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]:

```
''' Initializing the matrix values '''
A  = [[1, 3, 4],[2, 5, 7],[5, 9, 6]]
B  = [[1, 0, 0],[0, 1, 0],[0, 0, 1]]

def matrix_mul(A,B):
    '''Get the length of matrix A'''
    A_ROWS = len(A)
    A_COLS = len(A[0])
    '''Get the length of matrix B'''
    B_ROWS = len(B)
    B_COLS = len(B[0])
#     print(' length of A matrx is {} and b matrix is {}'.format((A_ROWS,A_COLS),(B_
    ''' Before Initializing with 0 value '''
    AB_MTRX = [[0 for a in range(A_ROWS)] for b in range(B_COLS)]
    if A_COLS== B_ROWS:
        "Matrix mulplication is possible"
        ''' First iterate through each row in A matrix '''
        for ar in range(A_ROWS):
            ''' Loop through each element by column of B matrix'''
            for bc in range(B_COLS):
                '''Finally for multiplying loop through the B Row matrix'''
                for br in range(B_ROWS):
                    ''' simple math calculation of two number and summing it and sto
                    AB_MTRX[ar][bc] += A[ar][br] * B[br][bc]
        print(" A*B =",AB_MTRX)
    else:
        print(" A*B = Not possible")
matrix_mul(A, B)
```

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

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 [1]:

```
from random import uniform
# 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 input exa
''' Given array'''
A = [0, 5, 27, 6, 13, 28, 100, 45, 10, 79]
# you can free to change all these codes/structure

'''Calculating the normalized values for the given array'''
def calculate_normalized_values(A):
    total_sum = sum(A)
    normalized_values = []
    for i in range(len(A)):
        normalized_values.append(A[i]/total_sum)
    return normalized_values

''' Calculatting the cummulative values fro the given array'''
def calculate_cummulative_nomalized_sum(get_normalized_values):
    cummulative_nomalized_values = []
    cummulative_nomalized_values.append(get_normalized_values[0])

    for i in range(1,len(get_normalized_values)):
        tmp = cummulative_nomalized_values[i-1] + get_normalized_values[i]
        cummulative_nomalized_values.append(tmp)
    return cummulative_nomalized_values

''' Finding the propability for a random number'''
def pick_a_number_from_list(A):
    # your code here for picking an element from with the probability propotional to
    ''' Getting the cummulative values of the given array '''
    get_cummulative_nomalized_values = calculate_cummulative_nomalized_sum(calculate
    ele = uniform(0,1)
    for val in range(0, len(get_cummulative_nomalized_values)):
        if ele <= get_cummulative_nomalized_values[val]:
            return A[val]
''' Getting of uniform random number in range - (0,1) '''
print("Doing random expirement:\n")

def sampling_based_on_magnitued():
    sample_magintude_results = {}
    final_resutls = {}
    for item in range(0,len(A)):
        sample_magintude_results[A[item]] = 0
    for i in range(101):
        #ele     = uniform(0,1)
        get_number = pick_a_number_from_list(A)
        if get_number in A:
            sample_magintude_results[get_number] +=1
        #print('For the number {} the propability is {}'.format(i,get_number))
    final_resutls = dict(sorted(sample_magintude_results.items(), reverse=True))
    print(final_resutls)
''' Calling sampling_based_on_magnitued function '''
sampling_based_on_magnitued()
```

```
Doing random expirement:
```

```
{100: 28, 79: 22, 45: 15, 28: 9, 27: 12, 13: 6, 10: 3, 6: 1, 5: 5, 0:
0}
```

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 [3]:

```
given_string = '#2a$#b%c%561#'
''' Finding each letter is digit or not and finding the # count '''
def replace_digits(given_string):
    output = ''
    for i in given_string:
        if i.isdigit():
            output +='#'
    return output # modified string which is after replacing the # with digits

replace_digits(given_string)
```

Out[3]:

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

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','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','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 [4]:

```
'''
    - For zip sor reference at: https://www.geeksforgeeks.org/python-ways-to-sort-a-
'''
# initilizing the values
Students=['student1','student2','student3','student4','student5','student6','student
Marks = [45, 78, 12, 14, 48, 43, 47, 98, 35, 80]
# Sorting the values based on students marks
sorted_values = sorted(zip(Students,Marks),key = lambda arr:arr[1])
print(sorted_values)
'''
    For getting the top five students:
        - get the last five students details
        - reverse the array in order to get descending order
'''
get_top_five_students = sorted_values[-5:][::-1]
print('\na.')
for top_student_details in get_top_five_students:
    print(top_student_details[0],top_student_details[1])

'''
    For getting the least five students:
        - get the first five students details
'''
get_least_five_students = sorted_values[:5]
print('\nb.')
for least_student_details in get_least_five_students:
    print(least_student_details[0],least_student_details[1])

'''
    For getting the 25%-75% students:
        - get the 25% and 75% integer index values with respect to length of student
'''
total_students = len(sorted_values)
get_twenty_five_percententage_index  =  int(0.25*total_students)
get_seventy_five_percententage_index =  int(0.75*total_students)
#print(total_students,get_twenty_five_percententage_index,get_seventy_five_percenter
print('\nc.')
for student_details in sorted_values[get_twenty_five_percententage_index:get_seventy
    print(student_details[0],student_details[1])
```

```
[('student3', 12), ('student4', 14), ('student9', 35), ('student6', 4
3), ('student1', 45), ('student7', 47), ('student5', 48), ('student2',
78), ('student10', 80), ('student8', 98)]

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 [5]:

```
# 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 input exa

# you can free to change all these codes/structure
def display_dash_board(students, marks):
    '''
    For getting the top five students:
        - get the last five students details
        - reverse the array in order to get descending order
    '''
    get_top_five_students = sorted_values[-5:][::-1]

    '''
        For getting the least five students:
            - get the first five students details
    '''
    get_least_five_students = sorted_values[:5]

    '''
        For getting the 25%-75% students:
            - get the 25% and 75% integer index values with respect to length of stu
    '''
    total_students = len(sorted_values)
    get_twenty_five_percententage_index  =  int(0.25*total_students)
    get_seventy_five_percententage_index =  int(0.75*total_students)
    #print(total_students,get_twenty_five_percententage_index,get_seventy_five_perce
    students_within_25_and_75 = sorted_values[get_twenty_five_percententage_index:ge

    return get_top_five_students, get_least_five_students, students_within_25_and_75


''' initilizing the values and sorting the values according to the requirments'''
students=['student1','student2','student3','student4','student5','student6','student
marks = [45, 78, 12, 14, 48, 43, 47, 98, 35, 80]
mark_tasks = ['a','b','c']
sorted_values = sorted(zip(Students,Marks),key = lambda arr:arr[1])
# print(sorted_values)


''' Display the results of each task '''
top_5_students, least_5_students, students_within_25_and_75 = display_dash_board(stu
for indx,task in enumerate([top_5_students, least_5_students, students_within_25_anc
    print('\n'+mark_tasks[indx]+".")
    for student_details in task:
        print(student_details[0],student_details[1])
```

```

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
```

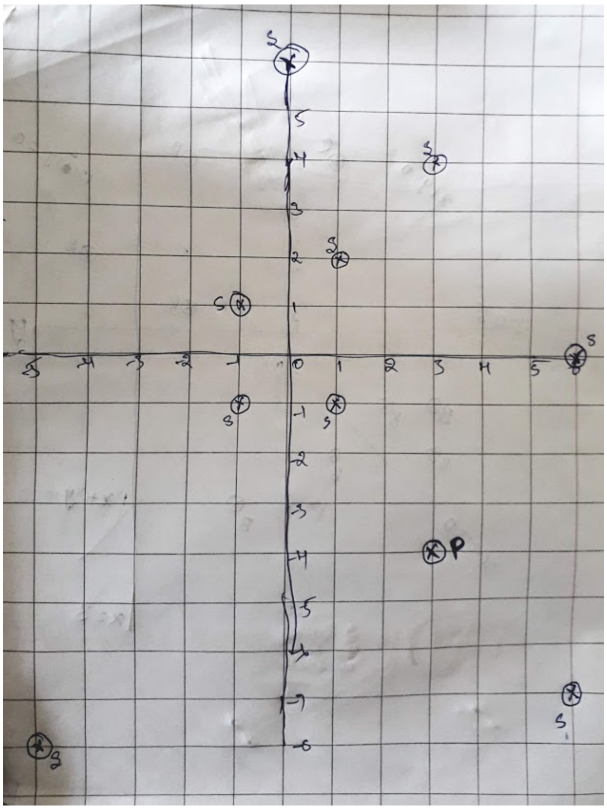
Q5: Find the closest points

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 [6]:

```
import math

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

''' Function to fint the closet points'''
def closest_points_to_p(S, P):
    cos_disntance = []
    for t in S:
        ''' Applying theformula '''
        numerator   = int(((t[0]*P[0])+(t[1]*P[1])))
        denominator = (math.sqrt(float( (t[0]**2)+(t[1]**2) ))* math.sqrt(float( (P[
        cos_disntance.append(math.acos(float(numerator/denominator)))
    '''To get the five cloest points, sorting it by zipping the values'''
    sorted_points = sorted(zip(S,cos_disntance),key = lambda arr:arr[1])
    return sorted_points[:5]



get_five_near_points  = closest_points_to_p(S, P)
print("Output:")
for pnt_details in get_five_near_points:
    print(pnt_details[0])
```

```
Output:
(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

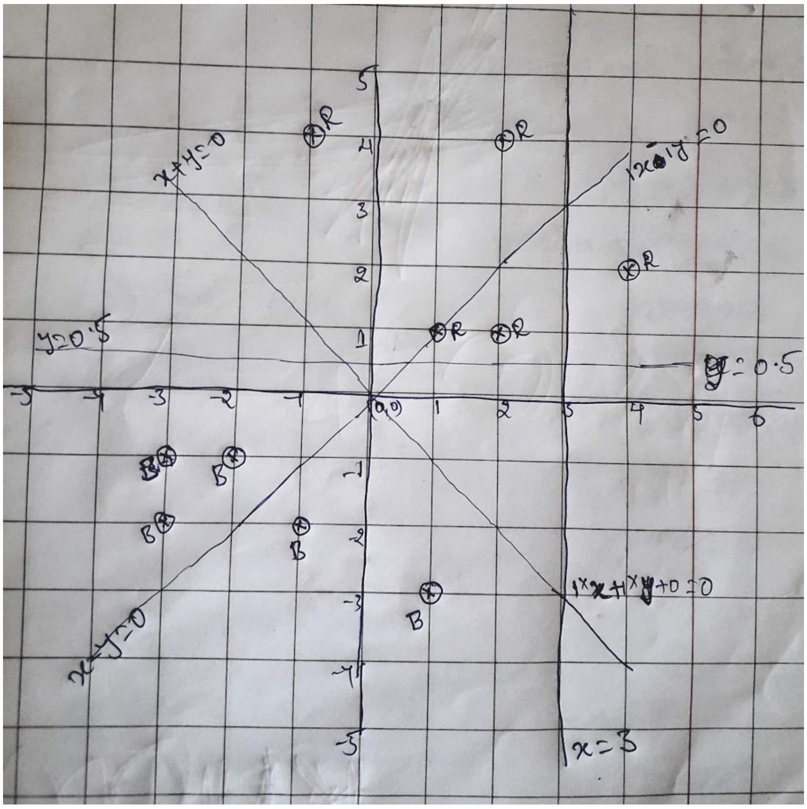
```
Red =[(R11,R12),(R21,R22),(R31,R32),(R41,R42),(R51,R52),..,(Rn1,Rn2)]
Blue=[(B11,B12),(B21,B22),(B31,B32),(B41,B42),(B51,B52),..,(Bm1,Bm2)]
```

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 [7]:

```
import math
import re
''' initializing the values '''
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"]

''' check all the points are on the same side'''
def check_points_on_same_side(line_coeff,points):
    postives,negatives = 0,0
    for each_point in points:
        ''' calculating the value'''
        get_value = (each_point[0]*line_coeff[0])+(each_point[1]*line_coeff[1])+(lir
        if get_value >0:
            postives +=1
        else:
            negatives +=1
    ''' finding whether all points are on the same side or not'''
    if max(postives,negatives)==len(points):
        return True
    else:
        return False

''' Get the co-efficents of lines'''
for line in Lines:
    all_coeff = [float(coef.strip()) for coef in re.split('x|y', line)]
    blue_results = check_points_on_same_side(all_coeff,Blue)
    red_results  = check_points_on_same_side(all_coeff,Red)
    ''' checking the two sets are on the both sides or not'''
    if blue_results and red_results:
        print("YES")
    else:
        print("NO")
```

```
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+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, 1
0, _, _, _, 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, 1
2, 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 [8]:

```
''' Function for calucluating the missing values'''
def set_values(initial_value,final_value,length,original_array):
    get_value = int((initial_value+final_value)/length)
    for i in range(length):
        original_array[i] = get_value
    return get_value,original_array

'''
    1. check first value if empty or not and allocate the respective value
    2. check the final value in the array is empty or not and allocating the respect
    3. if element is not empty the pass the start_value, final_value, length and req
    4. finally return the values
'''
def curve_smoothing(all_values):
    for indx,item in enumerate(all_values):
        ''' For inital value checking'''
        if item=="_" and indx==0:
            start_value = 0
            start_index = 0
            continue
        elif indx==0:
            start_value = int(item)
            start_index = 0
            continue

        ''' For Final value checking'''
        if item=="_" and indx==(len(all_values)-1):
            final_value = 0
            end_index     = (indx +1)
            total_elements= (indx +1)-(start_index)
            set_arr_vals= all_values[start_index:end_index]
#             print(start_value,final_value,total_elements,set_arr_vals)
            start_value, changed_array = set_values(start_value,final_value,total_el
            all_values[start_index:end_index] = changed_array
            continue

        ''' For the rest of the elements'''
        if item =='_':
            continue
        else:
            final_value   = int(item)
            end_index     = (indx +1)
            total_elements= (indx +1)-(start_index)
            set_arr_vals= all_values[start_index:end_index]
#             print(start_value,final_value,total_elements,set_arr_vals)
            start_value, changed_array = set_values(start_value,final_value,total_el
            all_values[start_index:end_index] = changed_array
            start_index = indx
    return all_values

''' Looping through all input string '''
all_input_strings = ["_,_,_,24","40,_,_,_,60","80,_,_,_,_","_,_,30,_,_,_,50,_,_"]
for cur_indx,each_string in enumerate(all_input_strings):
    all_values = each_string.split(',')
    print("\nInput"+str(cur_indx)+": ",*all_values, sep=", ")
    smoothed_values= curve_smoothing(all_values)
    print("Output"+str(cur_indx)+": ",*smoothed_values, sep=", ")
```

```
Input0: , _, _, _, 24
Output0: , 6, 6, 6, 6

Input1: , 40, _, _, _, 60
Output1: , 20, 20, 20, 20, 20

Input2: , 80, _, _, _, _
Output2: , 16, 16, 16, 16, 16

Input3: , _, _, 30, _, _, _, 50, _, _
Output3: , 10, 10, 12, 12, 12, 12, 4, 4, 4
```

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],[F
5,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 [9]:

```
def compute_conditional_probabilites(details):
    all_indivisual_probability  = {}
    all_conditional_probability = {}
    unique_values = []
    for each_list in details:

        ''' Finding the unique values of first columns'''
        if each_list[0] not in unique_values:
            unique_values.append(each_list[0])

        ''' Finding the propabilty of second columns'''
        ele = each_list[1]
        if ele in all_indivisual_probability.keys():
            all_indivisual_probability[ele] += 1
        else:
            all_indivisual_probability[ele] = 1

        ''' Finding the propabilty of first columns'''
        conditional_key = "F="+each_list[0]+"|S=="+each_list[1]
        if conditional_key in all_conditional_probability.keys():
            all_conditional_probability[conditional_key] += 1
        else:
            all_conditional_probability[conditional_key] = 1

    '''
        1. Looping through the unique values in first columns
        2. In loop through the second unique propabilites
        3. checking whether the propabilty for all propabilties when second column v

    '''
    for each_prop in unique_values:
        print("\nPrining propability for:"+each_prop)
        get_indivisual_keys = all_indivisual_probability.keys()
        for cond_prop in get_indivisual_keys:
            prop_key        = "F="+each_prop+"|S=="+cond_prop
            cond_prop_value = all_indivisual_probability[cond_prop]
            ''' If key exsits print the value probabilty'''
            if prop_key in  all_conditional_probability.keys():
                print("P("+prop_key+")="+str(all_conditional_probability[prop_key])+
            else:
                print("P("+prop_key+")=0/"+str(cond_prop_value))


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

```
Prining propability for:F1
P(F=F1|S==S1)=1/4
P(F=F1|S==S2)=1/3
P(F=F1|S==S3)=0/3

Prining propability for:F2
P(F=F2|S==S1)=1/4
P(F=F2|S==S2)=1/3
P(F=F2|S==S3)=1/3


Prining propability for:F3
```

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

Prining propability for:F4
P(F=F4|S==S1)=1/4
P(F=F4|S==S2)=0/3
P(F=F4|S==S3)=1/3

Prining propability for:F5
P(F=F5|S==S1)=1/4
P(F=F5|S==S2)=0/3
P(F=F5|S==S3)=0/3
```

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 [10]:

```
'''
    1. Decalre variables for storing required results
    2. First looping through first string and finding the common & non-common words
    3. Second looping through second string and finding the common & non-common word
    4. Applying set on common_words to get unique words
    5. Finally resturning the results

'''
def compute_string_comparisions(first_string,second_string):
    common_words            = []
    only_in_first_string  = []
    only_in_second_string = []
    first_string  = first_string.split()
    second_string = second_string.split()
    ''' checking the each word in first string with respect to second string '''
    for wrd in first_string:
        if wrd in second_string:
            common_words.append(wrd)
        else:
            only_in_first_string.append(wrd)
    ''' checking the each word in seconf string with respect to first string '''
    for wrd in second_string:
        if wrd in first_string:
            common_words.append(wrd)
        else:
            only_in_second_string.append(wrd)
    # your code
    return len(set(common_words)),only_in_first_string,only_in_second_string

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 = compute_string_comparisions(S1, S2)
''' FInally printing the results'''
print("\na.", a)
print("b.", b)
print("c.", c)
```

```

a. 7
b. ['first', 'F', '5']
c. ['second', 'S', '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
```

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

In [11]:

```
import math
''' Function for calculating loss'''
def compute_log_loss(errors_mapped_arr):
    # intitalising the variables, for calculating sum
    total = 0
    for item in errors_mapped_arr:
        ''' Applying the formula '''
        total += ( ( item[0]*math.log(item[1],10) ) + ((1.0-item[0])*math.log(1.0-it
    return (-1)*(1.0/len(errors_mapped_arr))*(total)


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)
''' Finally printing the loss '''
print(loss)
```

```
0.42430993457031635
```