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
```

In [ ]:

```
z = []
y=0
def zero_m():
    p = len(c)
    #print(p)
    q = len(d[0])
    x = []

    for i in range(q): #for columns of zero matrix
        x.append(0)
    for j in range(p): #for rows of zero matrix
        z.append(x.copy())
        #print(z)
    #return zero_mat
def matrix_multi(c,d):
    global y
    #print(c,len(c))
    #print(d,len(d))

    #print(z,len(z),len(z[0]))
    for i in range(len(z)):
        for j in range(len(z[0])):
            for k in range(len(d)):
                y += c[i][k]*d[k][j]
```

```
            z[i][j]=y
            y=0
    return z


c = [[1, 3, 4],[2, 5, 7],[5, 9, 6]]#input('first matrix:')
d = [[1, 0, 0],[0, 1, 0],[0, 0, 1]]#input('second matrix:')

(zero_m())


if len(c[0]) == len(d):
    print('matrix multiplication - ',matrix_multi(c,d))
else:
    print('matrix multiplication not possible')
```

```
matrix multiplication -  [[1, 3, 4], [2, 5, 7], [5, 9, 6]]
```

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)
```

In [ ]:

```
import random
A = [0, 5, 27, 6, 13, 28, 100, 45, 10, 79]
A_sum = sum(A)
def pick_a_number_from_list(A):
    s = 0
    c_sum = []
    for i in range(len(A)):
        s = s + A[i]
        c_sum.append(s)
    print(c_sum)

    r_value = int(random.uniform(0,A_sum))

    for j in range(len(c_sum)):
        if(r_value >= c_sum[j] and r_value < c_sum[j+1]):
            return A[j+1]

def sampling_based_on_magnitued():
    for i in range(1,100):
        number = pick_a_number_from_list(A)
        print(number)

sampling_based_on_magnitued()
# reference - https://codeutility.org/python-probability-proportional-to-its-magnitu
```

```
[0, 5, 32, 38, 51, 79, 179, 224, 234, 313]
79
[0, 5, 32, 38, 51, 79, 179, 224, 234, 313]
```

```
100
[0, 5, 32, 38, 51, 79, 179, 224, 234, 313]
28
[0, 5, 32, 38, 51, 79, 179, 224, 234, 313]
28
[0, 5, 32, 38, 51, 79, 179, 224, 234, 313]
100
[0, 5, 32, 38, 51, 79, 179, 224, 234, 313]
100
[0, 5, 32, 38, 51, 79, 179, 224, 234, 313]
100
[0, 5, 32, 38, 51, 79, 179, 224, 234, 313]
13
[0, 5, 32, 38, 51, 79, 179, 224, 234, 313]
79
[0, 5, 32, 38, 51, 79, 179, 224, 234, 313]
28
[0, 5, 32, 38, 51, 79, 179, 224, 234, 313]
100
[0, 5, 32, 38, 51, 79, 179, 224, 234, 313]
27
[0, 5, 32, 38, 51, 79, 179, 224, 234, 313]
45
[0, 5, 32, 38, 51, 79, 179, 224, 234, 313]
79
[0, 5, 32, 38, 51, 79, 179, 224, 234, 313]
100
[0, 5, 32, 38, 51, 79, 179, 224, 234, 313]
27
[0, 5, 32, 38, 51, 79, 179, 224, 234, 313]
100
[0, 5, 32, 38, 51, 79, 179, 224, 234, 313]
100
[0, 5, 32, 38, 51, 79, 179, 224, 234, 313]
100
[0, 5, 32, 38, 51, 79, 179, 224, 234, 313]
79
[0, 5, 32, 38, 51, 79, 179, 224, 234, 313]
28
[0, 5, 32, 38, 51, 79, 179, 224, 234, 313]
28
[0, 5, 32, 38, 51, 79, 179, 224, 234, 313]
27
[0, 5, 32, 38, 51, 79, 179, 224, 234, 313]
79
[0, 5, 32, 38, 51, 79, 179, 224, 234, 313]
13
[0, 5, 32, 38, 51, 79, 179, 224, 234, 313]
28
[0, 5, 32, 38, 51, 79, 179, 224, 234, 313]
100
[0, 5, 32, 38, 51, 79, 179, 224, 234, 313]
100
[0, 5, 32, 38, 51, 79, 179, 224, 234, 313]
27
[0, 5, 32, 38, 51, 79, 179, 224, 234, 313]
100
[0, 5, 32, 38, 51, 79, 179, 224, 234, 313]
28
[0, 5, 32, 38, 51, 79, 179, 224, 234, 313]
79
[0, 5, 32, 38, 51, 79, 179, 224, 234, 313]
13
[0, 5, 32, 38, 51, 79, 179, 224, 234, 313]
45
[0, 5, 32, 38, 51, 79, 179, 224, 234, 313]
27
[0, 5, 32, 38, 51, 79, 179, 224, 234, 313]
79
```

```
[0, 5, 32, 38, 51, 79, 179, 224, 234, 313]
27
[0, 5, 32, 38, 51, 79, 179, 224, 234, 313]
100
[0, 5, 32, 38, 51, 79, 179, 224, 234, 313]
27
[0, 5, 32, 38, 51, 79, 179, 224, 234, 313]
27
[0, 5, 32, 38, 51, 79, 179, 224, 234, 313]
13
[0, 5, 32, 38, 51, 79, 179, 224, 234, 313]
100
[0, 5, 32, 38, 51, 79, 179, 224, 234, 313]
45
[0, 5, 32, 38, 51, 79, 179, 224, 234, 313]
100
[0, 5, 32, 38, 51, 79, 179, 224, 234, 313]
100
[0, 5, 32, 38, 51, 79, 179, 224, 234, 313]
100
[0, 5, 32, 38, 51, 79, 179, 224, 234, 313]
100
[0, 5, 32, 38, 51, 79, 179, 224, 234, 313]
100
[0, 5, 32, 38, 51, 79, 179, 224, 234, 313]
27
[0, 5, 32, 38, 51, 79, 179, 224, 234, 313]
10
[0, 5, 32, 38, 51, 79, 179, 224, 234, 313]
100
[0, 5, 32, 38, 51, 79, 179, 224, 234, 313]
100
[0, 5, 32, 38, 51, 79, 179, 224, 234, 313]
45
[0, 5, 32, 38, 51, 79, 179, 224, 234, 313]
100
[0, 5, 32, 38, 51, 79, 179, 224, 234, 313]
27
[0, 5, 32, 38, 51, 79, 179, 224, 234, 313]
100
[0, 5, 32, 38, 51, 79, 179, 224, 234, 313]
79
[0, 5, 32, 38, 51, 79, 179, 224, 234, 313]
100
[0, 5, 32, 38, 51, 79, 179, 224, 234, 313]
28
[0, 5, 32, 38, 51, 79, 179, 224, 234, 313]
27
[0, 5, 32, 38, 51, 79, 179, 224, 234, 313]
79
[0, 5, 32, 38, 51, 79, 179, 224, 234, 313]
45
[0, 5, 32, 38, 51, 79, 179, 224, 234, 313]
79
[0, 5, 32, 38, 51, 79, 179, 224, 234, 313]
100
[0, 5, 32, 38, 51, 79, 179, 224, 234, 313]
79
[0, 5, 32, 38, 51, 79, 179, 224, 234, 313]
45
[0, 5, 32, 38, 51, 79, 179, 224, 234, 313]
45
[0, 5, 32, 38, 51, 79, 179, 224, 234, 313]
10
[0, 5, 32, 38, 51, 79, 179, 224, 234, 313]
79
[0, 5, 32, 38, 51, 79, 179, 224, 234, 313]
100
[0, 5, 32, 38, 51, 79, 179, 224, 234, 313]
```

```
100
[0, 5, 32, 38, 51, 79, 179, 224, 234, 313]
100
[0, 5, 32, 38, 51, 79, 179, 224, 234, 313]
6
[0, 5, 32, 38, 51, 79, 179, 224, 234, 313]
100
[0, 5, 32, 38, 51, 79, 179, 224, 234, 313]
79
[0, 5, 32, 38, 51, 79, 179, 224, 234, 313]
100
[0, 5, 32, 38, 51, 79, 179, 224, 234, 313]
100
[0, 5, 32, 38, 51, 79, 179, 224, 234, 313]
79
[0, 5, 32, 38, 51, 79, 179, 224, 234, 313]
27
[0, 5, 32, 38, 51, 79, 179, 224, 234, 313]
27
[0, 5, 32, 38, 51, 79, 179, 224, 234, 313]
45
[0, 5, 32, 38, 51, 79, 179, 224, 234, 313]
79
[0, 5, 32, 38, 51, 79, 179, 224, 234, 313]
45
[0, 5, 32, 38, 51, 79, 179, 224, 234, 313]
100
[0, 5, 32, 38, 51, 79, 179, 224, 234, 313]
45
[0, 5, 32, 38, 51, 79, 179, 224, 234, 313]
10
[0, 5, 32, 38, 51, 79, 179, 224, 234, 313]
100
[0, 5, 32, 38, 51, 79, 179, 224, 234, 313]
79
[0, 5, 32, 38, 51, 79, 179, 224, 234, 313]
79
[0, 5, 32, 38, 51, 79, 179, 224, 234, 313]
45
[0, 5, 32, 38, 51, 79, 179, 224, 234, 313]
79
[0, 5, 32, 38, 51, 79, 179, 224, 234, 313]
45
[0, 5, 32, 38, 51, 79, 179, 224, 234, 313]
79
[0, 5, 32, 38, 51, 79, 179, 224, 234, 313]
27
[0, 5, 32, 38, 51, 79, 179, 224, 234, 313]
45
[0, 5, 32, 38, 51, 79, 179, 224, 234, 313]
28
[0, 5, 32, 38, 51, 79, 179, 224, 234, 313]
45
[0, 5, 32, 38, 51, 79, 179, 224, 234, 313]
100
[0, 5, 32, 38, 51, 79, 179, 224, 234, 313]
28
```

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: ####
```

In [ ]:

```
from dataclasses import replace
import re
# 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
# try to complete this question using regular expressions
# you can free to change all these codes/structure
# String: it will be the input to your program
def replace_digits(string):
    r1 = re.sub('\D','',string)
    res = re.sub("\d","#",r1)
    if len(res) == 0:
        return ('empty string')
    else:
        return res


string = '234'
print(replace_digits(string))
string1 = 'a2b3c4'
print(replace_digits(string1))
string2 = '#2a$#b%c%561#'
print(replace_digits(string2))
string3 = 'abc'
print(replace_digits(string3))
```

```
###
###
####
empty 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','stude

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
```

In [ ]:

```
Students=['student1','student2','student3','student4','student5','student6','student
Marks = [45, 78, 12, 14, 48, 43, 47, 98, 22, 80]
sm = {}
def lst_dct(s,m):
    if len(s) == len(m):
        for i in range(len(s)):
            sm[s[i]] = m[i]
    return sm

def get_percentile(p):
    students_within_25_and_75 = []
    n = (25/100)*int(len(Marks))
    m = (75/100)*int(len(Marks))
    for j in range(len(p)):
        if j+1 > n and j+1 < m:
            students_within_25_and_75.append(p[j])
    return students_within_25_and_75

print(lst_dct(Students,Marks))
sm_sort = sorted(sm.items(), key= lambda x:x[1])
print(sm_sort)
print('least_5_students -',sm_sort[:5])
print('top_5_students -',sm_sort[-5:])
print('students_within_25_and_75 -',get_percentile(sm_sort))
```

```
{'student1': 45, 'student2': 78, 'student3': 12, 'student4': 14, 'student5': 48, 'st
udent6': 43, 'student7': 47, 'student8': 98, 'student9': 22, 'student10': 80}
[('student3', 12), ('student4', 14), ('student9', 22), ('student6', 43), ('student
1', 45), ('student7', 47), ('student5', 48), ('student2', 78), ('student10', 80),
('student8', 98)]
least_5_students - [('student3', 12), ('student4', 14), ('student9', 22), ('student
6', 43), ('student1', 45)]
top_5_students - [('student7', 47), ('student5', 48), ('student2', 78), ('student1
0', 80), ('student8', 98)]
students_within_25_and_75 - [('student9', 22), ('student6', 43), ('student1', 45),
('student7', 47), ('student5', 48)]
```

Q5: Find the closest points

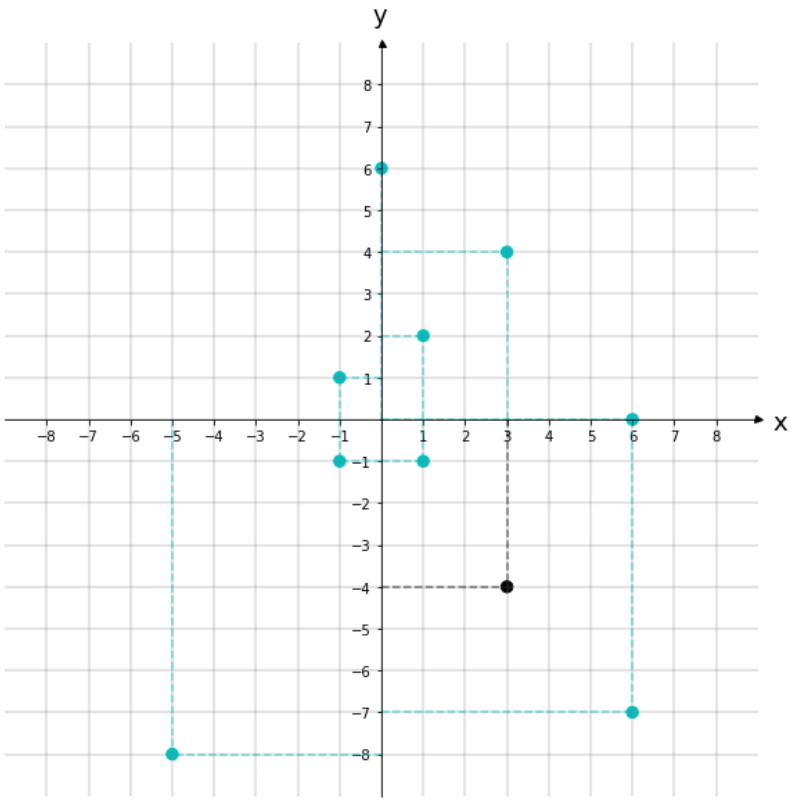
Consider you have given n data points in the form of list of tuples like S=[(x1,y1),(x2,y2),(x3,y3),(x4,y4),(x5,y5),..,(xn,yn)] and a point P=(p,q)

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

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

In [ ]:

```
import math
S= [(1,2),(3,4),(-1,1),(6,-7),(0, 6),(-5,-8),(-1,-1),(6,0),(1,-1)]
P= (3,-4)
d = {}
def closest_points_to_p(s, P):
    for i in range(len(s)):
        dist = math.acos(((s[i][0]*P[0]) + (s[i][1]*P[1]))/((math.sqrt((s[i][0]**2)
        d[i] = dist
    d_sort = sorted(d.items(), key= lambda x: x[1])
    findex = []
    for j in d_sort:
        findex.append(j[0])
    point_req = []
    for k in findex:
```

```
        point_req.append(S[k])
    return point_req[:5]
print('closest points - ',closest_points_to_p(S,P))
```

```
closest 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

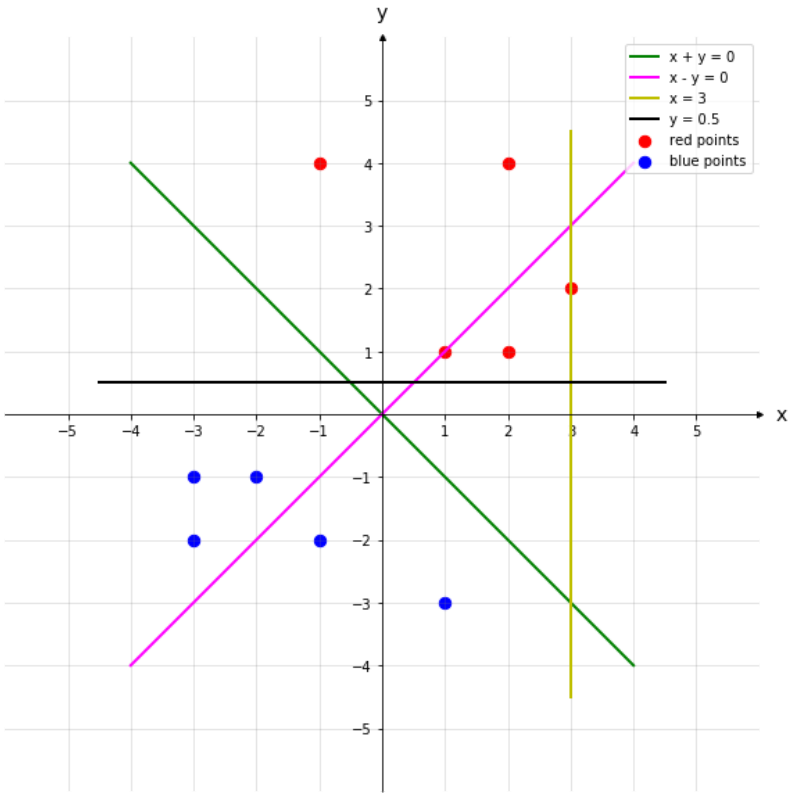
```
Red =[(R11,R12),(R21,R22),(R31,R32),(R41,R42),(R51,R52),..,(Rn1,Rn2)]
Blue=[(B11,B12),(B21,B22),(B31,B32),(B41,B42),(B51,B52),..,(Bm1,Bm2)]
```

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
```

In [ ]:
```
import re
import math
def i_am_the_one(red,blue,line):
    res_red = []
    res_blue = []
    for i in line:
        eq_res = []
        for j in red:
            eq_res.append(eval(i[0]+'*'+str(j[0])+i[2]+i[3]+'*'+str(j[1])+i[5]+i[6])
        res_red.append(eq_res)
    #print(res_red)
    for i in line:
        eq_res1 = []
        for k in blue:
            eq_res1.append(eval(i[0]+'*'+str(k[0])+i[2]+i[3]+'*'+str(k[1])+i[5]+i[6]
        res_blue.append(eq_res1)
    #print(res_blue)
    m = 0
    while m < len(res_blue):
        if res_red[m] == res_blue[m]:
            print('YES')
        else:
            print('NO')
        m = m+1

Lines=["1x+1y+0","1x-1y+0","1x+0y-3","0x+1y-0.5"]
Red= [(1,1),(2,1),(4,2),(2,4), (-1,4)]
Blue= [(-2,-1),(-1,-2),(-3,-2),(-3,-1),(1,-3)]
print(i_am_the_one(Red,Blue,Lines))
```

```
YES
NO
NO
YES
None
```

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 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, 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, 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 [ ]:

```
def curve_smoothing(s):
    L = []

    Listi = s.split(',')

    for i in range(len(Listi)):
        try:
            int(Listi[i]) == True
            L.append(i)
        except:
            pass
    #print('L',L)
    #print(len(L))
    for i in range(len(Listi)):
        if Listi[i] == '_':
            Listi[i] = 0
        else:
            Listi[i] = int(Listi[i])
    #print('Listi', Listi)

    L_f = []
    Lasti = 0
    ic = 0
    for i in range(len(L)):

        Lint = Listi[ic:L[i]+1]
        if i > 0:
            Lint.append(Lasti)
            L_f.pop()
        #print(Lint)

        for j in range(len(Lint)):
            a = (sum(Lint))/len(Lint)
            L_f.append(a)
        ic = L[i]+1
        #print(ic)
        Lasti = L_f[-1]
        #print(Lasti)
    #print("L_f -- ", L_f)

    Lint1 = Listi[L[-1]+1:len(Listi)]
    Lint1.append(Lasti)
    #print(Lint1)
```

```
    L_f.pop()
    n = 0
    while n < len(Lint1):
        b = (sum(Lint1))/len(Lint1)
        L_f.append(b)
        n = n+1
    return L_f

#S = "80,_,_,_,_"
#S = "_,_,_,24"
#S = "40,_,_,_,60"
S = "_,_,30,_,_,_,50,_,_"
print('output - ',curve_smoothing(S))
```

```
output -  [10.0, 10.0, 12.0, 12.0, 12.0, 12.0, 4.0, 4.0, 4.0]
```

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

In [ ]:

```
from typing import List


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

def compute_conditional_probabilites(f,s):
    count_f = 0
    count_s = 0
    for i in range(len(A)):
        if A[i][1] == s:
            count_s = count_s + 1
            if A[i][0] == f:
                count_f = count_f + 1
    return(f'P(F = {f} | S == {s}) = {count_f}/{count_s}')


L_f = []
L_s = []
```

```
for i in range(len(A)):
    L_f.append(A[i][0])
f_l = list(set(L_f))
f_l.sort()
print(f_l)
for i in range(len(A)):
    L_s.append(A[i][1])
s_l = list(set(L_s))
s_l.sort()
print(s_l)
for i in s_l:
    for j in f_l:
        print(compute_conditional_probabilites(j,i))

#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
```

```
['F1', 'F2', 'F3', 'F4', 'F5']
['S1', 'S2', 'S3']
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']
```

In [ ]:

```
def string_features(S1, S2):
    sen1 = S1.split(' ')
    sen2 = S2.split(' ')
    a = 0
    b = []
    for i in sen1:
```

```
        for j in sen2:
            if i == j:
                a = a+1
        if i not in sen2:
            b.append(i)
    c = []
    for j in sen2:
        if j not in sen1:
            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_count, b_list, c_list = string_features(S1, S2)

print(a_count,'\n',b_list,'\n', c_list)
```

```
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
$f(Y, Y_{score}) = -1 * \frac{1}{n}\Sigma_{foreachY,Y_{score}pair}(Ylog10(Y_{score}) + (1-Y)log10(1-Y_{score}))$ 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
```

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

In [ ]:

```
import math
def compute_log_loss(A):
    b = 0
    for i in range(len(A)):
        a = A[i][0]*math.log10(A[i][1]) + (1-A[i][0])*math.log10(1-A[i][1])
        b = b + a
    loss = ((-1)/len(A))*b

    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
```