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
```

```
import sys


def dot_mul(a, b, i, j):
  """
  Call this function to find each element of the resultant matrix by sending the arrays and the

  Parameters
  ----------
```

```
  a,b - input arrays
  i,j - current position

  """

  #i-th row in 'a' and j-th column in 'b'
  len_ax = len(a)
  len_ay = len(a[0])
  len_bx = len(b)
  len_by = len(b[0])

  #a[i][0] to a[i][len_ay-1]
  #b[j][0] to b[j][len_bx-1]
  sum = 0
  for ai, aj in zip(range(len_ay), range(len_bx)):  #iterating together both the arrays
    sum += a[i][ai]*b[aj][j]

  return sum

def matrix_mul(A, B):
  """
    Proper matrix multiplication by using dot product
  """
  if type(A) != list or type(B) != list:
    raise RuntimeError("A and B must be of list types")

  len_x1 = len(A)
  len_y1 = len(A[0])
  len_x2 = len(B)
  len_y2 = len(B[0])

  if len_y1 != len_x2:
    raise ValueError("Matrix is of mis-shape")

  result = [[0 for j in range(len_y2)] for i in range(len_x1)]
  for i in range(len_x1):
    for j in range(len_y2):
      #dot product of row_i in A and col j in B
      result[i][j] = dot_mul(A, B, i, j)

  print(result)

  return

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


A = [[1, 3, 4], [2,5,7], [5,9, 6]]
B = [[1,0,0], [0,1,0], [0,0,1]]
matrix_mul(A,B)

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

```
try:
  matrix_mul(A,B)
except:
  print(sys.exc_info()[1])
```

```
    [[11, 14, 17, 20, 23], [23, 30, 37, 44, 51]]
    [[1, 3, 4], [2, 5, 7], [5, 9, 6]]
    Matrix is of mis-shape
```

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)
```

```
from random import uniform

A = [0,5,27,6,13,28,100,45,10,79]
times = 1000

def pick_a_number_from_list(A):
    # your code here for picking an element from with the probability propotional to its magnit
    rand = uniform(0,1)
    A.sort() #important for the below logic to work
    tot = sum(A)
    s = [0]
    index = -1

    ## IN REF TO : https://www.quora.com/How-can-we-pick-an-element-from-an-array-with-probabil
    for i in range(1, len(A)):
      s.append(s[i-1]+A[i])
      if (s[i]/tot > rand): #it works because we keep increasing the cumulative sum each time
        index = i
        break

    return A[index]

def sampling_based_on_magnitued():
    picked_numbers = []
    for i in range(1,times+1):
        number = pick_a_number_from_list(A)
        picked_numbers.append(number)
    return picked_numbers

picked_numbers = sampling_based_on_magnitued()
```

```
print("After testing {} times".format(times))
A.sort() #sorting the array A
for item in A: #checking our logic by printing the frequency of occurrence of each element
  count = picked_numbers.count(item)
  print("{} occurred {}-times".format(item, count))
```

```
After testing 1000 times
0 occurred 0-times
5 occurred 17-times
6 occurred 18-times
10 occurred 25-times
13 occurred 41-times
27 occurred 96-times
28 occurred 82-times
45 occurred 154-times
79 occurred 239-times
100 occurred 328-times
```

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(input):
  p = re.compile('[^0-9.]') #finding the non-digit elements
  s = p.sub('', input)      #replacing those with empty string
  p=re.compile('[0-9.]')    #finding the digits elements
  return p.sub('#', s)      #replacing those with #

inputs = ["234", "a2b3c4", "abc", "#2a$#b%c%561#"]
for input in inputs:
  print(replace_digits(input))
```

```
###
###

####
```

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','stud
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
```

```
def display_dash_board(students, marks):

  data = [[student, mark] for (student,mark) in zip(students,marks)]
  data.sort(key = lambda x: x[1]) #sorting based on the marks
  N = len(data)

  #interquartile range code fixed as below logic
  max_mark = data[-1][1]
  min_mark = data[0][1]
  diff = max_mark - min_mark
  p25 = diff*0.25
  p75 = diff*0.75

  #Fetchting top5 students with only the name and mark
  top_5_students = [(x[0],x[1]) for x in data[-5:]]
  top_5_students.reverse()

  #Fetchting least 5 students with only the name and mark
  least_5_students = [(x[0],x[1]) for x in data[0:5]]
```

```
  #Fetchting 25 to 75 percentile students with only the name and mark
  interquartile = list(filter(lambda x: x[1] >= p25 and x[1]<=p75, data))
  students_within_25_and_75 = [(x[0],x[1]) for x in interquartile ]

  return top_5_students, least_5_students, students_within_25_and_75


students=['student1','student2','student3','student4','student5','student6','student7','student
marks = [45, 78, 12, 14, 48, 43, 47, 98, 35, 80]


#students=['student1','student2','student3','student4','student5','student6','student7','studen
#marks = [45, 78, 12, 14, 48, 43, 47, 98, 35, 80, 10, 20, 99, 15, 56]

def proper_print(lst):
  """
  Just to print the output in the way shown in the question
  """
  for ele in lst:
    print(ele[0], ' ', ele[1])

top_5_students, least_5_students, students_within_25_and_75 = display_dash_board(students, mark
print("Top 5 students : ")
proper_print(top_5_students)
print("\nLeast 5 students : ")
proper_print(least_5_students)
print("\nStudents in 25 to 75 percentile range : ")
proper_print(students_within_25_and_75)
```

```
Top 5 students :
student8   98
student10   80
student2   78
student5   48
student7   47

Least 5 students :
student3   12
student4   14
student9   35
student6   43
student1   45

Students in 25 to 75 percentile range :
student9   35
student6   43
student1   45
student7   47
student5   48
```

Q5: Find the closest points

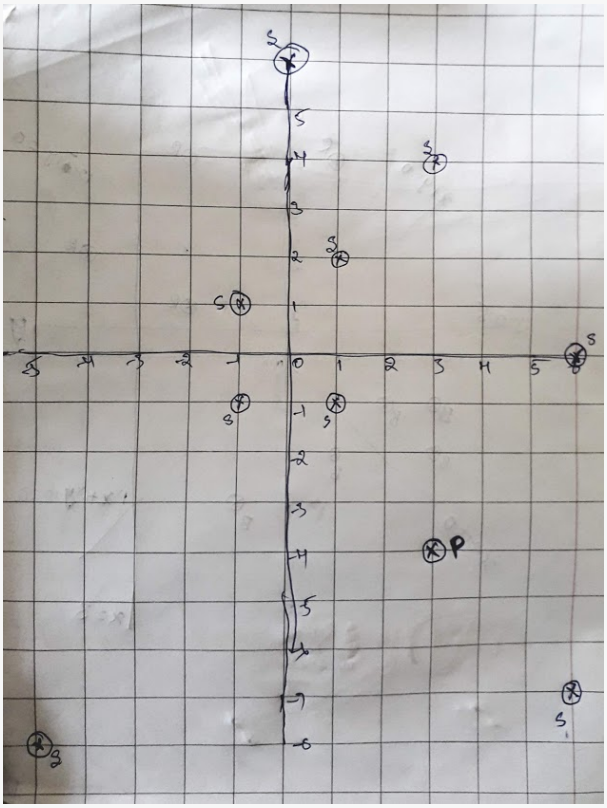
consider you have given n data points in the form of list of tuples like S=[(x1,y1),(x2,y2),(x3,y3),(x4,y4),(x5,y5),.., (xn,yn)] and a point P=(p,q)

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)
```

```
import math

def cosine_dist(s,p):
  """
  Find the cosine distance

  Parameters:
  s - tuple of 2 integers
  p - tuple of 2 integers
  """
  sx = s[0]
  sy = s[1]
  px = p[0]
  py = p[1]
```

```
  return math.acos((sx*px+sy*py)/((math.sqrt(sx*sx+sy*sy))*(px*px+py*py)))

# here S is list of tuples and P is a tuple ot len=2
def closest_points_to_p(S, P):

  # Find the cosine distance and add it to the list
  S_with_dist = [[point, cosine_dist(point, P)] for point in S ]

  #Sort the list with the cosine distance (which is in the index-1)
  S_with_dist.sort(key = (lambda x: x[1]))

  #Return only the points
  return [point[0] for point in S_with_dist[0:5]]

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)
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)]
```

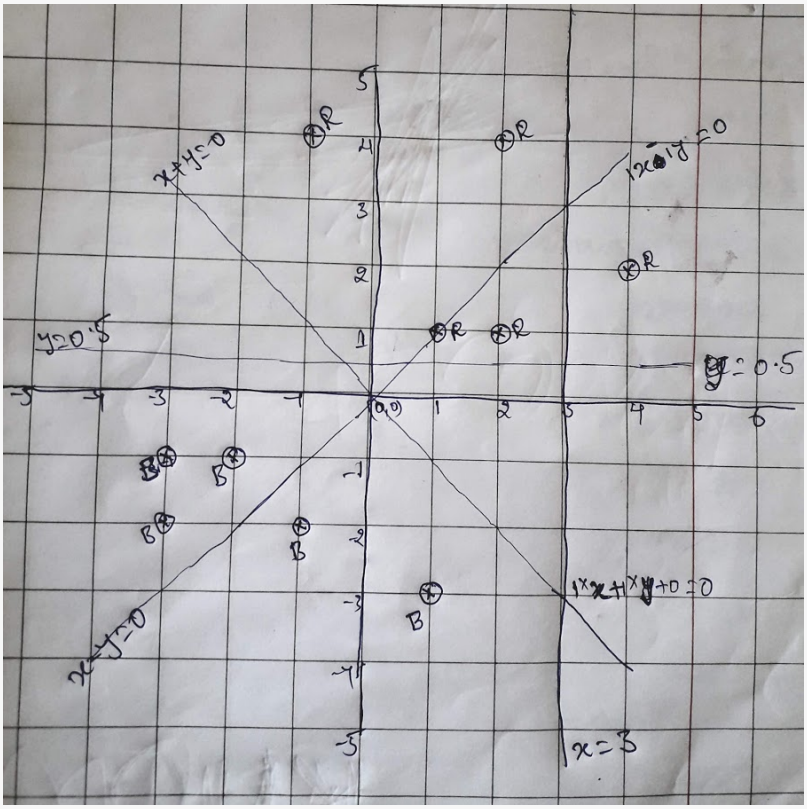
and set of line equations(in the string formate, i.e list of strings)

```
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
from functools import reduce

def compare(x,y):
  """
  A - if all are true
  B - if all are false
  C - if combination of true and false
  """
  if (x==True and y==True) or (x=='A' and y==True):
    return 'A'
  if (y==False and x==False) or (x=='B' and y==False):
    return 'B'
  return 'C'

def checkSign(point, line):
  """
  Apply the point on the line and get the sign
  """
  return line[0]*point[0]+line[1]*point[1]+line[2]

def i_am_the_one(red,blue,line):
  r = []
  b = []
```

```
  #get the sign
  for point in red:
    r.append(checkSign(point, line))
  for point in blue:
    b.append(checkSign(point, line))

  #True if a number is +ve, False if a number is -v2
  #Then, if all numbers are +ve, reduce it to 'A', in case of -ve, reduce it to 'B'. Otherwise
  r = reduce(compare, [True if a > 0 else False for a in r])
  b = reduce(compare, [True if a > 0 else False for a in b])

  #Compare based on the signs of all points of Red and Blue
  if (r == 'A' and b == 'B') or (r == 'B' and b == 'A'):
    return "YES"
  return "NO"

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"]
Eqn = []

for line in Lines: #Splitting each equation into a tuple of 3 values of (x,y,c)
  eqn = line.split("x")
  x = float(eqn[0])
  eqn = eqn[1].split("y")
  y = float(eqn[0])
  z = float(eqn[1])
  eqn = (x,y,z)
  Eqn.append(eqn)

for i in Eqn:
    yes_or_no = i_am_the_one(Red, Blue, i)
    print(yes_or_no)
```

```
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 dis

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
```

```
# 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 strings

def fill(num, ran, s):
  """
  Fill the range with the values
  """
  to_fill = s/(ran[1]-ran[0]+1)
  for i in range(ran[0], ran[1]+1):
    num[i] = to_fill

def curve_smoothing(string):
  s = string.split(",")
  num = [int(part) if part != "_" else None for part in s] #split the string into list of numbe

  s = 0 #sum for a range
  filling = False
  ran=[-1, -1] #track what range of the list to fill

  for i in range(len(num)):
    if filling == False : #start of a range
      ran[0] = i
      s = 0 if num[0] is None else num[i]
      filling = True
    elif filling and num[i] != None: #end of a range
      ran[1] = i
      s = s + num[i]
      fill(num, ran, s)
```

```
      filling = True
      ran[0] = i
      s = num[i]
      i-=1
    else: #blank space, so extending the range to fill
      ran[1] = i

  if filling:
    fill(num, ran, s)
  return num


S=  ["_,_,30,_,_,_,50,_,_", "_,_,_,24", "40,_,_,_,60", "80,_,_,_,_"]
for s in S:
  print(s)
  smoothed_values= curve_smoothing(s)
  print(smoothed_values)
```

```
    _,_,30,_,_,_,50,_,_
    [10.0, 10.0, 12.0, 12.0, 12.0, 12.0, 4.0, 4.0, 4.0]
    _,_,_,24
    [6.0, 6.0, 6.0, 6.0]
    40,_,_,_,60
    [20.0, 20.0, 20.0, 20.0, 20.0]
    80,_,_,_,_
    [16.0, 16.0, 16.0, 16.0, 16.0]
```

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
```

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

  result = [] #2d list to have the result of all conditional probabilities
  for i in range(len(F)):
    result.append([])

  for s in S:
    temp = [a for a in A if a[1] == s ]       #getting the list of particular 'S'
    count = len(temp)                         #getting the count
    for index,f in enumerate(F, start=0):
        f_count = temp.count([f,s])           #getting the occurrences of F with the particula
        result[index].append((f_count, count))#Saving the result in the 2d list


  char_i = ord('a')
  for f_i, f_c in enumerate(result, start=0): #Iterating the list for each F
    line = chr(char_i)+". "                   #for printing a, b, c, d, e in the result
    for s_i, s_c in enumerate(f_c, start=0):
      line = line + str("P(F={}|S=={})={}/{}".format(F[f_i], S[s_i], s_c[0], s_c[1]))
      if s_i < len(s_c) :
        line = line + ", "
    print(line)
    char_i += 1                               #Next character for printing


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

compute_conditional_probabilites(A)
```

```
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
```

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']
```

```
# 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 strings

# you can free to change all these codes/structure
def string_features(S1, S2):
  a,b,c = None, None, None

  #Since it's just said that no of common words. It is assumed that if a sentence has two same
  s1 = set()
  s2 = set()

  s1.update(S1.split()) #adding the split words to the set
  s2.update(S2.split())

  a = len(s1.intersection(s2))  #find the common words using intersection and get the count
  b = s1.difference(s2)         #set difference s1 - s2
  c = s2.difference(s1)         #set difference s2 - s1

  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', '5', 'F'}
c.  {'S', 'second', '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
```

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

```
import math

def compute_log_loss(A):
  loss = float(0.0) #initialize
  for a in A:
    loss += ((a[0]*math.log10(a[1])) + (1-a[0])*math.log10(1-a[1]) ) #error for each
  return loss*-1/len(A)   #loss * -1/n

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
```