



**University of Mumbai**

**PRACTICAL JOURNAL**

**PSDS2P2: SOFT COMPUTING**  
**(MSc. Computer Science with specialization in Data Science 2021-2023)**

**SEM-II**

**Submitted by**

**Sumon Singh  
Roll No. 16**

## : INDEX :

Sr. No	Title
1)	McCulloch Pitt
2)	Hebb's Network
3)	Max Net
4)	Hamming Network
5)	Kohonen Self-organizing map
6)	BAM Network
7)	Operations on Fuzzy sets
8)	Cartesian product of Fuzzy sets
9)	De-Fuzzification
10)	Arithmetic Operations

# Practical-1

## McCulloch Pitt

1] Solve single input McCulloch-Pitts (NOT gate)

For  $W_1 = -1, b=0$

Condition  $X_1 = 0, X_2 = 1$

Solution:

Solve the single input McCulloch-Pitts (NOT gate) for  $w_1 = -1, b = 0$   
Condition  $X_1 = 0, X_2 = 1$

a)  $X_1 = 0$

$$t = X_1 w_1$$

$$= 0 \times -1$$

$= 0$ , As  $t > 0$  then  $y = 1$

$$\therefore y = 1$$

b)  $X_2 = 1$

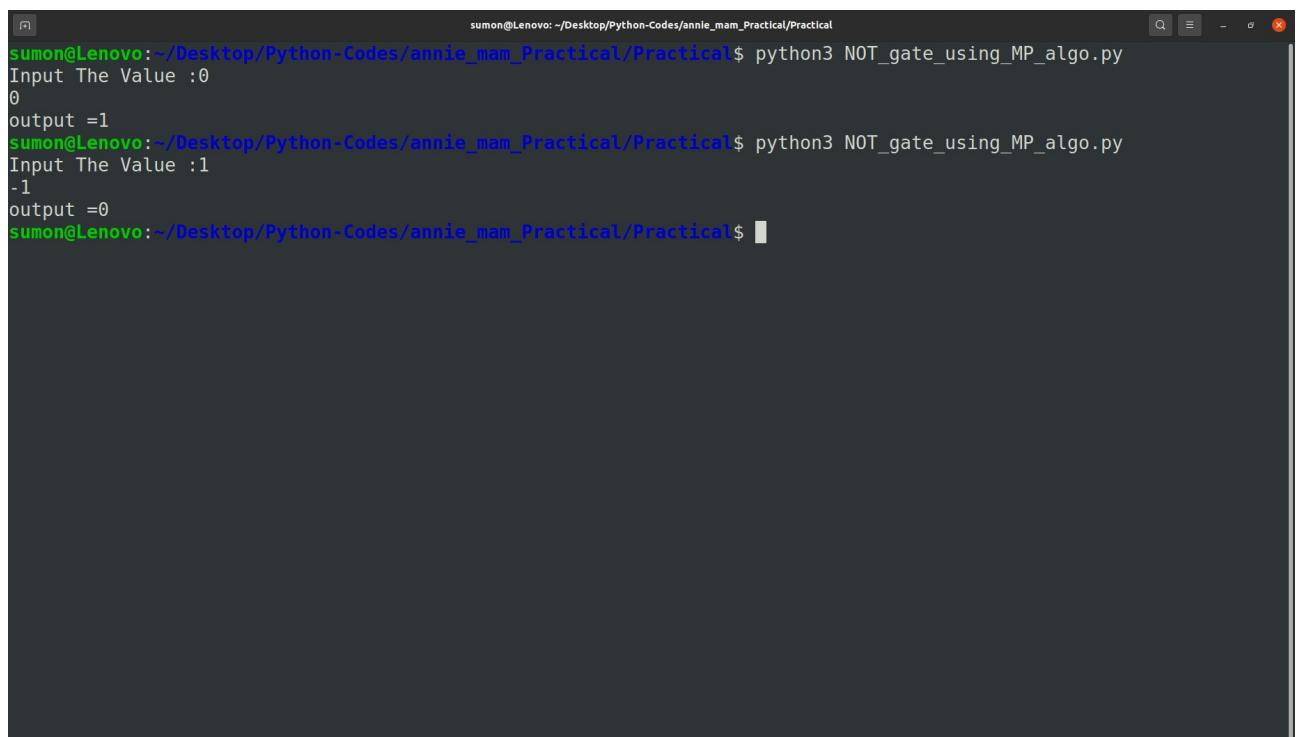
$$t = X_2 w_1 = 1 \times -1 = -1, \text{ As } t < 0 \text{ then } y = 0$$

$$\therefore y = 0$$

Code:

```
import sys
x1=int(input('Input The Value :'))
if x1 not in [0,1]:
    sys.exit('Wrong input ,it should be 1/0')
# weight = -1
w1=-1
t=w1*x1
print(t)
tr_val=0 # Threshold value
if t>=tr_val:
    print('output =1')
else:
    print('output =0')
```

Output:



```
sumon@Lenovo:~/Desktop/Python-Codes/annie_mam_Practical/Practical$ python3 NOT_gate_using_MP_algo.py
Input The Value :0
0
output =1
sumon@Lenovo:~/Desktop/Python-Codes/annie_mam_Practical/Practical$ python3 NOT_gate_using_MP_algo.py
Input The Value :1
-1
output =0
sumon@Lenovo:~/Desktop/Python-Codes/annie_mam_Practical/Practical$
```

2] Solve two-input McCulloch Pitts (OR gate)

For  $W_1 = W_2 = 1$ ,  $b = -1$

Conditions [a]  $X_1 = X_2 = 0$

[b]  $X_1 = 0, X_2 = 1$

[c]  $X_1 = 1, X_2 = 0$

[d]  $X_1 = X_2 = 1$

Draw a Hyper-line for Class A and Class B of the Boolean function

Class A is  $b + W_1 X_1 + W_2 X_2 \geq 0$

Class B is  $b + W_1 X_1 + W_2 X_2 < 0$

Solution:

2) solve two input McCullagh Pitts (OR gate)  
for  $\omega_1 = \omega_2 = 1, b = -1$

Condition:- a)  $x_1 = x_2 = 0$  b)  $x_1 = 0, x_2 = 1$   
c)  $x_1 = 1, x_2 = 0$  d)  $x_1 = 1, x_2 = 1$

$$\Rightarrow a) x_1 = x_2 = 0, \omega_1 = \omega_2 = 1, b = -1$$

$$\therefore t = x_1\omega_1 + x_2\omega_2 + b = -1$$

$$\therefore t = -1$$

As, ~~t < 0~~  $\therefore Y = 0$

$$b) x_1 = 0, x_2 = 1, \omega_1 = \omega_2 = 1, b = -1$$

$$\therefore t = x_1\omega_1 + x_2\omega_2 + b = 0 \times 1 + 1 \times 1 - 1 = 0$$

As,  $t > 0 \therefore Y = 1$

$$c) x_1 = 1, x_2 = 0, \omega_1 = \omega_2 = 1, b = -1$$

$$\therefore t = x_1\omega_1 + x_2\omega_2 + b = 1 \times 1 + 0 \times 1 - 1 = 0$$

As,  $t > 0 \therefore Y = 1$

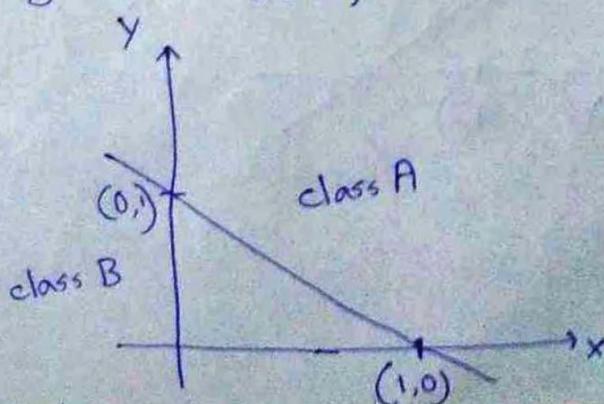
$$d) x_1 = x_2 = 1, \omega_1 = \omega_2 = 1, b = -1$$

$$\therefore t = 1 \times 1 + 1 \times 1 - 1 = 1$$

As,  $t > 0 \therefore Y = 1$

Class A where  $t > 0$ , i.e. when  $Y = 1$

Class B where  $t < 0$ , i.e. when  $Y = 0$



Code:

```
import sys
# For inputs
x1 = int(input('The first value is: '))
x2 = int(input('The second value is: '))
if x1 not in [0,1]:
    sys.exit('Wrong input, it should be 1/0')
if x2 not in [0,1]:
    sys.exit('Wrong input, it should be 1/0')
# For weights
w1 = 1
w2 = 1
b = -1
t = w1*x1 + w2*x2 + b
print('t=',t)
#Threshold value
6
tr_val = 0
if t>=tr_val:
    print('Output = 1')
else:
    print('Output = 0')
```

Output:

```
sumon@Lenovo:~/Desktop/Python-Codes/annie_mam_Practical/Practical$ python3 OR_gate_using_MP_algo.py
Input The First Value :0
Input The Second Value :1
0
output =1
sumon@Lenovo:~/Desktop/Python-Codes/annie_mam_Practical/Practical$ python3 OR_gate_using_MP_algo.py
Input The First Value :0
Input The Second Value :0
-1
output =0
sumon@Lenovo:~/Desktop/Python-Codes/annie_mam_Practical/Practical$ python3 OR_gate_using_MP_algo.py
Input The First Value :1
Input The Second Value :0
0
output =1
sumon@Lenovo:~/Desktop/Python-Codes/annie_mam_Practical/Practical$ python3 OR_gate_using_MP_algo.py
Input The First Value :1
Input The Second Value :1
1
output =1
sumon@Lenovo:~/Desktop/Python-Codes/annie_mam_Practical/Practical$ █
```

3] Solve two-input McCulloch Pitts for AND gate

$$B = -2, w_1 = w_2 = 1.$$

Conditions [a]  $X_1=X_2 = 0$

[b]  $X_1= 0, X_2= 1$

[c]  $X_1= 1, X_2= 0$

[d]  $X_1 = X_2= 1$

Solution:

3) Solve ~~the~~ two input McCulloch Pitts for AND gate  $b = -2, \omega_1 = \omega_2 = 1$   
 condition a)  $x_1 = x_2 = 0$  b)  $x_1 = 0, x_2 = 1$  c)  $x_1 = 1, x_2 = 0$   
 d)  $x_1 = x_2 = 1$

$\Rightarrow$

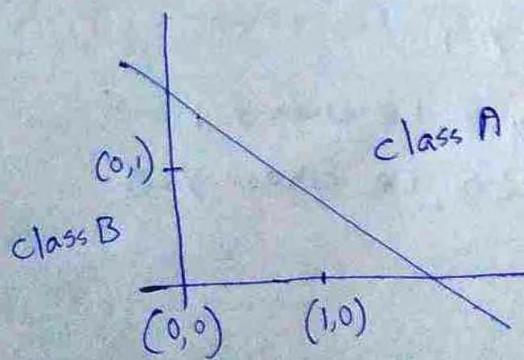
<u>to</u>	<u><math>x_1</math></u>	<u><math>x_2</math></u>	<u><math>\omega_1</math></u>	<u><math>\omega_2</math></u>	<u><math>b</math></u>	<u><math>t</math></u>	<u><math>Y</math></u>
a) 1	0	0	1	1	-2	-2	0
b) 2	0	1	1	1	-2	-1	0
c) 3	1	0	1	1	-2	-1	0
d) 4	1	1	1	1	-2	0	1

$$t = \omega_1 x_1 + \omega_2 x_2 + b$$

if  $t > 0$  then  $Y = 1$

if  $t < 0$  then  $Y = 0$

class A is  $\omega_1 x_1 + \omega_2 x_2 + b > 0$ , i.e where  $Y = 1$   
 class B is  $\omega_1 x_1 + \omega_2 x_2 + b < 0$ , i.e where  $Y = 0$



Code:

```
# The inputs
x1=int(input('Input The First Value :'))
x2=int(input('Input The Second Value :'))
if x1 not in [0,1]:
    sys.exit('Wrong input ,it should be 1/0')
if x2 not in [0,1]:
    sys.exit('Wrong input ,it should be 1/0')
# Weights and bias
w1 = 1
w2 = 1
b = -2
t = w1*x1 + w2*x2 + b
print('t = ',t)
# Threshold value
tr_val = 0
if t>=tr_val:
    print('output = 1')
else:
    print('output = 0')
```

Output:

```
sumon@Lenovo:~/Desktop/Python-Codes/annie_mam_Practical/Practical$ python3 And_gate_using_MP_algo.py
Input The First Value :0
Input The Second Value :0
-2
output =0
sumon@Lenovo:~/Desktop/Python-Codes/annie_mam_Practical/Practical$ python3 And_gate_using_MP_algo.py
Input The First Value :0
Input The Second Value :1
-1
output =0
sumon@Lenovo:~/Desktop/Python-Codes/annie_mam_Practical/Practical$ python3 And_gate_using_MP_algo.py
Input The First Value :1
Input The Second Value :0
-1
output =0
sumon@Lenovo:~/Desktop/Python-Codes/annie_mam_Practical/Practical$ python3 And_gate_using_MP_algo.py
Input The First Value :1
Input The Second Value :1
0
output =1
sumon@Lenovo:~/Desktop/Python-Codes/annie_mam_Practical/Practical$
```

4] In a public transport there is a seat for specially-abled and senior citizen.

There are four given situations. You need to decide when the seat can be occupied.

[a] Mention the situations or scenarios.

[b] Analyse the situations using the McCulloch-Pitts neural model.

[c] Use the value of both weights as 1 and scenarios as 1 and 0, b= 1, threshold as 1.

Solution:

4) a) Mention the scenarios:-

1) Scenario 1: The person is specially abled and senior citizen

$$x_1 = 1, x_2 = 1$$

2) scenario 2: The person is specially abled but not senior citizen

$$x_1 = 1, x_2 = 0$$

3) scenario 3: The person is senior citizen but not specially abled

$$x_1 = 0, x_2 = 1$$

4) scenario 4: The person is neither senior citizen nor specially abled

$$x_1 = 0, x_2 = 0$$

b) Analyse situation with McCullagh Pitts

a)  $x_1 = x_2 = 1, w_1 = w_2 = 1, b = 1$   
 $\therefore t = x_1 w_1 + x_2 w_2 + b = 3$ , As  $t > 2 \therefore y = 1$

b)  $x_1 = 1, x_2 = 0, w_1 = w_2 = 1, b = 1$   
 $\therefore t = 1 \times 1 + 0 \times 1 + 1 = 2$ , As  $t > 2 \therefore y = 1$

c)  $x_1 = 0, x_2 = 1, w_1 = w_2 = 1, b = 1$   
 $\therefore t = 0 \times 1 + 1 \times 1 + 1 = 2$ , As  $t > 2 \therefore y = 1$

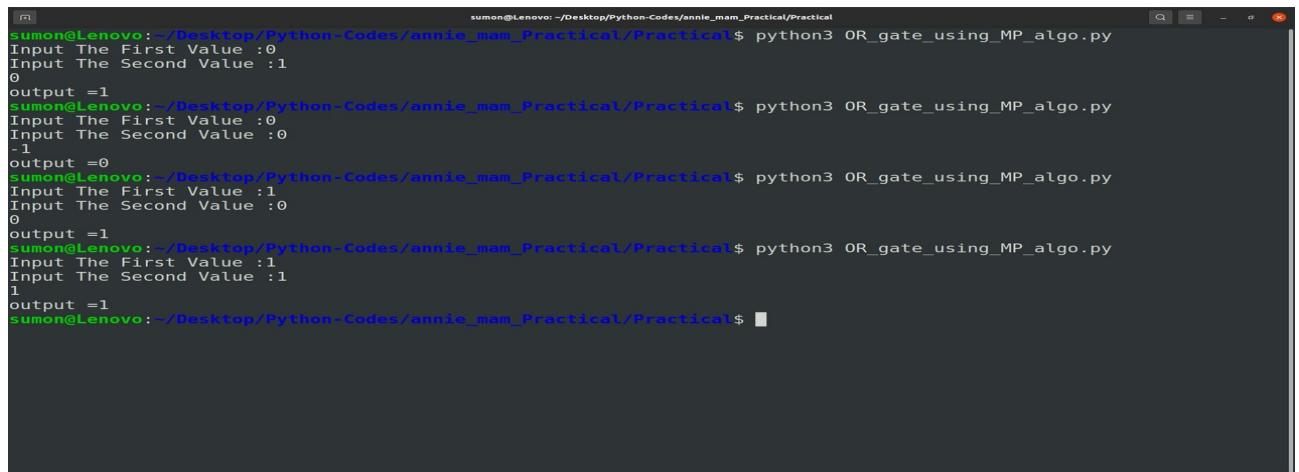
d)  $x_1 = 0, x_2 = 0, w_1 = w_2 = 1, b = 1$   
 $\therefore t = 0 \times 1 + 0 \times 1 + 1 = 1$ , As  $t < 2 \therefore y = 0$

Code:

```
import sys  
# For inputs  
x1 = int(input('The first value is: '))  
x2 = int(input('The second value is: '))  
if x1 not in [0,1]:  
    sys.exit('Wrong input, it should be 1/0')
```

```
if x2 not in [0,1]:  
    sys.exit('Wrong input, it should be 1/0')  
# For weights  
w1 = 1  
w2 = 1  
b = -1  
t = w1*x1 + w2*x2 + b  
print('t=',t)  
#Threshold value  
tr_val = 0  
if t>=tr_val:  
    print('Output = 1')  
else:  
    print('Output = 0')
```

## Output:



The image shows a terminal window with four separate command-line sessions. Each session starts with the command `python3 OR_gate_using_MP_algo.py`. In each session, the user is prompted for two inputs: "Input The First Value" and "Input The Second Value". The responses are as follows:

- Session 1: Input 0, Input 1. Output: 1
- Session 2: Input 0, Input 0. Output: 0
- Session 3: Input 1, Input 0. Output: 1
- Session 4: Input 1, Input 1. Output: 1

# Practical-2

## Hebb's Network

1] Design a Hebb's network to implement logical AND function (use bipolar inputs and targets).

- Predict the accuracy.
- Construct the decision boundary
- Draw the network.

Solution:

1) AND gate :-

$x_1$	$x_2$	$t$	bipolar 1, -1
-1	-1	-1	
-1	1	-1	
1	-1	-1	
1	1	1	

Hebb's network formula :-

$$w_{ij} = w_{ij} + mn_i y$$

$$b_n = b_n + my$$

random initial weight:  $w_1 = w_2 = 0$

learning rate:  $m = 1$

bias:  $b = 0$

Iteration	$x_1$	$x_2$	$t$	$y$	$w_1$	$w_2$	$b$
Initial					0	0	0
1	-1	-1	-1	-1	1	1	-1
2	-1	1	-1	-1	2	0	-2
3	1	-1	-1	-1	1	1	-3
4	1	1	1	1	2	2	-2

Final weights  $w_1 = w_2 = 2$ , bias  $b = -2$

Prediction Accuracy:-

a)  $x_1 = x_2 = -1 \therefore x_1 w_1 + x_2 w_2 + b = -2 + (-2) - 2 = -6 \neq 0, Y = -1$

$$b) \quad x_1 = -1, x_2 = 1$$

$$\therefore x_1\omega_1 + x_2\omega_2 + b = -2 + 2 - 2 = -2 < 0; y = -1$$

$$c) \quad x_1 = 1, x_2 = -1$$

$$\therefore x_1\omega_1 + x_2\omega_2 + b = 2 - 2 - 2 = -2 < 0; y = -1$$

$$d) \quad x_1 = 1, x_2 = 1$$

$$\therefore x_1\omega_1 + x_2\omega_2 + b = 2 + 2 - 2 = 2 > 0; y = 1$$

so, accuracy is 100%

Decision Line:

$$\omega_1 x_1 + \omega_2 x_2 + b \geq 0$$

$$2x_1 + 2x_2 - 2 \geq 0$$

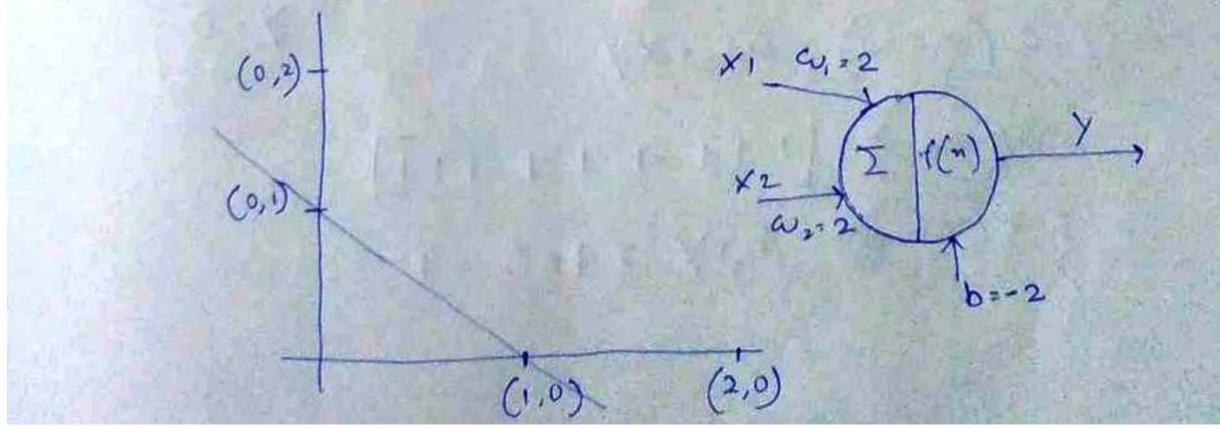
$$\therefore 2x_1 + 2x_2 - 2 = 0$$

$$(x_1 + x_2) = 1$$

$$x_1 = 1 - x_2$$

$$\text{if } x_2 = 0, x_1 = 1 \quad \text{if } x_1 = 0, x_2 = 1$$

Decision condition  $(1, 0), (0, 1)$



Code:

```
vector = []
# input
input_vector = int(input('Enter the Number of inputs: '))
for i in range(input_vector):
    vector.append(list(map(float, input(f'Enter the input-{i} : ').strip().split())))
# output
output = list(map(float, input(f'Enter the output-{i} : ').strip().split()))
# weights
weight = list(map(float, input(f'Enter the weights : ').strip().split()))
# learning rate
n = float(input(f'Enter the learning rate: '))
# bias
bias = float(input(f'Enter the bias: '))
for i in range(len(vector[0])):
    for j in range(input_vector):
        weight[j] = weight[j] + n*output[i]*vector[j][i]
    bias = bias + n*output[i]
print('-----')
print(f'For input-{i} : ')
print(f'weight : {weight}')
print(f'bias : {bias}' )
```

Output:

```
sumon@Lenovo:~/Desktop/Python-Codes/annie_mam_Practical/Practical$ python3 Hebb.py
Enter the Number of inputs : 2
Enter the input-0 : -1 -1 1 1
Enter the input-1 : -1 1 -1 1
Enter the output : -1 -1 -1 1
Enter the weights : 0 0
Enter the learning rate : 0.5
Enter the bias : 0
-----
For input-0 :
weight : [0.5, 0.5]
bias : -0.5
-----
For input-1 :
weight : [1.0, 0.0]
bias : -1.0
-----
For input-2 :
weight : [0.5, 0.5]
bias : -1.5
-----
For input-3 :
weight : [1.0, 1.0]
bias : -1.0
sumon@Lenovo:~/Desktop/Python-Codes/annie_mam_Practical/Practical$
```

2] Using HEBB network solve the OR gate problem if the learning rate is 0.5.

- Predict the accuracy.
- Construct the decision boundary
- Draw the network.

Solution:

Q2 OR gate;

	$n_1$	$n_2$	$t$
-1	-1	-1	
-1	1	1	
1	-1	1	
1	1	1	

Hebb's network formula:-

$$w_{in} = w_0 + \eta n_1 y$$

$$b_{in} = b_0 + \eta n_2 y$$

$$w_1 = w_2 = 0 \text{ (Initial weight)}$$

$$\eta = 0.5 \text{ (Learning rate)}$$

$$b = 0 \quad t = y$$
  

It <sup>o</sup> Initial Values	$n_1$	$n_2$	$w_1$	$w_2$	$t$	$b$	$y$
1	-1	-1	0	0	-1	-0.5	-1
2	-1	1	0	1	1	0	1
3	1	-1	0.5	0.5	1	0.5	1
4	1	1	1	1	1	1	1

\*  $w_1 = w_2 = b = 1$  (Final weight and bias)

Prediction accuracy:-

$$(x_1, x_2) = (-1, -1) = n_1 w_1 + n_2 w_2 + b = -1$$

$$(x_1, x_2) = (-1, 1) = 1$$

$$(x_1, x_2) = (1, -1) = 1$$

$$(x_1, x_2) = (1, 1) = 3$$

Acc. - 100%

Code:

```
vector = []
# input
input_vector = int(input('Enter the Number of inputs: '))
for i in range(input_vector):
    vector.append(list(map(float, input(f'Enter the input-{i} : ').strip().split())))
# output
output = list(map(float, input(f'Enter the output-{i} : ').strip().split()))
# weights
weight = list(map(float, input(f'Enter the weights : ').strip().split()))
# learning rate
n = float(input(f'Enter the learning rate: '))
# bias
bias = float(input(f'Enter the bias: '))
for i in range(len(vector[0])):
    for j in range(input_vector):
        weight[j] = weight[j] + n*output[i]*vector[j][i]
    bias = bias + n*output[i]
print('-----')
print(f'For input-{i} : ')
print(f'weight : {weight}')
print(f'bias : {bias}' )
```

## Output:

```
sumon@Lenovo:~/Desktop/Python-Codes/annie_mam_Practical/Practical$ python3 Hebb.py
Enter the Number of inputs : 2
Enter the input-0 : -1 1 1 1
Enter the input-1 : -1 1 -1 1
Enter the output : -1 1 1 1
Enter the weights : 0 0
Enter the learning rate : 0.5
Enter the bias : 0
-----
For input-0 :
weight : [0.5, 0.5]
bias : -0.5
-----
For input-1 :
weight : [0.0, 1.0]
bias : 0.0
-----
For input-2 :
weight : [0.5, 0.5]
bias : 0.5
-----
For input-3 :
weight : [1.0, 1.0]
bias : 1.0
sumon@Lenovo:~/Desktop/Python-Codes/annie_mam_Practical/Practical$
```

3] Using HEBB rule, find the weights required to perform the classification pattern of the inputs given. The input pattern "+" has value 1 and the empty squares have value -1. Draw the network architecture. Pattern 1 output value is 1, Pattern 2 output value is 0

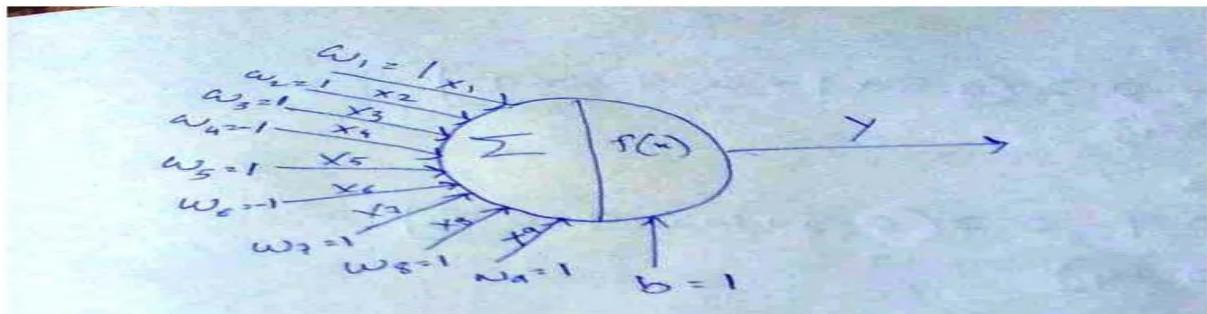
Solution:

3) Pattern 1 :  $x_1 = [1 \ 1 \ 1 \ -1 \ 1 \ -1 \ 1 \ 1]$ , output:  $-Y=1$   
 Pattern 2 :  $x_2 = [1 \ 1 \ 1 \ 1 \ -1 \ 1 \ 1 \ 1]$ , output:  $-Y=0$

Pattern 1:-  
 Initial weights  $w_0 = [0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0]$   
 weight updation formula :-  $w(\text{new}) = w_0 + x$   
 $w(\text{new}) = [1 \ 1 \ 1 \ -1 \ 1 \ -1 \ 1 \ 1]$

Initial bias  $b_0 = 0$   
 bias updation formula :-  $b(\text{new}) = b_0 + Y$   
 $= 0 + 1$   
 $b(\text{new}) = 1$

Pattern 2:-  
 $w_0 = w(\text{new}) \rightarrow$  Initial weight for pattern 2  
 $b_0 = b(\text{new}) \rightarrow$  Initial bias for pattern 2  
 $\therefore w(\text{new}) = w_0 + xy \quad Y=0$   
 $w(\text{new}) = [1 \ 1 \ 1 \ -1 \ 1 \ -1 \ 1 \ 1]$   
 $b(\text{new}) = b_0 + Y = 1 + 0 = 1$



Code:

```

vector = []
# input
input_vector = int(input('Enter the Number of inputs: '))
for i in range(input_vector):
    vector.append(list(map(float, input(f'Enter the input-{i} : ').strip().split())))
# output
output = list(map(float, input(f'Enter the output-{i} : ').strip().split()))
# weights
weight = list(map(float, input(f'Enter the weights : ').strip().split()))
# learning rate
n = float(input(f'Enter the learning rate: '))
# bias
bias = float(input(f'Enter the bias: '))
for i in range(len(vector[0])):
    for j in range(input_vector):
        weight[j] = weight[j] + n*output[i]*vector[j][i]
    
```

```
bias = bias + n*output[i]
print('-----')
print(f'For input-{i} : ')
print(f'weight : {weight}')
print(f'bias : {bias} ')
```

## Output:



A terminal window titled 'sumon@Lenovo: ~/Desktop/Python-Codes/annie\_mam\_Practical/Practical' is shown. The window contains the output of a Python script named 'Hebb.py'. The script prompts the user for various parameters: the number of inputs (9), individual input values (1, 1, 1, 1, -1, 1, 1, -1, 1), the output value (0), weights (0, 0, 0, 0, 0, 0, 0, 0, 0), the learning rate (1), and the bias (0). After these inputs, the script prints two sets of information for 'input-0' and 'input-1'. For 'input-0', it shows a weight vector [1.0, 1.0, 1.0, -1.0, 1.0, -1.0, 1.0, 1.0, 1.0] and a bias of 1.0. Similarly, for 'input-1', it shows the same weight vector and a bias of 1.0.

```
sumon@Lenovo:~/Desktop/Python-Codes/annie_mam_Practical/Practical$ python3 Hebb.py
Enter the Number of inputs : 9
Enter the input-0 : 1 1
Enter the input-1 : 1 1
Enter the input-2 : 1 1
Enter the input-3 : -1 1
Enter the input-4 : 1 -1
Enter the input-5 : -1 1
Enter the input-6 : 1 1
Enter the input-7 : 1 1
Enter the input-8 : 1 1
Enter the output : 1 0
Enter the weights : 0 0 0 0 0 0 0 0 0
Enter the learning rate : 1
Enter the bias : 0
-----
For input-0 :
weight : [1.0, 1.0, 1.0, -1.0, 1.0, -1.0, 1.0, 1.0, 1.0]
bias : 1.0
-----
For input-1 :
weight : [1.0, 1.0, 1.0, -1.0, 1.0, -1.0, 1.0, 1.0, 1.0]
bias : 1.0
sumon@Lenovo:~/Desktop/Python-Codes/annie_mam_Practical/Practical$
```

4] Using HEBB rule, classify the two-dimensional input pattern. Draw the network architecture. Pattern 1 output value is 1, Pattern 2 output value is -1

Solution:

Q-4

$$X_1 = [-1, -1, -1, -1, +1, +1, -1, -1, -1]$$

$$X_2 = [-1, -1, -1, +1, -1, +1, +1, -1, +1]$$

Pattern 1,  $y_1 = 1$   
 Pattern 2,  $y_2 = -1$

Pattern: 1

$$\omega_0 = [0, 0, 0, 0, 0, 0, 0, 0, 0], \quad y_1 = 1$$

$$\omega_n = \omega_0 + n_1 y_1, \quad b_n = b_0 + y_1$$

Updated bias:  $b = 1$   
 Updated weight:-

$$\omega_1 = [-1, -1, -1, -1, +1, +1, -1, -1, -1]$$

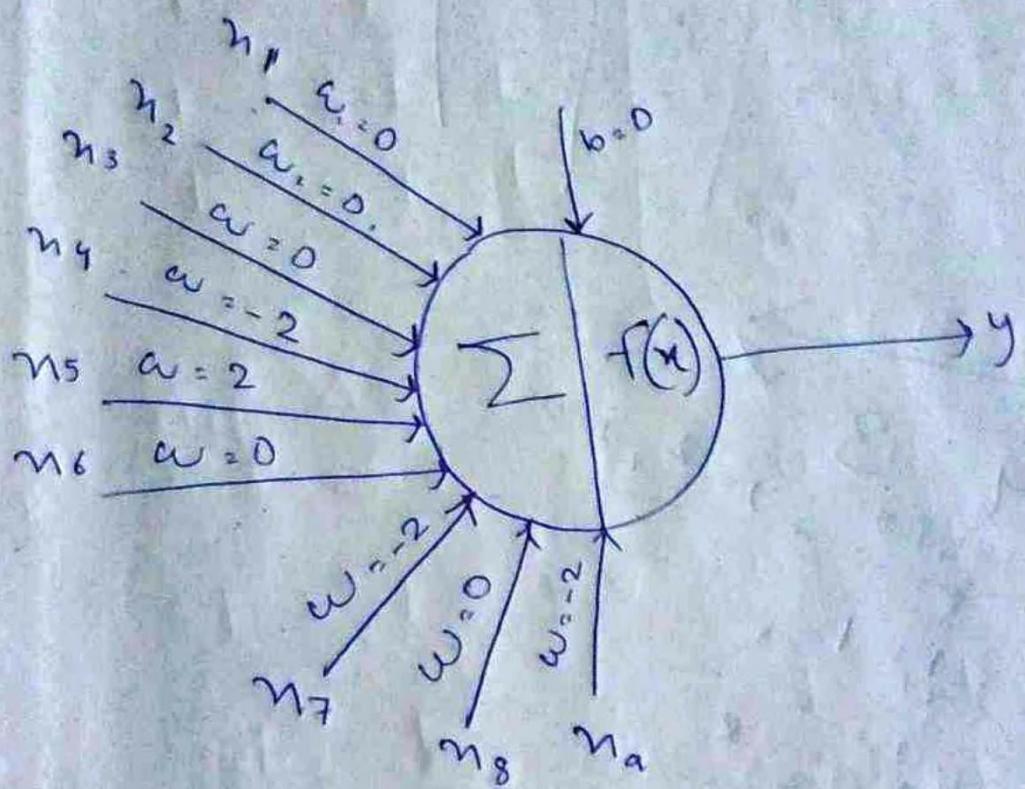
Pattern: 2

$$y_2 = -1 \quad b_n = b_0 + y_2$$

$$\omega_n = \omega_0 + n_2 y_2 \quad b = 0$$
 ~~$\omega_2 = [0, 0, 0, 0, 2, 0, 2, 0, 2]$~~ 

$$\omega_2 = [0, 0, 0, -2, 2, 0, -2, 0, -2]$$

## Network



Code:

```
vector = []
# input
input_vector = int(input('Enter the Number of inputs: '))
for i in range(input_vector):
    vector.append(list(map(float, input(f'Enter the input-{i} : ').strip().split())))
# output
output = list(map(float, input(f'Enter the output-{i} : ').strip().split()))
# weights
weight = list(map(float, input(f'Enter the weights : ').strip().split())))
# learning rate
n = float(input(f'Enter the learning rate: '))
# bias
bias = float(input(f'Enter the bias: '))
for i in range(len(vector[0])):
    for j in range(input_vector):
        weight[j] = weight[j] + n*output[i]*vector[j][i]
bias = bias + n*output[i]
print('-----')
print(f'For input-{i} : ')
print(f'weight : {weight}')
print(f'bias : {bias} ')
```

## Output:

```
sumon@Lenovo:~/Desktop/Python-Codes/annie_mam_Practical/Practical$ python3 Hebb.py
Enter the Number of inputs : 9
Enter the input-0 : -1 -1
Enter the input-1 : -1 -1
Enter the input-2 : -1 -1
Enter the input-3 : -1 1
Enter the input-4 : 1 -1
Enter the input-5 : 1 1
Enter the input-6 : -1 1
Enter the input-7 : -1 -1
Enter the input-8 : -1 1
Enter the output : 1 -1
Enter the weights : 0 0 0 0 0 0 0 0 0
Enter the learning rate : 1
Enter the bias : 0
-----
For input-0 :
weight : [-1.0, -1.0, -1.0, -1.0, 1.0, 1.0, -1.0, -1.0, -1.0]
bias : 1.0
-----
For input-1 :
weight : [0.0, 0.0, 0.0, -2.0, 2.0, 0.0, -2.0, 0.0, -2.0]
bias : 0.0
sumon@Lenovo:~/Desktop/Python-Codes/annie_mam_Practical/Practical$
```

## Practical-3

### Max-Net

1] A MaxNet has three inhibitory weights with  $\epsilon = 0.25$ .  
The net is initially activated by the input signal [0.1, 0.3, 0.9].  
The activation function  $f(x) = x$  if  $x \geq 0$   
 $0$  if  $x < 0$   
Find the winning neuron.

Solution:

$$1) [0.1, 0.3, 0.9], \epsilon = 0.25$$

Condition 1:-

$$a_1 = 0.1, \epsilon = 0.25$$

$$a_1 = f[a_1(t) - \epsilon \sum(a_i(t))] \rightarrow \text{formula}$$

$$\therefore a_1 = f[0.1 - 0.25(0.3 + 0.9)] \\ = f(-0.2) < 0$$

$$\therefore a_1 = 0$$

$$a_2 = f[0.3 - 0.25(0.1 + 0.9)] \\ = f(0.05) \therefore a_2 = 0.05$$

$$a_3 = f[0.9 - 0.25(0.3 + 0.1)] \\ = f(0.8) \\ \therefore a_3 = 0.8$$

New updated weights are  $[0, 0.05, 0.8]$

Condition 2:-

~~$a_1 = 0$~~

$$a_2 = f[0.05 - 0.25(0 + 0.8)] \\ = f(-0.15) < 0$$

$$\therefore a_2 = 0$$

$$a_3 = f[0.8 - 0.25 \times 0.05] \\ = f(0.7875) > 0 \\ \therefore a_3 = 0.7875$$

New weights are  $[0, 0, 0.7875]$

$\therefore$  The winning neuron is 3rd neuron

Code:

Code:

```
def calculate(l,e):
    total = sum(l)
    u_l = []
    for i in l:
        n_w = i - (e*(total - i))
        if n_w < 0:
            n_w = 0
        u_l.append(n_w)
    return u_l

def check_active(l):
    p = 0
    loc = 0
    n = 0
    for i in range(len(l)):
        if l[i] > 0:
            p = p + 1
            loc = i + 1
        else:
            n = n + 1

    if (p==1):
        return loc
    else:
        return 0
# weights
w = list(map(float, input('Enter the initial values of
activations one by one
separated by space : ').strip().split())))

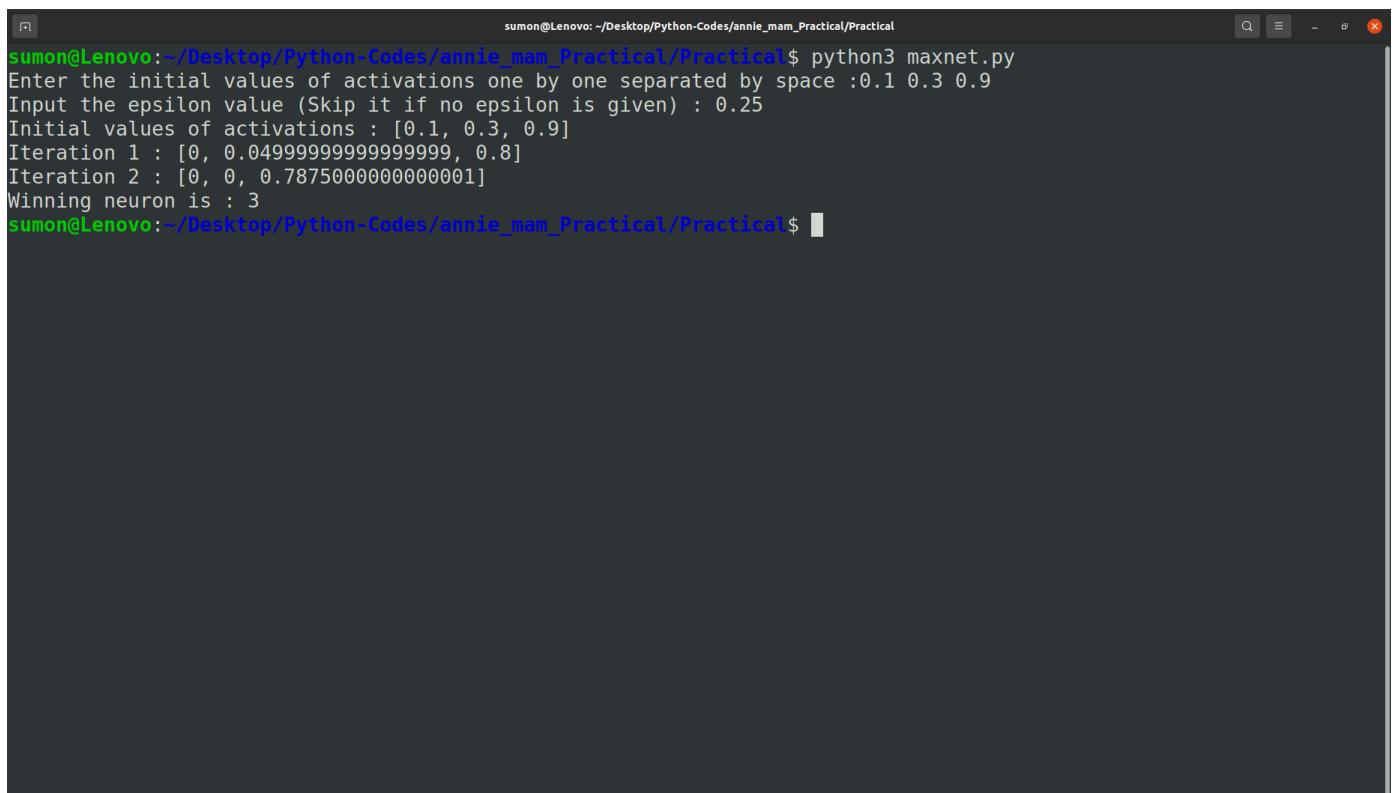
```

```

n = len(w)
temp_e = 1/n # epsilon
e = float(input('Input the epsilon value (Skip if no epsilon is given) : ') or
temp_e)
k = 0
print(f'Initial values of activations : {w}')
while(1):
    w = calculate(w,e)
    k = k+1
    print(f'Iteration {k} : {w}')
    c = check_active(w)
    if(c):
        print('Winning neuron is : '+str(c))
        break

```

## Output:



```

sumon@Lenovo:~/Desktop/Python-Codes/annie_mam_Practical/Practical$ python3 maxnet.py
Enter the initial values of activations one by one separated by space :0.1 0.3 0.9
Input the epsilon value (Skip it if no epsilon is given) : 0.25
Initial values of activations : [0.1, 0.3, 0.9]
Iteration 1 : [0, 0.0499999999999999, 0.8]
Iteration 2 : [0, 0, 0.7875000000000001]
Winning neuron is : 3
sumon@Lenovo:~/Desktop/Python-Codes/annie_mam_Practical/Practical$ █

```

2] Construct MaxNet with five neurons having initial activation of five nodes [0.5, 0.9, 1, 0.9, 0.9].

The activation function  $f(x) = \begin{cases} x & \text{if } x \geq 0 \\ 0 & \text{if } x < 0 \end{cases}$

Solution:

$$2) \epsilon = \frac{1}{n} = \frac{1}{5} = 0.2 ; [0.5, 0.9, 1, 0.9, 0.9]$$

Condition 1:

$$a_1 = 0.5, \epsilon = 0.2$$

$$\begin{aligned} a_1 &= f[a_1(t) - \epsilon \sum a_i(t)] \rightarrow \text{formula} \\ &= f[0.5 - 0.2(0.9 + 1 + 0.9 + 0.9)] \\ &= f(-0.24) = 0 \end{aligned}$$

$$a_2 = 0.9, \epsilon = 0.2$$

$$\begin{aligned} &= f[0.9 - 0.2(0.5 + 1 + 0.9 + 0.9)] \\ &= f(0.24) \end{aligned}$$

$$a_3 = 1, \epsilon = 0.2$$

$$\begin{aligned} &= f[1 - 0.2(0.5 + 0.9 + 0.9 + 0.9)] \\ &= f(0.36) \end{aligned}$$

$$a_4 = 0.9, \epsilon = 0.2$$

$$\therefore f [0.9 - 0.2(0.5 + 1 + 0.9 + 0.9)]$$

$$= f(0.24)$$

$$a_5 = 0.9, \epsilon = 0.2$$

$$\therefore f [0.9 - 0.2(0.5 + 1 + 0.9 + 0.9)]$$

$$= f(0.24)$$

Condition 2 :-

$$[0, 0.24, 0.36, 0.24, 0.24]$$

$$a_1(2) = [0 - 0.2(0.24 + 0.36 + 0.24 + 0.24)]$$

$$= (-0.216) = 0$$

$$a_2(2) = [0.24 - 0.2(0 + 0.36 + 0.24 + 0.24)]$$

$$= (0.072)$$

$$a_3(2) = [0.36 - 0.2(0 + 0.24 + 0.24 + 0.24)]$$

$$= (0.216)$$

$$a_4(2) = [0.24 - 0.2(0 + 0.24 + 0.36 + 0.24)]$$

$$= (0.072)$$

$$a_5(2) = [0.24 - 0.2(0 + 0.24 + 0.36 + 0.24)]$$

$$= (0.072)$$

Sond. Neuron 3 :-

$$[0, 0.072, 0.216, 0.072, 0.072]$$

$$\alpha_1(3) = [0 - 0.2(0.072 + 0.216 + 0.072 + 0.072)] \\ = (-0.0864) \times 0$$

$$\alpha_2(3) = [0.072 - 0.2(0 + 0.216 + 0.072 + 0.072)] \\ = 0$$

$$\alpha_3(3) = [0.216 - 0.2(0 + 0.072 + 0.072 + 0.072)] \\ = 0.1728$$

$$\alpha_4(3) = 0$$

$$\alpha_5(3) = 0$$

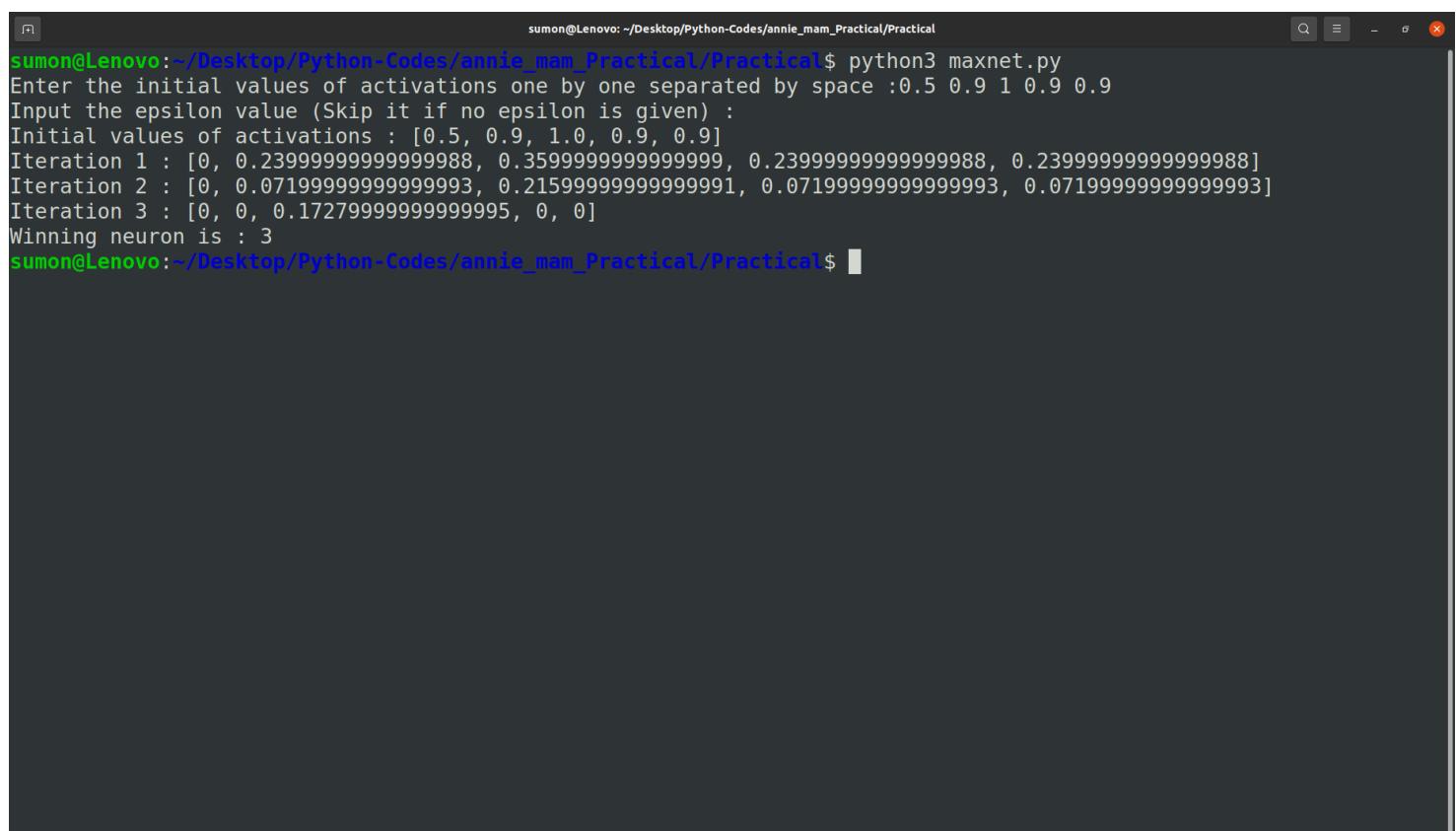
∴ The winning neuron is 3

Code:

```
def calculate(l,e):
total = sum(l)
u_l = []
for i in l:
n_w = i - (e*(total - i))
if n_w < 0:
n_w = 0
u_l.append(n_w)
return u_l
def check_active(l):
p = 0
loc = 0
n = 0
for i in range(len(l)):
if l[i] > 0:
p = p + 1
loc = i + 1
else:
n = n + 1
if (p==1):
return loc
else:
return 0
# weights
w = list(map(float, input('Enter the initial values of activations
one by one
separated by space : ').strip().split()))
n = len(w)
temp_e = 1/n # epsilon
e = float(input('Input the epsilon value (Skip if no epsilon is
given) : ') or
temp_e)
k = 0
print(f'Initial values of activations : {w}')
```

```
while(1):
w = calculate(w,e)
k = k+1
print(f'Iteration {k} : {w}')
c = check_active(w)
if(c):
print('Winning neuron is : '+str(c))
break
```

## Output:



A screenshot of a terminal window titled "sumon@Lenovo: ~/Desktop/Python-Codes/annie\_mam\_Practical/Practical". The window shows the command "python3 maxnet.py" being run. The program prompts for initial activation values and an epsilon value. It then displays three iterations of the maxnet algorithm, showing the activation values for each iteration. Finally, it prints the winning neuron index as 3.

```
sumon@Lenovo:~/Desktop/Python-Codes/annie_mam_Practical/Practical$ python3 maxnet.py
Enter the initial values of activations one by one separated by space :0.5 0.9 1 0.9 0.9
Input the epsilon value (Skip it if no epsilon is given) :
Initial values of activations : [0.5, 0.9, 1.0, 0.9, 0.9]
Iteration 1 : [0, 0.239999999999988, 0.359999999999999, 0.239999999999988, 0.239999999999988]
Iteration 2 : [0, 0.071999999999993, 0.215999999999991, 0.071999999999993, 0.071999999999993]
Iteration 3 : [0, 0, 0.172799999999995, 0, 0]
Winning neuron is : 3
sumon@Lenovo:~/Desktop/Python-Codes/annie_mam_Practical/Practical$
```

3] Construct the action of Maxnet with 4 neurons and inhibitory weight  $\epsilon = 0.3$ .

The neurons given initial activation input signal  $a_1(0) = 0.3$ ,  $a_2(0) = 0.3$ ,

$a_3(0) = 0.5$ ,  $a_4(0) = 0.7$ . Simulate the iteration until coverage.

Solution:

$$3) \epsilon = 0.3, [0.3, 0.3, 0.5, 0.7]$$

Condition 1:

$$\alpha_1 = 0.3 \quad \epsilon = 0.3$$

$$\alpha_1 = f[\alpha_j(t) - \sum_{i \neq j} \alpha_i(t)] \rightarrow \text{Formula}$$

$$\begin{aligned}\alpha_1 &= 0.3 - 0.3[0.3 + 0.5 + 0.7] \\ &= f(-0.15) = 0\end{aligned}$$

$$\alpha_2 = 0.3, \epsilon = 0.3$$

$$\begin{aligned}\alpha_2 &= 0.3 - 0.3[0.3 + 0.5 + 0.7] \\ &= f(-0.15) = 0\end{aligned}$$

$$\alpha_3 = 0.5 \quad \epsilon = 0.3$$

$$\begin{aligned}\alpha_3 &= 0.5 - 0.3[0.3 + 0.3 + 0.7] \\ &= f(0.11)\end{aligned}$$

$$a_4 = 0.7, e = 0.3$$

$$a_4 = 0.7 - 0.3[0.3 + 0.3 + 0.5]$$

$$= f(0.37)$$

Condition 2 :-

$$[0, 0, 0.11, 0.37]$$

$$a_1 = 0 - 0.3[0 + 0.11 + 0.37]$$

$$= -0.144 < 0 \therefore a_1 = 0$$

$$a_2 = 0 - 0.3[0 + 0.11 + 0.37]$$

$$= -0.144 < 0 \therefore a_2 = 0$$

$$a_3 = 0.11 - 0.3[0 + 0 + 0.37]$$

$$= -0.001 < 0 \therefore 0$$

$$a_4 = 0.37 - 0.3[0 + 0 + 0.11]$$

$$= 0.337$$

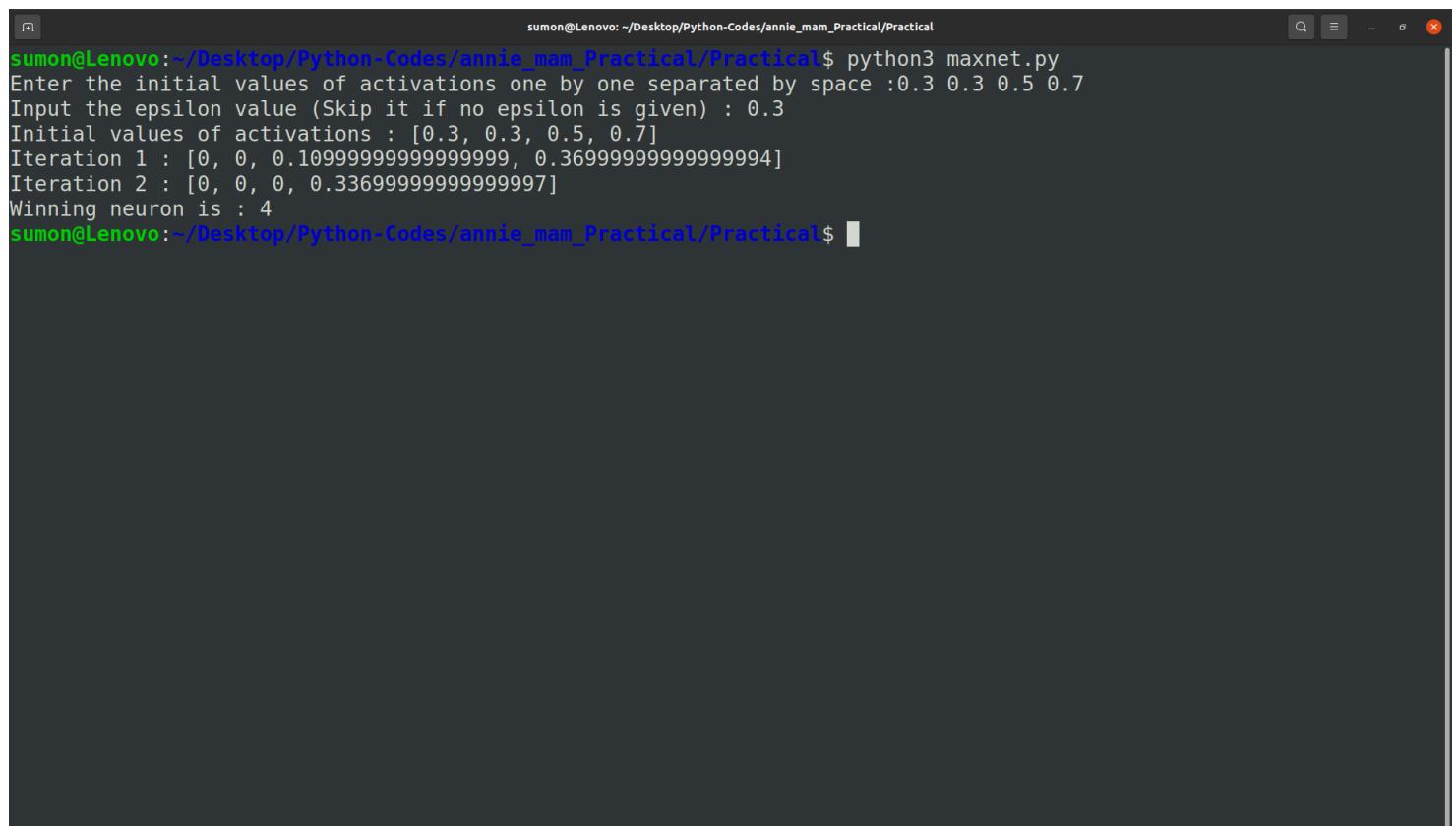
The winning neuron is 4

Code:

```
def calculate(l,e):
total = sum(l)
u_l = []
for i in l:
n_w = i - (e*(total - i))
if n_w < 0:
n_w = 0
u_l.append(n_w)
return u_l
def check_active(l):
p = 0
loc = 0
n = 0
for i in range(len(l)):
if l[i] > 0:
p = p + 1
loc = i + 1
else:
n = n + 1
if (p==1):
return loc
else:
return 0
# weights
w = list(map(float, input('Enter the initial values of activations
one by one
separated by space : ').strip().split()))
n = len(w)
temp_e = 1/n # epsilon
e = float(input('Input the epsilon value (Skip if no epsilon is
given) : ') or
temp_e)
k = 0
print(f'Initial values of activations : {w}')
while(1):
```

```
w = calculate(w,e)
k = k+1
print(f'Iteration {k} : {w}')
c = check_active(w)
if(c):
    print('Winning neuron is : '+str(c))
    break
```

## Output:



A terminal window titled "sumon@Lenovo: ~/Desktop/Python-Codes/annie\_mam\_Practical/Practical". The window shows the command "python3 maxnet.py" being run. It prompts for initial activation values (0.3, 0.3, 0.5, 0.7) and an epsilon value (0.3). It then displays two iterations of activations and identifies the winning neuron as 4.

```
sumon@Lenovo:~/Desktop/Python-Codes/annie_mam_Practical/Practical$ python3 maxnet.py
Enter the initial values of activations one by one separated by space :0.3 0.3 0.5 0.7
Input the epsilon value (Skip it if no epsilon is given) : 0.3
Initial values of activations : [0.3, 0.3, 0.5, 0.7]
Iteration 1 : [0, 0, 0.1099999999999999, 0.3699999999999994]
Iteration 2 : [0, 0, 0, 0.3369999999999997]
Winning neuron is : 4
sumon@Lenovo:~/Desktop/Python-Codes/annie_mam_Practical/Practical$
```

4] Construct a Maxnet with four neurons and inhibitory weight  $\epsilon = 0.2$ , given  
the initial activations as follows  $a_1(0) = 0.3$ ;  $a_2(0) = 0.5$ ;  $a_3(0) = 0.7$ ;  $a_4(0) = 0.9$   
The activation function  $f(x) = \begin{cases} x & \text{if } x > 0 \\ 0 & \text{if } x \leq 0 \end{cases}$

Solution:

$$4) \quad a_1(0) = 0.3, \quad \epsilon = 0.2$$

$$\text{Condition 1:} \\ a_1(1) = f[a_1(t) - \epsilon [\sum(a_i(t))]]$$

$$a_1(1) = f[0.3 - 0.2 \times (0.5 + 0.3 + 0.9)] \\ = f[-0.12] < 0 \\ \therefore a_1(1) < 0$$

$$a_3(1) = f[a_3(t) - \epsilon [\sum(a_i(t))]] \\ = f[0.7 - 0.2(0.3 + 0.5 + 0.9)] \\ = f[0.36] > 0 \\ \therefore a_3(1) = 0.36$$

$$a_2(1) = f[a_2(t) - \epsilon [\sum(a_i(t))]] \\ = f[0.5 - 0.2(0.3 + 0.7 + 0.9)] \\ = f[0.12] > 0 \\ \therefore a_2(1) = 0.12$$

$$a_4(1) = f[a_4(t) - \epsilon [\sum(a_i(t))]] \\ = f[0.9 - 0.2(0.3 + 0.5 + 0.7)] \\ = f(0.6) > 0 \\ \therefore a_4 = 0.6$$

New weights are :- [0, 0.12, 0.36, 0.6]

Condition 2:-

$$\alpha_1(2) = 0$$

$$\alpha_2(2) = f[0.12 - 0.2(0.36 + 0.6)]$$

$$= f(-0.072) < 0$$

$$\therefore \alpha_2(2) = 0$$

$$\alpha_3(2) = f[0.36 - 0.2(0.12 + 0.6)]$$

$$= f(0.216) > 0$$

$$\therefore \alpha_3(2) = 0.216$$

$$\alpha_4(2) = f[0.6 - 0.2(0.36 + 0.12)]$$

$$= f(0.504) > 0$$

$$\therefore \alpha_4(2) = 0.504$$

New weights are :- [0, 0, 0.216, 0.504]

Condition 3:-

$$\alpha_1(3) = 0$$

$$\alpha_2(3) = 0$$

$$\alpha_3(3) = f[0.216 - 0.2(0.504)]$$

$$= f(0.116) > 0$$

$$\therefore \alpha_3(3) = 0.116$$

$$\alpha_4(3) = f[0.504 - 0.2 \times 0.216]$$

$$= f(0.4608) > 0$$

$$\therefore \alpha_4(3) = 0.4608$$

New weights are :- [0, 0, 0.116, 0.4608]

Condition 4 :-

$$\alpha_1(4) = 0$$

$$\alpha_2(4) = 0$$

$$\begin{aligned}\alpha_3(4) &= f[0.116 - 0.2(0.468)] \\ &= f(0.02304) > 0 \\ \therefore \alpha_3(4) &= 0.02304\end{aligned}$$

$$\begin{aligned}\alpha_4(4) &= f[0.4608 - 0.2 \times 0.116] \\ &= f(0.4376) > 0\end{aligned}$$

$$\therefore \alpha_4(4) = 0.4376$$

New weights are :- [0, 0, 0.02304, 0.4376]

Condition 5 :-

$$\alpha_1(5) = 0$$

$$\alpha_2(5) = 0$$

$$\begin{aligned}\alpha_3(5) &= f[0.024 - 0.2(0.4376)] \\ &= f(-0.06) < 0 \\ \therefore \alpha_3(5) &= 0\end{aligned}$$

$$\begin{aligned}\alpha_4(5) &= f[0.438 - 0.2(0.024)] \\ &= f(0.4332) > 0 \\ \therefore \alpha_4(5) &= 0.4332\end{aligned}$$

New weights are [0, 0, 0, 0.4332]

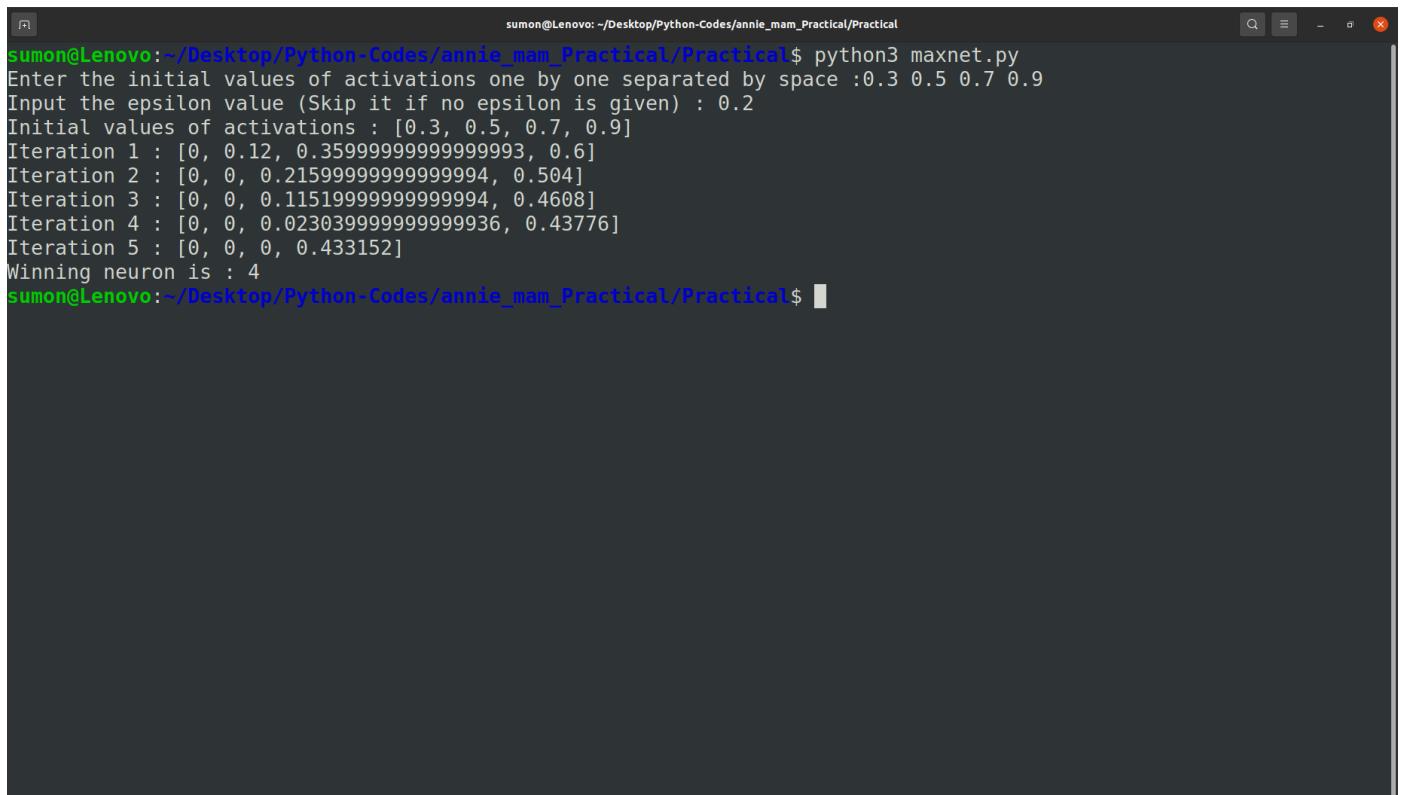
$\therefore$  The winning neuron is the 4th neuron.

Code:

```
def calculate(l,e):
total = sum(l)
u_l = []
for i in l:
n_w = i - (e*(total - i))
if n_w < 0:
n_w = 0
u_l.append(n_w)
return u_l
def check_active(l):
p = 0
loc = 0
n = 0
for i in range(len(l)):
if l[i] > 0:
p = p + 1
loc = i + 1
else:
n = n + 1
if (p==1):
return loc
else:
return 0
# weights
w = list(map(float, input('Enter the initial values of
activations one by one
separated by space : ').strip().split()))
n = len(w)
temp_e = 1/n # epsilon
e = float(input('Input the epsilon value (Skip if no epsilon is
given) : ') or
temp_e)
k = 0
print(f'Initial values of activations : {w}')
while(1):
```

```
w = calculate(w,e)
k = k+1
print(f'Iteration {k} : {w}')
c = check_active(w)
if(c):
    print('Winning neuron is : '+str(c))
    break
```

## Output:



```
sumon@Lenovo:~/Desktop/Python-Codes/annie_mam_Practical/Practical$ python3 maxnet.py
Enter the initial values of activations one by one separated by space :0.3 0.5 0.7 0.9
Input the epsilon value (Skip it if no epsilon is given) : 0.2
Initial values of activations : [0.3, 0.5, 0.7, 0.9]
Iteration 1 : [0, 0.12, 0.3599999999999993, 0.6]
Iteration 2 : [0, 0, 0.2159999999999994, 0.504]
Iteration 3 : [0, 0, 0.1151999999999994, 0.4608]
Iteration 4 : [0, 0, 0.02303999999999936, 0.43776]
Iteration 5 : [0, 0, 0, 0.433152]
Winning neuron is : 4
sumon@Lenovo:~/Desktop/Python-Codes/annie_mam_Practical/Practical$
```

# Practical-4

## Hamming Network

1] Solve the Hamming network to cluster four vectors.

Given the exemplar

vectors  $e(1) = [1 -1 -1 -1]$  and  $e(2) = [-1 -1 -1 1]$ , for the bipolar input vectors

$$X_1 = [-1 -1 1 -1]$$

$$X_2 = [-1 -1 1 1]$$

$$X_3 = [-1 -1 -1 1]$$

$$X_4 = [1 1 -1 -1]$$

Solution:

$$1) e(1) = [1 \ -1 \ -1 \ -1], \ e(2) = [-1 \ -1 \ -1 \ 1]$$

$$x_1 = [-1 \ -1 \ 1 \ -1], \ x_2 = [-1 \ -1 \ 1 \ 1], \ x_3 = [-1 \ -1 \ -1 \ 1],$$

$$x_4 = [1 \ 1 \ -1 \ -1]$$

\* my  $\rightarrow$  exemplar vectors

$$\omega = \begin{bmatrix} 1 & -1 \\ -1 & -1 \\ -1 & -1 \\ -1 & 1 \end{bmatrix}$$

initial weight is:  $\frac{1}{2} \times$  exemplar vector

$$= \begin{bmatrix} \frac{1}{2} & -\frac{1}{2} \\ -\frac{1}{2} & -\frac{1}{2} \\ -\frac{1}{2} & -\frac{1}{2} \\ -\frac{1}{2} & \frac{1}{2} \end{bmatrix} = \begin{bmatrix} \omega_1 & \omega_2 \\ 0.5 & -0.5 \\ -0.5 & -0.5 \\ -0.5 & -0.5 \\ -0.5 & 0.5 \end{bmatrix}$$

$$\therefore \text{bias} :- b = \frac{n}{2} = \frac{4}{2} = 2$$

1st iteration :-

$$x_1 = [-1 \ -1 \ 1 \ -1] \text{ with } \omega_1$$

$$y(1) = b + \sum n_i \omega_i$$

$$= 2 + [-1 \ -1 \ 1 \ -1] \begin{bmatrix} 0.5 \\ -0.5 \\ -0.5 \\ -0.5 \end{bmatrix}$$

$$= 2 + [-0.5 + 0.5 - 0.5 + 0.5]$$

$$= 2 + [0]$$

$$y(1) = 2$$

$$y(2) = 2 + [-1 \ -1 \ 1 \ -1] \begin{bmatrix} -0.5 \\ -0.5 \\ -0.5 \\ 0.5 \end{bmatrix}$$

$$y(2) = 2 + [0.5 + 0.5 - 0.5 - 0.5]$$

$$= 2 + [0]$$

$$y(2) = 2$$

As  $y(1) = y(2)$  so any one of them has best match ~~be~~ exemplar for  $x$ .

2nd iteration:-

$$x_2 = [-1 \ -1 \ 1 \ 1]$$

$$y(1) = 2 + [-1 \ -1 \ 1 \ 1] \begin{bmatrix} 0.5 \\ -0.5 \\ -0.5 \\ -0.5 \end{bmatrix} = 2 + [0.5 + 0.5 - 0.5 - 0.5] \\ = 2 + (-1)$$

$$\therefore y(1) = 1$$

$$y(2) = 2 + [-1 \ -1 \ 1 \ 1] \begin{bmatrix} -0.5 \\ -0.5 \\ -0.5 \\ 0.5 \end{bmatrix} = 2 + [0.5 + 0.5 - 0.5 + 0.5] \\ = 2 + [1] \\ = 3$$

$$\therefore y(2) = 3$$

As,  $y(2) > y(1)$

∴  $y(2)$  has best match exemplar for  $x_2$

3rd iteration:-

$$x_3 = [-1 \ -1 \ -1]$$

$$y(1) = 2 + [-1 \ -1 \ -1 \ 1] \begin{bmatrix} 0.5 \\ -0.5 \\ -0.5 \\ -0.5 \end{bmatrix} = 2 + [-0.5 + 0.5 + 0.5 - 0.5] \\ = 2 + [0] = 2$$

$$y(2) = 2 + [-1 \ -1 \ -1 \ 1] \begin{bmatrix} -0.5 \\ -0.5 \\ -0.5 \\ 0.5 \end{bmatrix} = 2 + [0.5 + 0.5 + 0.5 + 0.5] \\ = 2 + [2] = 4$$

As,  $y(2) > y(1)$  so  $y(2)$  has best match exemplar for  $x_3$

4th iteration:-

$$x_4 = [1 \ 1 \ -1 \ -1]$$

$$y(1) = 2 + [1 \ 1 \ -1 \ -1] \begin{bmatrix} 0.5 \\ -0.5 \\ -0.5 \\ -0.5 \end{bmatrix}$$

$$= 2 + [0.5 - 0.5 + 0.5 - 0.5]$$

$$= 2 + 1 = 3$$

$$y(2) = 2 + [1 \ 1 \ -1 \ -1] \begin{bmatrix} -0.5 \\ -0.5 \\ -0.5 \\ 0.5 \end{bmatrix}$$

$$= 2 + [-0.5 - 0.5 + 0.5 - 0.5] = 2 + [-1] = 1$$

As,  $y(1) > y(2)$  so  $y(1)$  has best match  
exemplar for  $x_4$

Code:

```
import numpy as np
input_vector_number = int(input('Enter the number of input
vectors : '))
exemplar_vector_number = int(input('Enter the number of
exemplar vectors :
'))
input_vector = []
exemplar_vector = []
exemplar_vector_update = []
for i in range(input_vector_number):
    input_vector.append(np.array(input(f'Enter the elements of
Input[{i}] :
').strip().split(),int))
for i in range(exemplar_vector_number):
    exemplar_vector.append(np.array(input(f'Enter the
elements of the
Exemplar[{i}] :
').strip().split(),int))
B = input_vector_number/2
b = np.array(B)
for i in range(0,exemplar_vector_number):
    exemplar_vector_update.append(exemplar_vector[i]/2)
print('Initial weights are : ')
for i in range(0, exemplar_vector_number):
    print(f'W[{i}]', end = ' ')
    print('\n')
for j in range(0, len(exemplar_vector[0])):
    for i in range(exemplar_vector_number):
        print(f' {exemplar_vector_update[i][j]}', end = ' ')
    print('\n')
for i in range(input_vector_number):
    y = []
    print(f'For input vector {i}: ')
```

```
for j in range(exemplar_vector_number):
    k = np.matmul(input_vector[i],exemplar_vector[j].T)
    y.append(np.add(k,b))
    print(f'Value of Y[{j}] = {y[j]}')
min_in = y.index(max(y))
print(f'Y[{min_in}] has the best exemplar vector for Input
vector {i}' )
```

## Output:

```
sumon@Lenovo:~/Desktop/Python-Codes/annie_mam_Practical/Practical$ python3 Hamming_code.py
Enter the number of input vectors : 4
Enter the number of exemplar vectors : 2
Enter the elements of Input[0] : -1 -1 1 -1
Enter the elements of Input[1] : -1 -1 1 1
Enter the elements of Input[2] : -1 -1 -1 1
Enter the elements of Input[3] : 1 1 -1 -1
Enter the elements of Exemplar[0] : 1 -1 -1 -1
Enter the elements of Exemplar[1] : -1 -1 -1 1
Initial weights are :
W[0]      W[1]
0.5      -0.5
-0.5      -0.5
-0.5      -0.5
-0.5      0.5

For Input vector 0:
Value of Y[0]=2.0
Value of Y[1]=2.0
Y[0] has the best exemplar for Input vector 0
For Input vector 1:
Value of Y[0]=1.0
Value of Y[1]=3.0
Y[1] has the best exemplar for Input vector 1
For Input vector 2:
Value of Y[0]=2.0
Value of Y[1]=4.0
Y[1] has the best exemplar for Input vector 2
For Input vector 3:
Value of Y[0]=3.0
Value of Y[1]=1.0
Y[0] has the best exemplar for Input vector 3
sumon@Lenovo:~/Desktop/Python-Codes/annie_mam_Practical/Practical$
```

2] Hamming net that has the following two exemplar vectors:

$$e(1) = [1 -1 -1 -1], e(2) = [-1 -1 -1 1].$$

following four vectors

$$x(1) = [1 1 -1 -1]$$

$$x(2) = [1 -1 -1 -1]$$

$$x(3) = [-1 -1 -1 1]$$

$$x(4) = [-1 -1 1 1]$$

Solution:

$$\Rightarrow e(1) = [1 \ -1 \ 1 \ -1], e(2) = [-1 \ -1 \ -1 \ 1]$$

$$x_1 = [1 \ 1 \ -1 \ -1], x_2 = [1 \ -1 \ -1 \ -1], x_3 = [-1 \ -1 \ -1 \ 1],$$

$$x_4 = [-1 \ -1 \ 1 \ 1]$$

$$\therefore w_i = \frac{1}{2} [e_1, e_2] = \begin{bmatrix} 0.5 & -0.5 \\ -0.5 & 0.5 \\ 0.5 & 0.5 \\ -0.5 & 0.5 \end{bmatrix}$$

$$\therefore y(i) = b + x_i w_i$$

$$\text{Now, bias is } b + \frac{w_0}{2} = \frac{4}{2} = 2$$

1st iteration:-

$$y(1) = 2 + [1 \ 1 \ -1 \ -1] \begin{bmatrix} 0.5 \\ -0.5 \\ -0.5 \\ -0.5 \end{bmatrix} = 2 + [0.5 - 0.5 + 0.5 - 0.5] \\ = 2 + 1$$

$$y(2) = 2 + [1 \ 1 \ -1 \ -1] \begin{bmatrix} -0.5 \\ -0.5 \\ -0.5 \\ 0.5 \end{bmatrix} = 2 + [-0.5 - 0.5 + 0.5 - 0.5] \\ = 2 + (-1)$$

As,  $y(1) > y(2)$  so  $y(1)$  is best match  
example for  $x_1$ .

2nd iteration:-

$$y(1) = 2 + [1 \ -1 \ -1 \ -1] \begin{bmatrix} 0.5 \\ -0.5 \\ -0.5 \\ -0.5 \end{bmatrix} = 2 + [0.5 + 0.5 + 0.5 + 0.5] = 2 + 2 = 4$$

$$y(2) = 2 + [-1 \ -1 \ -1 \ -1] \begin{bmatrix} -0.5 \\ -0.5 \\ -0.5 \\ 0.5 \end{bmatrix} = 2 + [-0.5 + 0.5 + 0.5 - 0.5] = 2$$

As,  $y(1) > y(2)$  so  $y(1)$  is best match exemplar for  $x_2$

3rd iteration:-

$$y(1) = 2 + [-1 \ -1 \ -1 \ 1] \begin{bmatrix} 0.5 \\ -0.5 \\ -0.5 \\ -0.5 \end{bmatrix} = 2 + [-0.5 + 0.5 + 0.5 - 0.5] = 2$$

$$y(2) = 2 + [-1 \ -1 \ -1 \ 1] \begin{bmatrix} -0.5 \\ 0.5 \\ -0.5 \\ 0.5 \end{bmatrix} = 4$$

As,  $y(2) > y(1)$  so  $y(2)$  is best match exemplar for  $x_3$

4th iteration:-

$$y(1) = 2 + [-1 \ -1 \ 1 \ 1] \begin{bmatrix} 0.5 \\ -0.5 \\ -0.5 \\ -0.5 \end{bmatrix} = 2 + [-0.5 + 0.5 - 0.5 - 0.5] = 2 + (-1) = 1$$

$$y(2) = 2 + [-1 \ -1 \ 1 \ 1] \begin{bmatrix} -0.5 \\ -0.5 \\ -0.5 \\ 0.5 \end{bmatrix} = 2 + [0.5 + 0.5 - 0.5 + 0.5] = 3$$

As,  $y(2) > y(1)$  so  $y(2)$  is best match exemplar for  $x_4$

Code:

```
import numpy as np
input_vector_number = int(input('Enter the number of input
vectors : '))
exemplar_vector_number = int(input('Enter the number of
exemplar vectors :
'))
input_vector = []
exemplar_vector = []
exemplar_vector_update = []
for i in range(input_vector_number):
    input_vector.append(np.array(input(f'Enter the elements of
Input[{i}] :
').strip().split(),int))
for i in range(exemplar_vector_number):
    exemplar_vector.append(np.array(input(f'Enter the
elements of the
Exemplar[{i}] :
').strip().split(),int))
B = input_vector_number/2
b = np.array(B)
for i in range(0,exemplar_vector_number):
    exemplar_vector_update.append(exemplar_vector[i]/2)
print('Initial weights are : ')
for i in range(0, exemplar_vector_number):
    print(f'W[{i}]', end = ' ')
    print('\n')
for j in range(0, len(exemplar_vector[0])):
    for i in range(exemplar_vector_number):
        print(f' {exemplar_vector_update[i][j]}', end = ' ')
    print('\n')
for i in range(input_vector_number):
    y = []
    print(f'For input vector {i}: ')
    for j in range(exemplar_vector_number):
```

```

k = np.matmul(input_vector[i],exemplar_vector[j].T)
y.append(np.add(k,b))
print(f'Value of Y[{j}] = {y[j]}')
min_in = y.index(max(y))
print(f'Y[{min_in}] has the best exemplar vector for Input vector {i}')

```

## Output:

```

sumon@Lenovo:~/Desktop/Python-Codes/annie_mam_Practical/Practical$ python3 Hamming_code.py
Enter the number of input vectors : 4
Enter the number of exemplar vectors : 2
Enter the elements of Input[0] : 1 1 -1 -1
Enter the elements of Input[1] : 1 -1 -1 -1
Enter the elements of Input[2] : -1 -1 -1 1
Enter the elements of Input[3] : -1 -1 1 1
Enter the elements of Exemplar[0] : 1 -1 -1 -1
Enter the elements of Exemplar[1] : -1 -1 -1 1
Initial weights are :
W[0]          W[1]
0.5           -0.5
-0.5          -0.5
-0.5          -0.5
-0.5          0.5

For Input vector 0:
Value of Y[0]=3.0
Value of Y[1]=1.0
Y[0] has the best exemplar for Input vector 0
For Input vector 1:
Value of Y[0]=4.0
Value of Y[1]=2.0
Y[0] has the best exemplar for Input vector 1
For Input vector 2:
Value of Y[0]=2.0
Value of Y[1]=4.0
Y[1] has the best exemplar for Input vector 2
For Input vector 3:
Value of Y[0]=1.0
Value of Y[1]=3.0
Y[1] has the best exemplar for Input vector 3
sumon@Lenovo:~/Desktop/Python-Codes/annie_mam_Practical/Practical$ 

```

# Practical-5

## Kohonen Self-organizing map

1] Consider a Kohonen self-organizing net with two cluster units and five input units. The weight vectors for the cluster units are given as  
 $W_1 = [1.0 \ 0.9 \ 0.7 \ 0.5 \ 0.3]$   $W_2 = [0.3 \ 0.5 \ 0.7 \ 0.9 \ 1.0]$   
Using the Euclidean distance find the winning cluster unit for input pattern  
[0.0 0.5 1.0 0.5 0.0]. the learning rate is 0.25

Solution:

Kohonen

- $\omega = \begin{bmatrix} \omega_1 & \omega_2 \\ 1.0 & 0.3 \\ 0.9 & 0.5 \\ 0.7 & 0.7 \\ 0.5 & 0.9 \\ 0.3 & 1 \end{bmatrix}$
- $x_r = [0.0 \ 0.5 \ 1.0 \ 0.5 \ 0.0]$
- $\alpha = 0.25$

$D(i) = \sum (\omega_{ij} - n_i)^2 \rightarrow$  square of euclidean distance

$i = 1 \text{ to } 5$

$$\begin{aligned}
 \underset{\omega_1}{\text{for}} \leftarrow D(1) &= (1.0 - 1)^2 + (0.9 - 0.5)^2 + (0.7 - 1)^2 \\
 &\quad + (0.5 - 0.5)^2 + (0.3 - 0.0)^2 \\
 &= 1 + 0.16 + 0.09 + 0.09 \\
 &= 1.34
 \end{aligned}$$

$$\begin{aligned}
 \underset{\omega_2}{\text{for}} \leftarrow D(2) &= (0.3 - 0.0)^2 + (0.5 - 0.5)^2 + (0.7 - 1)^2 \\
 &\quad + (0.9 - 0.5)^2 + (1 - 0.0)^2 \\
 &= 0.09 + 0.09 + 0.16 + 1 = 1.34
 \end{aligned}$$

Since,  $D(1) = D(2)$

$\alpha = 0.25$

$$\omega_{ij}(\text{new}) = \omega_{ij}(\text{old}) + \alpha (n_i - \omega_{ij}(\text{old}))$$

$$\omega_{11}(\text{new}) = 1 + 0.25(0 - 1)$$

$\omega_{11} = 1$   
 $n = 0$

~~$\omega_{11}$~~   $\omega_{11}(\text{new}) = 0.75$

$$\omega_{21}(\text{new}) = 0.9 + 0.25(0.5 - 0.9)$$

$$= 0.8$$

$$\omega_{31}(\text{new}) = 0.7 + 0.25(1 - 0.7)$$
 ~~$\omega_{31}$~~   $= 0.775$

$$\omega_{41}(\text{new}) = 0.5 + 0.25(0.5 - 0.5)$$

$$= 0.5$$

$$\omega_{51}(\text{new}) = 0.3 + 0.25(0.0 - 0.3)$$

$$= 0.225$$

The updated weight is

~~$\omega_1 = \begin{bmatrix} 0.75 \\ 0.8 \\ 0.77 \\ 0.5 \\ 0.22 \end{bmatrix}$~~ 

$$\omega_2 = \begin{bmatrix} 0.75 & 0.3 \\ 0.8 & 0.5 \\ 0.77 & 0.7 \\ 0.5 & 0.9 \\ 0.22 & 1 \end{bmatrix}$$

Code:

```
def distance(W,X):
D = 0
for i in range(0, len(W)):
D = D + pow((W[i]-X[i]), 2)
D = round(D,2)
return D
def update_weight(W,R,X):
U_W = []
for i in range(0, len(W)):
U_W.append(W[i] + (R * (X[i]-W[i])))
return U_W
def print_weight(W, w_n):
for i in range(0, w_n):
print(f'W[{i}] = {w[i]}')
def print_input(X,v):
for i in range(0,v):
print(f'X[{i}] = {X[i]}')
W = []
w_n = int(input('Input the number of weight vectors : '))
for i in range(0, w_n):
w_i = list(map(float, input(f'Enter the weights of {i} weight
vector :
').strip().split())))
W.append(w_i)
X = []
v = int(input('Input the number of input vectors : '))
for i in range(0,v):
x_i = list(map(float, input(f'Enter inputs of {i} input vector :
').strip().split())))
X.append(x_i)
L_R = float(input(f'Enter the learning rate : '))
print('Initial Weights : ')
```

```
print_weight(W,w_n)
print('Initial Input vectors : ')
print_input(X,v)
k = w_n
if (k == len(X[0])):
p = []
for i in range(0, len(W[0])):
l = []
for j in range(0,k):
l.append(W[j][i])
p.append(l)
W = []
for i in range(0, len(p)):
W.append(p[i])
w_n = len(W)
D = [None]*w_n
for i in range(0,v):
for j in range(0,w_n):
D[i] = distance(W[i], X[j])
min_d = D.index(min(D))
W[min_d] = update_weight(W[mid_d],L_R, X[j])
if(k == len(X[0])):
q = []
for i in range(0, len(W[0])):
l = []
for j in range(0, len(W)):
l.append(W[j][i])
q.append(l)
W = []
for i in range(0, len(q)):
W.append(q[i])
w_n = len(W)
print('Final Weights : ')
print_weight(W,w_n)
```

## Output:

```
sumon@Lenovo:~/Desktop/Python-Codes/annie_mam_Practical/Practical$ python3 Kohonen_self.py
Input the number of weight vectors : 2
Enter the weights of 0 weight vector : 1.0 0.9 0.7 0.5 0.3
Enter the weights of 1 weight vector : 0.3 0.5 0.7 0.9 1.0
Input the number of input vectors : 1
Enter the inputs of 0 input vector : 0.0 0.5 1.0 0.5 0.0
Enter the learning rate : 0.25
Initial Weights :
W[0]=[1.0, 0.9, 0.7, 0.5, 0.3]
W[1]=[0.3, 0.5, 0.7, 0.9, 1.0]
Initial Input Vectors :
X[0]=[0.0, 0.5, 1.0, 0.5, 0.0]
Final Weights :
W[0]=[0.75, 0.8, 0.7749999999999999, 0.5, 0.2249999999999998]
W[1]=[0.3, 0.5, 0.7, 0.9, 1.0]
sumon@Lenovo:~/Desktop/Python-Codes/annie_mam_Practical/Practical$
```

2] Given a kohonen self-organizing map with weights  $w_1 = [0.7 \ 0.3 \ 0.1 \ 0.6 \ 1.0]$  and  $w_2 = [0.5 \ 0.2 \ 0.4 \ 0.9 \ 0.5]$ . Find the cluster unit close to input vector  $(0.35, 0.05)$ , the learning rate is 0.25.

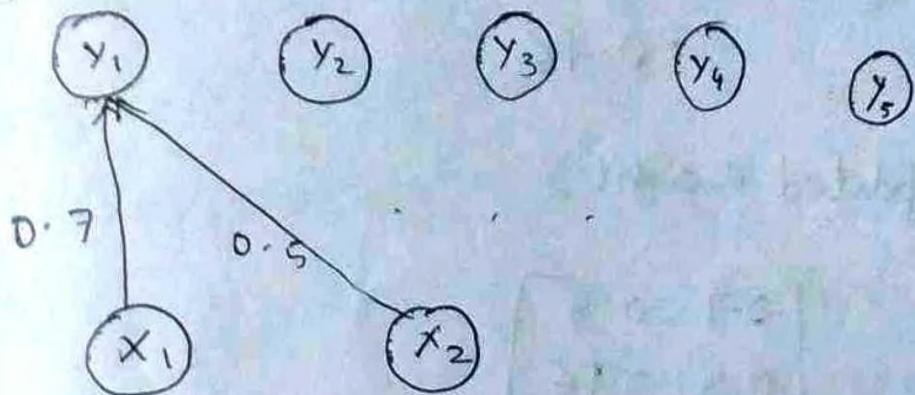
Solution:

$$2) \text{ Given } \omega = [\omega_1, \omega_2]$$

$$\begin{matrix} 0 \\ 1 \\ 2 \\ 3 \\ 4 \end{matrix} = \begin{bmatrix} 0.7 & 0.5 \\ 0.3 & 0.2 \\ 0.1 & 0.4 \\ 0.6 & 0.9 \\ 1.0 & 0.5 \end{bmatrix}$$

$$x = (0.35, 0.05)$$

$$\alpha = 0.25$$



$$D(1) = (0.7 - 0.35)^2 + (0.5 - 0.05)^2 \\ = 0.325$$

$$D(2) = (0.35 - 0.35)^2 + (0.2 - 0.05)^2 \\ = 0.025$$

$$D(3) = (0.1 - 0.35)^2 + (0.4 - 0.05)^2 \\ = 0.185$$

$$D(4) = (0.6 - 0.35)^2 + (0.9 - 0.05)^2 \\ = 0.785$$

$$D(5) = (1 - 0.35)^2 + (0.5 - 0.05)^2 = 0.625$$

Mm :-  $D(z) = 0.25$ , so update the  
weight of  $(w_{21}, w_{22})$

$$w_{21}(\text{new}) = 0.3 + 0.25(0.35 - 0.3) \\ \approx 0.31$$

$$w_{22}(\text{new}) = 0.2 + 0.25(0.05 - 0.2) \\ \approx 0.16$$

Updated weight :-

$$\begin{bmatrix} 0.7 & 0.5 \\ 0.3 & 0.16 \\ 0.1 & 0.4 \\ 0.6 & 0.9 \\ 1.0 & 0.5 \end{bmatrix}$$

Code:

```
def distance(W,X):
D = 0
for i in range(0, len(W)):
D = D + pow((W[i]-X[i]), 2)
D = round(D,2)
return D
def update_weight(W,R,X):
U_W = []
for i in range(0, len(W)):
U_W.append(W[i] + (R * (X[i]-W[i])))
return U_W
def print_weight(W, w_n):
for i in range(0, w_n):
print(f'W[{i}] = {w[i]}')
def print_input(X,v):
for i in range(0,v):
print(f'X[{i}] = {X[i]}')
W = []
w_n = int(input('Input the number of weight vectors : '))
for i in range(0, w_n):
w_i = list(map(float, input(f'Enter the weights of {i} weight
vector :
')).strip().split()))
W.append(w_i)
X = []
v = int(input('Input the number of input vectors : '))
for i in range(0,v):
x_i = list(map(float, input(f'Enter inputs of {i} input vector :
')).strip().split()))
X.append(x_i)
L_R = float(input(f'Enter the learning rate : '))
print('Initial Weights : ')
print_weight(W,w_n)
```

```

print('Initial Input vectors : ')
print_input(X,v)
k = w_n
if (k == len(X[0])):
p = []
for i in range(0, len(W[0])):
l = []
for j in range(0,k):
l.append(W[j][i])
p.append(l)
W = []
for i in range(0, len(p)):
W.append(p[i])
w_n = len(W)
D = [None]*w_n
for i in range(0,v):
for j in range(0,w_n):
D[i] = distance(W[i], X[j])
min_d = D.index(min(D))
W[min_d] = update_weight(W[mid_d],L_R, X[j])
if(k == len(X[0])):
q = []
for i in range(0, len(W[0])):
l = []
for j in range(0, len(W)):
l.append(W[j][i])
q.append(l)
W = []
for i in range(0, len(q)):
W.append(q[i])
w_n = len(W)
print('Final Weights : ')
print_weight(W,w_n)

```

## Output:

```
sumon@Lenovo:~/Desktop/Python-Codes/annie_mam_Practical/Practical$ python3 Kohonen_self.py
Input the number of weight vectors : 2
Enter the weights of 0 weight vector : 0.7 0.3 0.1 0.6 1.0
Enter the weights of 1 weight vector : 0.5 0.2 0.4 0.9 0.5
Input the number of input vectors : 1
Enter the inputs of 0 input vector : 0.35 0.05
Enter the learning rate : 0.25
Initial Weights :
W[0]=[0.7, 0.3, 0.1, 0.6, 1.0]
W[1]=[0.5, 0.2, 0.4, 0.9, 0.5]
Initial Input Vectors :
X[0]=[0.35, 0.05]
Final Weights :
W[0]=[0.7, 0.3125, 0.1, 0.6, 1.0]
W[1]=[0.5, 0.1625, 0.4, 0.9, 0.5]
sumon@Lenovo:~/Desktop/Python-Codes/annie_mam_Practical/Practical$
```

3] Construct a Kohonen self-organizing feature map to cluster the four given vectors, [0 0 1 1], [1 0 0 0], [0 1 1 0] and [0 0 0 1]. The number of clusters to be formed is two. Initial learning rate is 0.5.

Solution:

$$\omega = \begin{bmatrix} 0.2 & 0.9 \\ 0.4 & 0.7 \\ 0.6 & 0.5 \\ 0.8 & 0.3 \end{bmatrix}$$

~~$$x_1 = [0 \ 0 \ 1 \ 1]$$

$$x_2 = [1 \ 0 \ 0 \ 0]$$

$$x_3 = [0 \ 1 \ 1 \ 0]$$

$$x_4 = [0 \ 0 \ 0 \ 1]$$~~

$$\alpha = 0.5$$

Condition (1) :-

~~$D(1) \neq 1(0.2/8)$~~

For 1st input  $D(i) = \sum (w_{ij} - n_j) \quad i \in \{1 \text{ to } 4\}$

$$D(1) = (0.2-0)^2 + (0.4-0)^2 + (0.6-1)^2 + (0.8-1)^2$$

$$= 0.4$$

~~$D(2) = 0.9 +$~~

$$(0.9-0)^2 + (0.7-0)^2 + (0.5-1)^2 + (0.3-1)^2$$

$$= 2.04$$

As  $D(2) > D(1)$  so update weights  $w_i$  col

$$\text{new } \omega_0 = \begin{bmatrix} 0.1 & 0.9 \\ 0.2 & 0.7 \\ 0.8 & 0.5 \\ 0.9 & 0.3 \end{bmatrix}$$

for 2nd  
input:  $\underline{\underline{x}_2} [1 \ 0 \ 0 \ 0]$

$$D(1) = (0.1 - 1)^2 + (0.2)^2 + (0.8)^2 + (0.9)^2$$

$$= 2.5$$

$$D(2) = (0.9 - 1)^2 + (0.7)^2 + (0.5)^2 + (0.3)^2$$

$$= 0.84$$

$D(1) > D(2)$  so update  $\omega_2$ , col

$$\omega_{21}(\text{new}) = 0.9 + 0.5 \times (1 - 0.9)$$

$$= \cancel{0.85} \quad \cancel{0.95} \quad 0.95$$

$$\omega_{22}(\text{new}) = 0.7 + 0.5 \times (-0.7)$$

$$= \cancel{1.05} \quad 0.35$$

$$\omega_{23}(\text{new}) = 0.8 + 0.5 \times (-0.5)$$

$$= \cancel{1.05} \quad 0.55$$

$$w_{03} = 0.3 + 0.5(-0.3)$$

$$= \cancel{0.45} \quad 0.15$$

Updated weight:-

$$w_1 = \begin{bmatrix} 0.1 & 0.8 \\ 0.2 & 1 \\ 0.8 & 0.4 \end{bmatrix} \quad w_2 = \begin{bmatrix} 0.1 & 0.95 \\ 0.2 & 0.35 \\ 0.8 & 0.55 \\ 0.9 & 0.15 \end{bmatrix}$$

for 3rd.  $x_3 = [0 \ 1 \ 1 \ 0]$ ,  
input:

$$D(1) = (0.1)^2 + (1-0.2)^2 + (0.8)^2 + (0.9)^2$$

$$+ (1-0.8)^2 + (0.9)^2$$

$$= 1.5$$

$$D(2) = (0.95)^2 + (1-0.35)^2 + (1-0.55)^2$$

$$+ (0.15)^2$$

$$= 1.55$$

$$D(1) < D(2) \quad \text{update col 4}(w_1)$$

$$w_{11}(\text{new}) = 0.25 + 0.5(0 - 0.25) \\ = 0.125$$

Updated weights :-

$$w_2 = \begin{bmatrix} 0.375 & 0 \\ 0.5 & 1 \\ 0.75 & 0 \\ 0.125 & 1 \end{bmatrix}$$

for 4th input :-  $x_4 = [0 \ 0 \ 0 \ 1]$

$$D(1) = 1.71$$

$$D(2) = 1$$

$$D(1) > D(2)$$

update  $w_2$  col

$$w_{21}(\text{new}) = 0 + 0.5(0) = 0$$

$$w_{22}(\text{new}) = 1 + 0.5(0 - 1) = 0.5$$

$$w_{23}(\text{new}) = 0 \times 0.5(0 - 0) = 0$$

$$w_{24}(\text{new}) = 1 + 0.5(1 - 1) = 1$$

updated weights :-

$$w = \begin{bmatrix} 0.375 & 0 \\ 0.5 & 0.5 \\ 0.75 & 0 \\ 0.125 & 1 \end{bmatrix}$$

Code:

```
def distance(W,X):
D = 0
for i in range(0, len(W)):
D = D + pow((W[i]-X[i]), 2)
D = round(D,2)
return D
def update_weight(W,R,X):
U_W = []
for i in range(0, len(W)):
U_W.append(W[i] + (R * (X[i]-W[i])))
return U_W
def print_weight(W, w_n):
for i in range(0, w_n):
print(f'W[{i}] = {w[i]}')
def print_input(X,v):
for i in range(0,v):
print(f'X[{i}] = {X[i]}')
W = []
w_n = int(input('Input the number of weight vectors : '))
for i in range(0, w_n):
w_i = list(map(float, input(f'Enter the weights of {i} weight
vector :
').strip().split())))
W.append(w_i)
X = []
v = int(input('Input the number of input vectors : '))
for i in range(0,v):
x_i = list(map(float, input(f'Enter inputs of {i} input vector :
').strip().split())))
X.append(x_i)
L_R = float(input(f'Enter the learning rate : '))
print('Initial Weights : ')
print_weight(W,w_n)
print('Initial Input vectors : ')
```

```
print_input(X,v)
k = w_n
if (k == len(X[0])):
p = []
for i in range(0, len(W[0])):
l = []
for j in range(0,k):
l.append(W[j][i])
p.append(l)
W = []
for i in range(0, len(p)):
W.append(p[i])
w_n = len(W)
D = [None]*w_n
for i in range(0,v):
for j in range(0,w_n):
D[i] = distance(W[i], X[j])
min_d = D.index(min(D))
W[min_d] = update_weight(W[mid_d],L_R, X[j])
if(k == len(X[0])):
q = []
for i in range(0, len(W[0])):
l = []
for j in range(0, len(W)):
l.append(W[j][i])
q.append(l)
W = []
for i in range(0, len(q)):
W.append(q[i])
w_n = len(W)
print('Final Weights : ')
print_weight(W,w_n)
```

## Output:

```
sumon@Lenovo:~/Desktop/Python-Codes/annie_mam_Practical/Practical$ python3 Kohenen_self.py
Input the number of weight vectors : 2
Enter the weights of 0 weight vector : 0.2 0.4 0.6 0.8
Enter the weights of 1 weight vector : 0.9 0.7 0.5 0.3
Input the number of input vectors : 4
Enter the inputs of 0 input vector : 0 0 1 1
Enter the inputs of 1 input vector : 1 0 0 0
Enter the inputs of 2 input vector : 0 1 1 0
Enter the inputs of 3 input vector : 0 0 0 1
Enter the learning rate : 0.5
Initial Weights :
W[0]=[0.2, 0.4, 0.6, 0.8]
W[1]=[0.9, 0.7, 0.5, 0.3]
Initial Input Vectors :
X[0]=[0.0, 0.0, 1.0, 1.0]
X[1]=[1.0, 0.0, 0.0, 0.0]
X[2]=[0.0, 1.0, 1.0, 0.0]
X[3]=[0.0, 0.0, 0.0, 1.0]
Final Weights :
W[0]=[0.025, 0.3000000000000004, 0.45, 0.7250000000000001]
W[1]=[0.95, 0.35, 0.25, 0.15]
sumon@Lenovo:~/Desktop/Python-Codes/annie_mam_Practical/Practical$
```

4] Construct a Kohonen self-organizing feature map to cluster the four given vectors, [0 0 1 1], [1 0 0 0], [0 1 1 0] and [0 0 0 1]. The number of clusters to be formed is two. Initial learning rate is 0.5.

Solution:

q) Initial weights:-

$$\omega = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}$$

$$\alpha = 0.5$$

For 1st. -  
input:  $x_1 = [0 \ 0 \ 1 \ 1]$

$$D(1) = 1 + 0 + 0 + 1 = 2$$

$$D(2) = 0 + 1 + 1 + 0 = 2$$

$$\therefore D(1) = D(2)$$

so, update ~~the~~  $\omega_1$ , col:

$$\begin{aligned}\omega_{11}(\text{new}) &= 1 + 0.5(0-1) \\ &= 0.5\end{aligned}$$

$$\begin{aligned}\omega_{12}(\text{new}) &= 0 + 0.5(0-0) \\ &= 0\end{aligned}$$

$$\begin{aligned}\omega_{13}(\text{new}) &= 1 + 0.5(1-1) \\ &= 1\end{aligned}$$

$$\omega_{11} = 0 + 0.5(1-0) \\ = 0.5$$

Updated weight

$$\omega_b = \begin{bmatrix} 0.5 & 0 \\ 0 & 1 \\ 1 & 0 \\ 0.5 & 1 \end{bmatrix}$$

for 2nd:  $x_2 = [1 \ 0 \ 0 \ 0]$

np

$$\therefore D(1) = 0.5^2 + 0^2 + 1^2 + 0.5^2 \\ = 1.5$$

$$D(2) = 1^2 + 1^2 + 1^2 + 1^2 = 4$$

$D(1) < D(2)$  update  $\omega_1$ , col

$$\omega_{11}(\text{new}) = 0.5 + 0.5(1-0.5) \\ = 0.75$$

$$\omega_{12}(\text{new}) = 0 + 0.5(0-0) = 0$$

$$\omega_{13}(\text{new}) = 1 + 0.5(0-1) = 0.5$$

$$\omega_{14}(\text{new}) = 0.5 + 0.5(-0.5) = 0.25$$

Updated weight:-

$$a_1 = \begin{bmatrix} 0.75 & 0 \\ 0 & 1 \\ 0.5 & 0 \\ 0.25 & 1 \end{bmatrix}$$

To find  
input :-  $x_3 = [0 \ 1 \ 1 \ 0]$

$$D(1) = 1.875$$

$$D(2) = \cancel{0.2}$$

As  $D(1) < D(2)$

Update col  $w_i$

$$w_{11}(\text{new}) = 0.75 + 0.5(0 - 0.75) \\ = 0.375$$

$$w_{12}(\text{new}) = 0 + 0.5(1 - 0) \\ = 0.5$$

$$w_{13}(\text{new}) = 0.5 + 0.5(1 - 0.5) \\ = 0.75$$

$$w_{11}(\text{new}) = 0.25 + 0.5(0 - 0.25) \\ = 0.125$$

Updated weights :-

$$w_2 = \begin{bmatrix} 0.375 & 0 \\ 0.5 & 1 \\ 0.75 & 0 \\ 0.125 & 1 \end{bmatrix}$$

for 4th input :-  $x_4 = [0 \ 0 \ 0 \ 1]$

$$D(1) = 1.71$$

$$D(2) = 1$$

$$D(1) > D(2)$$

update  $w_2$  col

$$w_{21}(\text{new}) = 0 + 0.5(0) = 0$$

$$w_{22}(\text{new}) = 1 + 0.5(0 - 1) = 0.5$$

$$w_{23}(\text{new}) = 0 \times 0.5(0 - 0) = 0$$

$$w_{24}(\text{new}) = 1 + 0.5(1 - 1) = 1$$

updated weights :-

$$w = \begin{bmatrix} 0.375 & 0 \\ 0.5 & 0.5 \\ 0.75 & 0 \\ 0.125 & 1 \end{bmatrix}$$

Code:

```
def distance(W,X):
D = 0
for i in range(0, len(W)):
D = D + pow((W[i]-X[i]), 2)
D = round(D,2)
return D
def update_weight(W,R,X):
U_W = []
for i in range(0, len(W)):
U_W.append(W[i] + (R * (X[i]-W[i])))
return U_W
def print_weight(W, w_n):
for i in range(0, w_n):
print(f'W[{i}] = {w[i]}')
def print_input(X,v):
for i in range(0,v):
print(f'X[{i}] = {X[i]}')
W = []
w_n = int(input('Input the number of weight vectors : '))
for i in range(0, w_n):
w_i = list(map(float, input(f'Enter the weights of {i} weight
vector :
').strip().split())))
W.append(w_i)
X = []
v = int(input('Input the number of input vectors : '))
for i in range(0,v):
x_i = list(map(float, input(f'Enter inputs of {i} input vector :
').strip().split())))
X.append(x_i)
L_R = float(input(f'Enter the learning rate : '))
print('Initial Weights : ')
print_weight(W,w_n)
print('Initial Input vectors : ')
```

```
print_input(X,v)
k = w_n
if (k == len(X[0])):
p = []
for i in range(0, len(W[0])):
l = []
for j in range(0,k):
l.append(W[j][i])
p.append(l)
W = []
for i in range(0, len(p)):
W.append(p[i])
w_n = len(W)
D = [None]*w_n
for i in range(0,v):
for j in range(0,w_n):
D[i] = distance(W[i], X[j])
min_d = D.index(min(D))
W[min_d] = update_weight(W[mid_d],L_R, X[j])
if(k == len(X[0])):
q = []
for i in range(0, len(W[0])):
l = []
for j in range(0, len(W)):
l.append(W[j][i])
q.append(l)
W = []
for i in range(0, len(q)):
W.append(q[i])
w_n = len(W)
print('Final Weights : ')
print_weight(W,w_n)
```

## Output :

```
sumon@Lenovo:~/Desktop/Python-Codes/annie_mam_Practical/Practical$ python3 Kohonen_self.py
Input the number of weight vectors : 2
Enter the weights of 0 weight vector : 1 0 1 0
Enter the weights of 1 weight vector : 0 1 0 1
Input the number of input vectors : 4
Enter the inputs of 0 input vector : 0 0 1 1
Enter the inputs of 1 input vector : 1 0 0 0
Enter the inputs of 2 input vector : 0 1 1 0
Enter the inputs of 3 input vector : 0 0 0 1
Enter the learning rate : 0.5
Initial Weights :
W[0]=[1.0, 0.0, 1.0, 0.0]
W[1]=[0.0, 1.0, 0.0, 1.0]
Initial Input Vectors :
X[0]=[0.0, 0.0, 1.0, 1.0]
X[1]=[1.0, 0.0, 0.0, 0.0]
X[2]=[0.0, 1.0, 1.0, 0.0]
X[3]=[0.0, 0.0, 0.0, 1.0]
Final Weights :
W[0]=[0.375, 0.5, 0.75, 0.125]
W[1]=[0.0, 0.5, 0.0, 1.0]
sumon@Lenovo:~/Desktop/Python-Codes/annie_mam_Practical/Practical$ █
```

## Practical-6 BAM Network

1] Construct and test BAM network to associate letters E and F with simple bipolar input-output vectors. The target output for E is (-1, 1) and for F is (1, 1).  
The display matrix is 5 x 3.

```
### ###
#** ###
### #**
#** #**
### #**
'E' 'F'
```

Solution:

1)

Input pattern

E

F

Input(s)

$$\begin{bmatrix} 1 & 1 & 1 & 1 & -1 & -1 & 1 & 1 & 1 & 1 & -1 & -1 & 1 & 1 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 & -1 & 1 & -1 & -1 \end{bmatrix}$$

Target(t)

$$\begin{bmatrix} -1 \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

$$\omega = \sum S^T t$$

$$\omega_1 = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} -1 \\ 1 \end{bmatrix} =$$

$$\begin{bmatrix} -1 \\ -1 \\ -1 \\ -1 \\ -1 \\ -1 \\ -1 \\ -1 \\ -1 \\ -1 \\ -1 \\ -1 \\ -1 \\ -1 \end{bmatrix}$$

$$\rightarrow \omega_1$$

$$\omega_2 = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} 1 \\ 1 \end{bmatrix} =$$

$$\begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

$$\rightarrow \omega_2$$

$$\therefore \omega = \omega_1 + \omega_2$$

$$\begin{bmatrix} 0 & 2 \\ 0 & 2 \\ 0 & 2 \\ 0 & 2 \\ 2 & 0 \\ 2 & 0 \\ 0 & 2 \\ -2 & 0 \\ -2 & 0 \\ 0 & 2 \\ 0 & -2 \\ 0 & -2 \\ 0 & 2 \\ -2 & 0 \\ -2 & 0 \end{bmatrix}$$

$$\text{Now, } X_m \propto \omega$$

$$\begin{bmatrix} 1 & 1 & 1 & 1 & -1 & -1 & 1 & 1 & 1 & 1 & -1 & -1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 0 & 2 \\ 0 & 2 \\ 0 & 2 \\ 0 & 2 \\ 2 & 0 \\ 2 & 0 \\ 0 & 2 \\ -2 & 0 \\ -2 & 0 \\ 0 & 2 \\ 0 & -2 \\ 0 & -2 \\ 0 & 2 \\ -2 & 0 \\ -2 & 0 \end{bmatrix}$$

$$= \begin{bmatrix} -12 & 18 \\ -1 & 1 \end{bmatrix}$$

Now, for  $\omega$

$$[1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1]$$

$$\begin{bmatrix} 0 & 2 \\ 0 & 2 \\ 0 & 2 \\ 0 & 2 \\ 2 & 0 \\ 2 & 0 \\ 0 & 2 \\ -2 & 0 \\ -2 & 0 \\ 0 & 2 \\ 0 & -2 \\ 0 & -2 \\ 0 & 2 \\ -2 & 0 \\ -2 & 0 \end{bmatrix}$$

$$\cdot [12, 16]$$

$$\cdot (1, 1)$$

$$\omega^T, \begin{bmatrix} 0 & 0 & 0 & 0 & 2 & 2 & 0 & -2 & -2 & 0 & 0 & 0 & 0 & -2 & -2 \\ 2 & 2 & 2 & 2 & 0 & 0 & 2 & 0 & 0 & 2 & -2 & -2 & -2 & 0 & 0 \end{bmatrix}$$

Now, for pattern E (-1 1)

$$x * \omega^T * [1 \ 1 \ 1 \ 1 \ -1 \ -1 \ 1 \ 1 \ 1 \ -1 \ 1 \ 1]$$

Now, for pattern F (1, 1)

$$x * \omega^T * [1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ -1 \ -1 \ 1 \ -1 \ -1 \ 1 \ 1 \ 1]$$

Code:

```
from turtle import shape
import numpy as np
input_number=int(input('Enter the number of inputs : '))
target_number = input_number
input_vector = []
target_vector = []
for i in range(input_number):
    input_vector.append(list(map(int,input(f'Enter the elements
of Input[{i}] :
')).strip().split())))
for i in range(target_number):
    target_vector.append(list(map(int, input(f'Enter the
elements of Target[{i}] :
')).strip().split())))
r = len(input_vector[0])
c = len(target_vector[0])
W = np.zeros([])
for i in range(input_number):
    input_vector_T = np.array([input_vector[i]]).T
    target_vector_A = np.array([target_vector[i]])
    W = W + np.matmul(input_vector_T, target_vector_A)
print('Final weight for X will be : ')
print(W)
print('Final Weight for Y will be : ')
print(W.T)
```

## Output:

```
sumon@Lenovo:~/Desktop/Python-Codes/annie_mam_Practical/Practical$ python3 BAM.py
Enter the number of inputs : 2
Enter the elements of Input[0] : 1 1 1 1 -1 -1 1 1 1 1 -1 -1 1 1 1
Enter the elements of Input[1] : 1 1 1 1 1 1 1 -1 -1 1 -1 -1 1 -1 -1
Enter the elements of Target[0] : -1 1
Enter the elements of Target[1] : 1 1
Final weight for X will be :
[[ 0.  2.]
 [ 0.  2.]
 [ 0.  2.]
 [ 0.  2.]
 [ 2.  0.]
 [ 2.  0.]
 [ 0.  2.]
 [-2.  0.]
 [-2.  0.]
 [ 0.  2.]
 [ 0. -2.]
 [ 0. -2.]
 [ 0.  2.]
 [-2.  0.]
 [-2.  0.]]
Final weight for Y will be :
[[ 0.  0.  0.  0.  2.  2.  0. -2. -2.  0.  0.  0.  0. -2. -2.]
 [ 2.  2.  2.  2.  0.  0.  2.  0.  0.  2. -2. -2.  2.  0.  0.]]
```

2] Construct and test BAM network to associate letters I and C with simple bipolar input-output vectors. The target output for I is (1, -1) and for C is (-1, 1).

###           ###  
\*##           ##\*  
###           ###

'I' 'C'

Solution:

$$2) I = [1 \ 1 \ 1 \ -1 \ 1 \ 1 \ 1 \ 1]$$

$$C = [1 \ 1 \ 1 \ 1 \ 1 \ -1 \ 1 \ 1]$$

$$\text{target } I = [1, -1] \quad C = [-1, 1]$$

$$\text{weight: } \sum S^T \cdot t = \sum I^T t$$

$$\omega_1 = \begin{bmatrix} 1 \\ 1 \\ -1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} [1, -1] = \begin{bmatrix} 1 & -1 \\ 1 & -1 \\ -1 & 1 \\ 1 & -1 \\ 1 & -1 \\ 1 & -1 \\ 1 & -1 \\ 1 & -1 \end{bmatrix}$$

$$\omega_2 = \begin{bmatrix} 1 \\ 1 \\ 1 \\ -1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} [-1, 1] = \begin{bmatrix} -1 & 1 \\ -1 & 1 \\ -1 & 1 \\ 1 & -1 \\ -1 & 1 \\ 1 & -1 \\ 1 & -1 \\ 1 & -1 \end{bmatrix}$$

Total weight :-  $\omega = \omega_1 + \omega_2 = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ -2 & 2 \\ 0 & 0 \\ 2 & -2 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}$

$$x_1 \times \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

$$x_1 \times w = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ -2 & 2 \\ 0 & 0 \\ 2 & -2 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} 2 & 2 & -2 & -2 \end{bmatrix} \rightarrow [8 & -8]$$

$$x_2 \times w = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ -2 & 2 \\ 0 & 0 \\ 2 & -2 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} -2 & -2 & 2 & 2 \end{bmatrix} \rightarrow [-4 & 4]$$

NOW,  $w^T = \begin{bmatrix} 0 & 0 & 0 & -2 & 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2 & 0 & -2 & 0 & 0 & 0 \end{bmatrix}$

For pattern I :-  $[1, -1]$

$$\text{so } I \times w^T = [1 \quad -1] \begin{bmatrix} 0 & 0 & 0 & -2 & 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2 & 0 & -2 & 0 & 0 & 0 \end{bmatrix}$$

$$= [0 \quad 0 \quad 0 \quad -4 \quad 0 \quad 4 \quad 0 \quad 0 \quad 0]$$

As per activation funcn :-  $[1 \quad 1 \quad 1 \quad -1 \quad 1 \quad 1 \quad 1]$

For pattern C :-  $[-1, 1]$

$$\text{so, } [-1 \quad 1] \begin{bmatrix} 0 & 0 & 0 & -2 & 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2 & 0 & -2 & 0 & 0 & 0 \end{bmatrix}$$

$$= [0 \quad 0 \quad 0 \quad 4 \quad 0 \quad -4 \quad 0 \quad 0 \quad 0]$$

As per activation funcn :-  $[1 \quad 1 \quad 1 \quad 1 \quad -1 \quad 1 \quad 1] = C$

Code:

```
from turtle import shape
import numpy as np
input_number=int(input('Enter the number of inputs : '))
target_number = input_number
input_vector = []
target_vector = []
for i in range(input_number):
    input_vector.append(list(map(int,input(f'Enter the elements
of Input[{i}] :
')).strip().split())))
for i in range(target_number):
    target_vector.append(list(map(int, input(f'Enter the
elements of Target[{i}] :
')).strip().split())))
r = len(input_vector[0])
c = len(target_vector[0])
W = np.zeros([])
for i in range(input_number):
    input_vector_T = np.array([input_vector[i]]).T
    target_vector_A = np.array([target_vector[i]])
    W = W + np.matmul(input_vector_T, target_vector_A)
print('Final weight for X will be : ')
print(W)
print('Final Weight for Y will be : ')
print(W.T)
```

## Output:

```
sumon@Lenovo:~/Desktop/Python-Codes/annie_mam_Practical/Practical$ python3 BAM.py
Enter the number of inputs : 2
Enter the elements of Input[0] : 1 1 1 -1 1 1 1 1 1
Enter the elements of Input[1] : 1 1 1 1 1 -1 1 1 1
Enter the elements of Target[0] : 1 -1
Enter the elements of Target[1] : -1 1
Final weight for X will be :
[[ 0.  0.]
 [ 0.  0.]
 [ 0.  0.]
 [-2.  2.]
 [ 0.  0.]
 [ 2. -2.]
 [ 0.  0.]
 [ 0.  0.]
 [ 0.  0.]]
Final weight for Y will be :
[[ 0.  0. -2.  0.  2.  0.  0.  0.]
 [ 0.  0.  0.  2.  0. -2.  0.  0.  0.]]
sumon@Lenovo:~/Desktop/Python-Codes/annie_mam_Practical/Practical$
```

# Practical-7

## Operations of Fuzzy set

1] Implement Union, Intersection, Complement and Difference on Fuzzy set.

$$L = \{0/100 + 0.1/200 + 0.3/300\}$$

$$M = \{0.5/100 + 0.57/200 + 0.6/300\}$$

Solution:

$$L = \left\{ \frac{0}{100} + \frac{0.1}{200} + \frac{0.3}{300} \right\}$$

$$\bar{M} = \left\{ \frac{0.5}{100} + \frac{0.57}{200} + \frac{0.6}{300} \right\}$$

union :-  $L \cup \bar{M} = \mu_L(x) \cup \mu_{\bar{M}}(x) = \max(\mu_L(x), \mu_{\bar{M}}(x))$

$$= \left\{ \frac{0.5}{100} + \frac{0.57}{200} + \frac{0.6}{300} \right\}$$

Intersection :-  $\mu_L(x) \cap \mu_{\bar{M}}(x) = \min(\mu_L(x), \mu_{\bar{M}}(x))$

$$= \left\{ \frac{0}{100} + \frac{0.1}{200} + \frac{0.3}{300} \right\}$$

Complement :-

$$\bar{L} = \left\{ \frac{1}{100} + \frac{0.9}{200} + \frac{0.7}{300} \right\}$$

$$\bar{M} = \left\{ \frac{0.5}{100} + \frac{0.43}{200} + \frac{0.4}{300} \right\}$$

Difference :-

$$L \setminus \bar{M} = L \cap \bar{M} = \underline{\mu_L(x)} \cap \underline{\mu_{\bar{M}}(x)}$$

$$= \left\{ \frac{0}{100} + \frac{0.1}{200} + \frac{0.3}{300} \right\}$$

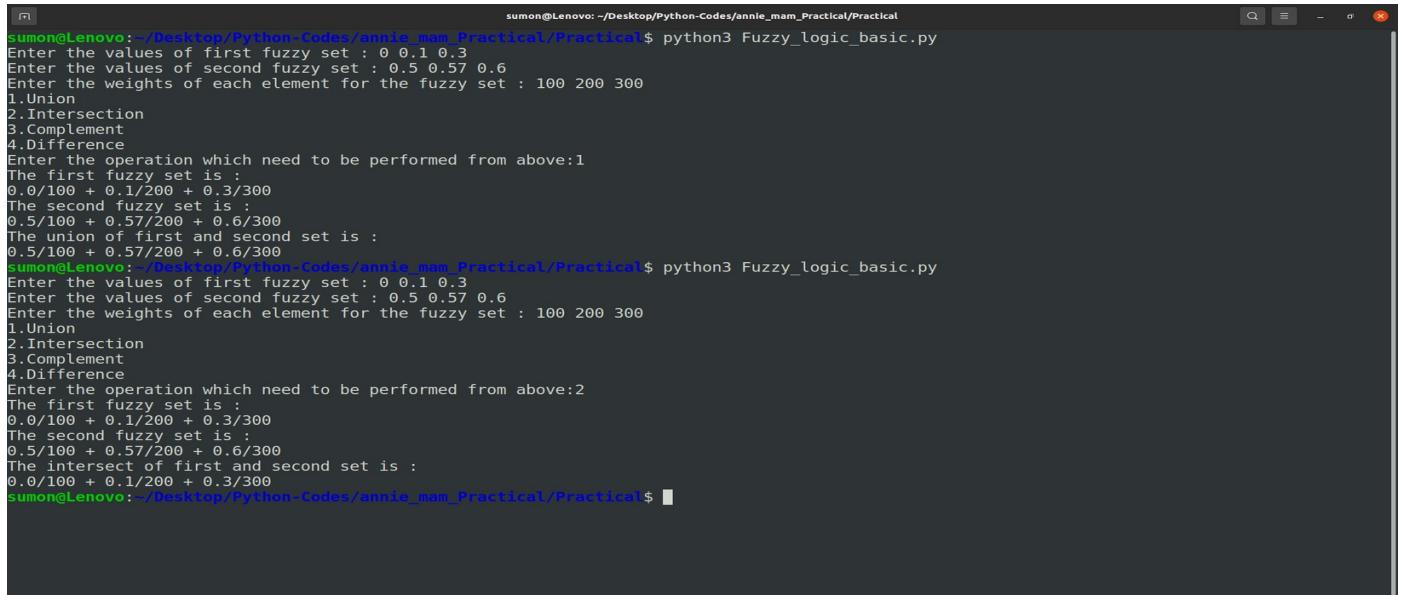
Code:

```
def union(A,B):
R = []
for i in range(0, len(A)):
if(A[i] > B[i]):
R.append(A[i])
else:
R.append(B[i])
return R
def intersect(A,B):
R = []
for i in range(0, len(A)):
if(A[i] < B[i]):
R.append(A[i])
else:
R.append(B[i])
return R
def complement(A):
R = []
for i in range(0,len(A)):
V = 1 - A[i]
R.append(V)
return R
def diff(A,B):
COM_B = complement(B)
R = []
for i in range(0, len(A)):
if(A[i] < COM_B[i]):
R.append(A[i])
else:
R.append(COM_B[i])
return R
def print_fuzzy_set(A,W):
for i in range(0, len(A)-1):
print(f'{round(A[-1], 2)}/{round(W[-1], 1)}', end=' + ')
```

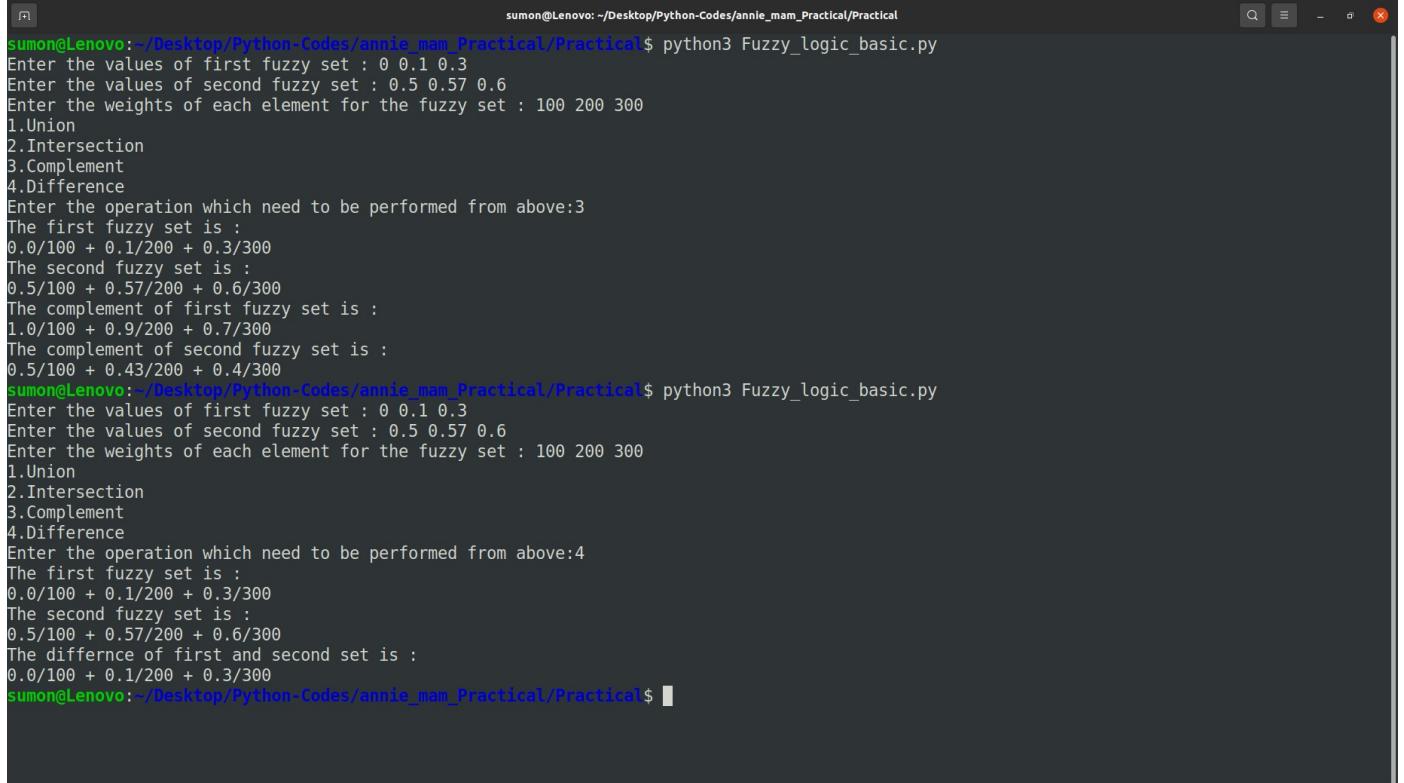
```
print(f'{round(A[-1], 2)}/{round(W[-1], 1)})')
A = list(map(float, input('Enter the values of the first fuzzy
set : ').strip().split()))
B = list(map(float, input('Enter the values of the second
fuzzy set :
').strip().split()))
W = list(map(float, input('Enter the values of the
weights(denominators) of
each elements for the fuzzy set : ').strip().split()))
OP =
float(input('1.Union\n2.Intersection\n3.Complement\
n4.Difference\nEnter the
operation which need to be performed above:'))
print('The first fuzzy set is : ')
print_fuzzy_set(A,W)
print('The second fuzzy set is : ')
print_fuzzy_set(B,W)
if (OP==1):
R = union(A,B)
print('The union of the first and second set is :')
print_fuzzy_set(R,W)
elif (OP==2):
R = intersect(A,B)
print('The intersection of the first and second set is :')
print_fuzzy_set(R,W)
elif (OP==3):
R = complement(A)
print('The complement of the first fuzzy set is :')
print_fuzzy_set(R,W)
R = complement(B)
print('The complement of the second fuzzy set is :')
print_fuzzy_set(R,W)
elif (OP == 4):
R = diff(A,B)
print('The difference of the first and second set is :')
print_fuzzy_set(R,W)
```

```
else:  
    print('Please select any valid number : 1,2,3,4')  
    exit()
```

## Output:



```
sumon@Lenovo:~/Desktop/Python-Codes/annie_mam_Practical/Practical$ python3 Fuzzy_logic_basic.py  
Enter the values of first fuzzy set : 0 0.1 0.3  
Enter the values of second fuzzy set : 0.5 0.57 0.6  
Enter the weights of each element for the fuzzy set : 100 200 300  
1.Union  
2.Intersection  
3.Complement  
4.Difference  
Enter the operation which need to be performed from above:1  
The first fuzzy set is :  
0.0/100 + 0.1/200 + 0.3/300  
The second fuzzy set is :  
0.5/100 + 0.57/200 + 0.6/300  
The union of first and second set is :  
0.5/100 + 0.57/200 + 0.6/300  
sumon@Lenovo:~/Desktop/Python-Codes/annie_mam_Practical/Practical$ python3 Fuzzy_logic_basic.py  
Enter the values of first fuzzy set : 0 0.1 0.3  
Enter the values of second fuzzy set : 0.5 0.57 0.6  
Enter the weights of each element for the fuzzy set : 100 200 300  
1.Union  
2.Intersection  
3.Complement  
4.Difference  
Enter the operation which need to be performed from above:2  
The first fuzzy set is :  
0.0/100 + 0.1/200 + 0.3/300  
The second fuzzy set is :  
0.5/100 + 0.57/200 + 0.6/300  
The intersect of first and second set is :  
0.0/100 + 0.1/200 + 0.3/300  
sumon@Lenovo:~/Desktop/Python-Codes/annie_mam_Practical/Practical$
```



```
sumon@Lenovo:~/Desktop/Python-Codes/annie_mam_Practical/Practical$ python3 Fuzzy_logic_basic.py  
Enter the values of first fuzzy set : 0 0.1 0.3  
Enter the values of second fuzzy set : 0.5 0.57 0.6  
Enter the weights of each element for the fuzzy set : 100 200 300  
1.Union  
2.Intersection  
3.Complement  
4.Difference  
Enter the operation which need to be performed from above:3  
The first fuzzy set is :  
0.0/100 + 0.1/200 + 0.3/300  
The second fuzzy set is :  
0.5/100 + 0.57/200 + 0.6/300  
The complement of first fuzzy set is :  
1.0/100 + 0.9/200 + 0.7/300  
The complement of second fuzzy set is :  
0.5/100 + 0.43/200 + 0.4/300  
sumon@Lenovo:~/Desktop/Python-Codes/annie_mam_Practical/Practical$ python3 Fuzzy_logic_basic.py  
Enter the values of first fuzzy set : 0 0.1 0.3  
Enter the values of second fuzzy set : 0.5 0.57 0.6  
Enter the weights of each element for the fuzzy set : 100 200 300  
1.Union  
2.Intersection  
3.Complement  
4.Difference  
Enter the operation which need to be performed from above:4  
The first fuzzy set is :  
0.0/100 + 0.1/200 + 0.3/300  
The second fuzzy set is :  
0.5/100 + 0.57/200 + 0.6/300  
The difference of first and second set is :  
0.0/100 + 0.1/200 + 0.3/300  
sumon@Lenovo:~/Desktop/Python-Codes/annie_mam_Practical/Practical$
```

2] Implement Union, Intersection, Complement and Difference on Fuzzy set.

$$A = \{1/2.0 + 0.65/4.0 + 0.5/6.0 + 0.35/8.0 + 0/10.0\}$$

$$B = \{0/2.0 + 0.35/4.0 + 0.5/6.0 + 0.65/8.0 + 1/10.0\}$$

Solution:

$$2) \underline{A} = \left\{ \frac{1}{2.0} + \frac{0.65}{4.0} + \frac{0.5}{6.0} + \frac{0.35}{8.0} + \frac{0}{10.0} \right\}$$

$$\underline{B} = \left\{ \frac{0}{2.0} + \frac{0.35}{4.0} + \frac{0.5}{6.0} + \frac{0.65}{8.0} + \frac{1}{10.0} \right\}$$

union:-  $\underline{A} \cup \underline{B} = \underline{\mu_A}(n) \cup \underline{\mu_B}(n)$

$$= \left\{ \frac{1}{2.0} + \frac{0.65}{4.0} + \frac{0.5}{6.0} + \frac{0.65}{8.0} + \frac{1}{10.0} \right\}$$

Intersection:-  $\underline{A} \cap \underline{B} = \underline{\mu_A}(n) \cap \underline{\mu_B}(n)$

$$= \left\{ \frac{0}{2.0} + \frac{0.35}{4.0} + \frac{0.5}{6.0} + \frac{0.35}{8.0} + \frac{0}{10.0} \right\}$$

complement:-

$$\overline{\underline{A}} = \left\{ \frac{0}{2.0} + \frac{0.35}{4.0} + \frac{0.5}{6.0} + \frac{0.65}{8.0} + \frac{1}{10.0} \right\}$$

$$\overline{\underline{B}} = \left\{ \frac{1}{2.0} + \frac{0.65}{4.0} + \frac{0.5}{6.0} + \frac{0.35}{8.0} + \frac{0}{10.0} \right\}$$

Difference:-

$$\underline{A} | \underline{B} = \underline{\mu_A}(n) \cap \overline{\underline{\mu_B}(n)}$$

$$= \left\{ \frac{1}{2.0} + \frac{0.65}{4.0} + \frac{0.5}{6.0} + \frac{0.35}{8.0} + \frac{0}{10.0} \right\}$$

Code:

```
def union(A,B):
R = []
for i in range(0, len(A)):
if(A[i] > B[i]):
R.append(A[i])
else:
R.append(B[i])
return R
def intersect(A,B):
R = []
for i in range(0, len(A)):
if(A[i] < B[i]):
R.append(A[i])
else:
R.append(B[i])
return R
def complement(A):
R = []
for i in range(0,len(A)):
V = 1 - A[i]
R.append(V)
return R
def diff(A,B):
COM_B = complement(B)
R = []
for i in range(0, len(A)):
if(A[i] < COM_B[i]):
R.append(A[i])
else:
R.append(COM_B[i])
return R
def print_fuzzy_set(A,W):
for i in range(0, len(A)-1):
print(f'{round(A[-1], 2)}/{round(W[-1], 1)}', end=' + ')
```

```
print(f'{round(A[-1], 2)}/{round(W[-1], 1)})')
A = list(map(float, input('Enter the values of the first fuzzy
set : ').strip().split()))
B = list(map(float, input('Enter the values of the second
fuzzy set :
').strip().split()))
W = list(map(float, input('Enter the values of the
weights(denominators) of
each elements for the fuzzy set : ').strip().split()))
OP =
float(input('1.Union\n2.Intersection\n3.Complement\
\n4.Difference\nEnter the
operation which need to be performed above:'))
print('The first fuzzy set is : ')
print_fuzzy_set(A,W)
print('The second fuzzy set is : ')
print_fuzzy_set(B,W)
if (OP==1):
R = union(A,B)
print('The union of the first and second set is :')
print_fuzzy_set(R,W)
elif (OP==2):
R = intersect(A,B)
print('The intersection of the first and second set is :')
print_fuzzy_set(R,W)
elif (OP==3):
R = complement(A)
print('The complement of the first fuzzy set is :')
print_fuzzy_set(R,W)
R = complement(B)
print('The complement of the second fuzzy set is :')
print_fuzzy_set(R,W)
elif (OP == 4):
R = diff(A,B)
print('The difference of the first and second set is :')
print_fuzzy_set(R,W)
```

```

else:
print('Please select any valid number : 1,2,3,4')
exit()

```

## Output:

```

sumon@Lenovo:~/Desktop/Python-Codes/annie_mam_Practical/Practical$ python3 Fuzzy_logic_basic.py
Enter the values of first fuzzy set : 1.0 0.65 0.5 0.35 0
Enter the values of second fuzzy set : 0.0 0.35 0.5 0.65 1
Enter the weights of each element for the fuzzy set : 2 4 6 8 10
1.Union
2.Intersection
3.Complement
4.Difference
Enter the operation which need to be performed from above:1
The first fuzzy set is :
1.0/2 + 0.65/4 + 0.5/6 + 0.35/8 + 0.0/10
The second fuzzy set is :
0.0/2 + 0.35/4 + 0.5/6 + 0.65/8 + 1.0/10
The union of first and second set is :
1.0/2 + 0.65/4 + 0.5/6 + 0.65/8 + 1.0/10
sumon@Lenovo:~/Desktop/Python-Codes/annie_mam_Practical/Practical$ python3 Fuzzy_logic_basic.py
Enter the values of first fuzzy set : 1.0 0.65 0.5 0.35 0
Enter the values of second fuzzy set : 0.0 0.35 0.5 0.65 1
Enter the weights of each element for the fuzzy set : 2 4 6 8 10
1.Union
2.Intersection
3.Complement
4.Difference
Enter the operation which need to be performed from above:2
The first fuzzy set is :
1.0/2 + 0.65/4 + 0.5/6 + 0.35/8 + 0.0/10
The second fuzzy set is :
0.0/2 + 0.35/4 + 0.5/6 + 0.65/8 + 1.0/10
The intersect of first and second set is :
0.0/2 + 0.35/4 + 0.5/6 + 0.35/8 + 0.0/10
sumon@Lenovo:~/Desktop/Python-Codes/annie_mam_Practical/Practical$ 

```

```

sumon@Lenovo:~/Desktop/Python-Codes/annie_mam_Practical/Practical$ python3 Fuzzy_logic_basic.py
Enter the values of first fuzzy set : 1.0 0.65 0.5 0.35 0
Enter the values of second fuzzy set : 0.0 0.35 0.5 0.65 1
Enter the weights of each element for the fuzzy set : 2 4 6 8 10
1.Union
2.Intersection
3.Complement
4.Difference
Enter the operation which need to be performed from above:3
The first fuzzy set is :
1.0/2 + 0.65/4 + 0.5/6 + 0.35/8 + 0.0/10
The second fuzzy set is :
0.0/2 + 0.35/4 + 0.5/6 + 0.65/8 + 1.0/10
The complement of first fuzzy set is :
0.0/2 + 0.35/4 + 0.5/6 + 0.65/8 + 1.0/10
The complement of second fuzzy set is :
1.0/2 + 0.65/4 + 0.5/6 + 0.35/8 + 0.0/10
sumon@Lenovo:~/Desktop/Python-Codes/annie_mam_Practical/Practical$ python3 Fuzzy_logic_basic.py
Enter the values of first fuzzy set : 1.0 0.65 0.5 0.35 0
Enter the values of second fuzzy set : 0.0 0.35 0.5 0.65 1
Enter the weights of each element for the fuzzy set : 2 4 6 8 10
1.Union
2.Intersection
3.Complement
4.Difference
Enter the operation which need to be performed from above:4
The first fuzzy set is :
1.0/2 + 0.65/4 + 0.5/6 + 0.35/8 + 0.0/10
The second fuzzy set is :
0.0/2 + 0.35/4 + 0.5/6 + 0.65/8 + 1.0/10
The difference of first and second set is :
1.0/2 + 0.65/4 + 0.5/6 + 0.35/8 + 0.0/10
sumon@Lenovo:~/Desktop/Python-Codes/annie_mam_Practical/Practical$ 

```

3] Implement Union, Intersection, Complement and Difference on Fuzzy set.

$$I_1 = \{0/0 + 0.5/20 + 0.35/40 + 0.75/60 + 0.95/80 + 1/100\}$$
$$I_2 = \{0/0 + 0.45/20 + 0.55/40 + 0.65/60 + 0.9/80 + 1/100\}$$

Solution:

$$3) \quad I_1 = \left\{ \frac{0}{0} + \frac{0.5}{20} + \frac{0.35}{40} + \frac{0.75}{60} + \frac{0.95}{80} + \frac{1}{100} \right\}$$

$$I_2 = \left\{ \frac{0}{0} + \frac{0.45}{20} + \frac{0.55}{40} + \frac{0.65}{60} + \frac{0.9}{80} + \frac{1}{100} \right\}$$

Union :-  $\underline{I_1} \cup \underline{I_2} = \mu_{I_1}(\sim) \cup \mu_{I_2}(\sim)$

$$= \left\{ \frac{0}{0} + \frac{0.5}{20} + \frac{0.55}{40} + \frac{0.75}{60} + \frac{0.95}{80} + \frac{1}{100} \right\}$$

Intersection :-  $\underline{I_1} \cap \underline{I_2} = \mu_{I_1}(\sim) \cap \mu_{I_2}(\sim)$

$$= \left\{ \frac{0}{0} + \frac{0.45}{20} + \frac{0.35}{40} + \frac{0.65}{60} + \frac{0.9}{80} + \frac{1}{100} \right\}$$

Complement :-

$$\bar{I}_1 = \left\{ \frac{1}{0} + \frac{0.5}{20} + \frac{0.65}{40} + \frac{0.25}{60} + \frac{0.05}{80} + \frac{0}{100} \right\}$$

$$\bar{I}_2 = \left\{ \frac{1}{0} + \frac{0.55}{20} + \frac{0.45}{40} + \frac{0.35}{60} + \frac{0.1}{80} + \frac{0}{100} \right\}$$

Difference :-

$$\underline{I_1} | \underline{I_2} = \underline{\mu_{I_1}(\sim)} \cap \underline{\mu_{I_2}(\sim)}$$

$$= \left\{ \frac{0}{0} + \frac{0.5}{20} + \frac{0.35}{40} + \frac{0.35}{60} + \frac{0.1}{80} + \frac{0}{100} \right\}$$

Code:

```
def union(A,B):
R = []
for i in range(0, len(A)):
if(A[i] > B[i]):
R.append(A[i])
else:
R.append(B[i])
return R
def intersect(A,B):
R = []
for i in range(0, len(A)):
if(A[i] < B[i]):
R.append(A[i])
else:
R.append(B[i])
return R
def complement(A):
R = []
for i in range(0,len(A)):
V = 1 - A[i]
R.append(V)
return R
def diff(A,B):
COM_B = complement(B)
R = []
for i in range(0, len(A)):
if(A[i] < COM_B[i]):
R.append(A[i])
else:
R.append(COM_B[i])
return R
def print_fuzzy_set(A,W):
for i in range(0, len(A)-1):
print(f'{round(A[-1], 2)}/{round(W[-1], 1)}', end=' + ')
```

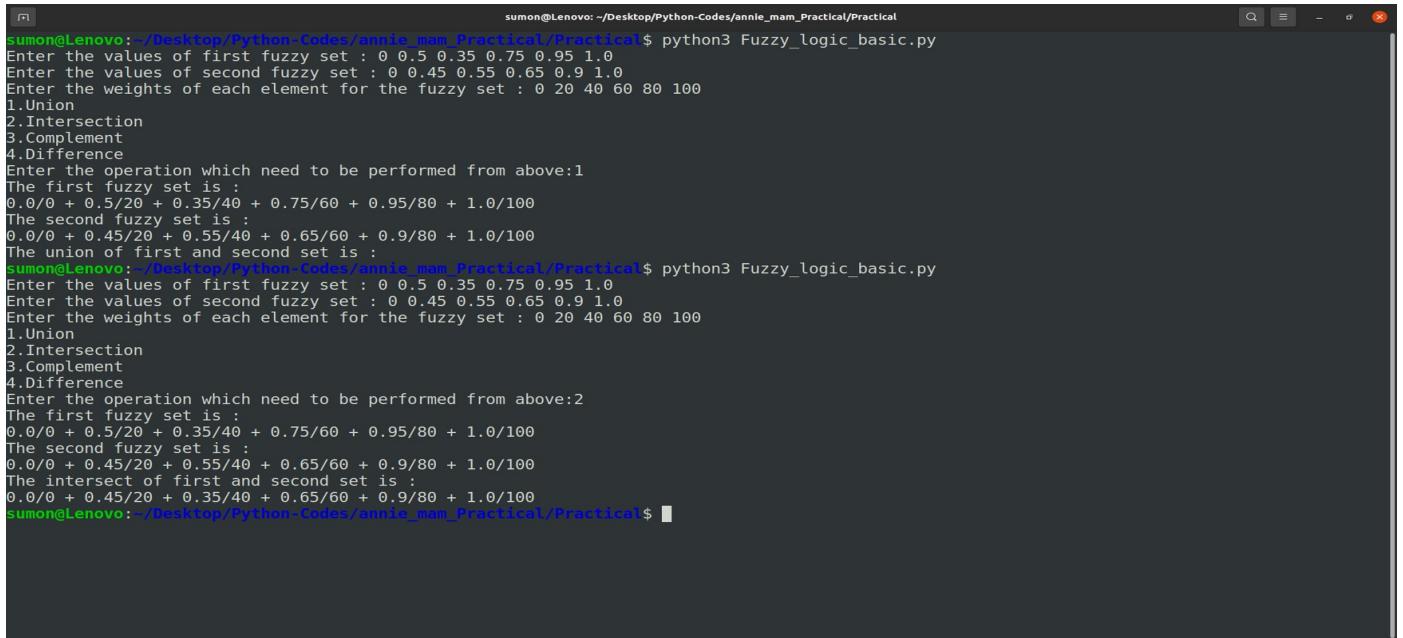
```
print(f'{round(A[-1], 2)}/{round(W[-1], 1)})')
A = list(map(float, input('Enter the values of the first fuzzy
set : ').strip().split()))
B = list(map(float, input('Enter the values of the second
fuzzy set :
').strip().split()))
W = list(map(float, input('Enter the values of the
weights(denominators) of
each elements for the fuzzy set : ').strip().split()))
OP =
float(input('1.Union\n2.Intersection\n3.Complement\
n4.Difference\nEnter the
operation which need to be performed above:'))
print('The first fuzzy set is : ')
print_fuzzy_set(A,W)
print('The second fuzzy set is : ')
print_fuzzy_set(B,W)
if (OP==1):
R = union(A,B)
print('The union of the first and second set is :')
print_fuzzy_set(R,W)
elif (OP==2):
R = intersect(A,B)
print('The intersection of the first and second set is :')
print_fuzzy_set(R,W)
elif (OP==3):
R = complement(A)
print('The complement of the first fuzzy set is :')
print_fuzzy_set(R,W)
R = complement(B)
print('The complement of the second fuzzy set is :')
print_fuzzy_set(R,W)
elif (OP == 4):
R = diff(A,B)
print('The difference of the first and second set is :')
print_fuzzy_set(R,W)
```

```

else:
print('Please select any valid number : 1,2,3,4')
exit()

```

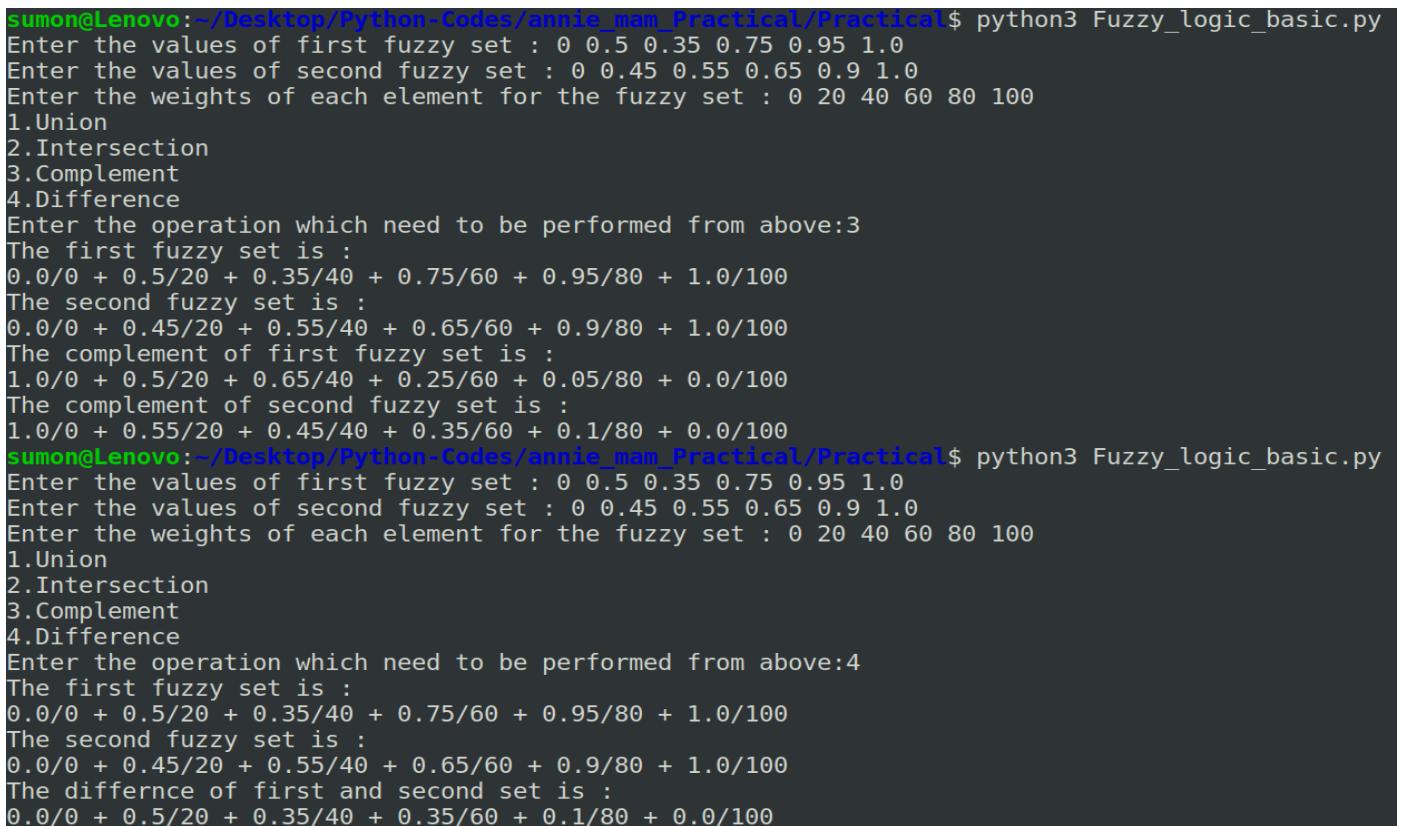
## Output:



```

sumon@Lenovo:~/Desktop/Python-Codes/annie_mam_Practical/Practical$ python3 Fuzzy_logic_basic.py
Enter the values of first fuzzy set : 0 0.5 0.35 0.75 0.95 1.0
Enter the values of second fuzzy set : 0 0.45 0.55 0.65 0.9 1.0
Enter the weights of each element for the fuzzy set : 0 20 40 60 80 100
1.Union
2.Intersection
3.Complement
4.Difference
Enter the operation which need to be performed from above:1
The first fuzzy set is :
0.0/0 + 0.5/20 + 0.35/40 + 0.75/60 + 0.95/80 + 1.0/100
The second fuzzy set is :
0.0/0 + 0.45/20 + 0.55/40 + 0.65/60 + 0.9/80 + 1.0/100
The union of first and second set is :
sumon@Lenovo:~/Desktop/Python-Codes/annie_mam_Practical/Practical$ python3 Fuzzy_logic_basic.py
Enter the values of first fuzzy set : 0 0.5 0.35 0.75 0.95 1.0
Enter the values of second fuzzy set : 0 0.45 0.55 0.65 0.9 1.0
Enter the weights of each element for the fuzzy set : 0 20 40 60 80 100
1.Union
2.Intersection
3.Complement
4.Difference
Enter the operation which need to be performed from above:2
The first fuzzy set is :
0.0/0 + 0.5/20 + 0.35/40 + 0.75/60 + 0.95/80 + 1.0/100
The second fuzzy set is :
0.0/0 + 0.45/20 + 0.55/40 + 0.65/60 + 0.9/80 + 1.0/100
The intersect of first and second set is :
0.0/0 + 0.45/20 + 0.35/40 + 0.65/60 + 0.9/80 + 1.0/100
sumon@Lenovo:~/Desktop/Python-Codes/annie_mam_Practical/Practical$ 

```



```

sumon@Lenovo:~/Desktop/Python-Codes/annie_mam_Practical/Practical$ python3 Fuzzy_logic_basic.py
Enter the values of first fuzzy set : 0 0.5 0.35 0.75 0.95 1.0
Enter the values of second fuzzy set : 0 0.45 0.55 0.65 0.9 1.0
Enter the weights of each element for the fuzzy set : 0 20 40 60 80 100
1.Union
2.Intersection
3.Complement
4.Difference
Enter the operation which need to be performed from above:3
The first fuzzy set is :
0.0/0 + 0.5/20 + 0.35/40 + 0.75/60 + 0.95/80 + 1.0/100
The second fuzzy set is :
0.0/0 + 0.45/20 + 0.55/40 + 0.65/60 + 0.9/80 + 1.0/100
The complement of first fuzzy set is :
1.0/0 + 0.5/20 + 0.65/40 + 0.25/60 + 0.05/80 + 0.0/100
The complement of second fuzzy set is :
1.0/0 + 0.55/20 + 0.45/40 + 0.35/60 + 0.1/80 + 0.0/100
sumon@Lenovo:~/Desktop/Python-Codes/annie_mam_Practical/Practical$ python3 Fuzzy_logic_basic.py
Enter the values of first fuzzy set : 0 0.5 0.35 0.75 0.95 1.0
Enter the values of second fuzzy set : 0 0.45 0.55 0.65 0.9 1.0
Enter the weights of each element for the fuzzy set : 0 20 40 60 80 100
1.Union
2.Intersection
3.Complement
4.Difference
Enter the operation which need to be performed from above:4
The first fuzzy set is :
0.0/0 + 0.5/20 + 0.35/40 + 0.75/60 + 0.95/80 + 1.0/100
The second fuzzy set is :
0.0/0 + 0.45/20 + 0.55/40 + 0.65/60 + 0.9/80 + 1.0/100
The differnce of first and second set is :
0.0/0 + 0.5/20 + 0.35/40 + 0.35/60 + 0.1/80 + 0.0/100

```

4] Implement Union, Intersection, Complement and Difference on Fuzzy set.

$$T_1 = \{0/0 + 0.2/1 + 0.3/2 + 0.6/3 + 0.9/4 + 1/5\}$$

$$T_2 = \{0/0 + 0.1/1 + 0.2/2 + 0.3/3 + 0.4/4 + 0.7/5\}$$

Solution:

$$4) T_1 = \left\{ \frac{0}{0} + \frac{0.2}{1} + \frac{0.3}{2} + \frac{0.6}{3} + \frac{0.9}{4} + \frac{1}{5} \right\}$$

$$T_2 = \left\{ \frac{0}{0} + \frac{0.1}{1} + \frac{0.2}{2} + \frac{0.3}{3} + \frac{0.4}{4} + \frac{0.7}{5} \right\}$$

union :-  $\underline{T_1} \cup \underline{T_2} = \mathcal{M}_{T_1}(n) \cup \mathcal{M}_{T_2}(n)$

$$\rightarrow \left\{ \cancel{\frac{0}{0}} + \cancel{\frac{0.1}{1}} + \cancel{\frac{0.2}{2}} \right.$$

$$= \left\{ \frac{0}{0} + \frac{0.2}{1} + \frac{0.3}{2} + \frac{0.6}{3} + \frac{0.9}{4} + \frac{1}{5} \right\}$$

Intersection :-  $\underline{T_1} \cap \underline{T_2} = \mathcal{M}_{T_1}(n) \cap \mathcal{M}_{T_2}(n)$

$$= \left\{ \frac{0}{0} + \frac{0.1}{1} + \frac{0.2}{2} + \frac{0.3}{3} + \frac{0.4}{4} + \frac{0.7}{5} \right\}$$

complement :-

$$\overline{T_1} = \left\{ \frac{1}{0} + \frac{0.8}{1} + \frac{0.7}{2} + \frac{0.4}{3} + \frac{0.1}{4} + \frac{0}{5} \right\}$$

$$\overline{T_2} = \left\{ \cancel{\frac{0}{0}} + \frac{0.9}{1} + \frac{0.8}{2} + \frac{0.7}{3} + \frac{0.6}{4} + \frac{0.3}{5} \right\}$$

Difference :-

$$\underline{T_1} | \underline{T_2} = \underline{\mathcal{M}_{T_1}(n)} \cap \overline{\mathcal{M}_{T_2}(n)}$$

$$= \left\{ \frac{0}{0} + \frac{0.2}{1} + \frac{0.3}{2} + \frac{0.6}{3} + \frac{0.6}{4} + \frac{0.3}{5} \right\}$$

Code:

```
def union(A,B):
R = []
for i in range(0, len(A)):
if(A[i] > B[i]):
R.append(A[i])
else:
R.append(B[i])
return R
def intersect(A,B):
R = []
for i in range(0, len(A)):
if(A[i] < B[i]):
R.append(A[i])
else:
R.append(B[i])
return R
def complement(A):
R = []
for i in range(0,len(A)):
V = 1 - A[i]
R.append(V)
return R
def diff(A,B):
COM_B = complement(B)
R = []
for i in range(0, len(A)):
if(A[i] < COM_B[i]):
R.append(A[i])
else:
R.append(COM_B[i])
return R
def print_fuzzy_set(A,W):
for i in range(0, len(A)-1):
print(f'{round(A[-1], 2)}/{round(W[-1], 1)}', end=' + ')
```

```
print(f'{round(A[-1], 2)}/{round(W[-1], 1)})')
A = list(map(float, input('Enter the values of the first fuzzy
set : ').strip().split()))
B = list(map(float, input('Enter the values of the second
fuzzy set :
').strip().split()))
W = list(map(float, input('Enter the values of the
weights(denominators) of
each elements for the fuzzy set : ').strip().split()))
OP =
float(input('1.Union\n2.Intersection\n3.Complement\
\n4.Difference\nEnter the
operation which need to be performed above:'))
print('The first fuzzy set is : ')
print_fuzzy_set(A,W)
print('The second fuzzy set is : ')
print_fuzzy_set(B,W)
if (OP==1):
R = union(A,B)
print('The union of the first and second set is :')
print_fuzzy_set(R,W)
elif (OP==2):
R = intersect(A,B)
print('The intersection of the first and second set is :')
print_fuzzy_set(R,W)
elif (OP==3):
R = complement(A)
print('The complement of the first fuzzy set is :')
print_fuzzy_set(R,W)
R = complement(B)
print('The complement of the second fuzzy set is :')
print_fuzzy_set(R,W)
elif (OP == 4):
R = diff(A,B)
print('The difference of the first and second set is :')
print_fuzzy_set(R,W)
```

```

else:
print('Please select any valid number : 1,2,3,4')
exit()

```

## Output:

```

sumon@Lenovo:~/Desktop/Python-Codes/annie_mam_Practical/Practical$ python3 Fuzzy_logic_basic.py
Enter the values of first fuzzy set : 0.0 0.2 0.3 0.6 0.9 1
Enter the values of second fuzzy set : 0.0 0.1 0.2 0.3 0.4 0.7
Enter the weights of each element for the fuzzy set : 0 1 2 3 4 5
1.Union
2.Intersection
3.Complement
4.Difference
Enter the operation which need to be performed from above:1
The first fuzzy set is :
0.0/0 + 0.2/1 + 0.3/2 + 0.6/3 + 0.9/4 + 1.0/5
The second fuzzy set is :
0.0/0 + 0.1/1 + 0.2/2 + 0.3/3 + 0.4/4 + 0.7/5
The union of first and second set is :
sumon@Lenovo:~/Desktop/Python-Codes/annie_mam_Practical/Practical$ python3 Fuzzy_logic_basic.py
Enter the values of first fuzzy set : 0.0 0.2 0.3 0.6 0.9 1
Enter the values of second fuzzy set : 0.0 0.1 0.2 0.3 0.4 0.7
Enter the weights of each element for the fuzzy set : 0 1 2 3 4 5
1.Union
2.Intersection
3.Complement
4.Difference
Enter the operation which need to be performed from above:2
The first fuzzy set is :
0.0/0 + 0.2/1 + 0.3/2 + 0.6/3 + 0.9/4 + 1.0/5
The second fuzzy set is :
0.0/0 + 0.1/1 + 0.2/2 + 0.3/3 + 0.4/4 + 0.7/5
The intersect of first and second set is :
0.0/0 + 0.1/1 + 0.2/2 + 0.3/3 + 0.4/4 + 0.7/5
sumon@Lenovo:~/Desktop/Python-Codes/annie_mam_Practical/Practical$ 

```

```

sumon@Lenovo:~/Desktop/Python-Codes/annie_mam_Practical/Practical$ python3 Fuzzy_logic_basic.py
Enter the values of first fuzzy set : 0.0 0.2 0.3 0.6 0.9 1.0
Enter the values of second fuzzy set : 0.0 0.1 0.2 0.3 0.4 0.7
Enter the weights of each element for the fuzzy set : 0 1 2 3 4 5
1.Union
2.Intersection
3.Complement
4.Difference
Enter the operation which need to be performed from above:3
The first fuzzy set is :
0.0/0 + 0.2/1 + 0.3/2 + 0.6/3 + 0.9/4 + 1.0/5
The second fuzzy set is :
0.0/0 + 0.1/1 + 0.2/2 + 0.3/3 + 0.4/4 + 0.7/5
The complement of first fuzzy set is :
1.0/0 + 0.8/1 + 0.7/2 + 0.4/3 + 0.1/4 + 0.0/5
The complement of second fuzzy set is :
sumon@Lenovo:~/Desktop/Python-Codes/annie_mam_Practical/Practical$ python3 Fuzzy_logic_basic.py
Enter the values of first fuzzy set : 0.0 0.2 0.3 0.6 0.9 1.0
Enter the values of second fuzzy set : 0.0 0.1 0.2 0.3 0.4 0.7
Enter the weights of each element for the fuzzy set : 0 1 2 3 4 5
1.Union
2.Intersection
3.Complement
4.Difference
Enter the operation which need to be performed from above:4
The first fuzzy set is :
0.0/0 + 0.2/1 + 0.3/2 + 0.6/3 + 0.9/4 + 1.0/5
The second fuzzy set is :
0.0/0 + 0.1/1 + 0.2/2 + 0.3/3 + 0.4/4 + 0.7/5
The differnce of first and second set is :
0.0/0 + 0.2/1 + 0.3/2 + 0.6/3 + 0.6/4 + 0.3/5
sumon@Lenovo:~/Desktop/Python-Codes/annie_mam_Practical/Practical$ 

```

# Practical-8

## Cartesian Product

1] Cartesian product of two fuzzy sets and perform max-min composition and max product composition.

$$A = \{0.1/30 + 0.2/60 + 0.3/90 + 0.4/120\}$$

$$B = \{1/1 + 0.2/2 + 0.5/3 + 0.7/4 + 0.3/5 + 0/6\}$$

Solution:

$$1) A = \left\{ \frac{0.1}{30} + \frac{0.2}{60} + \frac{0.3}{90} + \frac{0.4}{120} \right\}$$

$$B = \left\{ \frac{0.2}{2} + \frac{0.5}{3} + \frac{0.7}{4} + \frac{0.3}{5} + \frac{0.1}{6} \right\}$$

$$R = A \times B$$

$n_i \rightarrow A$   
 $y_j \rightarrow B$

$$R = \left[ \min(n_i, y_j) \right]_{\substack{i=1,2,3,4 \\ j=1,2,3,4,5,6}}$$

$$\mu(n_1, y_1) = \min(0.1, 1) = 0.1$$

$$\mu(n_1, y_2) = \min(0.1, 0.2) = 0.1$$

$$\mu(n_1, y_3) = \min(0.1, 0.5) = 0.1$$

$$\mu(n_1, y_4) = \min(0.1, 0.7) = 0.1$$

$$\mu(n_1, y_5) = \min(0.1, 0.3) = 0.1$$

$$\mu(n_1, y_6) = \min(0.1, 0) = 0$$

$$\mu(n_2, y_1) = \min(0.2, 1) = 0.2$$

$$\mu(n_2, y_2) = \min(0.2, 0.2) = 0.2$$

$$\mu(n_2, y_3) = \min(0.2, 0.5) = 0.2$$

$$\mu(n_2, y_4) = \min(0.2, 0.7) = 0.2$$

$$\mu(n_2, y_5) = \min(0.2, 0.3) = 0.2$$

$$\mu(n_2, y_6) = \min(0.2, 0) = 0$$

$$\mu(n_3, y_1) = \min(0.3, 1) = 0.3$$

$$\mu(n_3, y_2) = \min(0.3, 0.2) = 0.2$$

$$\mu(n_3, y_3) = \min(0.3, 0.5) = 0.3$$

$$\mu(n_3, y_4) = \min(0.3, 0.7) = 0.3$$

$$\mu(n_3, y_5) = \min(0.3, 0.3) = 0.3$$

$$\mu(n_3, y_6) = \min(0.3, 0) = 0$$

$$\mu(n_4, y_1) = \min(0.4, 1) = 0.4$$

$$\mu(n_4, y_2) = \min(0.4, 0.2) = 0.2$$

$$\mu(n_4, y_3) = \min(0.4, 0.5) = 0.4$$

$$\mu(n_4, y_4) = \min(0.4, 0.7) = 0.4$$

$$\mu(n_4, y_5) = \min(0.4, 0.3) = 0.3$$

$$\mu(n_4, y_6) = \min(0.4, 0) = 0$$

$$\therefore R = \begin{bmatrix} 0.1 & 0.1 & 0.1 & 0.1 & 0.1 & 0 \\ 0.2 & 0.2 & 0.2 & 0.2 & 0.2 & 0 \\ 0.3 & 0.2 & 0.3 & 0.3 & 0.3 & 0 \\ 0.4 & 0.2 & 0.4 & 0.4 & 0.3 & 0 \end{bmatrix}$$

Code:

```
import numpy as np
def cartesian_prod_of_two_sets(A,B):
val = []
for i in range(0,len(A)):
for j in range(0,len(B)):
if A[i]<B[j]:
val.append(A[i])
else:
val.append(B[j])
mat = np.array(val).reshape(len(A),len(B))
return mat
def print_fuzzy_set(A,W):
for i in range(0, len(A)-1):
print(f'{round(A[i],2)}/{round(W[i],1)}', end = ' + ')
print(f'{round(A[-1], 2)}/{round(W[-1], 1)})')
A=list(map(float,input('Enter the values of first fuzzy set : ').strip().split()))
B=list(map(float,input('Enter the values of second fuzzy set : ').strip().split()))
W1=list(map(int,input('Enter the weights of each element for the first fuzzy set : ').strip().split()))
W2=list(map(int,input('Enter the weights of each element for the second fuzzy set : ').strip().split()))
print('The first fuzzy set is :')
print_fuzzy_set(A,W1)
print('The second fuzzy set is :')
print_fuzzy_set(B,W2)
print('The cartesian product of first and second fuzzy set is : \n')
R = cartesian_prod_of_two_sets(A,B)
print(f'____|____{W2[0]}____|')
```

```

for i in range(1, len(W2)-1):
print(f'__{W2[i]}__', end = '|')
print(f'__{W2[-1]}__')
for i in range(0, R.shape[0]-1):
print(f'{W1[i]} ', end = '|')
for j in range(0, R.shape[1]):
print(f'__{R[i][j]}__', end = '|')
print('\n')
print(f'{W1[-1]} ', end = '|')
for i in range(0, R.shape[1]):
print(f'__{R[-1][i]}__', end = '|')
print('\n')

```

## Output:

```

sumon@Lenovo:~/Desktop/Python-Codes/annie_mam_Practical/Practical$ python3 cartesian_product.py
Enter the values of first fuzzy set : 0.1 0.2 0.3 0.4
Enter the values of second fuzzy set : 1.0 0.2 0.5 0.7 0.3 0
Enter the weights of each element for the first fuzzy set : 30 60 90 120
Enter the weights of each element for the second fuzzy set : 1 2 3 4 5 6
The first fuzzy set is :
0.1/30 + 0.2/60 + 0.3/90 + 0.4/120
The second fuzzy set is :
1.0/1 + 0.2/2 + 0.5/3 + 0.7/4 + 0.3/5 + 0.0/6
The cartesian product of first and second fuzzy set is :

 1   2   3   4   5   6
30 | 0.1 | 0.1 | 0.1 | 0.1 | 0.1 | 0.0 |
60 | 0.2 | 0.2 | 0.2 | 0.2 | 0.2 | 0.0 |
90 | 0.3 | 0.2 | 0.3 | 0.3 | 0.3 | 0.0 |
120 | 0.4 | 0.2 | 0.4 | 0.4 | 0.3 | 0.0 |

sumon@Lenovo:~/Desktop/Python-Codes/annie_mam_Practical/Practical$ 

```

2] Cartesian product of two fuzzy sets and perform max-min composition and max product composition.

$$R_1 = \{0.1/2 + 0.3/4 + 0.7/6 + 0.4/8\}$$

$$R_2 = \{0.1/0.1 + 0.3/0.2 + 0.3/0.3 + 0.4/0.4 + 0.5/0.5 + 0.2/0.6\}$$

Solution:

$$2) R_1 = \left\{ \frac{0.1}{2} + \frac{0.3}{4} + \frac{0.7}{6} + \frac{0.4}{8} + \frac{0.2}{10} \right\}$$

$$R_2 = \left\{ \frac{0.1}{0.1} + \frac{0.3}{0.2} + \frac{0.7}{0.3} + \frac{0.4}{0.4} + \frac{0.5}{0.5} + \frac{0.2}{0.6} \right\}$$

$$R_1 \times R_2 = \left[ \mu_{ij}(R_i, R_j) \right]_{i=1..5, j=1..6} \quad \begin{matrix} n_i \rightarrow R_1 \\ y_j \rightarrow R_2 \end{matrix}$$

$$\mu(n_1, y_1) = \min(0.1, 0.1) = 0.1$$

$$\mu(n_1, y_2) = \min(0.1, 0.3) = 0.1$$

$$\mu(n_1, y_3) = \min(0.1, 0.3) = 0.1$$

$$\mu(n_1, y_4) = \min(0.1, 0.4) = 0.1$$

$$\mu(n_1, y_5) = \min(0.1, 0.5) = 0.1$$

$$\mu(n_1, y_6) = \min(0.1, 0.2) = 0.1$$

$$\mu(n_2, y_1) = \min(0.3, 0.1) = 0.1$$

$$\mu(n_2, y_2) = \min(0.3, 0.3) = 0.3$$

$$\mu(n_2, y_3) = \min(0.3, 0.3) = 0.3$$

$$\mu(n_2, y_4) = \min(0.3, 0.4) = 0.3$$

$$\mu(n_2, y_5) = \min(0.3, 0.5) = 0.3$$

$$\mu(n_2, y_6) = \min(0.3, 0.2) = 0.2$$

$$\mu(n_3, y_1) = \min(0.7, 0.1) = 0.1$$

$$\mu(n_3, y_2) = \min(0.7, 0.3) = 0.3$$

$$\mu(n_3, y_3) = \min(0.7, 0.3) = 0.3$$

$$\mu(n_3, y_4) = \min(0.7, 0.4) = 0.4$$

$$\mu(n_3, y_5) = \min(0.7, 0.5) = 0.5$$

$$\mu(n_3, y_6) = \min(0.7, 0.2) = 0.2$$

$$\mu(n_4, y_1) = \min(0.4, 0.1) = 0.1$$

$$\mu(n_4, y_2) = \min(0.4, 0.3) = 0.3$$

$$\mu(n_4, y_3) = \min(0.4, 0.3) = 0.3$$

$$\mu(n_4, y_4) = \min(0.4, 0.4) = 0.4$$

$$\mu(n_4, y_5) = \min(0.4, 0.5) = 0.4$$

$$\mu(n_4, y_6) = \min(0.4, 0.2) = 0.2$$

$$\mu(n_5, y_1) = \min(0.2, 0.1) = 0.1$$

$$\mu(n_5, y_2) = \min(0.2, 0.3) = 0.2$$

$$\mu(n_5, y_3) = \min(0.2, 0.3) = 0.2$$

$$\mu(n_5, y_4) = \min(0.2, 0.4) = 0.2$$

$$\mu(n_5, y_5) = \min(0.2, 0.5) = 0.2$$

$$\mu(n_5, y_6) = \min(0.2, 0.2) = 0.2$$

$$R_i \times R_j = \begin{bmatrix} 0.1 & 0.1 & 0.1 & 0.1 & 0.1 & 0.1 \\ 0.1 & 0.3 & 0.3 & 0.3 & 0.3 & 0.2 \\ 0.1 & 0.3 & 0.3 & 0.4 & 0.5 & 0.2 \\ 0.1 & 0.3 & 0.3 & 0.4 & 0.4 & 0.2 \\ 0.1 & 0.3 & 0.2 & 0.2 & 0.2 & 0.2 \end{bmatrix}$$

Code:

```
import numpy as np
def cartesian_prod_of_two_sets(A,B):
val = []
for i in range(0,len(A)):
for j in range(0,len(B)):
if A[i]<B[j]:
val.append(A[i])
else:
val.append(B[j])
mat = np.array(val).reshape(len(A),len(B))
return mat
def print_fuzzy_set(A,W):
for i in range(0, len(A)-1):
print(f'{round(A[i],2)}/{round(W[i],1)}', end = ' + ')
print(f'{round(A[-1], 2)}/{round(W[-1], 1)})')
A=list(map(float,input('Enter the values of first fuzzy set : ').strip().split()))
B=list(map(float,input('Enter the values of second fuzzy set : ').strip().split()))
W1=list(map(int,input('Enter the weights of each element for the first fuzzy set : ').strip().split()))
W2=list(map(int,input('Enter the weights of each element for the second fuzzy set : ').strip().split()))
print('The first fuzzy set is :')
print_fuzzy_set(A,W1)
print('The second fuzzy set is :')
print_fuzzy_set(B,W2)
print('The cartesian product of first and second fuzzy set is : \n')
R = cartesian_prod_of_two_sets(A,B)
print(f'____|____ {W2[0]}____|____', end = '|')
for i in range(1, len(W2)-1):
```

```

print(f'____{W2[i]}____', end = '|')
print(f'____{W2[-1]}____')
for i in range(0, R.shape[0]-1):
    print(f'{W1[i]} ', end = '|')
    for j in range(0, R.shape[1]):
        print(f'____{R[i][j]}____', end = '|')
    print('\n')
    print(f'{W1[-1]} ', end = '|')
    for i in range(0, R.shape[1]):
        print(f'____{R[-1][i]}____', end = '|')
    print('\n')

```

## Output:

```

sumon@Lenovo:~/Desktop/Python-Codes/annie_mam_Practical/Practical$ python3 cartesian_product.py
Enter the values of first fuzzy set : 0.1 0.3 0.7 0.4 0.2
Enter the values of second fuzzy set : 0.1 0.3 0.3 0.4 0.5 0.2
Enter the weights of each element for the first fuzzy set : 2 4 6 8 10
Enter the weights of each element for the second fuzzy set : 0.1 0.2 0.3 0.4 0.5 0.6
The first fuzzy set is :
0.1/2.0 + 0.3/4.0 + 0.7/6.0 + 0.4/8.0 + 0.2/10.0
The second fuzzy set is :
0.1/0.1 + 0.3/0.2 + 0.3/0.3 + 0.4/0.4 + 0.5/0.5 + 0.2/0.6
The cartesian product of first and second fuzzy set is :
-----+-----+-----+-----+-----+-----+-----+-----+
2.0 | 0.1 | 0.2 | 0.3 | 0.4 | 0.5 | 0.6 |
-----+-----+-----+-----+-----+-----+-----+
4.0 | 0.1 | 0.3 | 0.3 | 0.3 | 0.2 | 0.2 |
-----+-----+-----+-----+-----+-----+-----+
6.0 | 0.1 | 0.3 | 0.3 | 0.4 | 0.5 | 0.2 |
-----+-----+-----+-----+-----+-----+-----+
8.0 | 0.1 | 0.3 | 0.3 | 0.4 | 0.4 | 0.2 |
-----+-----+-----+-----+-----+-----+-----+
10.0 | 0.1 | 0.2 | 0.2 | 0.2 | 0.2 | 0.2 |
-----+-----+-----+-----+-----+-----+-----+
sumon@Lenovo:~/Desktop/Python-Codes/annie_mam_Practical/Practical$ 

```

3] Cartesian product of two fuzzy sets and perform max-min composition and max product composition.

$$M_1 = \{0/0.72 + 0.8/0.725 + 1/0.75 + 0.8/0.775 + 0/0.78\}$$

$$M_2 = \{0/21k + 0.2/22k + 0.7/23k + 1/24k + 0.7/25k + 0.2/26k + 0/27k\}$$

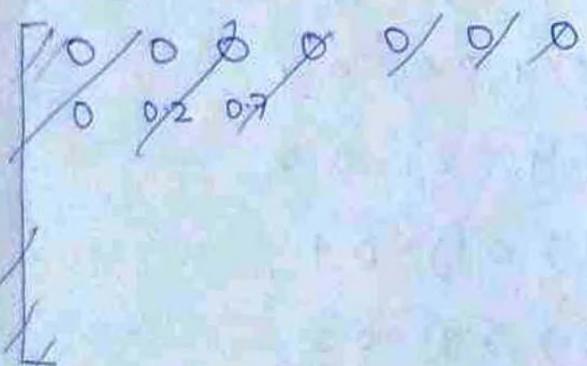
Solution:

3)

$$M_1 = \left\{ \frac{0}{0.72} + \frac{0.8}{0.725} + \frac{1}{0.75} + \frac{0.8}{0.775} + \frac{0}{0.78} \right\}$$

$$M_2 = \left\{ \frac{0}{21K} + \frac{0.2}{22K} + \frac{0.7}{23K} + \frac{1}{24K} + \frac{0.7}{25K} + \frac{0.2}{26K} + \frac{0}{27K} \right\}$$

$$M_1 \times M_2 = \left[ \mu(n_i, y_j) \right]_{i=1 \dots 5, j=1 \dots 7} \quad \begin{matrix} n_i \rightarrow M_1 \\ y_j = M_2 \end{matrix}$$



$$M_1 \times M_2 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.2 & 0.7 & 0.8 & 0.7 & 0.2 & 0 \\ 0 & 0.2 & 0.7 & 1 & 0.7 & 0.2 & 0 \\ 0 & 0.2 & 0.7 & 0.8 & 0.7 & 0.2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Code:

```
import numpy as np
def cartesian_prod_of_two_sets(A,B):
val = []
for i in range(0,len(A)):
for j in range(0,len(B)):
if A[i]<B[j]:
val.append(A[i])
else:
val.append(B[j])
mat = np.array(val).reshape(len(A),len(B))
return mat
def print_fuzzy_set(A,W):
for i in range(0, len(A)-1):
print(f'{round(A[i],2)}/{round(W[i],1)}', end = ' + ')
print(f'{round(A[-1], 2)}/{round(W[-1], 1)})')
A=list(map(float,input('Enter the values of first fuzzy set : ').strip().split()))
B=list(map(float,input('Enter the values of second fuzzy set : ').strip().split()))
W1=list(map(int,input('Enter the weights of each element for the first fuzzy set : ').strip().split()))
W2=list(map(int,input('Enter the weights of each element for the second fuzzy set : ').strip().split()))
print('The first fuzzy set is :')
print_fuzzy_set(A,W1)
print('The second fuzzy set is :')
print_fuzzy_set(B,W2)
print('The cartesian product of first and second fuzzy set is : \n')
R = cartesian_prod_of_two_sets(A,B)
print(f'____|____ {W2[0]}____|____', end = '|')
for i in range(1, len(W2)-1):
```

```
print(f'____{W2[i]}____', end = '|')
print(f'____{W2[-1]}____')
for i in range(0, R.shape[0]-1):
    print(f'{W1[i]} ', end = '|')
    for j in range(0, R.shape[1]):
        print(f'____{R[i][j]}____', end = '|')
    print('\n')
    print(f'{W1[-1]} ', end = '|')
    for i in range(0, R.shape[1]):
        print(f'____{R[-1][i]}____', end = '|')
    print('\n')
```

## Output:

```
sumon@Lenovo: ~/Desktop/Python-Codes/annie_mam_Practical/Practical$ python3 cartesian_product.py
Enter the values of first fuzzy set : 0.0 0.8 1.0 0.8 0.0
Enter the values of second fuzzy set : 0.0 0.2 0.7 1.0 0.7 0.2 0.0
Enter the weights of each element for the first fuzzy set : 0.72 0.725 0.75 0.775 0.78
Enter the weights of each element for the second fuzzy set : 21 22 23 24 25 26 27
The first fuzzy set is :
0.0/0.7 + 0.8/0.7 + 1.0/0.8 + 0.8/0.8 + 0.0/0.8
The second fuzzy set is :
0.0/21.0 + 0.2/22.0 + 0.7/23.0 + 1.0/24.0 + 0.7/25.0 + 0.2/26.0 + 0.0/27.0
The cartesian product of first and second fuzzy set is :

| 21.0 | 22.0 | 23.0 | 24.0 | 25.0 | 26.0 | 27.0 |
0.72 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
0.725 | 0.0 | 0.2 | 0.7 | 0.8 | 0.7 | 0.2 | 0.0 |
0.75 | 0.0 | 0.2 | 0.7 | 1.0 | 0.7 | 0.2 | 0.0 |
0.775 | 0.0 | 0.2 | 0.7 | 0.8 | 0.7 | 0.2 | 0.0 |
0.78 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |

sumon@Lenovo: ~/Desktop/Python-Codes/annie_mam_Practical/Practical$
```

4] Cartesian product of two fuzzy sets and perform max-min composition and max product composition.

$$I_1 = \{0.2/20 + 0.3/40 + 0.6/60 + 0.8/80 + 1.0/100 + 0.2/120\}$$

$$I_2 = \{0.35/500 + 0.67/1000 + 0.97/1500 + 0.25/1800\}$$

Solution:

$$4) I_1 = \left\{ \frac{0.2}{20} + \frac{0.3}{40} + \frac{0.6}{60} + \frac{0.8}{80} + \frac{1.0}{100} + \frac{0.2}{120} \right\}$$

$$I_2 = \left\{ \frac{0.35}{500} + \frac{0.67}{1000} + \frac{0.97}{1500} + \frac{0.25}{1800} \right\}$$

$$I_1 \times I_2 = \begin{bmatrix} \mu(n_i, y_j) \end{bmatrix}_{\substack{i=1 \dots 6 \\ j=1 \dots 4}} \quad \begin{array}{l} n_i \rightarrow I_1 \\ y_j \rightarrow I_2 \end{array}$$

$$I_1 \times I_2 = \begin{bmatrix} 0.2 & 0.2 & 0.2 & 0.2 \\ 0.3 & 0.3 & 0.3 & 0.25 \\ 0.35 & 0.6 & 0.6 & 0.25 \\ 0.35 & 0.67 & 0.8 & 0.25 \\ 0.35 & 0.67 & 0.97 & 0.25 \\ 0.2 & 0.2 & 0.2 & 0.2 \end{bmatrix}$$

Code:

```
import numpy as np
def cartesian_prod_of_two_sets(A,B):
val = []
for i in range(0,len(A)):
for j in range(0,len(B)):
if A[i]<B[j]:
val.append(A[i])
else:
val.append(B[j])
mat = np.array(val).reshape(len(A),len(B))
return mat
def print_fuzzy_set(A,W):
for i in range(0, len(A)-1):
print(f'{round(A[i],2)}/{round(W[i],1)}', end = ' + ')
print(f'{round(A[-1], 2)}/{round(W[-1], 1)})')
A=list(map(float,input('Enter the values of first fuzzy set : ').strip().split()))
B=list(map(float,input('Enter the values of second fuzzy set : ').strip().split()))
W1=list(map(int,input('Enter the weights of each element for the first fuzzy set : ').strip().split()))
W2=list(map(int,input('Enter the weights of each element for the second fuzzy set : ').strip().split()))
print('The first fuzzy set is :')
print_fuzzy_set(A,W1)
print('The second fuzzy set is :')
print_fuzzy_set(B,W2)
print('The cartesian product of first and second fuzzy set is : \n')
R = cartesian_prod_of_two_sets(A,B)
print(f'____|____ {W2[0]}____', end = '|')
for i in range(1, len(W2)-1):
```

```

print(f'____{W2[i]}____', end = '|')
print(f'____{W2[-1]}____')
for i in range(0, R.shape[0]-1):
    print(f'{W1[i]} ', end = '|')
    for j in range(0, R.shape[1]):
        print(f'____{R[i][j]}____', end = '|')
    print('\n')
    print(f'{W1[-1]} ', end = '|')
    for i in range(0, R.shape[1]):
        print(f'____{R[-1][i]}____', end = '|')
    print('\n')

```

## Output:

```

sumon@Lenovo:~/Desktop/Python-Codes/annie_mam_Practical/Practical$ python3 cartesian_product.py
Enter the values of first fuzzy set : 0.2 0.3 0.6 0.8 1.0 0.2
Enter the values of second fuzzy set : 0.35 0.67 0.97 0.25
Enter the weights of each element for the first fuzzy set : 20 40 60 80 100 120
Enter the weights of each element for the second fuzzy set : 500 1000 1500 1800
The first fuzzy set is :
0.2/20.0 + 0.3/40.0 + 0.6/60.0 + 0.8/80.0 + 1.0/100.0 + 0.2/120.0
The second fuzzy set is :
0.35/500.0 + 0.67/1000.0 + 0.97/1500.0 + 0.25/1800.0
The cartesian product of first and second fuzzy set is :
      | 500.0 | 1000.0 | 1500.0 | 1800.0
20.0 | 0.2 | 0.2 | 0.2 | 0.2 |
40.0 | 0.3 | 0.3 | 0.3 | 0.25 |
60.0 | 0.35 | 0.6 | 0.6 | 0.25 |
80.0 | 0.35 | 0.67 | 0.8 | 0.25 |
100.0 | 0.35 | 0.67 | 0.97 | 0.25 |
120.0 | 0.2 | 0.2 | 0.2 | 0.2 |
sumon@Lenovo:~/Desktop/Python-Codes/annie_mam_Practical/Practical$ 

```

5] Cartesian product of two fuzzy sets and perform max-min composition and max product composition.

R = [0.2 0.5 0.7 1 0.9  
0.3 0.5 0.7 1 0.8  
0.4 0.6 0.8 0.9 0.4  
0.9 1 0.8 0.6 0.4]

S = [1 0.8 0.6 0.3 0.1  
0.7 1 0.7 0.5 0.4  
0.5 0.6 1 0.8 0.8  
0.3 0.4 0.6 1 0.9  
0.9 0.3 0.5 0.7 1]

Code:

```
import numpy as np
def min_val(l1,l2):
if l1>l2:
return l2
else:
return l1
C1= int(input('Enter the column number of first matrix : '))
C2= int(input('Enter the column number of second matrix :
'))
R1= int(input('Enter the row number of first matrix : '))
R2= int(input('Enter the row number of second matrix : '))
print("Enter the entries in a single line (separated by
space): ")
entries1 = list(map(float, input('Enter the values of first
matrix :
').strip().split()))
entries2 = list(map(float, input('Enter the values of second
```

```
matrix :  
').strip().split()))  
matrix1 = np.array(entries1).reshape(R1, C1)  
matrix2 = np.array(entries2).reshape(R2, C2)  
min_max=[]  
max_product=[]  
for i in range(0,R1):  
    for j in range(0,C2):  
        A=matrix1[i,:]  
        B=matrix2[:,j]  
        val_min=[]  
        val_product=[]  
        for k in range(0,len(A)):  
            val_min.append(min_val(A[k],B[k]))  
            val_product.append(A[k]*B[k])  
        min_max.append(max(val_min))  
        max_product.append(max(val_product))  
result_max_min=np.array(min_max).reshape(R1, C2)  
result_max_product=np.array(max_product).reshape(R1, C2)  
print('The First Matrix Is : \n',matrix1)  
print('The Second Matrix Is : \n',matrix2)  
print('The Max-Min Composition Is :\n',result_max_min)  
print('The Max-Product Composition Is :\n',result_max_product)
```

## Output:

```
sumon@Lenovo:~/Desktop/Python-Codes/annie_mam_Practical/Practical$ python3 composition.py
Enter the column number of first matrix : 5
Enter the column number of second matrix : 5
Enter the row number of first matrix : 4
Enter the row number of second matrix : 5
Enter the entries in a single line (separated by space):
Enter the values of first matrix : 0.2 0.5 0.7 1 0.9 0.3 0.5 0.7 1 0.8 0.4 0.6 0.8 0.9 0.4 0.9 1 0.8 0.6 0.4
Enter the values of second matrix : 1 0.8 0.6 0.3 0.1 0.7 1 0.7 0.5 0.4 0.5 0.6 1 0.8 0.8 0.3 0.4 0.6 1 0.9 0.9 0.3 0.5 0.7 1
The First Matrix Is :
[[0.2 0.5 0.7 1. 0.9]
 [0.3 0.5 0.7 1. 0.8]
 [0.4 0.6 0.8 0.9 0.4]
 [0.9 1. 0.8 0.6 0.4]]
The Second Matrix Is :
[[1. 0.8 0.6 0.3 0.1]
 [0.7 1. 0.7 0.5 0.4]
 [0.5 0.6 1. 0.8 0.8]
 [0.3 0.4 0.6 1. 0.9]
 [0.9 0.3 0.5 0.7 1. ]]
The Max-Min Composition Is :
[[0.9 0.6 0.7 1. 0.9]
 [0.8 0.6 0.7 1. 0.9]
 [0.6 0.6 0.8 0.9 0.9]
 [0.9 1. 0.8 0.8 0.8]]
The Max-Product Composition Is :
[[0.81 0.5 0.7 1. 0.9 ]
 [0.72 0.5 0.7 1. 0.9 ]
 [0.42 0.6 0.8 0.9 0.81]
sumon@Lenovo:~/Desktop/Python-Codes/annie_mam_Practical/Practical$
```

6] Cartesian product of two fuzzy sets and perform max-min composition and max product composition.

A1 = [0.8 1 0.5 0.1 0 0  
0.1 0.2 0.3 0.2 1 0  
0.1 0.6 0.2 0.7 1 0  
0.1 0.4 0.5 0.8 1 0.9]

A2 = [0.1 0.2 0.5 0.9 0  
0.1 0.2 0.5 0.9 0  
0.1 0.2 0.5 0.9 0  
0.3 0.4 0.7 0.6 1  
0.3 0 0.7 0.6 1  
0.3 0.4 0.7 0.1 1]

Code:

```
import numpy as np
def min_val(l1,l2):
if l1>l2:
return l2
else:
return l1
C1= int(input('Enter the column number of first matrix : '))
C2= int(input('Enter the column number of second matrix : '))
R1= int(input('Enter the row number of first matrix : '))
R2= int(input('Enter the row number of second matrix : '))
print("Enter the entries in a single line (separated by space): ")
entries1 = list(map(float, input('Enter the values of first matrix : ')))
```

```
'.strip().split()))
entries2 = list(map(float, input('Enter the values of second
matrix :
').strip().split()))
matrix1 = np.array(entries1).reshape(R1, C1)
matrix2 = np.array(entries2).reshape(R2, C2)
min_max=[]
max_product=[]
for i in range(0,R1):
for j in range(0,C2):
A=matrix1[i,:]
B=matrix2[:,j]
val_min=[]
val_product=[]
for k in range(0,len(A)):
val_min.append(min_val(A[k],B[k]))
val_product.append(A[k]*B[k])
min_max.append(max(val_min))
max_product.append(max(val_product))
result_max_min=np.array(min_max).reshape(R1, C2)
result_max_product=np.array(max_product).reshape(R1,
C2)
print('The First Matrix Is : \n',matrix1)
print('The Second Matrix Is : \n',matrix2)
print('The Max-Min Composition Is :\n',result_max_min)
print('The Max-Product Composition Is :\n',result_max_product)
```

## Output:

```
sumon@Lenovo:~/Desktop/Python-Codes/annie_mam_Practical/Practical$ python3 composition.py
Enter the column number of first matrix : 6
Enter the column number of second matrix : 5
Enter the row number of first matrix : 4
Enter the row number of second matrix : 6
Enter the entries in a single line (separated by space):
Enter the values of first matrix : 0.8 1 0.5 0.1 0 0 0.1 0.2 0.3 0.2 1 0 0.1 0.6 0.2 0.7 1 0 0.1 0.4 0.5 0.8 1 0.9
Enter the values of second matrix : 0.1 0.2 0.5 0.9 0 0.1 0.2 0.5 0.9 0 0.1 0.2 0.5 0.9 0 0.3 0.4 0.7 0.6 1 0.3 0 0.7 0.6 1 0.3 0.4 0.7 0.1 1
The First Matrix Is :
[[0.8 1. 0.5 0.1 0. 0. ]
 [0.1 0.2 0.3 0.2 1. 0. ]
 [0.1 0.6 0.2 0.7 1. 0. ]
 [0.1 0.4 0.5 0.8 1. 0.9]]
The Second Matrix Is :
[[0.1 0.2 0.5 0.9 0. ]
 [0.1 0.2 0.5 0.9 0. ]
 [0.1 0.2 0.5 0.9 0. ]
 [0.3 0.4 0.7 0.6 1. ]
 [0.3 0. 0.7 0.6 1. ]
 [0.3 0.4 0.7 0.1 1. ]]
The Max-Min Composition Is :
[[0.1 0.2 0.5 0.9 0.1]
 [0.3 0.2 0.7 0.6 1. ]
 [0.3 0.4 0.7 0.6 1. ]
 [0.3 0.4 0.7 0.6 1. ]]
The Max-Product Composition Is :
[[0.1 0.2 0.5 0.9 0.1]
 [0.3 0.08 0.7 0.6 1. ]
 [0.3 0.28 0.7 0.6 1. ]
 [0.3 0.36 0.7 0.6 1. ]]

sumon@Lenovo:~/Desktop/Python-Codes/annie_mam_Practical/Practical$
```

# Practical-9

## Defuzzification

1] Consider two Fuzzy sets A and B , both defined on x given

$\mu(x)$	X1	X2	X3	X4	X5
A	0.2	0.3	0.4	0.7	0.1
B	0.4	0.5	0.6	0.8	0.9

The following lambda-cut sets

- a)  $(\bar{A})_{0.7}$  b)  $(B)_{0.2}$
- c)  $(A \cup B)_{0.6}$  d)  $(A \cap B)_{0.5}$
- e)  $(A \cup \bar{A})_{0.7}$  f)  $(B \cap c B)_{0.3}$
- g)  $c(A \cap B)_{0.6}$  h)  $(\bar{A} \cup c B)_{0.8}$

Solution:

$$1) \underline{A} = \left\{ \frac{0.2}{n_1} + \frac{0.3}{n_2} + \frac{0.4}{n_3} + \frac{0.7}{n_4} + \frac{0.1}{n_5} \right\}$$

$$\underline{B} = \left\{ \frac{0.4}{n_1} + \frac{0.5}{n_2} + \frac{0.6}{n_3} + \frac{0.8}{n_4} + \frac{0.9}{n_5} \right\}$$

$$A_1 = \{ n / \mu_A(n) \geq 1 \}$$

$$a) (\bar{A})_{0.7} = \left\{ \frac{0.8}{n_1} + \frac{0.7}{n_2} + \frac{0.6}{n_3} + \frac{0.3}{n_4} + \frac{0.9}{n_5} \right\}_{0.7}$$

$$\Rightarrow \left\{ \frac{0.8}{n_1} + \frac{0.7}{n_2} + \frac{0.9}{n_5} \right\}$$

$$b) (\bar{B})_{0.2} = \left\{ \frac{0.4}{n_1} + \frac{0.5}{n_2} + \frac{0.6}{n_3} + \frac{0.8}{n_4} + \frac{0.9}{n_5} \right\}$$

$$c) (\underline{A} \cup \underline{B})_{0.6} = \max(\mu_A(n), \mu_B(n))$$

$$= \left\{ \frac{0.4}{n_1} + \frac{0.5}{n_2} + \frac{0.6}{n_3} + \frac{0.8}{n_4} + \frac{0.9}{n_5} \right\}_{0.6}$$

$$\Rightarrow \left\{ \frac{0.6}{n_3} + \frac{0.8}{n_4} + \frac{0.9}{n_5} \right\}$$

$$d) (\underline{A} \cap \underline{B})_{0.5} = \min(\mu_A(n), \mu_B(n))$$

$$= \left\{ \frac{0.2}{n_1} + \frac{0.3}{n_2} + \frac{0.4}{n_3} + \frac{0.7}{n_4} + \frac{0.1}{n_5} \right\}_{0.5}$$

$$= \left\{ \frac{0.7}{n_4} \right\}$$

$$e) (\underline{A} \cup \bar{A})_{0.7} = \max(\mu_A(n), \overline{\mu_A}(n))$$

$$= \left\{ \frac{0.8}{n_1} + \frac{0.7}{n_2} + \frac{0.6}{n_3} + \frac{0.7}{n_4} + \frac{0.9}{n_5} \right\}_{0.7}$$

$$\Rightarrow \left\{ \frac{0.8}{n_1} + \frac{0.7}{n_2} + \frac{0.7}{n_4} + \frac{0.9}{n_5} \right\}$$

$$\text{ii) } (\underline{B} \cap \underline{B})_{0.5} = \left\{ \frac{0.4}{n_1}, \frac{0.5}{n_2}, \frac{0.4}{n_3}, \frac{0.2}{n_4}, \frac{0.1}{n_5} \right\}_{0.5}$$

$$= \left\{ \frac{0.4}{n_1}, \frac{0.5}{n_2}, \frac{0.4}{n_3} \right\}$$

$$\text{iii) } (\widehat{B} \cap \widehat{B})_{0.6} = \left\{ \frac{0.8}{n_1}, \frac{0.7}{n_2}, \frac{0.6}{n_3}, \frac{0.3}{n_4}, \frac{0.1}{n_5} \right\}_{0.6}$$

$$= \left\{ \frac{0.8}{n_1}, \frac{0.7}{n_2}, \frac{0.6}{n_3}, \frac{0.9}{n_5} \right\}$$

$$\text{iv) } (\overline{A} \cup \overline{B})_{0.8} = \left\{ \frac{0.8}{n_1}, \frac{0.7}{n_2}, \frac{0.6}{n_3}, \frac{0.3}{n_4}, \frac{0.9}{n_5} \right\}_{0.6}$$

$$= \left\{ \frac{0.8}{n_1}, \frac{0.9}{n_5} \right\}$$

Code:

```
def union(A,B):
R = []
for i in range(0, len(A)):
if (A[i]>B[i]):
R.append(A[i])
else:
R.append(B[i])
return R
def intersect(A,B):
R = []
for i in range(0, len(A)):
if (A[i]<B[i]):
R.append(A[i])
else:
R.append(B[i])
return R
def complement(A):
R = []
for i in range(0, len(A)):
V = 1- A[i]
R.append(V)
return R
def diff(A,B):
COM_B = complement(B)
R = []
for i in range(0, len(A)):
if (A[i]< COM_B[i]):
R.append(A[i])
else:
R.append(COM_B[i])
return R
def print_fuzzy_set(A,W):
for i in range(0, len(A)-1):
print(f'{round(A[i], 2)}/{round(W[i], 1)}', end = " + ")
```

```
print(f'{round(A[-1], 2})/ {round(W[-1], 1)})')
OP =
OP=float(input('1.Union\n2.Intersection\n3.Complement\
n4.Difference\n5.No
rmal-Transformation\nEnter the operation which need to be
performed from
above:'))
if (OP==1):
A=list(map(float,input('Enter the values of first fuzzy set :
').strip().split()))
B=list(map(float,input('Enter the values of second fuzzy set
: ').strip().split()))
W=list(map(int,input('Enter the weights of each element for
the fuzzy set :
').strip().split()))
lam=float(input('Enter the lamdha value : '))
R=union(A,B)
R=[x for x in R if x>lam]
print('The union of first and second set is :')
print_fuzzy_set(R,W)
elif (OP==2):
A=list(map(float,input('Enter the values of first fuzzy set :
').strip().split()))
B=list(map(float,input('Enter the values of second fuzzy set
: ').strip().split()))
W=list(map(int,input('Enter the weights of each element for
the fuzzy set :
').strip().split()))
lam=float(input('Enter the lamdha value : '))
R=intersect(A,B)
R=[x for x in R if x>lam]
print('The intersect of first and second set is :')
print_fuzzy_set(R,W)
elif (OP==3):
A=list(map(float,input('Enter the values of fuzzy set :
').strip().split()))
```

```
W=list(map(int,input('Enter the weights of each element for  
the fuzzy set :  
lam=float(input('Enter the lamdha value : '))  
R=complement(A)  
R=[x for x in R if x>lam]  
print('The complement of first fuzzy set is :')  
print_fuzzy_set(R,W)  
elif (OP==4):  
A=list(map(float,input('Enter the values of first fuzzy set :  
B=list(map(float,input('Enter the values of second fuzzy set :  
W=list(map(int,input('Enter the weights of each element for  
the fuzzy set :  
lam=float(input('Enter the lamdha value : '))  
R=diff(A,B)  
R=[x for x in R if x>lam]  
print('The differnce of first and second set is :')  
print_fuzzy_set(R,W)  
elif (OP==5):  
A=list(map(float,input('Enter the values of fuzzy set :  
W=list(input('Enter the weights of each element for the  
fuzzy set :  
lam=float(input('Enter the lamdha value : '))  
R=[x for x in A if x>lam]  
for i in range(0,len(R)-1):  
print(f'{round(R[i],2)}/{W[i]}',end=' + ')  
print(f'{round(R[-1],2)}/{W[-1]}')  
else:  
print('Please select any valid number : 1,2,3,4')  
exit()
```

## Output:

```
sumon@Lenovo:~/Desktop/Python-Codes/annie_mam_Practical/Practical$ python3 Alpha-cut.py
1.Union
2.Intersection
3.Complement
4.Difference
5.Normal-Transformation
Enter the operation which need to be performed from above:3
Enter the values of fuzzy set : 0.2 0.3 0.4 0.7 0.1
Enter the weights of each element for the fuzzy set : 1 2 3 4 5
Enter the lamdha value : 0.7
The complement of first fuzzy set is :
0.8/3 + 0.9/5
sumon@Lenovo:~/Desktop/Python-Codes/annie_mam_Practical/Practical$ python3 Alpha-cut.py
1.Union
2.Intersection
3.Complement
4.Difference
5.Normal-Transformation
Enter the operation which need to be performed from above:3
Enter the values of fuzzy set : 0.4 0.5 0.6 0.8 0.9
Enter the weights of each element for the fuzzy set : 1 2 3 4 5
Enter the lamdha value : 0.2
The complement of first fuzzy set is :
sumon@Lenovo:~/Desktop/Python-Codes/annie_mam_Practical/Practical$
```

```
sumon@Lenovo:~/Desktop/Python-Codes/annie_mam_Practical/Practical$ python3 Alpha-cut.py
1.Union
2.Intersection
3.Complement
4.Difference
5.Normal-Transformation
Enter the operation which need to be performed from above:1
Enter the values of first fuzzy set : 0.2 0.3 0.4 0.7 0.1
Enter the values of second fuzzy set : 0.4 0.5 0.6 0.8 0.9
Enter the weights of each element for the fuzzy set : 1 2 3 4 5
Enter the lamdha value : 0.6
The union of first and second set is :
0.8/1 + 0.9/5
sumon@Lenovo:~/Desktop/Python-Codes/annie_mam_Practical/Practical$ python3 Alpha-cut.py
1.Union
2.Intersection
3.Complement
4.Difference
5.Normal-Transformation
Enter the operation which need to be performed from above:2
Enter the values of first fuzzy set : 0.2 0.3 0.4 0.7 0.1
Enter the values of second fuzzy set : 0.4 0.5 0.6 0.8 0.9
Enter the weights of each element for the fuzzy set : 1 2 3 4 5
Enter the lamdha value : 0.5
The intersect of first and second set is :
sumon@Lenovo:~/Desktop/Python-Codes/annie_mam_Practical/Practical$
```

```
sumon@Lenovo:~/Desktop/Python-Codes/annie_mam_Practical/Practical$ python3 Alpha-cut.py
1.Union
2.Intersection
3.Complement
4.Difference
5.Normal-Transformation
Enter the operation which need to be performed from above:1
Enter the values of first fuzzy set : 0.2 0.3 0.4 0.7 0.1
Enter the values of second fuzzy set : 0.8 0.7 0.6 0.3 0.9
Enter the weights of each element for the fuzzy set : 1 2 3 4 5
Enter the lamdha value : 0.7
The union of first and second set is :
sumon@Lenovo:~/Desktop/Python-Codes/annie_mam$ sumon@Lenovo:~/Desktop/Python-Codes/annie_mam_Practical/Practical$ python3 Alpha-cut.py
1.Union
2.Intersection
3.Complement
4.Difference
5.Normal-Transformation
Enter the operation which need to be performed from above:2
Enter the values of first fuzzy set : 0.4 0.5 0.6 0.8 0.9
Enter the values of second fuzzy set : 0.6 0.5 0.4 0.2 0.1
Enter the weights of each element for the fuzzy set : 1 2 3 4 5
Enter the lamdha value : 0.3
The intersect of first and second set is :
sumon@Lenovo:~/Desktop/Python-Codes/annie_mam_Practical/Practical$
```

```
sumon@Lenovo:~/Desktop/Python-Codes/annie_mam_Practical/Practical$ python3 Alpha-cut.py
1.Union
2.Intersection
3.Complement
4.Difference
5.Normal-Transformation
Enter the operation which need to be performed from above:3
Enter the values of fuzzy set : 0.2 0.3 0.4 0.7 0.1
Enter the weights of each element for the fuzzy set : 1 2 3 4 5
Enter the lamdha value : 0.6
The complement of first fuzzy set is :
0.8/1 + 0.7/2 + 0.9/5
sumon@Lenovo:~/Desktop/Python-Codes/annie_mam_Practical/Practical$ python3 Alpha-cut.py
1.Union
2.Intersection
3.Complement
4.Difference
5.Normal-Transformation
Enter the operation which need to be performed from above:1
Enter the values of first fuzzy set : 0.8 0.7 0.6 0.3 0.9
Enter the values of second fuzzy set : 0.6 0.5 0.4 0.2 0.1
Enter the weights of each element for the fuzzy set : 1 2 3 4 5
Enter the lamdha value : 0.8
The union of first and second set is :
0.9/5
sumon@Lenovo:~/Desktop/Python-Codes/annie_mam_Practical/Practical$
```

2] Using the Zadeh's notation determine the lambda-cut sets for the given fuzzy sets

$$S_1 = \{0/0 + 0.5/20 + 0.65/40 + 0.85/60 + 1.0/80 + 1.0/100\}$$

$$S_2 = \{0/0 + 0.45/20 + 0.6/40 + 0.8/60 + 0.95/80 + 1.0/100\}$$

The following for lamda = 0.5

- a)  $(S_1 \cup S_2)$
- b)  $(S_1 \cap S_2)$
- c)  $c(S_1)$
- d)  $c(S_2)$
- e)  $c(S_1 \cup S_2)$
- f)  $c(S_1 \cap S_2)$

Solution:

7)  $(B \cap \bar{B})_{0.5} = \left\{ \frac{0.4}{n_1}, \frac{0.2}{n_2}, \frac{0.1}{n_3}, \frac{0.2}{n_4}, \frac{0.1}{n_5} \right\}_{0.5}$   
 $\cdot \left\{ \frac{0.4}{n_1}, \frac{0.2}{n_2}, \frac{0.1}{n_3} \right\}$

8)  $(\bar{B} \cap B)_{0.5} = \left\{ \frac{0.8}{n_1}, \frac{0.7}{n_2}, \frac{0.6}{n_3}, \frac{0.3}{n_4}, \frac{0.4}{n_5} \right\}_{0.5}$   
 $\cdot \left\{ \frac{0.8}{n_1}, \frac{0.7}{n_2}, \frac{0.6}{n_3}, \frac{0.4}{n_5} \right\}$

9)  $(\bar{B} \cup \bar{B})_{0.5} = \left\{ \frac{0.8}{n_1}, \frac{0.7}{n_2}, \frac{0.6}{n_3}, \frac{0.3}{n_4}, \frac{0.9}{n_5} \right\}_{0.5}$   
 $\cdot \left\{ \frac{0.8}{n_1}, \frac{0.9}{n_5} \right\}$

2)  $S_1 = \left\{ \frac{0}{0}, \frac{0.5}{20}, \frac{0.65}{40}, \frac{0.85}{60}, \frac{1.0}{80}, \frac{1.0}{100} \right\}$   
 $S_2 = \left\{ \frac{0}{0}, \frac{0.45}{20}, \frac{0.6}{40}, \frac{0.8}{60}, \frac{0.95}{80}, \frac{1.0}{100} \right\}$

a)  $(S_1 \cup S_2)_{0.5} = \left\{ \frac{0}{0} + \frac{0.5}{20} + \frac{0.65}{40} + \frac{0.85}{60} + \frac{1.0}{80} + \frac{1.0}{100} \right\}_{0.5}$   
 $\cdot \left\{ \frac{0.5}{20} + \frac{0.65}{40} + \frac{0.85}{60} + \frac{1.0}{80} + \frac{1.0}{100} \right\}$

b)  $(S_1 \cap S_2)_{0.5} = \left\{ \frac{0}{0} + \frac{0.45}{20} + \frac{0.6}{40} + \frac{0.8}{60} + \frac{0.95}{80} + \frac{1.0}{100} \right\}_{0.5}$   
 $\cdot \left\{ \frac{0.6}{40} + \frac{0.8}{60} + \frac{0.95}{80} + \frac{1.0}{100} \right\}$

c)  $(\bar{S}_1)_{0.5} = 1 - \mu_{S_1}(n) = \left\{ \frac{1}{0} + \frac{0.5}{20} + \frac{0.35}{40} + \frac{0.15}{60} + \frac{0}{80} + \frac{0}{100} \right\}_{0.5}$   
 $\cdot \left\{ \frac{0.5}{20} + \frac{1}{0} \right\}$

d)  $(\bar{S}_2)_{0.5} = \left\{ \frac{1}{0} + \frac{0.55}{20} + \frac{0.4}{40} + \frac{0.2}{60} + \frac{0.05}{80} + \frac{0}{100} \right\}_{0.5}$   
 $\cdot \left\{ \frac{1}{0} + \frac{0.55}{20} \right\}$

$$f) (\underline{S_1} \cap \underline{S_2})_{0.5} = \left\{ \frac{1}{0} + \frac{0.55}{20} + \frac{0.4}{40} + \frac{0.2}{60} + \frac{0.05}{80} + \frac{0}{100} \right\}_{0.5}$$
$$= \left\{ \frac{1}{0} + \frac{0.55}{20} \right\}$$

$$e) (\underline{S_1} \cup \underline{S_2})_{0.5} = \left\{ \frac{1}{0} + \frac{0.5}{20} + \frac{0.35}{40} + \frac{0.15}{60} + \frac{0}{80} + \frac{0}{100} \right\}_{0.5}$$
$$= \left\{ \frac{1}{0} + \frac{0.5}{20} \right\}$$

Code:

```
def union(A,B):
R = []
for i in range(0, len(A)):
if (A[i]>B[i]):
R.append(A[i])
else:
R.append(B[i])
return R
def intersect(A,B):
R = []
for i in range(0, len(A)):
if (A[i]<B[i]):
R.append(A[i])
else:
R.append(B[i])
return R
def complement(A):
R = []
for i in range(0, len(A)):
V = 1- A[i]
R.append(V)
return R
def diff(A,B):
COM_B = complement(B)
R = []
for i in range(0, len(A)):
if (A[i]< COM_B[i]):
R.append(A[i])
else:
R.append(COM_B[i])
return R
def print_fuzzy_set(A,W):
for i in range(0, len(A)-1):
print(f'{round(A[i], 2)}/{round(W[i], 1)}', end = " + ")
```

```
print(f'{round(A[-1], 2})/ {round(W[-1], 1)})')
OP =
OP=float(input('1.Union\n2.Intersection\n3.Complement\
n4.Difference\n5.No
rmal-Transformation\nEnter the operation which need to be
performed from
above:'))
if (OP==1):
A=list(map(float,input('Enter the values of first fuzzy set :
').strip().split()))
B=list(map(float,input('Enter the values of second fuzzy set
: ').strip().split()))
W=list(map(int,input('Enter the weights of each element for
the fuzzy set :
').strip().split()))
lam=float(input('Enter the lamdha value : '))
R=union(A,B)
R=[x for x in R if x>lam]
print('The union of first and second set is :')
print_fuzzy_set(R,W)
elif (OP==2):
A=list(map(float,input('Enter the values of first fuzzy set :
').strip().split()))
B=list(map(float,input('Enter the values of second fuzzy set
: ').strip().split()))
W=list(map(int,input('Enter the weights of each element for
the fuzzy set :
').strip().split()))
lam=float(input('Enter the lamdha value : '))
R=intersect(A,B)
R=[x for x in R if x>lam]
print('The intersect of first and second set is :')
print_fuzzy_set(R,W)
elif (OP==3):
A=list(map(float,input('Enter the values of fuzzy set :
').strip().split()))
```

```
W=list(map(int,input('Enter the weights of each element for  
the fuzzy set :  
lam=float(input('Enter the lamdha value : '))  
R=complement(A)  
R=[x for x in R if x>lam]  
print('The complement of first fuzzy set is :')  
print_fuzzy_set(R,W)  
elif (OP==4):  
A=list(map(float,input('Enter the values of first fuzzy set :  
B=list(map(float,input('Enter the values of second fuzzy set :  
W=list(map(int,input('Enter the weights of each element for  
the fuzzy set :  
lam=float(input('Enter the lamdha value : '))  
R=diff(A,B)  
R=[x for x in R if x>lam]  
print('The differnce of first and second set is :')  
print_fuzzy_set(R,W)  
elif (OP==5):  
A=list(map(float,input('Enter the values of fuzzy set :  
W=list(input('Enter the weights of each element for the  
fuzzy set :  
lam=float(input('Enter the lamdha value : '))  
R=[x for x in A if x>lam]  
for i in range(0,len(R)-1):  
print(f'{round(R[i],2)}/{W[i]}',end=' + ')  
print(f'{round(R[-1],2)}/{W[-1]}')  
else:  
print('Please select any valid number : 1,2,3,4')  
exit()
```

## Output:

```
sumon@Lenovo:~/Desktop/Python-Codes/annie_mam_Practical/Practical$ python3 Alpha-cut.py
1.Union
2.Intersection
3.Complement
4.Difference
5.Normal-Transformation
Enter the operation which need to be performed from above:1
Enter the values of first fuzzy set : 0 0.5 0.65 0.85 1.0 1.0
Enter the values of second fuzzy set : 0 0.45 0.6 0.8 0.95 1.0
Enter the weights of each element for the fuzzy set : 0 20 40 60 80 100
Enter the lamdha value : 0.5
The union of first and second set is :
0.65/0 + 0.85/20 + 1.0/40 + 1.0/100
sumon@Lenovo:~/Desktop/Python-Codes/annie_mam_Practical/Practical$ python3 Alpha-cut.py
1.Union
2.Intersection
3.Complement
4.Difference
5.Normal-Transformation
Enter the operation which need to be performed from above:2
Enter the values of first fuzzy set : 0 0.5 0.65 0.85 1.0 1.0
Enter the values of second fuzzy set : 0 0.45 0.6 0.8 0.95 1.0
Enter the weights of each element for the fuzzy set : 0 20 40 60 80 100
Enter the lamdha value : 0.5
The intersect of first and second set is :
0.6/0 + 0.8/20 + 0.95/40 + 1.0/100
sumon@Lenovo:~/Desktop/Python-Codes/annie_mam_Practical/Practical$
```

```
sumon@Lenovo:~/Desktop/Python-Codes/annie_mam_Practical/Practical$ python3 Alpha-cut.py
1.Union
2.Intersection
3.Complement
4.Difference
5.Normal-Transformation
Enter the operation which need to be performed from above:3
Enter the values of fuzzy set : 0 0.5 0.65 0.85 1.0 1.0
Enter the weights of each element for the fuzzy set : 0 20 40 60 80 100
Enter the lamdha value : 0.5
The complement of first fuzzy set is :
1.0/100
sumon@Lenovo:~/Desktop/Python-Codes/annie_mam_Practical/Practical$ python3 Alpha-cut.py
1.Union
2.Intersection
3.Complement
4.Difference
5.Normal-Transformation
Enter the operation which need to be performed from above:3
Enter the values of fuzzy set : 0 0.45 0.6 0.8 0.95 1.0
Enter the weights of each element for the fuzzy set : 0 20 40 60 80 100
Enter the lamdha value : 0.5
The complement of first fuzzy set is :
1.0/0 + 0.55/100
sumon@Lenovo:~/Desktop/Python-Codes/annie_mam_Practical/Practical$
```

```
sumon@Lenovo:~/Desktop/Python-Codes/annie_mam_Practical/Practical$ python3 Alpha-cut.py
1.Union
2.Intersection
3.Complement
4.Difference
5.Normal-Transformation
Enter the operation which need to be performed from above:3
Enter the values of fuzzy set : 0 0.5 0.65 0.85 1.0 1.0
Enter the weights of each element for the fuzzy set : 0 20 40 60 80 100
Enter the lamdha value : 0.5
The complement of first fuzzy set is :
1.0/100
sumon@Lenovo:~/Desktop/Python-Codes/annie_mam_Practical/Practical$ python3 Alpha-cut.py
1.Union
2.Intersection
3.Complement
4.Difference
5.Normal-Transformation
Enter the operation which need to be performed from above:3
Enter the values of fuzzy set : 0 0.45 0.6 0.8 0.95 1.0
Enter the weights of each element for the fuzzy set : 0 20 40 60 80 100
Enter the lamdha value : 0.5
The complement of first fuzzy set is :
1.0/0 + 0.55/100
sumon@Lenovo:~/Desktop/Python-Codes/annie_mam_Practical/Practical$
```

3] Consider the two fuzzy sets

$$A = \{0/0.2 + 0.8/0.4 + 1/0.6\}$$

$$B = \{0.9/0.2 + 0.7/0.4 + 0.3/0.6\}$$

The fuzzy sets into lambda-cut sets for lamda=0.4 and lamda=0.7 for the following operations.

- a)  $\bar{A}$
- b)  $c(B)$
- c)  $A \cup B$
- d)  $A \cap B$
- e)  $\bar{A} \cup c(B)$
- f)  $\bar{A} \cap c(B)$

Solution:

$$3) \underline{A} = \left\{ \frac{0}{0.2} + \frac{0.8}{0.4} + \frac{1}{0.6} \right\}, \underline{B} = \left\{ \frac{0.9}{0.2} + \frac{0.7}{0.4} + \frac{0.3}{0.6} \right\}$$

$$\text{a)} \bar{\underline{A}} = 1 - \mu_A(x) = \left\{ \frac{1}{0.2} + \frac{0.2}{0.4} + \frac{0}{0.6} \right\}$$

$$\bar{A}_{0.4} = \left\{ \frac{1}{0.2} \right\}, \bar{A}_{0.7} = \left\{ \frac{1}{0.2} \right\}$$

$$\text{b)} \bar{\underline{B}} = 1 - \mu_B(x) = \left\{ \frac{0.1}{0.2} + \frac{0.3}{0.4} + \frac{0.7}{0.6} \right\}$$

$$\bar{B}_{0.4} = \left\{ \frac{0.7}{0.6} \right\}, \bar{B}_{0.7} = \left\{ \frac{0.7}{0.6} \right\}$$

$$\text{c)} \underline{A} \cup \underline{B} = \left\{ \frac{0.9}{0.2} + \frac{0.8}{0.4} + \frac{1}{0.6} \right\}$$

$$(\underline{A} \cup \underline{B})_{0.4} = \left\{ \frac{0.9}{0.2} + \frac{0.8}{0.4} \right\} + \frac{1}{0.6} \}$$

$$(\underline{A} \cup \underline{B})_{0.7} = \left\{ \frac{0.9}{0.2} + \frac{0.8}{0.4} + \frac{1}{0.6} \right\}$$

$$\text{d)} \underline{A} \cap \underline{B} = \left\{ \frac{0}{0.2} + \frac{0.7}{0.4} + \frac{0.3}{0.6} \right\}$$

$$(\underline{A} \cap \underline{B})_{0.4} = \left\{ \frac{0.7}{0.4} \right\}, (\underline{A} \cap \underline{B})_{0.7} = \left\{ \frac{0.7}{0.4} \right\}$$

$$\text{e)} \bar{\underline{A}} \cup \bar{\underline{B}} = \left\{ \frac{1}{0.2} + \frac{0.3}{0.4} + \frac{0.7}{0.6} \right\}$$

$$(\bar{\underline{A}} \cup \bar{\underline{B}})_{0.4} = \left\{ \frac{1}{0.2} + \frac{0.7}{0.6} \right\}, (\bar{\underline{A}} \cup \bar{\underline{B}})_{0.7} = \left\{ \frac{1}{0.2} + \frac{0.7}{0.6} \right\}$$

$$\bar{A} \cap \bar{B} = \left\{ \frac{0.1}{0.2}, \frac{0.2}{0.4}, \frac{0.6}{0.6} \right\}$$
$$(\bar{A} \cap \bar{B})_{0.4} = \{3\}, (\bar{A} \cap \bar{B})_{0.9} = \{3\}$$

Code:

```
def union(A,B):
R = []
for i in range(0, len(A)):
if (A[i]>B[i]):
R.append(A[i])
else:
R.append(B[i])
return R
def intersect(A,B):
R = []
for i in range(0, len(A)):
if (A[i]<B[i]):
R.append(A[i])
else:
R.append(B[i])
return R
def complement(A):
R = []
for i in range(0, len(A)):
V = 1- A[i]
R.append(V)
return R
def diff(A,B):
COM_B = complement(B)
R = []
for i in range(0, len(A)):
if (A[i]< COM_B[i]):
R.append(A[i])
else:
R.append(COM_B[i])
return R
def print_fuzzy_set(A,W):
for i in range(0, len(A)-1):
print(f'{round(A[i], 2)}/{round(W[i], 1)}', end = " + ")
```

```
print(f'{round(A[-1], 2})/ {round(W[-1], 1)})')
OP =
OP=float(input('1.Union\n2.Intersection\n3.Complement\
n4.Difference\n5.No
rmal-Transformation\nEnter the operation which need to be
performed from
above:'))
if (OP==1):
A=list(map(float,input('Enter the values of first fuzzy set :
').strip().split()))
B=list(map(float,input('Enter the values of second fuzzy set
: ').strip().split()))
W=list(map(int,input('Enter the weights of each element for
the fuzzy set :
').strip().split()))
lam=float(input('Enter the lamdha value : '))
R=union(A,B)
R=[x for x in R if x>lam]
print('The union of first and second set is :')
print_fuzzy_set(R,W)
elif (OP==2):
A=list(map(float,input('Enter the values of first fuzzy set :
').strip().split()))
B=list(map(float,input('Enter the values of second fuzzy set
: ').strip().split()))
W=list(map(int,input('Enter the weights of each element for
the fuzzy set :
').strip().split()))
lam=float(input('Enter the lamdha value : '))
R=intersect(A,B)
R=[x for x in R if x>lam]
print('The intersect of first and second set is :')
print_fuzzy_set(R,W)
elif (OP==3):
A=list(map(float,input('Enter the values of fuzzy set :
').strip().split()))
```

```
W=list(map(int,input('Enter the weights of each element for  
the fuzzy set :  
lam=float(input('Enter the lamdha value : '))  
R=complement(A)  
R=[x for x in R if x>lam]  
print('The complement of first fuzzy set is :')  
print_fuzzy_set(R,W)  
elif (OP==4):  
A=list(map(float,input('Enter the values of first fuzzy set :  
B=list(map(float,input('Enter the values of second fuzzy set :  
W=list(map(int,input('Enter the weights of each element for  
the fuzzy set :  
lam=float(input('Enter the lamdha value : '))  
R=diff(A,B)  
R=[x for x in R if x>lam]  
print('The differnce of first and second set is :')  
print_fuzzy_set(R,W)  
elif (OP==5):  
A=list(map(float,input('Enter the values of fuzzy set :  
W=list(input('Enter the weights of each element for the  
fuzzy set :  
lam=float(input('Enter the lamdha value : '))  
R=[x for x in A if x>lam]  
for i in range(0,len(R)-1):  
print(f'{round(R[i],2)}/{W[i]}',end=' + ')  
print(f'{round(R[-1],2)}/{W[-1]}')  
else:  
print('Please select any valid number : 1,2,3,4')  
exit()
```

## Output:

```
sumon@Lenovo:~/Desktop/Python-Codes/annie_mam_Practical/Practical$ python3 Alpha-cut.py
1.Union
2.Intersection
3.Complement
4.Difference
5.Normal-Transformation
Enter the operation which need to be performed from above:3
Enter the values of fuzzy set : 0.0 0.8 1
Enter the weights of each element for the fuzzy set : 2 4 6
Enter the lamdha value : 0.4
The complement of first fuzzy set is :
1.0/6
sumon@Lenovo:~/Desktop/Python-Codes/annie_mam_Practical/Practical$ python3 Alpha-cut.py
1.Union
2.Intersection
3.Complement
4.Difference
5.Normal-Transformation
Enter the operation which need to be performed from above:3
Enter the values of fuzzy set : 0.0 0.8 1
Enter the weights of each element for the fuzzy set : 2 4 6
Enter the lamdha value : 0.7
The complement of first fuzzy set is :
sumon@Lenovo:~/Desktop/Python-Codes/annie_mam_Practical/Practical$
```

```
sumon@Lenovo:~/Desktop/Python-Codes/annie_mam_Practical/Practical$ python3 Alpha-cut.py
1.Union
2.Intersection
3.Complement
4.Difference
5.Normal-Transformation
Enter the operation which need to be performed from above:3
Enter the values of fuzzy set : 0.9 0.7 0.3
Enter the weights of each element for the fuzzy set : 2 4 6
Enter the lamdha value : 0.4
The complement of first fuzzy set is :
0.7/6
```

```
sumon@Lenovo:~/Desktop/Python-Codes/annie_mam_Practical/Practical$ python3 Alpha-cut.py
1.Union
2.Intersection
3.Complement
4.Difference
5.Normal-Transformation
Enter the operation which need to be performed from above:1
Enter the values of first fuzzy set : 0.0 0.8 1.0
Enter the values of second fuzzy set : 0.9 0.7 0.3
Enter the weights of each element for the fuzzy set : 2 4 6
Enter the lamdha value : 0.4
The union of first and second set is :
0.9/2 + 0.8/4 + 1.0/6
sumon@Lenovo:~/Desktop/Python-Codes/annie_mam_Practical/Practical$ python3 Alpha-cut.py
1.Union
2.Intersection
3.Complement
4.Difference
5.Normal-Transformation
Enter the operation which need to be performed from above:1
Enter the values of first fuzzy set : 0.0 0.8 1.0
Enter the values of second fuzzy set : 0.9 0.7 0.3
Enter the weights of each element for the fuzzy set : 2 4 6
Enter the lamdha value : 0.7
The union of first and second set is :
sumon@Lenovo:~/Desktop/Python-Codes/annie_mam_Practical/Practical$
```

```
sumon@Lenovo:~/Desktop/Python-Codes/annie_mam_Practical/Practical$ python3 Alpha-cut.py
1.Union
2.Intersection
3.Complement
4.Difference
5.Normal-Transformation
Enter the operation which need to be performed from above:2
Enter the values of first fuzzy set : 0.0 0.8 1.0
Enter the values of second fuzzy set : 0.9 0.7 0.3
Enter the weights of each element for the fuzzy set : 2 4 6
Enter the lamdha value : 0.4
The intersect of first and second set is :
sumon@Lenovo:~/Desktop/Python-Codes/annie_mam_Practical/Practical$
```

```
sumon@Lenovo:~/Desktop/Python-Codes/annie_mam_Practical/Practical$ python3 Alpha-cut.py
1.Union
2.Intersection
3.Complement
4.Difference
5.Normal-Transformation
Enter the operation which need to be performed from above:1
Enter the values of first fuzzy set : 1.0 0.2 0.0
Enter the values of second fuzzy set : 0.1 0.3 0.7
Enter the weights of each element for the fuzzy set : 2 4 6
Enter the lamdha value : 0.4
The union of first and second set is :
1.0/2 + 0.7/6
sumon@Lenovo:~/Desktop/Python-Codes/annie_mam_Practical/Practical$ python3 Alpha-cut.py
1.Union
2.Intersection
3.Complement
4.Difference
5.Normal-Transformation
Enter the operation which need to be performed from above:1
Enter the values of first fuzzy set : 1.0 0.2 0.0
Enter the values of second fuzzy set : 0.1 0.3 0.7
Enter the weights of each element for the fuzzy set : 2 4 6
Enter the lamdha value : 0.7
The union of first and second set is :
sumon@Lenovo:~/Desktop/Python-Codes/annie_mam_Practical/Practical$
```

4] Consider the discrete fuzzy set defined on the universe X = {a,b,c,d,e}

$$A = \{1/a + 0.9/b + 0.6/c + 0.3/d + 0/e\}$$

Using Zadeh's notation find the Alpha-cut sets for alpha=1,0.9,0.6,0.3,0+, , 0

Solution:

$$\text{4) } A = \left\{ \frac{1}{a} + \frac{0.9}{b} + \frac{0.6}{c} + \frac{0.3}{d} + \frac{0}{e} \right\}$$
$$A_{0.1} = \left\{ \frac{1}{a} + \frac{0.9}{b} + \frac{0.6}{c} + \frac{0.3}{d} \right\}$$
$$A_{0.9} = \left\{ \frac{1}{a} + \frac{0.9}{b} \right\}$$
$$A_{0.6} = \left\{ \frac{1}{a} + \frac{0.9}{b} + \frac{0.6}{c} \right\}, \quad A_{0.3} = \left\{ \frac{1}{a} + \frac{0.9}{b} + \frac{0.6}{c} + \frac{0.3}{d} \right\}$$
$$A_{0+} = \left\{ \frac{1}{a} + \frac{0.9}{b} + \frac{0.6}{c} + \frac{0.3}{d} \right\}$$
$$A_0 = \left\{ \frac{1}{a} + \frac{0.9}{b} + \frac{0.6}{c} + \frac{0.3}{d} + \frac{0}{e} \right\}$$

Code:

```
def union(A,B):
R = []
for i in range(0, len(A)):
if (A[i]>B[i]):
R.append(A[i])
else:
R.append(B[i])
return R
def intersect(A,B):
R = []
for i in range(0, len(A)):
if (A[i]<B[i]):
R.append(A[i])
else:
R.append(B[i])
return R
def complement(A):
R = []
for i in range(0, len(A)):
V = 1- A[i]
R.append(V)
return R
def diff(A,B):
COM_B = complement(B)
R = []
for i in range(0, len(A)):
if (A[i]< COM_B[i]):
R.append(A[i])
else:
R.append(COM_B[i])
return R
def print_fuzzy_set(A,W):
for i in range(0, len(A)-1):
print(f'{round(A[i], 2)}/{round(W[i], 1)}', end = " + ")
```

```
print(f'{round(A[-1], 2})/ {round(W[-1], 1)})')
OP =
OP=float(input('1.Union\n2.Intersection\n3.Complement\
n4.Difference\n5.No
rmal-Transformation\nEnter the operation which need to be
performed from
above:'))
if (OP==1):
A=list(map(float,input('Enter the values of first fuzzy set :
').strip().split()))
B=list(map(float,input('Enter the values of second fuzzy set
: ').strip().split()))
W=list(map(int,input('Enter the weights of each element for
the fuzzy set :
').strip().split()))
lam=float(input('Enter the lamdha value : '))
R=union(A,B)
R=[x for x in R if x>lam]
print('The union of first and second set is :')
print_fuzzy_set(R,W)
elif (OP==2):
A=list(map(float,input('Enter the values of first fuzzy set :
').strip().split()))
B=list(map(float,input('Enter the values of second fuzzy set
: ').strip().split()))
W=list(map(int,input('Enter the weights of each element for
the fuzzy set :
').strip().split()))
lam=float(input('Enter the lamdha value : '))
R=intersect(A,B)
R=[x for x in R if x>lam]
print('The intersect of first and second set is :')
print_fuzzy_set(R,W)
elif (OP==3):
A=list(map(float,input('Enter the values of fuzzy set :
').strip().split()))
```

```
W=list(map(int,input('Enter the weights of each element for  
the fuzzy set :  
lam=float(input('Enter the lamdha value : '))  
R=complement(A)  
R=[x for x in R if x>lam]  
print('The complement of first fuzzy set is :')  
print_fuzzy_set(R,W)  
elif (OP==4):  
A=list(map(float,input('Enter the values of first fuzzy set :  
B=list(map(float,input('Enter the values of second fuzzy set :  
W=list(map(int,input('Enter the weights of each element for  
the fuzzy set :  
lam=float(input('Enter the lamdha value : '))  
R=diff(A,B)  
R=[x for x in R if x>lam]  
print('The differnce of first and second set is :')  
print_fuzzy_set(R,W)  
elif (OP==5):  
A=list(map(float,input('Enter the values of fuzzy set :  
W=list(input('Enter the weights of each element for the  
fuzzy set :  
lam=float(input('Enter the lamdha value : '))  
R=[x for x in A if x>lam]  
for i in range(0,len(R)-1):  
print(f'{round(R[i],2)}/{W[i]}',end=' + ')  
print(f'{round(R[-1],2)}/{W[-1]}')  
else:  
print('Please select any valid number : 1,2,3,4')  
exit()
```

## Output:

```
sumon@Lenovo:~/Desktop/Python-Codes/annie_mam_Practical/Practical$ python3 Alpha-cut.py
1.Union
2.Intersection
3.Complement
4.Difference
5.Normal-Transformation
Enter the operation which need to be performed from above:5
Enter the values of fuzzy set : 1 0.9 0.6 0.3 0.0
Enter the weights of each element for the fuzzy set : a b c d e
Enter the lamdha value : 0.9
1.0/e
sumon@Lenovo:~/Desktop/Python-Codes/annie_mam_Practical/Practical$ python3 Alpha-cut.py
1.Union
2.Intersection
3.Complement
4.Difference
5.Normal-Transformation
Enter the operation which need to be performed from above:5
Enter the values of fuzzy set : 1 0.9 0.6 0.3 0.0
Enter the weights of each element for the fuzzy set : a b c d e
Enter the lamdha value : 0.6
1.0/a + 0.9/e
sumon@Lenovo:~/Desktop/Python-Codes/annie_mam_Practical/Practical$ python3 Alpha-cut.py
1.Union
2.Intersection
3.Complement
4.Difference
5.Normal-Transformation
Enter the operation which need to be performed from above:5
Enter the values of fuzzy set : 1 0.9 0.6 0.3 0.0
Enter the weights of each element for the fuzzy set : a b c d e
Enter the lamdha value : 0.3
1.0/a + 0.9/b + 0.6/e
sumon@Lenovo:~/Desktop/Python-Codes/annie_mam_Practical/Practical$
```

```
sumon@Lenovo:~/Desktop/Python-Codes/annie_mam_Practical/Practical$ python3 Alpha-cut.py
1.Union
2.Intersection
3.Complement
4.Difference
5.Normal-Transformation
Enter the operation which need to be performed from above:5
Enter the values of fuzzy set : 1.0 0.9 0.6 0.3 0.0
Enter the weights of each element for the fuzzy set : a b c d e
Enter the lamdha value : 0
1.0/a + 0.9/b + 0.6/c + 0.3/e
sumon@Lenovo:~/Desktop/Python-Codes/annie_mam_Practical/Practical$
```

# Practical-10

## Arithmetic Operations

1] Perform the following operation on interval

- a) Addition i)  $[3,2] + [4,3]$  ii)  $[5,3] + [4,2]$
- b) Subtraction i)  $[3,5] - [4,5]$  ii)  $[6,9] - [2,4]$
- c) Multiplication i)  $[2,1] \times [1,3]$  ii)  $[1,2] \times [5,3]$
- d) Division i)  $[4,6] / [1,2]$  ii)  $[7,3] / [3,6]$
- e) Image of A=  $[5,3]$  B =  $[1,3]$
- f) Inverse of A=  $[5,3]$  B =  $[1,3]$

Solution:

1) a) i)  $[3, 2] + [4, 3] = [3+4, 2+3] = [7, 5]$   
ii)  $[5, 3] + [4, 2] = [5+4, 3+2] = [9, 5]$

b) i)  $[3, 5] - [4, 5] = [3-5, 5-5] = [-2, 0]$   
ii)  $[6, 9] - [2, 4] = [6-4, 9-2] = [2, 7]$

c) i)  $[2, 1] \times [1, 3] = [2 \times 1, 1 \times 3] = [2, 3]$   
ii)  $[1, 2] \times [5, 3] = [1 \times 5, 2 \times 3] = [5, 6]$

d) i)  $[4, 6] \div [1, 2] = [4/2, 6/1] = [2, 6]$   
ii)  $[7, 3] \div [3, 6] = [7/6, 3/3] = [7/6, 1]$

e) i)  $A = [5, 3] = [-3, -5]$  ii)  $B = [1, 3] = [-3, -1]$

f) i)  $A = [5, 3] = [1/5, 1/3]$  ii)  $B = [1, 3] = [1/1, 1/3] = [1, 1/3]$

Code:

```
def ADD(x,y):
r=[]
r.append(x[0]+y[0])
r.append(x[1]+y[1])
return r
def SUB(x,y):
r=[]
r.append(x[0]-y[1])
r.append(x[1]-y[0])
return r
def MUL(x,y):
r=[]
r.append(x[0]*y[0])
r.append(x[1]*y[1])
return r
def DIV(x,y):
r=[]
r.append(x[0]/y[1])
r.append(x[1]/y[0])
return r
def IMG(x):
r=[]
r.append(-x[1])
r.append(-x[0])
return r
def INV(x):
r=[]
r.append(1/x[0])
r.append(1/x[1])
return r
def MIN(x,y):
r=[]
for i in range(0,len(x)):
if x[i]>y[i]:
```

```
r.append(y[i])
else:
r.append(x[i])
return r

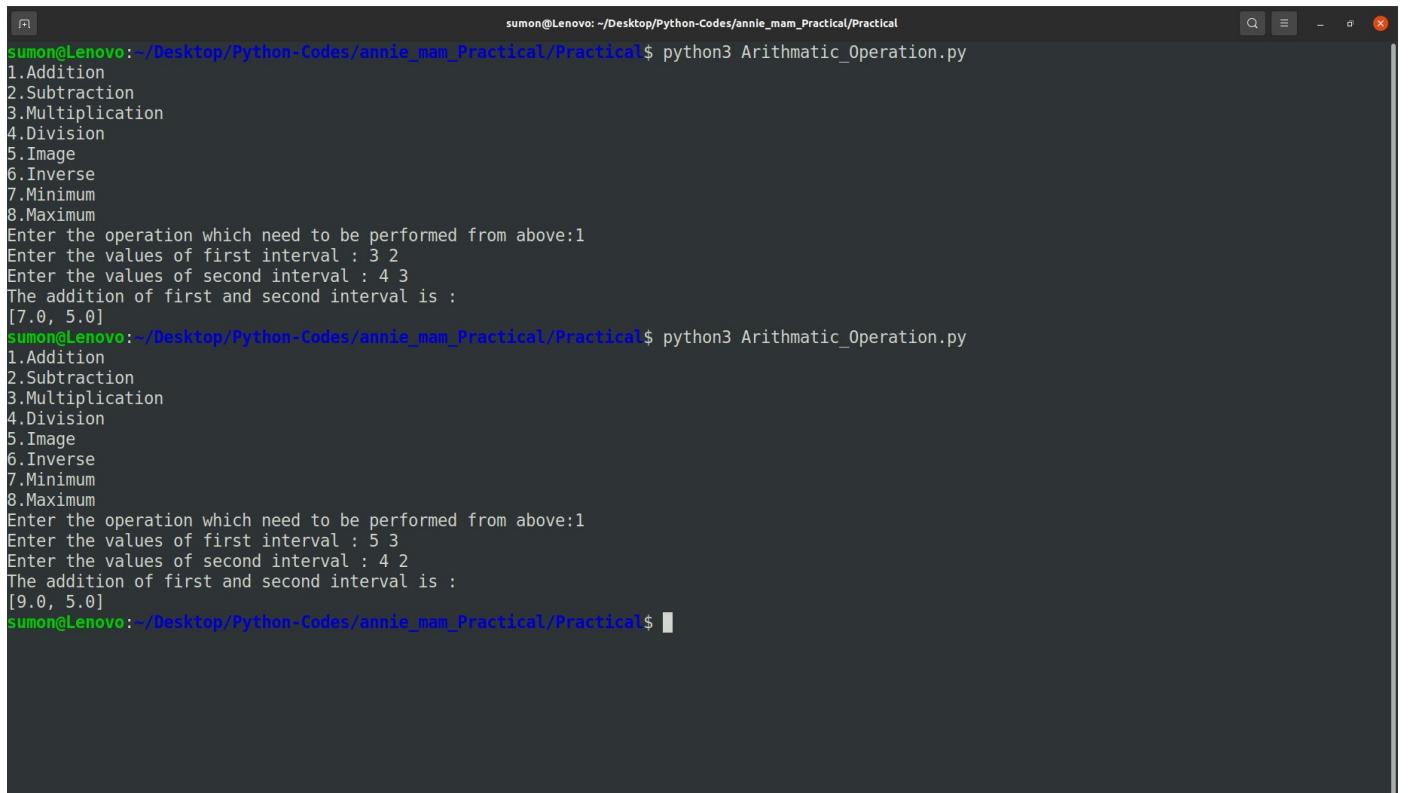
def MAX(x,y):
r=[]
for i in range(0,len(x)):
if x[i]<y[i]:
r.append(y[i])
else:
r.append(x[i])
return r

OP=float(input('1.Addition\n2.Subtraction\n3.Multiplication\n4.Division\n5.Im
age\n6.Inverse\n7.Minimum\n8.Maximum\nEnter the
operation which need
to be performed from above:'))
if (OP==1):
A=list(map(float,input('Enter the values of first interval :
').strip().split()))
B=list(map(float,input('Enter the values of second interval :
').strip().split()))
R=ADD(A,B)
print('The addition of first and second interval is :')
print(R)
elif (OP==2):
A=list(map(float,input('Enter the values of first interval :
').strip().split()))
B=list(map(float,input('Enter the values of second interval :
').strip().split()))
R=SUB(A,B)
print('The subtraction of first and second interval is :')
print(R)
elif (OP==3):
A=list(map(float,input('Enter the values of first interval :')))
```

```
'.strip().split()))
B=list(map(float,input('Enter the values of second interval :
').strip().split()))
R=MUL(A,B)
print('The multiplication of first and second interval is :')
print(R)
elif (OP==4):
A=list(map(float,input('Enter the values of first interval :
').strip().split()))
B=list(map(float,input('Enter the values of second interval :
').strip().split()))
R=DIV(A,B)
print('The division of first and second interval is :')
print(R)
elif (OP==5):
A=list(map(float,input('Enter the values of the interval :
').strip().split()))
R=IMG(A)
print('The image of the interval is :')
print(R)
elif (OP==6):
A=list(map(float,input('Enter the values of the interval :
').strip().split()))
R=INV(A)
print('The inverse of the interval is :')
print(R)
elif (OP==7):
A=list(map(float,input('Enter the values of first interval :
').strip().split()))
B=list(map(float,input('Enter the values of second interval :
').strip().split()))
R=MIN(A,B)
print('The minimam of first and second interval is :')
print(R)
elif (OP==8):
A=list(map(float,input('Enter the values of first interval :
```

```
'.strip().split()))
B=list(map(float,input('Enter the values of second interval :
').strip().split()))
R=MAX(A,B)
print('The maximum of first and second interval is :')
print(R)
else:
print('Please select any valid number : 1,2,3,4')
exit()
```

## Output:



A terminal window titled 'sumon@Lenovo: ~/Desktop/Python-Codes/annie\_mam\_Practical/Practical' is shown. It displays the execution of a Python script named 'Arithmatic\_Operation.py'. The script lists eight arithmetic operations (1. Addition, 2. Subtraction, 3. Multiplication, 4. Division, 5. Image, 6. Inverse, 7. Minimum, 8. Maximum) and asks for two intervals. For the first run, it takes intervals [3, 2] and [4, 3], resulting in [7.0, 5.0]. For the second run, it takes intervals [5, 3] and [4, 2], resulting in [9.0, 5.0].

```
sumon@Lenovo:~/Desktop/Python-Codes/annie_mam_Practical/Practical$ python3 Arithmatic_Operation.py
1.Addition
2.Subtraction
3.Multiplication
4.Division
5.Image
6.Inverse
7.Minimum
8.Maximum
Enter the operation which need to be performed from above:1
Enter the values of first interval : 3 2
Enter the values of second interval : 4 3
The addition of first and second interval is :
[7.0, 5.0]
sumon@Lenovo:~/Desktop/Python-Codes/annie_mam_Practical/Practical$ python3 Arithmatic_Operation.py
1.Addition
2.Subtraction
3.Multiplication
4.Division
5.Image
6.Inverse
7.Minimum
8.Maximum
Enter the operation which need to be performed from above:1
Enter the values of first interval : 5 3
Enter the values of second interval : 4 2
The addition of first and second interval is :
[9.0, 5.0]
sumon@Lenovo:~/Desktop/Python-Codes/annie_mam_Practical/Practical$
```

```
sumon@Lenovo:~/Desktop/Python-Codes/annie_mam_Practical/Practical$ python3 Arithmatic_Operation.py
1.Addition
2.Subtraction
3.Multiplication
4.Division
5.Image
6.Inverse
7.Minimum
8.Maximum
Enter the operation which need to be performed from above:2
Enter the values of first interval : 3 5
Enter the values of second interval : 4 5
The subtraction of first and second interval is :
[-2.0, 1.0]
sumon@Lenovo:~/Desktop/Python-Codes/annie_mam_Practical/Practical$ python3 Arithmatic_Operation.py
1.Addition
2.Subtraction
3.Multiplication
4.Division
5.Image
6.Inverse
7.Minimum
8.Maximum
Enter the operation which need to be performed from above:2
Enter the values of first interval : 6 9
Enter the values of second interval : 2 4
The subtraction of first and second interval is :
[2.0, 7.0]
sumon@Lenovo:~/Desktop/Python-Codes/annie_mam_Practical/Practical$
```

```
sumon@Lenovo:~/Desktop/Python-Codes/annie_mam_Practical/Practical$ python3 Arithmatic_Operation.py
1.Addition
2.Subtraction
3.Multiplication
4.Division
5.Image
6.Inverse
7.Minimum
8.Maximum
Enter the operation which need to be performed from above:3
Enter the values of first interval : 2 1
Enter the values of second interval : 1 3
The multiplication of first and second interval is :
[2.0, 3.0]
sumon@Lenovo:~/Desktop/Python-Codes/annie_mam_Practical/Practical$ python3 Arithmatic_Operation.py
1.Addition
2.Subtraction
3.Multiplication
4.Division
5.Image
6.Inverse
7.Minimum
8.Maximum
Enter the operation which need to be performed from above:3
Enter the values of first interval : 1 2
Enter the values of second interval : 5 3
The multiplication of first and second interval is :
[5.0, 6.0]
sumon@Lenovo:~/Desktop/Python-Codes/annie_mam_Practical/Practical$
```

```
sumon@Lenovo:~/Desktop/Python-Codes/annie_mam_Practical/Practical$ python3 Arithmatic_Operation.py
1.Addition
2.Subtraction
3.Multiplication
4.Division
5.Image
6.Inverse
7.Minimum
8.Maximum
Enter the operation which need to be performed from above:4
Enter the values of first interval : 4 6
Enter the values of second interval : 1 2
The division of first and second interval is :
[2.0, 6.0]
sumon@Lenovo:~/Desktop/Python-Codes/annie_mam_Practical/Practical$ python3 Arithmatic_Operation.py
1.Addition
2.Subtraction
3.Multiplication
4.Division
5.Image
6.Inverse
7.Minimum
8.Maximum
Enter the operation which need to be performed from above:4
Enter the values of first interval : 7 3
Enter the values of second interval : 3 6
The division of first and second interval is :
[1.1666666666666667, 1.0]
sumon@Lenovo:~/Desktop/Python-Codes/annie_mam_Practical/Practical$
```

```
sumon@Lenovo:~/Desktop/Python-Codes/annie_mam_Practical/Practical$ python3 Arithmatic_Operation.py
1.Addition
2.Subtraction
3.Multiplication
4.Division
5.Image
6.Inverse
7.Minimum
8.Maximum
Enter the operation which need to be performed from above:5
Enter the values of the interval : 5 3
The image of the interval is :
[-3.0, -5.0]
sumon@Lenovo:~/Desktop/Python-Codes/annie_mam_Practical/Practical$ python3 Arithmatic_Operation.py
1.Addition
2.Subtraction
3.Multiplication
4.Division
5.Image
6.Inverse
7.Minimum
8.Maximum
Enter the operation which need to be performed from above:5
Enter the values of the interval : 1 3
The image of the interval is :
[-3.0, -1.0]
sumon@Lenovo:~/Desktop/Python-Codes/annie_mam_Practical/Practical$
```

```
sumon@Lenovo:~/Desktop/Python-Codes/annie_mam_Practical/Practical$ python3 Arithmatic_Operation.py
1.Addition
2.Subtraction
3.Multiplication
4.Division
5.Image
6.Inverse
7.Minimum
8.Maximum
Enter the operation which need to be performed from above:6
Enter the values of the interval : 5 3
The inverse of the interval is :
[0.2, 0.333333333333333]
sumon@Lenovo:~/Desktop/Python-Codes/annie_mam_Practical/Practical$ python3 Arithmatic_Operation.py
1.Addition
2.Subtraction
3.Multiplication
4.Division
5.Image
6.Inverse
7.Minimum
8.Maximum
Enter the operation which need to be performed from above:6
Enter the values of the interval : 1 3
The inverse of the interval is :
[1.0, 0.333333333333333]
sumon@Lenovo:~/Desktop/Python-Codes/annie_mam_Practical/Practical$
```

- 2] For given interval perform the inner-outer product
- i)  $E = [2,4]$   $F = [-4,5]$
  - ii)  $F = [5,6]$   $G = [9,2]$

Solution:

2)  $\Rightarrow E = [2, 4], I = [-4, 5]$

a) Max operation:-

$$E \vee I = [2, 4] \vee [-4, 5] = [2, 5]$$

b) min operation:-

$$E \wedge I = [2, 4] \wedge [-4, 5] = [-4, 4]$$

i)  $E = F = [5, 6], G = [9, 2]$

a) Max operation:-

$$E \vee G = [5, 6] \vee [9, 2] = [9, 6]$$

b) min operation:-

$$E \wedge G = [5, 6] \wedge [9, 2] = [5, 2]$$

Code:

```
def ADD(x,y):
r=[]
r.append(x[0]+y[0])
r.append(x[1]+y[1])
return r
def SUB(x,y):
r=[]
r.append(x[0]-y[1])
r.append(x[1]-y[0])
return r
def MUL(x,y):
r=[]
r.append(x[0]*y[0])
r.append(x[1]*y[1])
return r
def DIV(x,y):
r=[]
r.append(x[0]/y[1])
r.append(x[1]/y[0])
return r
def IMG(x):
r=[]
r.append(-x[1])
r.append(-x[0])
return r
def INV(x):
r=[]
r.append(1/x[0])
r.append(1/x[1])
return r
def MIN(x,y):
r=[]
for i in range(0,len(x)):
```

```

if x[i]>y[i]:
r.append(y[i])
else:
r.append(x[i])
return r

def MAX(x,y):
r=[]
for i in range(0,len(x)):
if x[i]<y[i]:
r.append(y[i])
else:
r.append(x[i])
return r

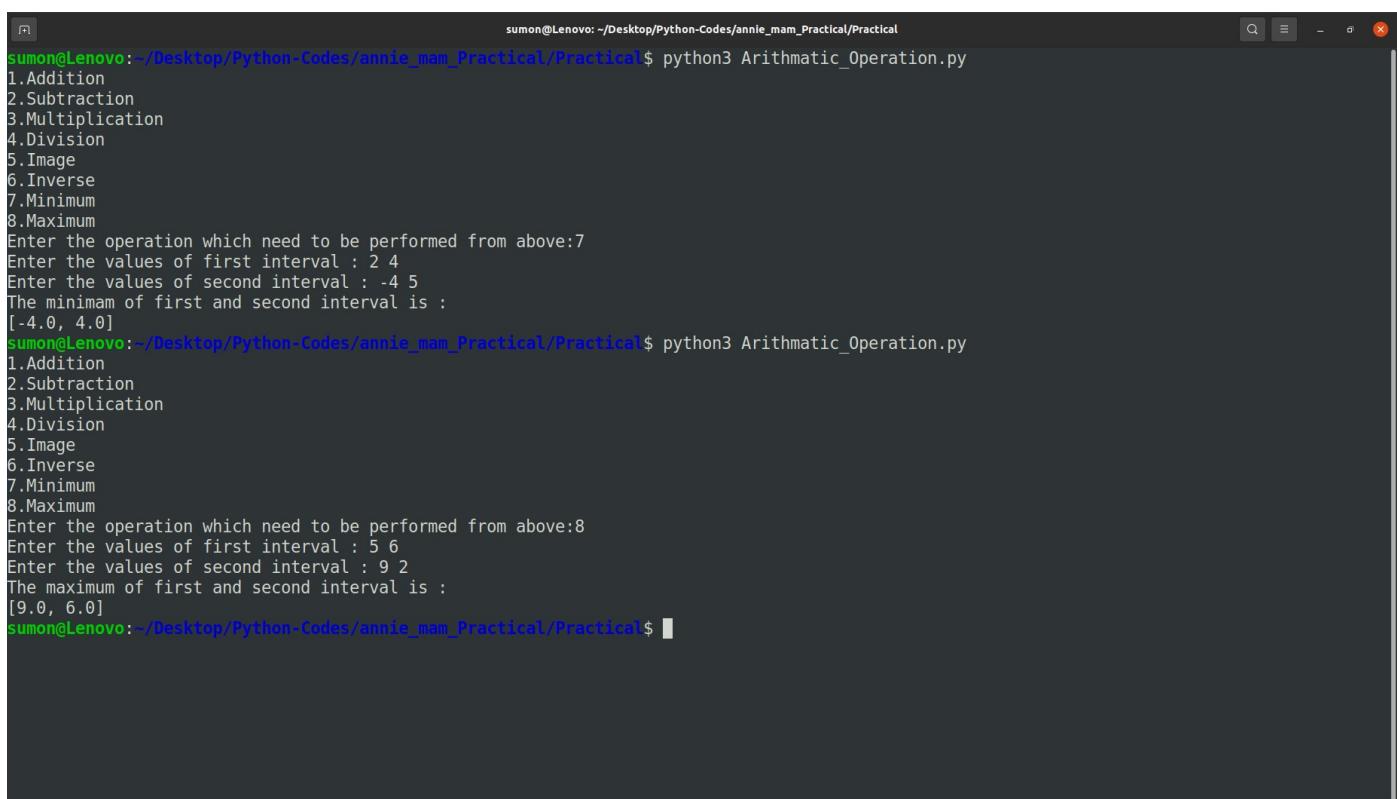
OP=float(input('1.Addition\n2.Subtraction\n3.Multiplication\n4.Division\n5.Im-
age\n6.Inverse\n7.Minimum\n8.Maximum\nEnter the
operation which need
to be performed from above:'))
if (OP==1):
A=list(map(float,input('Enter the values of first interval :
').strip().split()))
B=list(map(float,input('Enter the values of second interval :
').strip().split()))
R=ADD(A,B)
print('The addition of first and second interval is :')
print(R)
elif (OP==2):
A=list(map(float,input('Enter the values of first interval :
').strip().split()))
B=list(map(float,input('Enter the values of second interval :
').strip().split()))
R=SUB(A,B)
print('The subtraction of first and second interval is :')
print(R)
elif (OP==3):

```

```
A=list(map(float,input('Enter the values of first interval : ').strip().split()))
B=list(map(float,input('Enter the values of second interval : ').strip().split()))
R=MUL(A,B)
print('The multiplication of first and second interval is :')
print(R)
elif (OP==4):
A=list(map(float,input('Enter the values of first interval : ').strip().split()))
B=list(map(float,input('Enter the values of second interval : ').strip().split()))
R=DIV(A,B)
print('The division of first and second interval is :')
print(R)
elif (OP==5):
A=list(map(float,input('Enter the values of the interval : ').strip().split()))
R=IMG(A)
print('The image of the interval is :')
print(R)
elif (OP==6):
A=list(map(float,input('Enter the values of the interval : ').strip().split()))
R=INV(A)
print('The inverse of the interval is :')
print(R)
elif (OP==7):
A=list(map(float,input('Enter the values of first interval : ').strip().split()))
B=list(map(float,input('Enter the values of second interval : ').strip().split()))
R=MIN(A,B)
print('The minimam of first and second interval is :')
print(R)
elif (OP==8):
```

```
A=list(map(float,input('Enter the values of first interval : ').strip().split()))
B=list(map(float,input('Enter the values of second interval : ').strip().split()))
R=MAX(A,B)
print('The maximum of first and second interval is :')
print(R)
else:
print('Please select any valid number : 1,2,3,4')
exit()
```

## Output:



```
sumon@Lenovo:~/Desktop/Python-Codes/annie_mam_Practical/Practical$ python3 Arithmatic_Operation.py
1.Addition
2.Subtraction
3.Multiplication
4.Division
5.Image
6.Inverse
7.Minimum
8.Maximum
Enter the operation which need to be performed from above:7
Enter the values of first interval : 2 4
Enter the values of second interval : -4 5
The minimam of first and second interval is :
[-4.0, 4.0]
sumon@Lenovo:~/Desktop/Python-Codes/annie_mam_Practical/Practical$ python3 Arithmatic_Operation.py
1.Addition
2.Subtraction
3.Multiplication
4.Division
5.Image
6.Inverse
7.Minimum
8.Maximum
Enter the operation which need to be performed from above:8
Enter the values of first interval : 5 6
Enter the values of second interval : 9 2
The maximum of first and second interval is :
[9.0, 6.0]
sumon@Lenovo:~/Desktop/Python-Codes/annie_mam_Practical/Practical$
```

3] For fuzzy vectors perform the inner-outer product

i)  $a = (0.5, 0.2, 1, 0, 0.8)$   $b = (0.8, 0.1, 0.9, 0.3)$

ii)  $c = (0.5, 0.7, 0.2, 0.3, 1, 0.8)$   $d = (0, 0.2, 0.1, 0.4, 0.6, 1.0)$

Solution:

$$3) i) \underline{a} = (0.5, 0.2, 1.0, 0.8), \underline{b} = (0.8, 0.1, 0.9, 0.3)$$

\* inner product :-

$$\underline{a} \cdot \underline{b}^T = (0.5, 0.2, 1.0, 0.8) \begin{bmatrix} 0.8 \\ 0.1 \\ 0.9 \\ 0.3 \end{bmatrix}$$

$$= (0.5 \wedge 0.8) \vee (0.2 \wedge 0.1) \vee (1.0 \wedge 0.9) \vee (0.8 \wedge 0.3)$$

$$= (0.5 \vee 0.1 \vee 0.9 \vee 0.3) = 0.9$$

\* outer product :-

$$\underline{a} \otimes \underline{b}^T = (0.5, 0.2, 1.0, 0.8) \begin{bmatrix} 0.8 \\ 0.1 \\ 0.9 \\ 0.3 \end{bmatrix}$$

$$= (0.5 \vee 0.8) \wedge (0.2 \vee 0.1) \wedge (1.0 \vee 0.9) \wedge (0.8 \vee 0.3)$$

$$= 0.8 \wedge 0.2 \wedge 1.0 \wedge 0.8$$

$$= 1.0$$

$$\text{ii) } \frac{\mathbf{c}^T}{\mathbf{d}^T} \begin{pmatrix} 0.5, 0.7, 0.2, 0.3, 1, 0.8 \end{pmatrix}$$

inner product:-

$$\mathbf{c} \cdot \mathbf{d}^T = \begin{pmatrix} 0.5, 0.7, 0.2, 0.3, 1, 0.8 \end{pmatrix} \begin{bmatrix} 0 \\ 0.2 \\ 0.1 \\ 0.4 \\ 0.6 \\ 1.0 \end{bmatrix}$$

$$= (0.5 \wedge 0) \vee (0.7 \wedge 0.2) \vee (0.2 \wedge 0.1) \vee (0.3 \wedge 0.4) \vee (1 \wedge 0.6)$$

$$= (0.5 \wedge 0) \vee (0.8 \wedge 1.0)$$

$$= 0 \vee 0.2 \vee 0.1 \vee 0.3 \vee 0.6 \vee 0.8 = 0.8$$

outer product:-

~~$$(0.5 \wedge 0)^T$$~~

$$(0.5 \wedge 0) \wedge (0.7 \wedge 0.2) \wedge (0.2 \wedge 0.1) \wedge (0.3 \wedge 0.4) \wedge (1 \wedge 0.6)$$

$$\wedge (0.8 \wedge 1.0)$$

$$= 0.5 \wedge 0.7 \wedge 0.2 \wedge 0.4 \wedge 1 \wedge 1.0$$

$$= 0.2$$

Code:

```
from email.mime.base import MIMEBase
def ADD(x,y):
r=[]
r.append(x[0]+y[0])
r.append(x[1]+y[1])
return r
def SUB(x,y):
r=[]
r.append(x[0]-y[1])
r.append(x[1]-y[0])
return r
def MUL(x,y):
r=[]
r.append(x[0]*y[0])
r.append(x[1]*y[1])
return r
def DIV(x,y):
r=[]
r.append(x[0]/y[1])
r.append(x[1]/y[0])
return r
def IMG(x):
r=[]
r.append(-x[1])
r.append(-x[0])
return r
def INV(x):
r=[]
r.append(1/x[0])
r.append(1/x[1])
return r
def MIN(x,y):
r=[]
for i in range(0,len(x)):
```

```

if x[i]>y[i]:
r.append(y[i])
else:
r.append(x[i])
return r

def MAX(x,y):
r=[]
for i in range(0,len(x)):
if x[i]<y[i]:
r.append(y[i])
else:
r.append(x[i])
return r

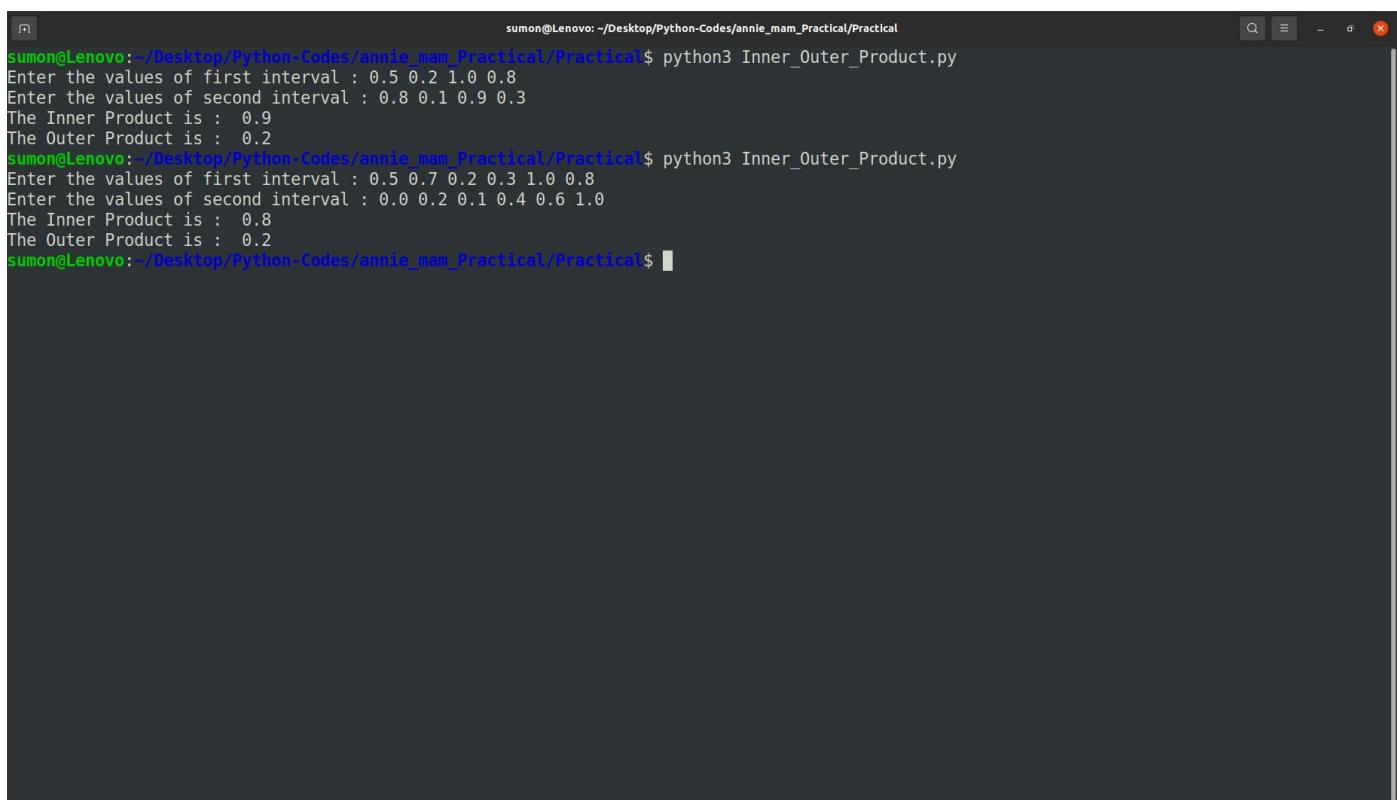
OP=float(input('1.Addition\n2.Subtraction\n3.Multiplication\n4.Division\n5.Im-
age\n6.Inverse\n7.Minimum\n8.Maximum\nEnter the
operation which need
to be performed from above:'))
if (OP==1):
A=list(map(float,input('Enter the values of first interval :
').strip().split()))
B=list(map(float,input('Enter the values of second interval :
').strip().split()))
R=ADD(A,B)
print('The addition of first and second interval is :')
print(R)
elif (OP==2):
A=list(map(float,input('Enter the values of first interval :
').strip().split()))
B=list(map(float,input('Enter the values of second interval :
').strip().split()))
R=SUB(A,B)
print('The subtraction of first and second interval is :')
print(R)
elif (OP==3):

```

```
A=list(map(float,input('Enter the values of first interval : ').strip().split()))
B=list(map(float,input('Enter the values of second interval : ').strip().split()))
R=MUL(A,B)
print('The multiplication of first and second interval is :')
print(R)
elif (OP==4):
A=list(map(float,input('Enter the values of first interval : ').strip().split()))
B=list(map(float,input('Enter the values of second interval : ').strip().split()))
R=DIV(A,B)
print('The division of first and second interval is :')
print(R)
elif (OP==5):
A=list(map(float,input('Enter the values of the interval : ').strip().split()))
R=IMG(A)
print('The image of the interval is :')
print(R)
elif (OP==6):
A=list(map(float,input('Enter the values of the interval : ').strip().split()))
R=INV(A)
print('The inverse of the interval is :')
print(R)
elif (OP==7):
A=list(map(float,input('Enter the values of first interval : ').strip().split()))
B=list(map(float,input('Enter the values of second interval : ').strip().split()))
R=MIN(A,B)
print('The minimam of first and second interval is :')
print(R)
elif (OP==8):
```

```
A=list(map(float,input('Enter the values of first interval : ').strip().split()))
B=list(map(float,input('Enter the values of second interval : ').strip().split()))
R=MAX(A,B)
print('The maximum of first and second interval is :')
print(R)
else:
print('Please select any valid number : 1,2,3,4')
exit()
```

## Output:



```
sumon@Lenovo:~/Desktop/Python-Codes/annie_mam_Practical/Practical$ python3 Inner_Outer_Product.py
Enter the values of first interval : 0.5 0.2 1.0 0.8
Enter the values of second interval : 0.8 0.1 0.9 0.3
The Inner Product is :  0.9
The Outer Product is :  0.2
sumon@Lenovo:~/Desktop/Python-Codes/annie_mam_Practical/Practical$ python3 Inner_Outer_Product.py
Enter the values of first interval : 0.5 0.7 0.2 0.3 1.0 0.8
Enter the values of second interval : 0.0 0.2 0.1 0.4 0.6 1.0
The Inner Product is :  0.8
The Outer Product is :  0.2
sumon@Lenovo:~/Desktop/Python-Codes/annie_mam_Practical/Practical$
```

