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 37 44 51]]

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

```
def matrix_mul(A, B):
    rows_A = len(A)
    columns_A = len(A[0])
    rows_B = len(B)
    columns_B = len(B[0])
    print("Shape of A :({},{})".format(rows_A,columns_A))
    print("Shape of B :({},{})".format(rows_B,columns_B))
    #Check the validity of matricies for multiplication i.e columns of A == rows of B
```

```
        if columns_A == rows_B:
            #resultant matrix
            result = []
            #To travesre through row of A
            for i in range(rows_A):
                #To create list of list
                lst = []
                #To travesre through  columns of B
                for j in range(columns_B):
                    #To store sum of mutiplication of elements
                    sum = 0
                    #To access corresponding row and column elements
                    for k in range(columns_A):
                        sum+=A[i][k]*B[k][j]
                    lst.append(sum)
                result.append(lst)
            return result
        #If there is mismatch in columns of A and rows of B
        else:
            print("A*B = Not possible")


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

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

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

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

    matrix_mul(A,B)
```

```
     Shape of A :(2,2)
     Shape of B :(2,5)
     [[11, 14, 17, 20, 23], [23, 30, 37, 44, 51]]
```

Q2: Proportional Sampling - 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.
```

```
from random import uniform
def pick_a_number_from_list(A):

    probability = []
    cummulative = []
    sum = 0
    for num in A:
        sum+=num

    for num in A:
        probability.append(num/sum)

    sum=0
    for num in probability:
        sum+=num
        cummulative.append(sum)

    pick = uniform(0.0,1.0)
    for i in range(len(cummulative)):
        if cummulative[i] >= pick:
            return A[i]


def sampling_based_on_magnitued():
    A = [0, 5, 27, 6, 13, 28, 100, 45, 10, 79]
    for i in range(1,100):
        number = pick_a_number_from_list(A)
        print(number)

sampling_based_on_magnitued()
```

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: ####
```

```
def replace_digits(String):
    k=0
    result = ''
    for i in range(len(String)):
        if String[i].isdigit():
            k+=1
    for i in range(k):
        result +="#"
    return result


String = "#2a$#b%c%561#"
replace_digits(String)
```

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, 22, 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','
Marks = [45, 78, 12, 14, 48, 43, 47, 98, 22, 80]
a.
student8  98
student10 80
student2  78
student5  48
student7  47
b.
student3 12
student4 14
student9 22
student6 43
student1 45
```

```
c.
student9 22
student6 43
student1 45
student7 47
student5 48
```

```
import math
def map_name_marks(students,marks):
    lst = []
    if len(marks) == len(students):
        for i in range(len(marks)):
            lst.append((students[i],marks[i]))
    result = dict(lst)
    return result


def display_dash_board(students, marks):
    mapped = map_name_marks(students,marks)

    Asc_sort_mapped ={k:v for k,v in sorted(mapped.items(),key = lambda item:item[1])}
    top_5_students = []
    k=0
    for item in Asc_sort_mapped.items():
        if k<5:
            top_5_students.append(item)
        else:
            break
        k+=1
    dict(top_5_students)

    Desc_sort_mapped = {k:v for k,v in sorted(mapped.items(),key = lambda item:item[1],rev
    least_5_students = []
    k=0
    for item in Desc_sort_mapped.items():
        if k<5:
            least_5_students.append(item)
        else:
            break
        k+=1
    dict(least_5_students)

    p25 = 25*len(students)/100
    p75 = 75*len(students)/100

    if p25.is_integer():
        p25 = p25
    else:
        p25 = math.ceil(p25)-1
```

```
    if p75.is_integer():
        p75 = p75
    else:
        p75 = math.ceil(p75)-1

    k=0
    lst = []
    for item in Asc_sort_mapped.items():
        if k>=p25 and k<=p75:
            lst.append(item)
        k+=1


    students_within_25_and_75 = lst

    return top_5_students, least_5_students, students_within_25_and_75


Students = ['student1','student2','student3','student4','student5','student6','student7','
Marks = [45, 78, 12, 14, 48, 43, 45, 98, 22, 80]
top_5_students, least_5_students, students_within_25_and_75 = display_dash_board(Students,
print(top_5_students)
print(least_5_students)
print(students_within_25_and_75)
```

```
[('student3', 12), ('student4', 14), ('student9', 22), ('student6', 43), ('student1',
[('student8', 98), ('student10', 80), ('student2', 78), ('student5', 48), ('student1
[('student9', 22), ('student6', 43), ('student1', 45), ('student7', 45), ('student5',
```

Q5: Find the closest points

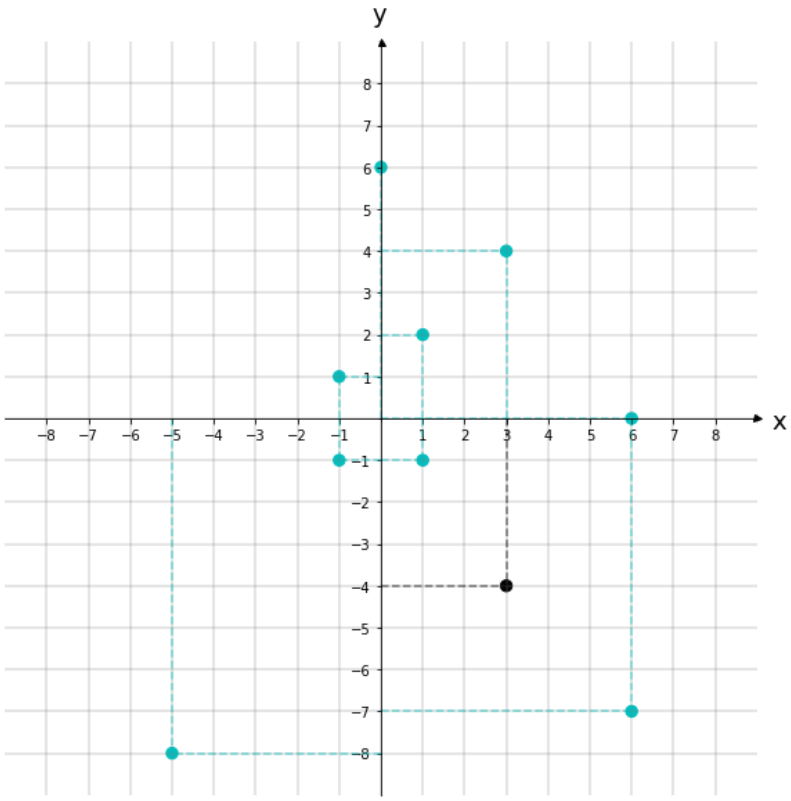
Consider you have given n data points in the form of list of tuples like S=[(x1,y1),(x2,y2),(x3,y3),(x4,y4),(x5,y5),..,(xn,yn)] and a point P=(p,q)

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)
```

Hint - If you write the formula correctly you'll get the distance between points (6,-7) and (3,-4) = 0.065

```
import math
def closest_points_to_p(S, P):
    lst = []
    E_distance_P = math.sqrt(P[0]**2 + P[1]**2)
    for point in S:
        dot_product = P[0]*point[0]+P[1]*point[1]
        E_distance_point =math.sqrt(point[0]**2 + point[1]**2)
        k = dot_product/(E_distance_point*E_distance_P)
        lst.append((point,math.acos(k)))
    lst = dict(lst)
    lst = {k:v for k,v in sorted(lst.items(), key =lambda item:item[1])}
    return lst

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)
k=0
```

```
for item in points:
    if k < 5:
        print("({},{})".format(item[0],item[1]))
    else:
        break
    k+=1
```

```
 (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)]
```

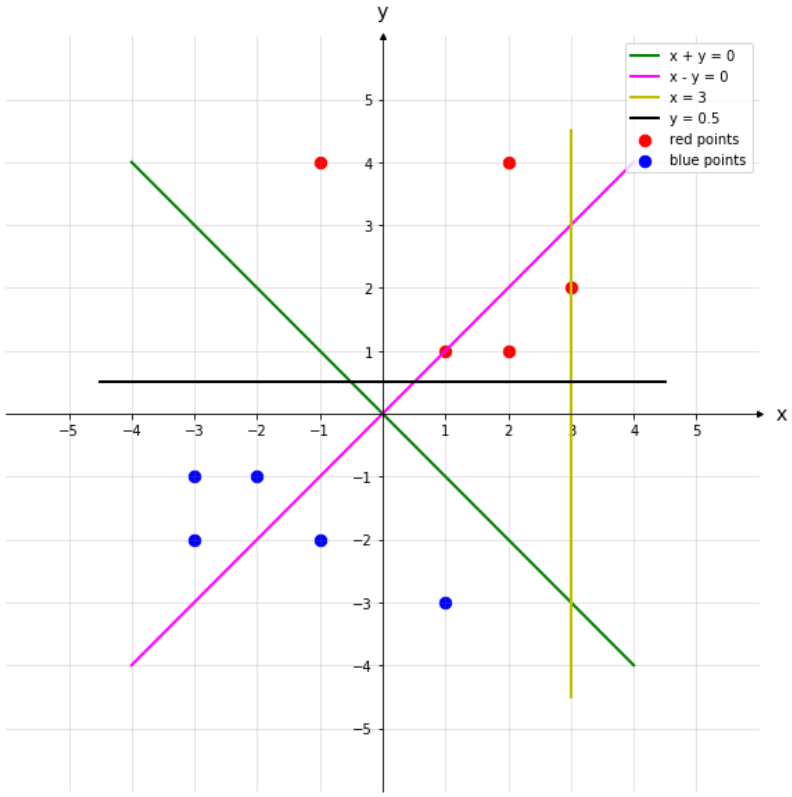
and set of line equations(in the string formate, i.e list of strings)

```
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

```
def i_am_the_one(red,blue,line):
    neg_flag_B = False; pos_flag_B = False
    X = 0; Y = 0; C = 0
    x=1;y=1

    for k in range(len(line)):
        if line[k] == 'x' or line[k] == 'X' :
            X = k
        elif line[k] == 'y' or line[k] == 'Y':
            Y =k

    x = float(line[0:X])
    y = float(line[X+1:Y])
    c = float(line[Y+1:len(line)])


    print("X = {},Y = {},C = {}".format(x,y,c))

    for point in blue:
        if neg_flag_B and pos_flag_B:
```

```
            return "No"

        k = point[0]*x + point[1]*y + c
        if k < 0 :
            neg_flag_B = True
        if k > 0 :
            pos_flag_B = True
        if k == 0 :
            return "No"

    if neg_flag_B == True and pos_flag_B == False:
        for point in red:
            k = point[0]*x + point[1]*y + c

            if k > 0:
                continue
            else:
                return "No"

    if pos_flag_B == True and neg_flag_B == False:
        for point in red:
            k = point[0]*x + point[1]*y + c

            if k < 0:
                continue
            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","1 x -1y +0","1 x+0 y-3 ","0 x+1 y  -0.5"]

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

     X = 1.0,Y = 1.0,C = 0.0
     Yes
     X = 1.0,Y = -1.0,C = 0.0
     No
     X = 1.0,Y = 0.0,C = -3.0
     No
     X = 0.0,Y = 1.0,C = -0.5
     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

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

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, _, _, _,
    b. now distribute the sum (10+50) missing values in between (10, 10, 12, 12, 12, 12,
    c. now we will distribute 12 to right side missing values (10, 10, 12, 12, 12, 12, 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 assign_value(lst,start_index,end_index,value):
    for i in range(start_index,end_index):
        lst[i] = value


def curve_smoothing(string):

    result = ""
    lst = string.split(',')
    start_index =-1; start = 0
    flag = False
    for k in range(len(lst)):
        if lst[k] == '_':
            flag = True
```

```
            if k == 0:
                start_index = k
                start = 0
            elif start < 0:
                start = float(lst[k-1])
                start_index = k-1
        if lst[k].isdecimal() and not flag:
            start_index = k
            start = float(lst[k])

        if lst[k].isdecimal() and flag :
            value=(start+float(lst[k]))/(k-start_index+1)
            assign_value(lst,start_index,k+1,value)
            flag = False
            start = -1
        if k == len(lst)-1 and flag:
            value = (start)/(k-start_index+1)
            assign_value(lst,start_index,k+1,value)

    for k in range(len(lst)):
        result +=str(int(lst[k]))
        if k != len(lst)-1:
            result +=","

    return result

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

```
 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

```
def compute_conditional_probabilites(A):
    count_S1 = 0
    count_S2 = 0
    count_S3 = 0

    F1_S1 = 0;F1_S2 = 0;F1_S3 = 0
    F2_S1 = 0;F2_S2 = 0;F2_S3 = 0
    F3_S1 = 0;F3_S2 = 0;F3_S3 = 0
    F4_S1 = 0;F4_S2 = 0;F4_S3 = 0
    F5_S1 = 0;F5_S2 = 0;F5_S3 = 0

    for i in range(len(A)):
        if A[i][1] == 'S1':
            count_S1+=1
        if A[i][1] == 'S2':
            count_S2+=1
        if A[i][1] == 'S3':
            count_S3+=1

    for i in range(len(A)):
        if A[i][0] == 'F1' and A[i][1] == 'S1':
            F1_S1+=1
        if A[i][0] == 'F2' and A[i][1] == 'S1':
            F2_S1+=1
        if A[i][0] == 'F3' and A[i][1] == 'S1':
            F3_S1+=1
        if A[i][0] == 'F4' and A[i][1] == 'S1':
            F4_S1+=1
        if A[i][0] == 'F5' and A[i][1] == 'S1':
            F5_S1+=1
        if A[i][0] == 'F1' and A[i][1] == 'S2':
            F1_S2+=1
        if A[i][0] == 'F2' and A[i][1] == 'S2':
            F2_S2+=1
        if A[i][0] == 'F3' and A[i][1] == 'S2':
            F3_S2+=1
        if A[i][0] == 'F4' and A[i][1] == 'S2':
            F4_S2+=1
        if A[i][0] == 'F5' and A[i][1] == 'S2':
            F5_S2+=1
        if A[i][0] == 'F1' and A[i][1] == 'S3':
            F1_S3+=1
        if A[i][0] == 'F2' and A[i][1] == 'S3':
            F2_S3+=1
        if A[i][0] == 'F3' and A[i][1] == 'S3':
            F3_S3+=1
        if A[i][0] == 'F4' and A[i][1] == 'S3':
            F4_S3+=1
```

```
            if A[i][0] == 'F5' and A[i][1] == 'S3':
                F5_S3+=1

        print("P(F=F1|S==S1)={}/{}".format(F1_S1,count_S1))
        print("P(F=F2|S==S1)={}/{}".format(F2_S1,count_S1))
        print("P(F=F3|S==S1)={}/{}".format(F3_S1,count_S1))
        print("P(F=F4|S==S1)={}/{}".format(F4_S1,count_S1))
        print("P(F=F5|S==S1)={}/{}\n".format(F5_S1,count_S1))

        print("P(F=F1|S==S2)={}/{}".format(F1_S2,count_S2))
        print("P(F=F2|S==S2)={}/{}".format(F2_S2,count_S2))
        print("P(F=F3|S==S2)={}/{}".format(F3_S2,count_S2))
        print("P(F=F4|S==S2)={}/{}".format(F4_S2,count_S2))
        print("P(F=F5|S==S2)={}/{}\n".format(F5_S2,count_S2))

        print("P(F=F1|S==S3)={}/{}".format(F1_S3,count_S3))
        print("P(F=F2|S==S3)={}/{}".format(F2_S3,count_S3))
        print("P(F=F3|S==S3)={}/{}".format(F3_S3,count_S3))
        print("P(F=F4|S==S3)={}/{}".format(F4_S3,count_S3))
        print("P(F=F5|S==S3)={}/{}".format(F5_S3,count_S3))
```

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

compute_conditional_probabilites(A)
```

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

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

     P(F=F1|S==S3)=0/3
     P(F=F2|S==S3)=1/3
     P(F=F3|S==S3)=1/3
     P(F=F4|S==S3)=1/3
     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']
```

```
def string_features(S1, S2):
    common = 0
    common1 =0
    match = False
    lst1 = S1.split(' ')
    lst2 = S2.split(' ')

    diff_S1 = []
    diff_S2 = []

    for word1 in lst1:
        match = False
        for word2 in lst2:
            if word1 == word2:
                common+=1
                match = True
        if match == False:
            diff_S1.append(word1)


    for word2 in lst2:
        match = False
        for word1 in lst1:
            if word1 == word2:
                match = True
        if match == False:
            diff_S2.append(word2)

    return common,diff_S1,diff_S2

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']
['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

$f(Y, Y_{score}) = -1 * \frac{1}{n}\Sigma_{foreach Y, Y_{score} pair}(Y log10(Y_{score}) + (1-Y) log10(1-Y_{score}))$

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
```

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

```
import math
def compute_pair_value(lst):
    result = lst[0]*math.log10(lst[1])+(1-lst[0])*math.log10(1-lst[1])
    return result

def compute_log_loss(A):
    n = len(A)
    values = list(map(compute_pair_value,A))
    result = 0

    for value in values:
        result +=value

    loss = -1*(1/n)*result

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

```
 0.42430993457031635
```