# Q.1: Given two matrices print product

In [10]:

```
def matrix_mul(A, B):

    if (len(A[0])!=len(B)):
        print("Not possible")
        return
    if(len(A[0])==len(B)):

        #https://stackoverflow.com/questions/54964557/how-to-create-a-zero-matrix-without-using-numpy
        result = [( [0]*len(B[0])) for row in range(len(A))]

        for i in range(len(A)):
            for j in range(len(B[0])):
                for k in range(len(A[0])):
                    result[i][j] += A[i][k] * B[k][j]

        for r in result:
            print(r)
        return
```

In [11]:

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

In [12]:

```
matrix_mul(A, B)
```

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

In [13]:

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

```
In [14]: matrix_mul(A,B)
```

```
[11, 14, 17, 20, 23]
[23, 30, 37, 44, 51]
```

```
In [15]: A = [[1,2],[3,4]]
         B= [[1,4],[5,6],[7,8],[9,6]]
```

```
In [16]: matrix_mul(A, B)
```

```
Not possible
```

# Q.2: Select a number randomly with probability proportional to its magnitude from the given array of n elements

In [162]:

```
import random
def pick_a_number_from_list(A):
    sum=0
    cum_sum=0
    cum_list=[]
    for i in range(len(A)):
        sum +=A[i]


    #normalize the list
    norm = [float(i)/sum for i in A]


    #cummulative sum of normalized list
    for i in range(len(norm)):
        cum_sum +=norm[i]
        cum_list.append(cum_sum)

    r = random.uniform(0,cum_sum)

    for index in range(len(cum_list)):

        if(r>=cum_list[index] and r<cum_list[index+1]):


            number= A[index+1]

    return number
```

In [167]:

```
def sampling_based_on_magnitued():
    a=dict()
    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

    sorted_a={k: v for k, v in sorted(a.items(), key=lambda item: item[1],reverse=True)}
    print(sorted_a)

sampling_based_on_magnitued()
```

```
{100: 37, 79: 26, 45: 10, 27: 9, 28: 7, 13: 4, 10: 3, 5: 2, 6: 1}
```

# Q.3: Replace the digits in the string with

In [3]:

```
import re

def replace_digits(str):
    replacements = [['[^0-9]',''], ['\d', '#']]
    for ele in replacements:
        str = re.sub(ele[0],ele[1] ,str)


    return str
```

In [4]:

```
#executing function for all four strings

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

```
###
###

####
```

# Q:4. Students marks dashboard

In [155]:

```
def display_dash_board(students, marks):
    dict = {Students[i]: Marks[i] for i in range(len(Students))}
    dict1=sorted(dict.items(), key=lambda kv: kv[1], reverse=True)[:5]
    dict2=sorted(dict.items(), key=lambda kv: kv[1])[:5]
    print("top_5_students\n")

    for item in dict1:
        print(item[0],item[1])

    print("\nleast_5_students\n")

    for item in dict2:
        print(item[0],item[1])

    print("\nmarks between >25th percentile <75th percentile\n")
    dict3=sorted(dict.items(), key=lambda kv: kv[1])
    marks=dict.values()
    max_marks=max(marks)
    min_marks=min(marks)
    dif=max_marks-min_marks
    perc_25=dif * 0.25
    perc_75=dif * 0.75

    for item in dict3:
        if perc_25 < item[1] < perc_75:
            print(item[0],item[1])
```

In [156]:

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

```
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

marks between >25th percentile <75th percentile

student9 35
student6 43
student1 45
student7 47
student5 48
```

# Q.5: Find the closest points

In [46]:

```
import math

def closest_points_to_p(S, P):

    cosine_dist=[]
    for ele in S:
        sum=((ele[0]*P[0])+(ele[1]*P[1]))

        sqrt=math.sqrt((math.pow(ele[0],2)+(math.pow(ele[1],2)))) * math.sqrt((math.pow(P[0],2)+(math.pow(P[1],2))))
        cosine_dist.append(math.acos(sum/sqrt))

    dict= dict = {S[i]: cosine_dist[i] for i in range(len(S))}
    sorted_dict= sorted(dict.items(), key=lambda kv: kv[1])[:5]

    print("closest points in S from P\n")
    for item in sorted_dict:
        print(item[0])
```

In [47]:

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

In [48]:

```
closest_points_to_p(S, P)
```

```
closest points in S from P

(6, -7)
(1, -1)
(6, 0)
(-5, -8)
(-1, -1)
```

# Q.6: Find which line seperates oranges and apples

In [40]:

```
red_list = []
blue_list = []

def i_am_the_one(red,blue,line):

    for k in Lines:
        a=k[0];
        b=k[2]+k[3];
        c=k[5]+k[6]
        R=[]
        B=[]

        for i in Red:
            x1=i[0];y1=i[1]
            eq1 = int(a) * int(x1) + int(b) * int(y1)-int(c)
            R.append(eq1)
        red_list.append(R)

        for j in Blue:

            x2=j[0];y2=j[1]
            eq2 = int(a) * int(x2) + int(b) * int(y2)-int(c)
            B.append(eq2)
        blue_list.append(B)


    red=[all(a>0 for a in i) for i in red_list]
    blue=[all(b<0 for b in i) for i in blue_list]

    for i in range(len(blue)):

        if(blue[i]==True and red[i]==True):
            print("Yes")
        else:
            print('No')
```

In [41]:

```
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"]
```

In [42]:

```
i_am_the_one(Red,Blue,Lines)
```

```
Yes
No
No
Yes
```

# Q.7 Filling the missing values in specified format

In [195]:

```
def curve_smoothing(string):

    splitstring = string.split(',')

    pos = 0
    next_value = 0
    last_pos = 0
    last_value = 0

    while pos < len(splitstring):
        if splitstring[pos] != '_' or (pos + 1 == len(splitstring)):
            if splitstring[pos] != '_':

                next_value = int(splitstring[pos])
            else:
                next_value = 0
            new_value = (next_value + last_value) / (pos - last_pos + 1)
            for i in range(last_pos, pos + 1):
                splitstring[i] = int(new_value)
            last_value = new_value
            last_pos = pos
        pos += 1

    return splitstring
```

In [185]:

```
string=["_,_,_,24","40,_,_,_,60","80,_,_,_,_","_,_,30,_,_,_,50,_,_"]
```

In [196]:

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

```
[6, 6, 6, 6]
[20, 20, 20, 20, 20]
[16, 16, 16, 16, 16]
[10, 10, 12, 12, 12, 12, 4, 4, 4]
```

# Q.8 : Find probabilities

In [80]:

```
A = [['F1','S1'],['F2','S2'],['F3','S3'],['F1','S2'],['F2','S3'],['F3','S2'],['F2','S1'],['F4','S1'],['F4','S3'],['F5'
,'S1']]
#Create list F
for i in A:
    if i[0] not in F:
        F.append(i[0])
print(F)

    #Create list S
for i in A:
    if i[1] not in S:
        S.append(i[1])
print(S)
```

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

In [84]:

```
def compute_conditional_probabilites(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)))
```

In [82]:

```
for k in F:
    for m in S:

        compute_conditional_probabilites(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
```

# Q.9: Given two sentences S1,S2

In [39]:

```
def string_features(S1, S2):
    S1List = S1.split(" ")
    S2List = S2.split(" ")
    a=len(list(set(S1List)&set(S2List)))
    b=[word for word in S1List if word not in S2List]
    c=[word for word in S2List if word not in S1List]
    return a,b,c
```

In [40]:

```
S1= "the first column F will contain only 5 uniques values"
S2= "the second column S will contain only 3 uniques values"
p,q,r = string_features(S1, S2)
print("Number of common words between S1, S2",p)
print("Words in S1 but not in S2",q)
print("Words in S2 but not in S1",r)
```

```
Number of common words between S1, S2 7
Words in S1 but not in S2 ['first', 'F', '5']
Words in S2 but not in S1 ['second', 'S', '3']
```

# Q.10: Compute log loss

In [57]:

```
import math
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))))

    log_loss=round((sum/8)*(-1),7)
    return log_loss
```

In [58]:

```
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]]
log_loss = compute_log_loss(A)
```

```
In [59]: log_loss
```

```
Out[59]: 0.4243099
```

```
In [198]: !jupyter nbconvert --to html PythonMandatory_solve.ipynb
```

```
[NbConvertApp] Converting notebook PythonMandatory_solve.ipynb to html
[NbConvertApp] Writing 316848 bytes to PythonMandatory_solve.html
```