->  2
5
[[1, 8], [7, 5]]
**************************************
---------------------- Second matrix---------------------
Enter the number of rows of the matrix:  4
Enter the number of columns of the matrix:  2
Element in row:-->  1  and column:-->  1
5
Element in row:-->  1  and column:-->  2
9
Element in row:-->  2  and column:-->  1
7
Element in row:-->  2  and column:-->  2
5
Element in row:-->  3  and column:-->  1
6
Element in row:-->  3  and column:-->  2
5
Element in row:-->  4  and column:-->  1
8
Element in row:-->  4  and column:-->  2
3
[[5, 9], [7, 5], [6, 5], [8, 3]]
**************************************
Result : --
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 experiment
s.
f(100) > f(79) > f(45) > f(28) > f(27) > f(13) > f(10) > f(6) > f(5) > f
(0)
```

In [1]:

```
# write your python code here
# you can take the above example as sample input for your program to test
# it should work for any general input try not to hard code for only given in
# you can free to change all these codes/structure
from collections import Counter
from bisect import bisect
from random import uniform

A = [0,5,27,6,13,28,100,45,10,79]
def pick_a_number_from_list(A):
    sum=0
    d_dash=[]

    #step 1 : getting sum of all elements in the list
    for i in range(0,len(A)):
        sum+=A[i]
    #print(sum)

    #step 2: normalising the value using the sum
    for i in range(0,len(A)):
        d_dash.append(A[i]/sum)
    #print('D_dash : ',d_dash)

    #step 3: finding commulative sum
    d_doulbe_dash=[]
    c_sum=0
    for x in d_dash:
        c_sum+=x
        d_doulbe_dash.append(c_sum)
    #print('d_doulbe_dash : ',d_doulbe_dash)

    #getting a random number between 0 and 1
    r=uniform(0.0,1.0)
    #print('R : ',r)

    idx=bisect(d_doulbe_dash,r*d_doulbe_dash[-1])
    '''d_doulbe_dash[-1] return the max value in d_doulbe_dash.multiplyong it
    d_doulbe_dash[-1].Finally bisect returns the index of the value in d_doul
    we can find the value from A prsent at that index'''
    result=A[idx]
    return result

def sampling_based_on_magnitued():
    out_list=[]
    for i in range(100):#repeating the experiment 100 times
        out_list.append(pick_a_number_from_list(A))
    #print(out_list)
    Result=Counter(out_list)#counting the value and it occurence
    print(Result)

sampling_based_on_magnitued()

```

```
Counter({100: 31, 79: 23, 45: 17, 28: 13, 27: 6, 13: 5, 6: 2, 10: 2, 5: 1})
```

Q3: Replace the digits in the string with

Consider a string that will have digits in that, we need to remove all the characters which are 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 [1]:

```python
import re #18.6
# you can free to change all these codes/structure
# String: it will be the input to your program
def replace_digits():
    '''This function replaces the digit in the string with # '''
    String=input("Enter the string : ")
    lst=[]
    #storing the digits from the string into list
    for i in String:
        if i.isdigit():
            lst.append(i)
    #replacing degit with "#"
    for i in range(len(lst)):
        lst[i]="#"
    print("".join(lst)) #converting list into string

replace_digits()
```

```
Enter the string : #2a$#b%c%561#
####
```

Q4: Students marks dashboard

Consider the marks list of class students given in 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','student
6','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 [3]:

```
def display_dash_board(students, marks):

    mapped=zip(students,marks)#zipping the two lists
    mapped=dict(mapped) #converting lists into dict

    top_5_students=[] #declaring empty list
    least_5_students=[] #declaring empty list
    students_within_25_and_75=[]#declaring empty list

    sorted_list=sorted(mapped.items(),key=lambda x:x[1],reverse=True) #sortin

    for items in sorted_list[:5]:
        top_5_students.append(items)

    for items in sorted_list[-1:-6:-1]:
        least_5_students.append(items)

    sorted_list=sorted(mapped.items(),key=lambda x:x[1])#sorting in ascending
    n=len(marks)
    q1=int(n/4)
    q3=int(3*n/4)
    for items in sorted_list[q1:q3]:
        students_within_25_and_75.append(items)

    return top_5_students,least_5_students,students_within_25_and_75

students=['student1','student2','student3','student4','student5','student6','
marks = [45, 78, 12, 14, 48, 43, 47, 98, 35, 80]

top_5_students,least_5_students,students_within_25_and_75=display_dash_board(
print("a. Top 5 students are : ")
for i in range(len(top_5_students)):
    print(top_5_students[i][0]," : ",top_5_students[i][1])
print("----------------------------------------")

print("b. Least 5 students are : ")
for i in range(len(least_5_students)):
    print(least_5_students[i][0]," : ",least_5_students[i][1])
print("----------------------------------------")

print("c. Student between 25th percentile and 75th percentile")
for i in range(len(students_within_25_and_75)):
    print(students_within_25_and_75[i][0]," : ",students_within_25_and_75[i][
print("----------------------------------------")


```

```
a. Top 5 students are :
student8  :  98
student10  :  80
student2  :  78
student5  :  48
student7  :  47
----------------------------------------
b. Least 5 students are :
```

```
student3  :  12
student4  :  14
student9  :  35
student6  :  43
student1  :  45
----------------------------------------
c. Student between 25th percentile and 75th percentile
student9  :  35
student6  :  43
student1  :  45
student7  :  47
student5  :  48
----------------------------------------
```

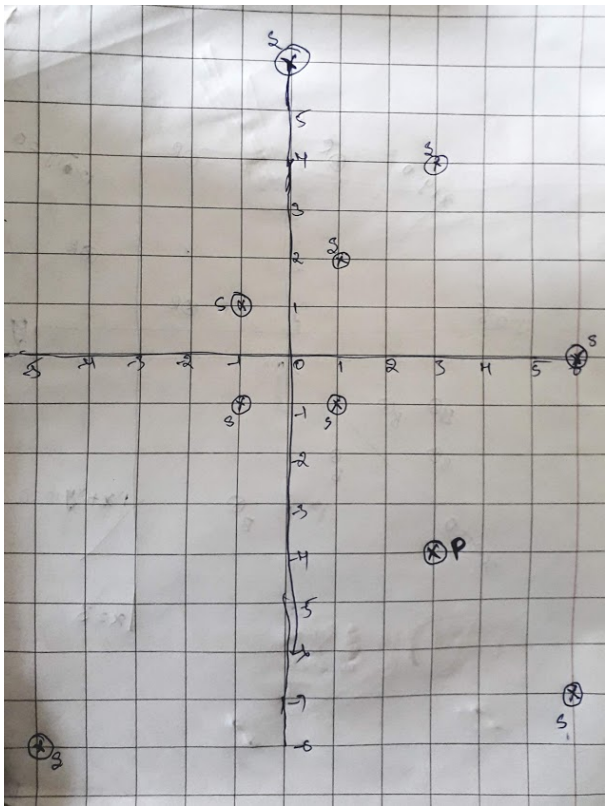
Q5: Find the closest points

Consider you are 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 defined 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 [13]:
```
import math

dis_lst=[]

def closest_points_to_p(S,P):
    '''This function returns 5 closeset points form a list of tuple containin
    for i in range(len(S)):
        d=((S[i][0]*P[0])+(S[i][1]*P[1]))/(math.sqrt(((S[i][0]**2)+(S[i][1]**
        #taking cos inverse and appeding the result of each iteration into an
        dis_lst.append(math.acos(d))
    mapped=zip(S,dis_lst)
    mapped=list(mapped)
    #sorting the list
    mapped.sort(key=lambda x:x[1])
    #unzipping the list to seperate points and angles obtained after applying
    points,distance=zip(*mapped)
    return points[:5]


S=[(1,2),(3,4),(-1,1),(6,-7),(0, 6),(-5,-8),(-1,-1),(6,0),(1,-1)]
P=(3,-4)
# calling the function and storing the result in a variable
closest_points=closest_points_to_p(S,P)
print(closest_points)


```

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

Q6: Find which line separates oranges and apples

Consider you are 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 format, i.e list of strings)

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

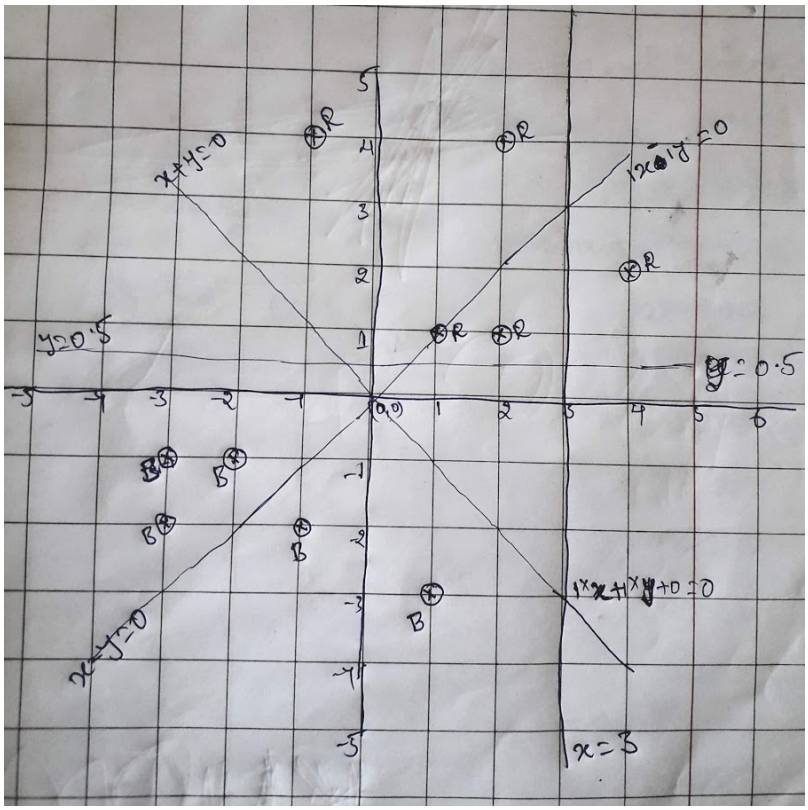
Your task here is to print "YES"/"NO" for each line given. You should print YES, if all the red points are one side of the line and blue points are on other side of the line, otherwise you should print NO.

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 [53]:
```
import math
import re

def i_am_the_one(red,blue,line):
    count_red=0
    count_blue=0

    #iterating through each line
    for li in line:
        #getting coefficients
        a= [float(coef.strip()) for coef in re.split('x|y', line)]
        #taking counts if red points line on either side of the line
        for i in range(0,len(Red)):
            x=(red[i][0]*a[0]+red[i][1]*a[1]+a[2])
            if(x>0):
                count_red+=1
            else:
                count_red-=1

        #taking counts if blue points line on either side of the line
        for i in range(0,len(Blue)):
            y=(blue[i][0]*a[0]+blue[i][1]*a[1]+a[2])
            if(y>0):
                count_blue+=1
            else:
                count_blue-=1

        #checking if points #lie on same side or not(counts will be equal if
        if abs(count_blue)==count_red:
            result="yes"
        else:
            result="no"
        #reinitializing counts for iterations
        count_blue=0
        count_red=0
    return result

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) # the returned value
```

```
yes
no
no
yes
```

Q7: Filling the missing values in the specified format

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+4
0)/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, 1
0, 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, 1
2, 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 [21]:
```
# took help from https://stackoverflow.com/questions/57179618/filling-the-mis
#still having doubt how to solve this question

def curve_smoothing(string):
    count=0
    value= 0
    a=string.split(",")
    tot_num = len(a)
    for i in a:
        if i=="_":
            count+=1
        else:
            value=int(i)+int(value)

    seq_num = int(value/tot_num)
    lst1=[]
    for i in range(tot_num):
        lst1.append(str(seq_num))
    print (','.join(lst1))

S1="_,_,_,24"
S2="40,_,_,_,60"
S3="80,_,_,_,_"
S4="_,_,30,_,_,_,50,_,_"
curve_smoothing(S1)
print('------------------------')
curve_smoothing(S2)
print('------------------------')
curve_smoothing(S3)
print('------------------------')
curve_smoothing(S4)
print('------------------------')
```

```
6,6,6,6
------------------------
20,20,20,20,20
------------------------
16,16,16,16,16
------------------------
8,8,8,8,8,8,8,8,8
------------------------
```

Q8: Find the probabilities

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 [22]:

```
#ref: https://stackoverflow.com/questions/57160252/find-conditional-probabili
from fractions import Fraction
A=[['F1','S1'],['F2','S2'],['F3','S3'],['F1','S2'],['F2','S3'],['F3','S2'],['
def compute_conditional_probabilites(F,S):
    num=0
    den=0
    for i in range(len(A)):
        if(A[i][1]==S):
            den=den+1
            if(A[i][0]==F):
                num=num+1

    print('P(F={}|S=={})='.format(F,S), Fraction(num,den))

for k in ['F1', 'F2', 'F3', 'F4', 'F5']:
    for m in ['S1', 'S2', 'S3']:
        compute_conditional_probabilites(k,m)


```

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

Q9: Operations on sentences

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

In [23]:

```
def string_features(S1, S2):
    # conveting the strings into list and the into sets
    set_1=set(S1.split())
    set_2=set(S2.split())
    #getting the number of comman words in set_1 and set_2
    a=len(set_1 & set_2)

    #getting Words in set_1 but not in set_2
    b=list(set_1-set_2)

    # getting Words in set_2 but not in set_1
    c=list(set_2-set_1
          )
    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
['first', 'F', '5']
['S', 'second', '3']
```

Q10: Error Function

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:
0.44982
```

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

In [24]:
```
import math as m

def compute_log_loss(A):
    n=len(A)
    loss=0
    for i in range(0,n):
        loss+=(A[i][0]*(m.log10(A[i][1])))+((1-A[i][0])*(m.log10(1-A[i][1])))
    loss=-(loss/n)
    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
loss = compute_log_loss(A)
print(loss)
```

```
0.42430993457031635
```

In [ ]:
```

```