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):
  number_col_A = len(A[0])
  number_row_B = len(B)
```

```
  # Creating empty list for result of size N initialized with 0's
  product = [[0 for _ in range(len(B[0]))] for _ in range(len(A))]

  if number_col_A == number_row_B:

    # Iterating over the rows of list A
    for i in range(len(A)):

      # Iterating over the columns of list B
      for j in range(len(B[0])):

        # Iterating over the rows of B
        for k in range(len(B)):
          product[i][j] += (A[i][k] * B[k][j])
  else:
    print('A*B = Not Possible')
  return product


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

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

# A = [[1,3,4],
#      [2,5,7],
#      [5,9,6]]
# B = [[1,0,0],
#      [0,1,0],
#      [0,0,1]]

result = matrix_mul(A, B)

# display result of two matrices A & B
for r in result:
  print(r)
```

```
    [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.
f(100) > f(79) > f(45) > f(28) > f(27) > f(13) > f(10) > f(6) > f(5) > f(0)
```

```
import random
from random import uniform

# Function to pick random number from list
def pick_a_number_from_list(A):
    cum_sum = []
    for i in A:
      # getting cummulative sum of each number to help proportionate probability
      cum_sum.append(i/sum(A))
    # Selecting random element with probability proportional to its magnitude
    selected_random_number = random.choices(A,weights=cum_sum,k=1)  # here K=1 will only sele
    return selected_random_number


def sampling_based_on_magnitued():
    A = [0, 5, 27, 6, 13, 28, 100, 45, 10, 79]
    # we are doing same experience 100 times with replacement
    for i in range(1,100):
        number = pick_a_number_from_list(A)
        print(number)

sampling_based_on_magnitued()
```

```
[79]
[27]
[100]
[79]
[100]
[79]
[45]
[100]
[28]
[28]
[27]
[27]
[100]
[28]
[79]
[45]
[28]
[100]
[100]
```

```
[100]
[79]
[79]
[100]
[13]
[79]
[79]
[79]
[79]
[100]
[100]
[28]
[45]
[28]
[79]
[45]
[100]
[45]
[79]
[100]
[79]
[100]
[45]
[28]
[45]
[5]
[28]
[79]
[6]
[79]
[28]
[10]
[13]
[45]
[45]
[79]
[100]
[45]
[100]
[79]
```

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: ####
```

```
import re
def replace_digits(String):
    replace_with = '#'
    print('The original string is :', String)
    # removing all special char and a-zA-Z char
    replaced_string = re.sub(r'[?|$|.|#|!|%|a-z|A-Z]',r'',String)
    # replaced digit with '#'
    return re.sub(r'\d',replace_with,replaced_string)


print("1. 234""\n""2. a2b3c4""\n""3. abc""\n""4. #2a$#b%c%561#""\n""5. Enter new string""\n""
print("\n")
choice = int(input('Enter your choice:'))
print("----------------------------------")
if(choice==1):
  print('The output string is: ',replace_digits('234'))
elif(choice==2):
  print('The output string is: ',replace_digits('a2b3c4'))
elif(choice==3):
  print('The output string is: ',replace_digits('abc'))
elif(choice==4):
  print('The output string is: ',replace_digits('#2a$#b%c%561#'))
elif(choice==5):
  custom = input()
  print('The output string is: ',replace_digits(custom))
elif(choice==6):
  exit()
else:
  print('Invalid choice. Enter between 1-6')
```

```
1. 234
2. a2b3c4
3. abc
4. #2a$#b%c%561#
5. Enter new string
6. Exit


Enter your choice:6
----------------------------------
```

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','stu
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
```

```
def get_top_or_least_marks(marks):
  student_list = []
  for i in range(len(marks)):
    if i <= 4:
      student_list.append(marks[i])
  return student_list;

# printing students and their marks
def print_students_marks(students_marks_dict):
  for k,v in students_marks_dict.items():
    print(k,v)
  print("----------------------")

# code to display dash board
def display_dash_board(students, marks):
    top_5_students = []
```

```
    # write code for computing top top 5 students
    top_marks = sorted(zip(marks,students),reverse=True)
    # getting top 5 students who got top marks
    top_5_students = get_top_or_least_marks(top_marks)
    # converting list to dict to get key value
    top_5_students = dict(top_5_students)
    # swap student and mark in dict
    top_5_students = dict((v,k)for k,v in top_5_students.items())

    # write code for computing top least 5 students
    least_marks = sorted(zip(marks,students),reverse=False)
    least_5_students = get_top_or_least_marks(least_marks)
    least_5_students = dict(least_5_students)
    least_5_students = dict((v,k) for k,v in least_5_students.items())

    #  code for computing students within 25 and 75 percentile
    max_mark = max(marks)
    min_mark = min(marks)
    diff_mark = max_mark - min_mark
    per_25 = diff_mark * 0.25
    per_75 = diff_mark * 0.75
    marks_in_ascending = dict(sorted(zip(marks,students),reverse=False))
    # getting all the students within 25 and 75 percentile in dict
    students_within_25_and_75 = {
        students : marks
        for(marks,students)
        in marks_in_ascending.items()
        if per_25 <= marks <= per_75
    }

    return top_5_students, least_5_students, students_within_25_and_75

students = ['student1','student2','student3','student4','student5','student6','student7','stu
marks = [45, 78, 12, 14, 48, 43, 47, 98, 22, 80]

top_5_students, least_5_students, students_within_25_and_75 = display_dash_board(students, ma

# print top 5 students
print_students_marks(top_5_students)
# print top least 5 students
print_students_marks(least_5_students)
# print students_within_25_and_75 percentile
print_students_marks(students_within_25_and_75)
```

```
 student8 98
 student10 80
 student2 78
 student5 48
 student7 47
 -----------------------
 student3 12
```

```
student4 14
student9 22
student6 43
student1 45
----------------------
student9 22
student6 43
student1 45
student7 47
student5 48
----------------------
```

Q5: Find the closest points

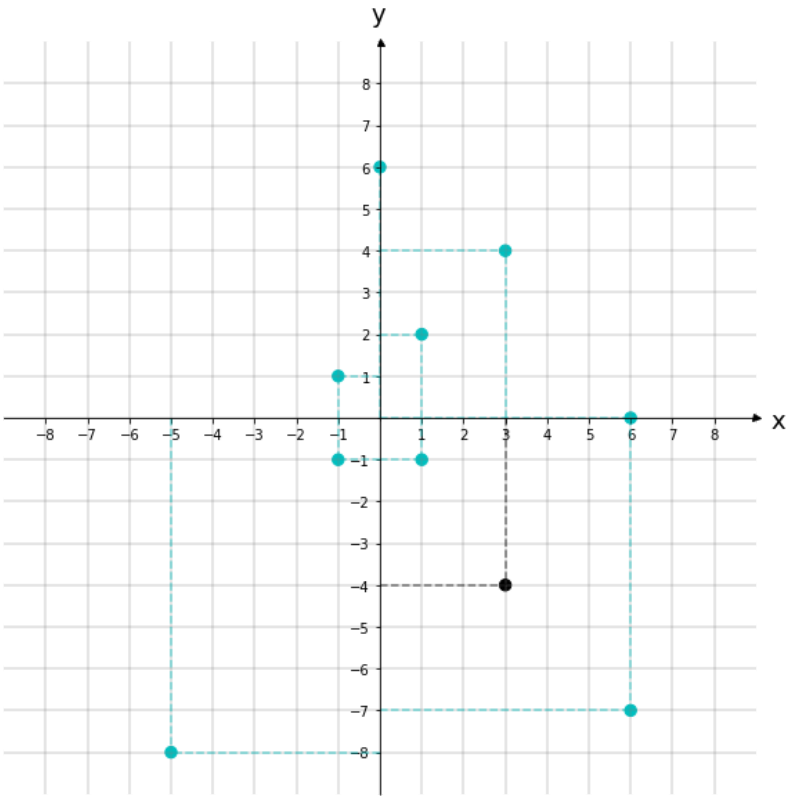
Consider you have given n data points in the form of list of tuples like S=[(x1,y1),(x2,y2),(x3,y3),(x4,y4),(x5,y5),..,(xn,yn)] and a point P=(p,q)

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 $cos^{-1}\left(\frac{(x\cdot p+y\cdot q)}{\sqrt{(x^2+y^2)}\cdot\sqrt{(p^2+q^2)}}\right)$

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):
  # setting value for p and q points
  p,q = P[0],P[1]
  result = []
  # code to compute the cosine distance between two points (x,y) and (p,q)
  for x,y in S:
    # calculating numerator values
    numerator = x*p + y*q
    # calculating denominator values
    denominator = math.sqrt(x*x + y*y) * math.sqrt(p*p + q*q)
    # divding num/den to get cosine distance
    result.append(math.acos(numerator/denominator))
  return result

S= [(1,2),(3,4),(-1,1),(6,-7),(0, 6),(-5,-8),(-1,-1),(6,0),(1,-1)]
P= (3,-4)
# getting all the data points distance from p,q points
data_points = closest_points_to_p(S,P)
# getting closest data points
closest_points_to_p = sorted(zip(S,data_points), key=lambda i:i[1])
for points,distance in closest_points_to_p[:5]:
  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

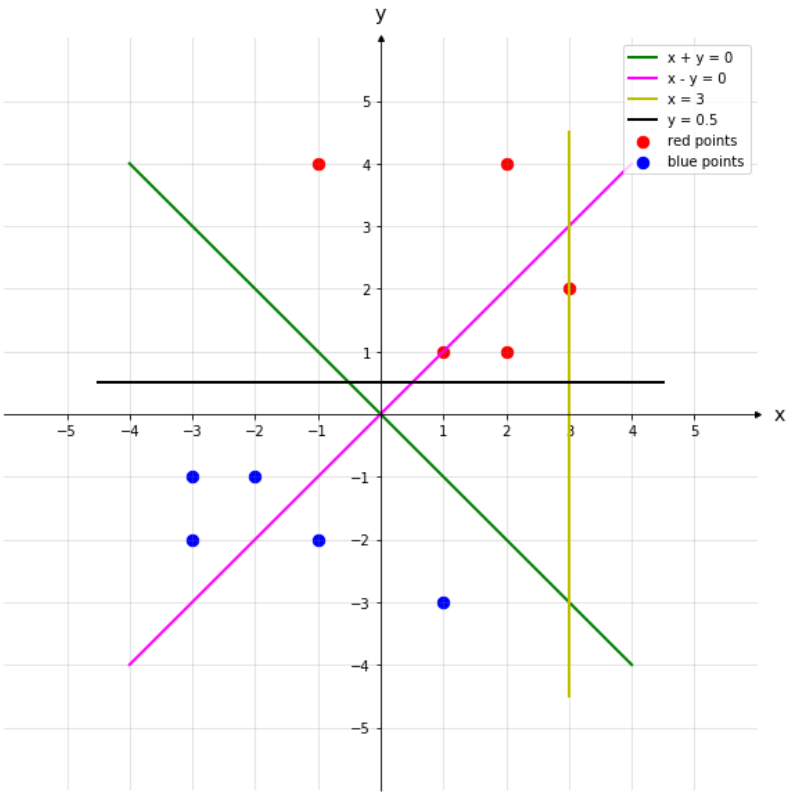
```
Red =[(R11,R12),(R21,R22),(R31,R32),(R41,R42),(R51,R52),..,(Rn1,Rn2)]
Blue=[(B11,B12),(B21,B22),(B31,B32),(B41,B42),(B51,B52),..,(Bm1,Bm2)]
```

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
```

```
import math
import matplotlib.pyplot as plt
```

```
import re
def get_side_of_line(a,b,c,p):
  # Equation to calculate intercept using ax+by+c
  calculate_itercept = (a*p[0]) + (b*p[1]) + c
  if calculate_itercept > 0:
    return 0
  elif calculate_itercept < 0:
    return -1
  elif calculate_itercept == 0:
    return 0
  else:
    return -2

# you can free to change all these codes/struct
def i_am_the_one(red,blue,a,b,c):
  # check for which side our 1st data point is on
  red_sign = get_side_of_line(a,b,c,red[0])
  blue_sign = get_side_of_line(a,b,c,blue[0])

  # checking the sides of all red data points
  for i in range(len(red)):
    if red_sign != get_side_of_line(a,b,c,red[i]):
      return 'NO'

  # checking the sides of all blue data points
  for j in range(len(blue)):
    if blue_sign != get_side_of_line(a,b,c,blue[j]):
      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:
  # extracting the coefficient from the lines of string
    a, b, c = [float(coef.strip()) for coef in re.split('x|y', i)]
    # getting the result whether all points belong to either side
    result = i_am_the_one(Red, Blue, a,b,c)
    print(result)
```

```
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

Ex 2: 40, _, _, _, 60 ==> (60+40)/5,(60+40)/5,(60+40)/5,(60+40)/5,(60+40)/5 ==> 20, 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 di

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
```

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
```

```
from matplotlib.transforms import DEBUG

import re

def curve_smoothing(string):
  # spliting string to get index
    split_string = string.split(',')
    # new_index is required after every loop to start again from next to new_index
```

```
    new_index = 0
    # new_value is required after every loop to start again to distribute values till next di
    new_value = 0

    # This loop will fill the missing values with distributed value till any digit found
    for i in range(len(split_string)):
      if str(split_string[i]).isdigit():
        for j in range(i+1):
          # getting distributed value and filling it in missing place
           split_string[j] = int(split_string[i])//(i+1)
        new_index = i
        new_value = int(split_string[i])
        break

    # This loop will fill all the missing value after any digit found till next digit found
    for number in split_string:
      if isinstance(number,(int)):
        for i in range(new_index+1, len(split_string)):
          if split_string[i].isdigit():
            # calculating distributed sum i.e(10+50) to fill in between missing values
            temp = (new_value + int(split_string[i]))//(i-new_index+1)
            # filling distributed value in between missing values
            for j in range(new_index, i+1):
              split_string[j] = temp
            new_index = i
            new_value = int(split_string[i])

    # Now will destribute last new_value till right missing places
    # for this part of loop taken help from StackOverFlow
    try:
      for i in range(new_index+1,len(split_string)):
        if not(split_string[i].isdigit()):
          # counting all missing values to fill in
          count_missing_value = split_string.count('_')
          break
      # distributing new value to remaining missing places
      temp = new_value // (count_missing_value + 1)
      for i in range(new_index, len(split_string)):
        split_string[i] = temp
    except:
      pass
    return split_string


Input1 = "_,_,_,24"
Input2 = "40,_,_,_,60"
Input3 = "80,_,_,_,_"
Input4 = "_,_,30,_,_,_,50,_,_"
smoothed_values= curve_smoothing(Input4)
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,F,S):
  # initializing numerator and denominator values to 0
  numerator,denominator = 0,0

  # function to calculate conditional probability of lists of list
  for i in range(len(A)):
    if A[i][1] == S:
      denominator += 1;
      if A[i][0] == F:
        numerator += 1;
  return numerator, denominator

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

# Five unique values of column F
First_Column = ['F1','F2','F3','F4','F5']
# Thre unique values of column S
```

```
Second_Column = ['S1','S2','S3']

result = []
# Iterating over the First Column values
for F in range(len(First_Column)):
  # Iterating over the Second Column values
  for S in range(len(Second_Column)):
    numerator, denominator = compute_conditional_probabilites(A,First_Column[F],Second_Column
    # Displaying probability of lists of list in specified formate
    print(('P(F={}|S=={})={}/{}'.format(First_Column[F], Second_Column[S], str(numerator), st
```

```
P(F=F1|S==S1)=1/4
P(F=F1|S==S2)=1/3
P(F=F1|S==S3)=0/3
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)=0/4
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)=0/3
P(F=F4|S==S3)=1/3
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']
```

```
def string_features(S1, S2):
  a = 0  # Common Words between S1 and S2
  b = [] # Words in S1 but not in S2
```

```
    b = []  # words in S1 but not in S2
    c = []  # Words in S2 but not in S1

    # converting string into list of string
    S1_list = list(S1.split(" "))
    S2_list = list(S2.split(" "))

    # Number of common words between S1 and S2
    for i in range(len(S1_list)):
      for j in range(len(S2_list)):
        if S1_list[i] == S2_list[j]:
          a += 1

    # Words in S1 but not in S2
    for i in S1_list:
      if i not in S2_list:
        b.append(i)

    # words in S2 but not in S1
    for j in S2_list:
      if j not in S1_list:
        c.append(j)

    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.',a)
print('b.',b)
print('c.',c)
```

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

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

```
import math
def compute_log_loss(A):
    loss = 0
    # length of a List A
    n = len(A)
    for i in range(len(A)):
      # Equation to get the loss of function f(Y,Yscore)
      loss += (A[i][0] * math.log10(A[i][1])) + ((1 - A[i][0]) * (math.log10(1-A[i][1])))
    result = (-1 * loss)/n
    return result

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(round(loss,7))
```

```
0.4243099
```

✓ 0s completed at 7:15 PM