1: Multiplication of Two Matrix

In [16]:
```
def matrix_mul(A,B):
    if len(A[0])!=len(B): #check the condition of matrix multiplication rule
        print('matrix multiplication between A and B is not possible')
#         return 0
    else:
        row = len(A) # find the number of row in first matrix
        col = len(B[0]) # find the number of column in second matrix

        result = [[0 for x in range(len(B[0]))] for y in range(len(A))] # find the dimension of resultant matrix

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

In [17]:
```
a=[[1,3,4],
   [2,5,7],
   [5,9,6]]
b=[[1,0,0],
   [0,1,0],
   [0,0,1]]
matrix_mul(a,b)
```

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

In [18]:
```
c=[[1,2],
   [3,4]]
d=[[1,4],
   [5,6],
   [7,8],
   [9,6]]
matrix_mul(c,d)
```

```
matrix multiplication between A and B is not possible
```

In [19]:
```
e=[[1,2],
   [3,4]]
f=[[1,2,3,4,5],
   [5,6,7,8,9]]
matrix_mul(e,f)
```

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

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

In [20]:

```python
from random import uniform

def pick_a_number_from_list(A):

    addition = 0

    A.sort()

    # calculate the length of the list A
    lenA = len(A)


    # find addition of total elements
    for i in range(lenA):
        addition += A[i]

    scaledA = [] # to store scaled elements

    for i in range(lenA):
        item = A[i]/addition
        scaledA.append(item)

    # cumulatie list to store cumulative value
    cum_list = []

    # append first element of list to cum_list form sclaedA list
    cum_list.append(scaledA[0])

    for i in range(1, lenA):
        cum_list.append(scaledA[i])
        cum_list[i] += cum_list[i-1]

    u_val = uniform(0.0,1.0) # generate uniform number in range [0,1]


    for i in range(1,lenA):
        if (u_val >= cum_list[i-1] and u_val < cum_list[i]):
            return A[i]
```

In [21]:
```
def sampling_based_on_magnitude():
    random_numbers = [] # to store found random numbers

    for i in range(1,101):
        num = pick_a_number_from_list(A)
        random_numbers.append(num)
        print(num,end=', ')
    return random_numbers
```

In [22]:
```
A = [0,5,27,6,13,28,100,45,10,79]

random_numbers = list(sampling_based_on_magnitude())
```

```
79, 5, 79, 5, 100, 79, 27, 100, 100, 100, 100, 100, 79, 27, 13, 100, 13, 10, 100, 100, 10, 27, 79, 100, 100, 100, 79,
45, 79, 79, 79, 13, 100, 10, 28, 79, 79, 79, 45, 45, 45, 79, 45, 45, 27, 79, 100, 100, 100, 79, 45, 100, 10, 13, 27,
100, 10, 6, 27, 100, 79, 27, 79, 100, 45, 100, 45, 28, 79, 13, 79, 28, 27, 79, 28, 79, 100, 79, 45, 100, 28, 10, 100,
79, 45, 100, 27, 100, 100, 27, 100, 100, 10, 79, 10, 45, 45, 45, 28, 100,
```

In [23]:
```
# find probability element-wise

prob = {}

for i in range(len(A)):
    count = 0
    for j in range(100):
        if A[i] == random_numbers[j]:
            count +=1

    prob[A[i]] = count
prob
```

Out[23]:
```
{0: 0, 5: 2, 6: 1, 10: 8, 13: 5, 27: 10, 28: 6, 45: 14, 79: 24, 100: 30}
```

3. Replace the digits in the string with

In [24]:
```
def replace_digits(String):
    numOfHash=0
    for i in range(len(String)):
        if(String[i].isdigit()):
#isdigit() is a built in method which is handle the strings. here we using this method for counting the digit
            numOfHash += 1
    if numOfHash==0:
        print('Empty string')
    else:
        for i in range(numOfHash):
            print("#",end='')
```

In [25]:
```
replace_digits('2348')
```

```
####
```

In [26]:
```
replace_digits('a2b3c4')
```

```
###
```

In [27]:
```
replace_digits('abc')
```

```
Empty string
```

In [28]:
```
replace_digits('#2a$#b%c%561#')
```

```
####
```

4: Students marks dashboard

In [29]:

```
def display_dash_board(students, marks):
    merge_list = zip(students,marks) # merge the list by using the zip() function
    merge_list = list(merge_list)
    percentile=[] #make empty list for counting the percentile of students
    top_5_students = sorted(merge_list,key=lambda k :k[1],reverse=True)[0:5]
    # sort the zipped list by marks in descending order
    least_5_students = sorted(merge_list,key=lambda k :k[1],reverse=False)[0:5]
    #sort the zipped list by marks in descending order
    percentile_students= sorted(merge_list,key=lambda k :k[1])


    percentile_75=(((3*len(marks))/4)/len(marks))*100
    percentile_25=(((len(marks))/4)/len(marks))*100
    for i in range(len(students)):
        j=((i+1)/10)*100
        if(j>percentile_25) & (j<percentile_75):
            percentile.append(percentile_students[i])
    return top_5_students,least_5_students,percentile


students=['student1','student2','student3','student4','student5','student6','student7','student8',
          'student9','student10']
marks = [45, 78, 12, 14, 48, 43, 47, 98, 35, 80]
top_5_students,least_5_students,percentile=display_dash_board(students, marks)
print("a. Top 5 students rankers")
for students,marks in top_5_students:
    print(students, marks)
print('\n')
print("b. lest 5 students rankers")
for students,marks in least_5_students:
    print(students, marks)
print('\n')
print("c. students who got percentile between 25 and 75")
for students,marks in percentile:
    print(students, marks)
```

a. Top 5 students rankers
student8 98
student10 80
student2 78
student5 48
student7 47

b. lest 5 students rankers
student3 12
student4 14
student9 35
student6 43
student1 45

c. students who got percentile between 25 and 75
student9 35
student6 43
student1 45
student7 47
student5 48

5. closest point using cosine distance :

formula of cosine distance=cos−1((𝑥·𝑝+𝑦·𝑞)/(√𝑥2+𝑦2)·(√𝑝2+𝑞2))

In [55]:
```
import math #import math for cos inverse and square root function

def closest_points_to_p(S, P):
    points=sorted(S,key=lambda x:  math.acos((x[0]*p[0]+x[1]*(p[1]))/(math.sqrt(x[0]**2+x[1]**2)
                                                                      *math.sqrt(((p[0])**2)+((p[1])**2)))))
    return points

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)
for i in range(5):
    print(points[i])
```

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

6. Line sepration of points:

passes line completely seprate the Blue and Orange points

In [2]:
```
#### Regular Expression concept study from geeksforgeeks####
import re
import math
def i_am_the_one(red,blue,line):
    check1=0
    check2=0
    res =re.findall(r'[0-9\-\+]+', line) # used for finding the coefficient of x,y and constant
    x1=int(res[0])
    y1=int(res[1])
    b1=int(res[2])
   #  if y1>mx+c that's mean point up on the line and
   #  if y1<mx+c that's mean point below the line
    for i, j in zip(red,blue):
        if(((y1*i[1]>((-1)*x1*i[0])+b1)and(y1*j[1]>((-1)*x1*j[0])+b1))
           or((y1*i[1]<((-1)*x1*i[0])+b1)and(y1*j[1]<((-1)*x1*j[0])+b1))):
            check1 +=1


        else:
            check2 +=1
    if  check2 ==len(red):
        print("YES")
    else:
        print('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"]
for i in Lines:
    i_am_the_one(Red, Blue, i)
```

```
YES
NO
NO
YES
```

7: Filling the missing values in the specified format

In [1]:

```
def condition_one(temp):
    # condition_two: _,_,_,40
    len_temp = len(temp)

    num_end = int(temp[-1])

    value = num_end/len_temp

    temp_list=[]
    for i in range(len_temp):
        temp_list.append(int(value))

    return temp_list

def condition_two(temp):
    # condition_one: 40,_,_,_
    len_temp = len(temp)

    num_start = int(temp[0])

    value = num_start/len_temp

    temp_list=[]
    for i in range(len_temp):
        temp_list.append(int(value))

    return temp_list



def condition_three(temp):
    # condition_one: 40,_,_,_,10
    len_temp = len(temp)

    num_start = int(temp[0])
    num_end = int(temp[-1])

    value = (num_start+num_end)/len_temp

    temp_list=[]
    for i in range(len_temp):
```

```
        temp_list.append(int(value))

    return temp_list
```

In [10]:
```
def curve_smoothing(string):

    str_list = string.split(',') # split string using ',' separator

    for i in range(len(str_list)):
        if str_list[i] != '_':
            str_list[i] = int(str_list[i])

    count = 0
    while(count<= len(str_list)):

        if str_list[0] == '_':

            temp_list = []

            condition = True

            while condition == True:

                if str_list[count] == '_':
                    temp_list.append(str_list[count])
                    count +=1

                elif str(str_list[count]).isdigit():

                    temp_list.append(str_list[count])
                    condition = False

            returned_list = condition_one(temp_list)

            str_list[0:count+1] = returned_list

            if len(str_list) == len(temp_list):
                return str_list

        elif str(str_list[count]).isdigit():

            in_start = count # starting index

            temp = []
```

```
            temp.append(str_list[count])

            count+=1

            condition = True

            while condition == True:


                if str_list[count] == '_':

                    if count != len(str_list):

                        temp.append(str_list[count])
                        count+=1

                        if count == len(str_list):

                            returned_list = condition_two(temp)

                            str_list[in_start:count] = returned_list

                            return str_list
                elif str(str_list[count]).isdigit():

                    if count!= len(str_list):

                        temp.append(str_list[count])

                        condition=False

                        returned_list = condition_three(temp)

                        str_list[in_start:count+1] = returned_list

                        if count==len(str_list) - 1:
                            return str_list
```

In [11]:
```
S ='_,_,_,40'
output_values=curve_smoothing(S)
output_values
```

Out[11]: [10, 10, 10, 10]

In [14]:
```
S= "40,_,_,_,60"
output_values=curve_smoothing(S)
output_values
```

Out[14]: [20, 20, 20, 20, 20]

In [15]:
```
S= "80,_,_,_,_"
output_values=curve_smoothing(S)
output_values
```

Out[15]: [16, 16, 16, 16, 16]

In [16]:
```
S= "_,_,30,_,_,_,50,_,_"
output_values=curve_smoothing(S)
output_values
```

Out[16]: [10, 10, 12, 12, 12, 12, 4, 4, 4]

8: probabilities

In [59]:

```
def findColumns(A):
    '''
    this function returns two columns of given 2d-list
    '''
    column_one ,column_two = [], []

    for item in A:
        column_one.append(item[0])
        column_two.append(item[1])

    return column_one, column_two

def findUnique(A):
    '''
    this function returns list of unique lists.
    '''
    temp_list = list(set(A))
    return list(temp_list)


def compute_conditional_probabilites(input_list):

    # find column_one and column_two
    column_one,column_two = findColumns(A)

    # find unique columns
    unique_column_one = findUnique(column_one)
    unique_column_two = findUnique(column_two)

    # sort unique found columns
    unique_column_one.sort()
    unique_column_two.sort()

    for i in unique_column_one: # iterate over all unique column one
        for j in unique_column_two: # iterate over all unique column two
            print('P(F=={})|S=={}) = {}/{}'.format(i,j, A.count([i,j]),column_two.count(j)),end=' ')

        print(' ')
```

In [60]:
```
A = [['F1','S1'],['F2','S2'],['F3','S3'],['F1','S2'],['F2','S3'],
     ['F3','S2'],['F2','S1'],['F4','S1'],['F4','S3'],['F5','S1']]
compute_conditional_probabilites(A)
```

```
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
```

In [ ]:

9: Operation on Sentences

In [19]:
```
def string_features(S1, S2):
    a=0
    b=[]
    c=[]
    listS1=S1.split() #split the string and change in list
    listS2=S2.split()
    for i,j in zip(listS1,listS2): # merge the lists
        if i==j:
            a +=1 #for counting the same words
        else:
            b.append(i)
            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']
```

10: Error function

In [24]:
```
import math
def compute_log_loss(A):
    loss=0
    for i in A:
        count=(-1/len(A))*((int(i[0])*(math.log(float(i[1]),10)))+((1-int(i[0]))*(math.log((1-float(i[1])),10))))
        loss += count
    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
```