# 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
```

In [19]:

```
# 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 examples

a = [[1, 3, 4],
      [2, 5, 7],
      [5, 9, 6]]
b = [[1, 0, 0],
     [0, 1, 0],
     [0, 0, 1]]

result = []
def matrix_mul(a, b):
    if (len(a[0])== len(b)):
        for i in range(len(a)):
            row=[]
            for j in range(len(b[0])):
                dig = 0
                for k in range(len(b)):
                    dig += a[i][k] * b[k][j]
                row.append(dig)
            result.append(row)
        for r in result:
            print(r)
```

```
    else:
        print("Given arrays are not compatible")

matrix_mul(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 [29]:

```
from random import uniform

def pick_a_number_from_list(A):
    sum=0
    cumm=[]
    for i in range(len(A)):
        sum = sum + A[i]
        cumm.append(sum)
    r = int(random.uniform(0,sum))
    print(r)
    number=0
    for index in range(len(cumm)):
        if(r>=cumm[index] and r<cumm[index+1]):
            return A[index+1]
    return number

def sampling_based_on_magnitude():
    A = [0, 5, 27, 6, 13, 28, 100, 45, 10, 79]
    a = dict()
    print(A,sum(A))
    for i in range(1,100):
        number = pick_a_number_from_list(A)
        if number not in a:
            a[number] = 1
        else:
            a[number]+=1
    print(a)

sampling_based_on_magnitude()
```

```
[0, 5, 27, 6, 13, 28, 100, 45, 10, 79] 313
215
114
119
17
245
299
27
130
309
54
273
5
303
171
175
9
```

293
300
305
53
297
89
167
67
231
231
286
308
103
6
280
170
140
33
269
217
113
192
169
262
280
105
167
228
161
285
275
256
191
201
12
217
213
21
62
18
167
126
94
173
86
80
146
79
268
211
73
175
257
163
119
198
172
301
25
304
114
26
54
44
204
85
120
252
1
159
227
104

```
167
117
205
131
4
229
54
128
179
125
242
{45: 12, 100: 37, 27: 10, 79: 24, 28: 7, 10: 5, 6: 1, 13: 1, 5: 2}
```

## 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 [5]:

```
import re
s="a2b3c4"
re.sub("\d","#",re.sub("\D", "", s))
```

Out[5]:

```
'###'
```

## 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','studen
t7','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 [6]:

```
import math


def display_dash_board(marks, students):
    marks_zipped = list(zip(marks, students))
    marks_zipped.sort(key=lambda item: item[0])
    top5 = last5 = marks_zipped

    if len(marks_zipped) > 5:
        top5 = marks_zipped[:-6: -1]

        last5 = marks_zipped[:5]

    Bw2575 = marks_zipped[math.ceil(0.25 * len(marks_zipped)):math.floor(0.75 * len(mark
s_zipped))]

    return top5, last5, Bw2575


Students = ['student1', 'student2', 'student3', 'student4', 'student5', 'student6', 'stu
dent7', 'student8', 'student9','student10']
Marks = [45, 78, 12, 14, 48, 43, 47, 98, 35, 80]

display_dash_board(Marks, Students)
```

Out[6]:

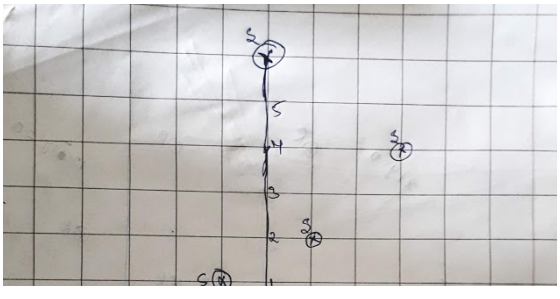
```
([(98, 'student8'),
  (80, 'student10'),
  (78, 'student2'),
  (48, 'student5'),
  (47, 'student7')],
 [(12, 'student3'),
  (14, 'student4'),
  (35, 'student9'),
  (43, 'student6'),
  (45, 'student1')],
 [(43, 'student6'), (45, 'student1'), (47, 'student7'), (48, 'student5')])
```

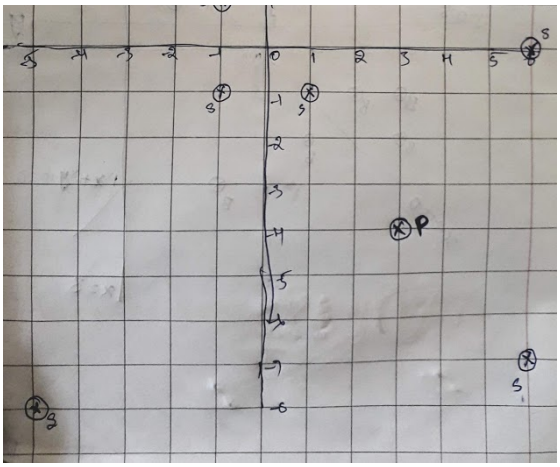
## Q5: Find the closest points

**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)

In [32]:

```
import math


def clst_pts(S, P):
    clst_pts = []
    final_list = []

    for point in S:
        den = math.sqrt((point[0] ** 2) + (point[1] ** 2)) * math.sqrt((P[0] ** 2) + (P
[1] ** 2))
        num = point[0] * P[0] + point[1] * P[1]

        if den != 0:
            cos_dis = math.acos(num / den)
            clst_pts.append((cos_dis, point))

    for i in sorted(clst_pts, key=lambda x: x[0])[:5]:
        final_list.append(i[1])

    return final_list


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

pts = clst_pts(S, P)
print("Closest point-cosine-distance - top 5:", *[point for point in pts], sep="\n")
```

```
Closest point-cosine-distance - top 5:
(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**

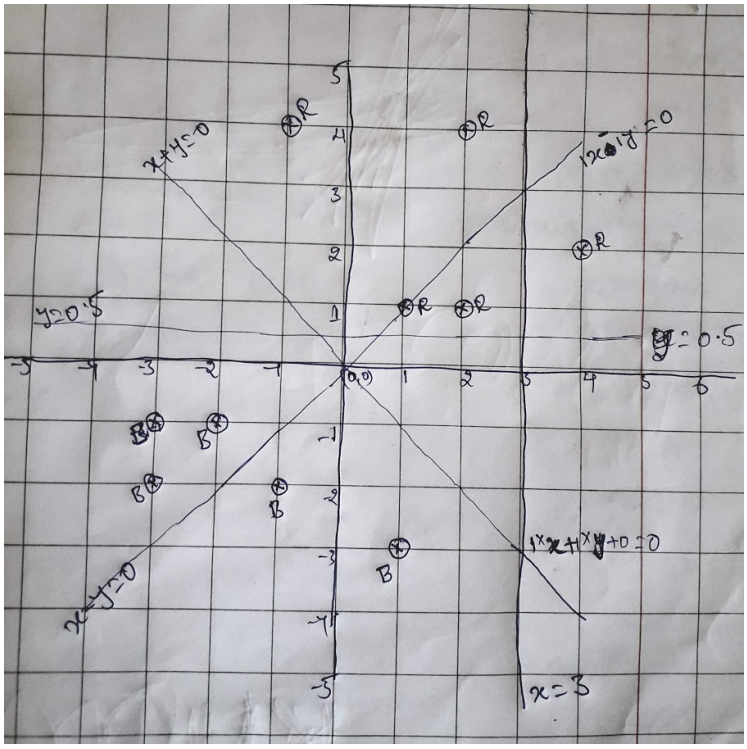
```
Red =[(R11,R12),(R21,R22),(R31,R32),(R41,R42),(R51,R52),..,(Rn1,Rn2)]
Blue=[(B11,B12),(B21,B22),(B31,B32),(B41,B42),(B51,B52),..,(Bm1,Bm2)]
```

**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 [7]:

```
import math

def i_am_the_one(red, blue, line):
    rsign = -1

    if eval(line.replace('x', '*%s' % red[0][0]).replace('y', '*%s' % red[0][1])) > 0:
        rsign = 1

    for r_pt in red:
        if rsign == 1 and eval(
                line.replace('x', '*%s' % r_pt[0]).replace('y', '*%s' % r_pt[1])) < 0:
            return 'NO'

        if rsign == -1 and eval(
                line.replace('x', '*%s' % r_pt[0]).replace('y', '*%s' % r_pt[1])) > 0:
            return 'NO'


    bsign = -1 * rsign

    for b_pts in blue:
        if bsign == 1 and eval(
                line.replace('x', '*%s' % b_pts[0]).replace('y', '*%s' % b_pts[1])) < 0:
            return 'NO'
```

```
        if bsign == -1 and eval(
                line.replace('x', '*%s' % b_pts[0]).replace('y', '*%s' % b_pts[1])) > 0:
            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:
    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 equall
y 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 [2]:

```
def fun(x, a, b):
    if a == -1:
```

```
        v = float(x[b])/(b+1)
        for i in range(a+1,b+1):
            x[i] = v
    elif b == -1:
        v = float(x[a])/(len(x)-a)
        for i in range(a, len(x)):
            x[i] = v
    else:
        v = (float(x[a])+float(x[b]))/(b-a+1)
        for i in range(a,b+1):
            x[i] = v
    return x

def curve_smoothing(text):
    x = text.replace(" ","").split(",")
    y = [i for i, v in enumerate(x) if v != '_']
    if y[0] != 0:
        y = [-1] + y
    if y[-1] != len(x)-1:
        y = y + [-1]
    for (a, b) in zip(y[:-1], y[1:]):
        fun(x,a,b)
    return x

tests = ["80,_,_,_,_"]

for i in tests:
    print (curve_smoothing(i))
```

```
[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
```

In [27]:

```
# 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



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

```
def values(F,S):
    num=0
    den=0
    for i in range(len(A)):
        if(A[i][1]==S):
            den=den+1
            if(A[i][0]==F):
                num=num+1
    print('P(F={}|S=={})={}/{}'.format(F, S, str(num), str(den)))

for k in ['F1', 'F2', 'F3', 'F4', 'F5']:
    for m in ['S1', 'S2', 'S3']:
        values(k,m)
```

```
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']

In [2]:

```
# 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=[]
    for ch in S1.split():
        for ch1 in S2.split():
            if(ch ==ch1):
                a.append(ch)
    print(len(a))
    b= list(set(S1.split())-set(S2.split()))
    print(b)
    c = list(set(S2.split())-set(S1.split()))
    print(c)
```

```
S1= "the first column F will contain only 5 uniques values"
S2= "the second column S will contain only 3 uniques values"
print(string_features(S1, S2))
```

```
7
['first', 'F', '5']
['3', 'second', 'S']
None
```

## 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 $f(Y, Y_{score})$ here n is the number of rows in the matrix**

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

```
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
```

$$\frac{-1}{8} \cdot ((1 \cdot log_{10}(0.4) + 0 \cdot log_{10}(0.6)) + (0 \cdot log_{10}(0.5) + 1 \cdot log_{10}(0.5)) + \ldots + (1 \cdot log_{10}(0.8) + 0 \cdot log_{10}(0.2)))$$

In [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

import math

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]]
```

```
# you can free to change all these codes/structure
def compute_log_loss(A):
    sum = 0
    for i in range(len(A)):
        sum = sum + ((A[i][0] * math.log(A[i][1],10)) + ((1-A[i][0]) * (math.log(1-A[i]
[1],10))))
    return sum

print((compute_log_loss(A)/8)*(-1))
```

```
0.42430993457031635
```

In [ ]: