University of Mumbai

PRACTICAL JOURNAL

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

SEM-II

Submitted by

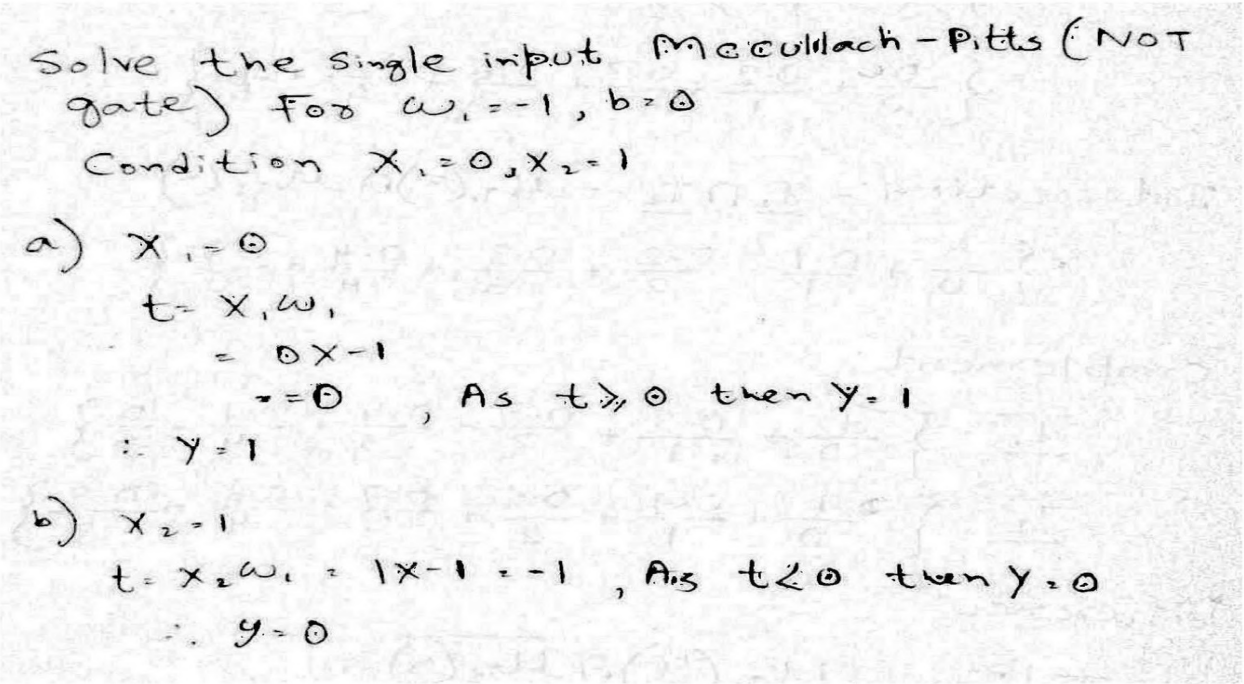
Sumon Singh
Roll No. 16

: INDEX :

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

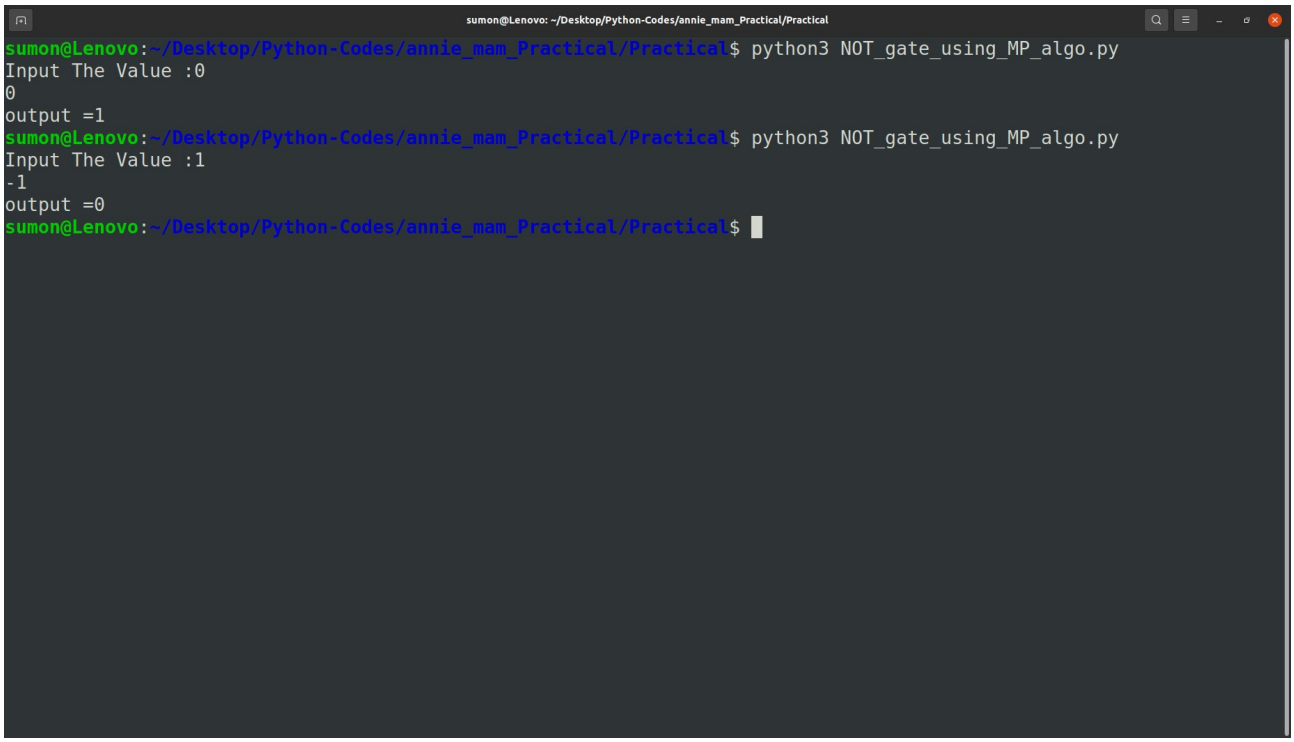
Solution:



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:



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 Mccullach Pitts (OR gate) for $w_1 = w_2 = 1$, $b = -1$

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

=> a) $x_1 = x_2 = 0$, $w_1 = w_2 = 1$, $b = -1$

$\therefore t = x_1 w_1 + x_2 w_2 + b = -1$

$\therefore t = -1$

As, $t < 0 \therefore y = 0$

b) $x_1 = 0, x_2 = 1$, $w_1 = w_2 = 1$, $b = -1$

$\therefore t = x_1 w_1 + x_2 w_2 + b = 0 \times 1 + 1 \times 1 - 1 = 0$

As, $t \geq 0 \therefore y = 1$

c) $x_1 = 1, x_2 = 0$, $w_1 = w_2 = 1$, $b = -1$

$\therefore t = x_1 w_1 + x_2 w_2 + b = 1 \times 1 + 0 \times 1 - 1 = 0$

As, $t \geq 0 \therefore y = 1$

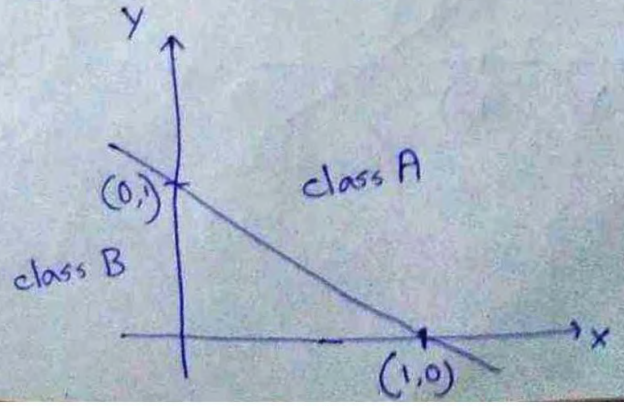
d) $x_1 = x_2 = 1$, $w_1 = w_2 = 1$, $b = -1$

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

As, $t \geq 0 \therefore y = 1$

Class A where $t \geq 0$, i.e when $y = 1$

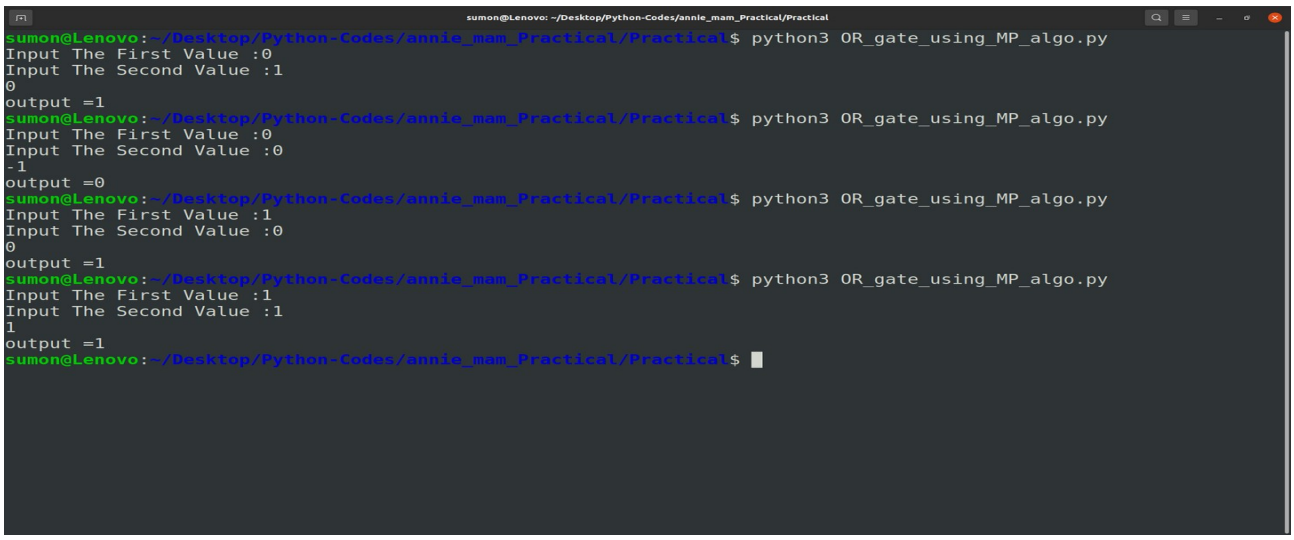
Class B where $t < 0$, i.e where $y = 0$



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:



3] Solve two-input McCulloch Pitts for AND gate
B = -2, w1 = w2 = 1.
Conditions [a] X1=X2 = 0
[b] X1= 0, X2= 1
[c] X1= 1, X2= 0
[d] X1 = X2= 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) $x_1=x_2=1$

=>

its	x_1	x_2	ω_1	ω_2	b	t	y
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

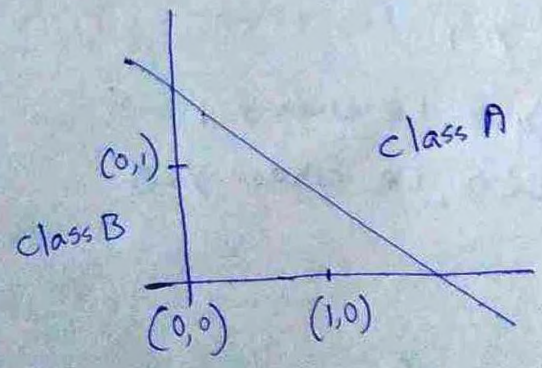
$t = \omega_1 x_1 + \omega_2 x_2 + b$

if $t \geq 0$ then $y=1$

if $t < 0$ then $y=0$

class A is $\omega_1 x_1 + \omega_2 x_2 + b \geq 0$, i.e where $y=1$

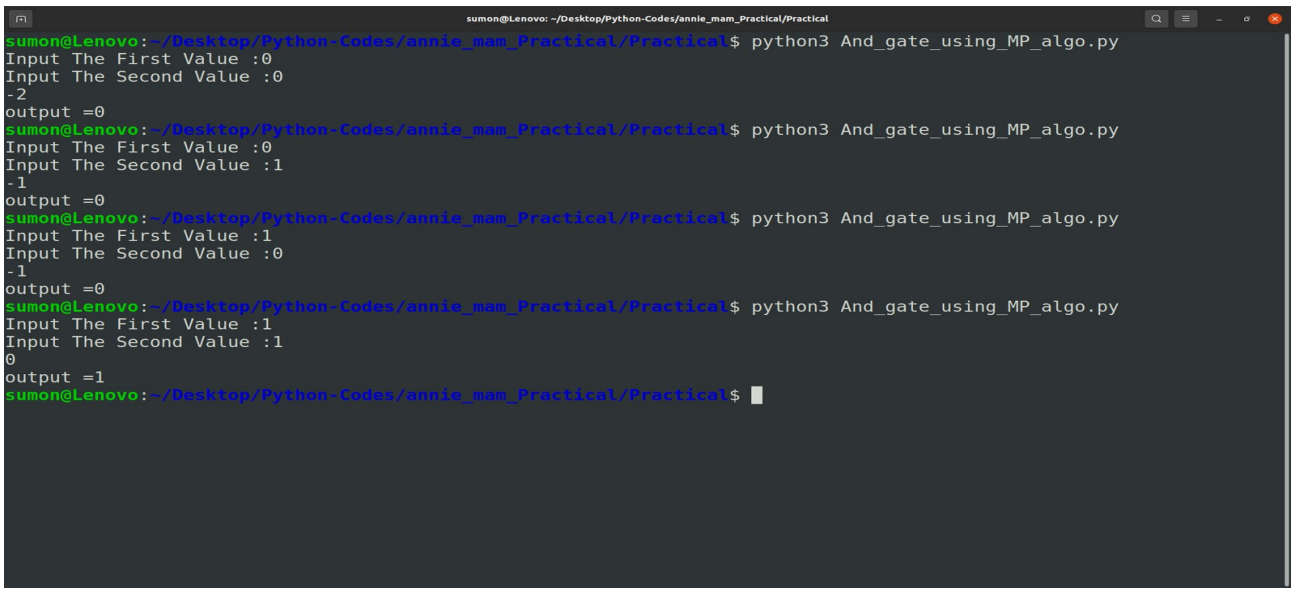
class B is $\omega_1 x_1 + \omega_2 x_2 + b < 0$, i.e where $y=0$



Code:

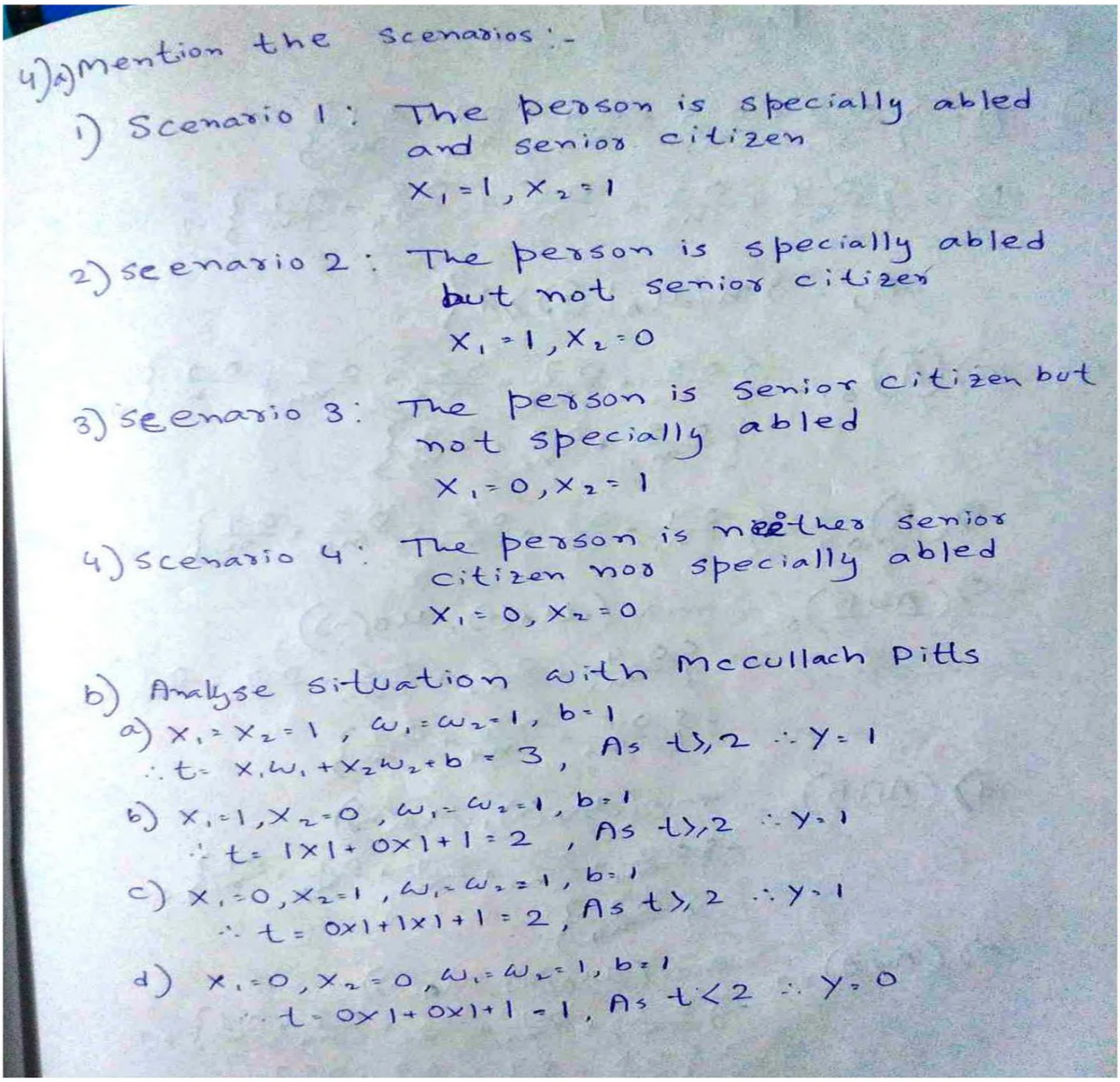
```
# The inputs
x1=int(input('Input The First Value :'))
x2=int(input('Input The Second Value :'))
if x1 not in [0,1]:
sys.exit('Wrong input ,it should be 1/0')
if x2 not in [0,1]:
sys.exit('Wrong input ,it should be 1/0')
# Weights and bias
w1 = 1
w2 = 1
b = -2
t = w1*x1 + w2*x2 + b
print('t = ',t)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

Output:



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:

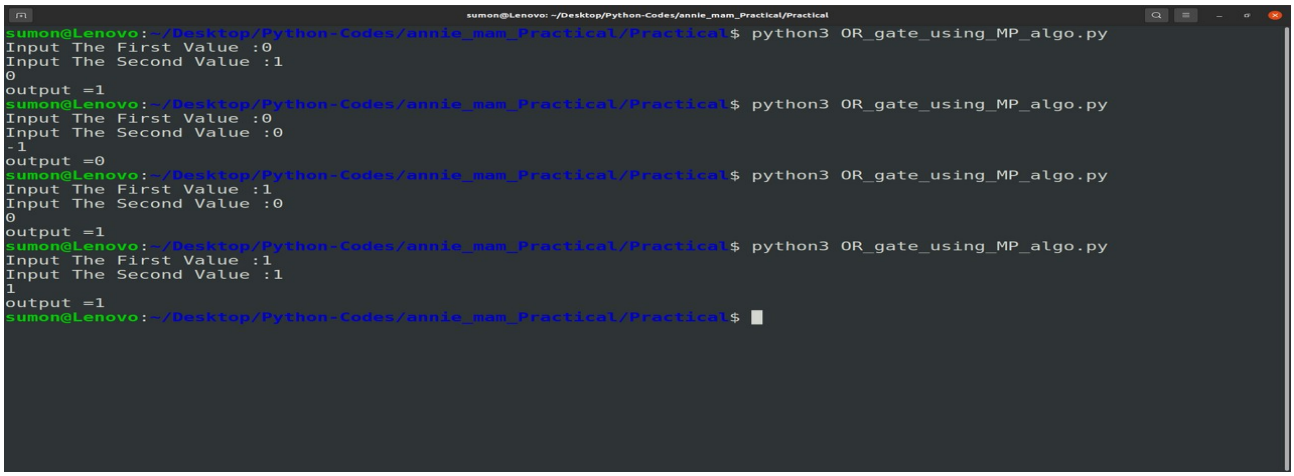


Code:

```
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')
```

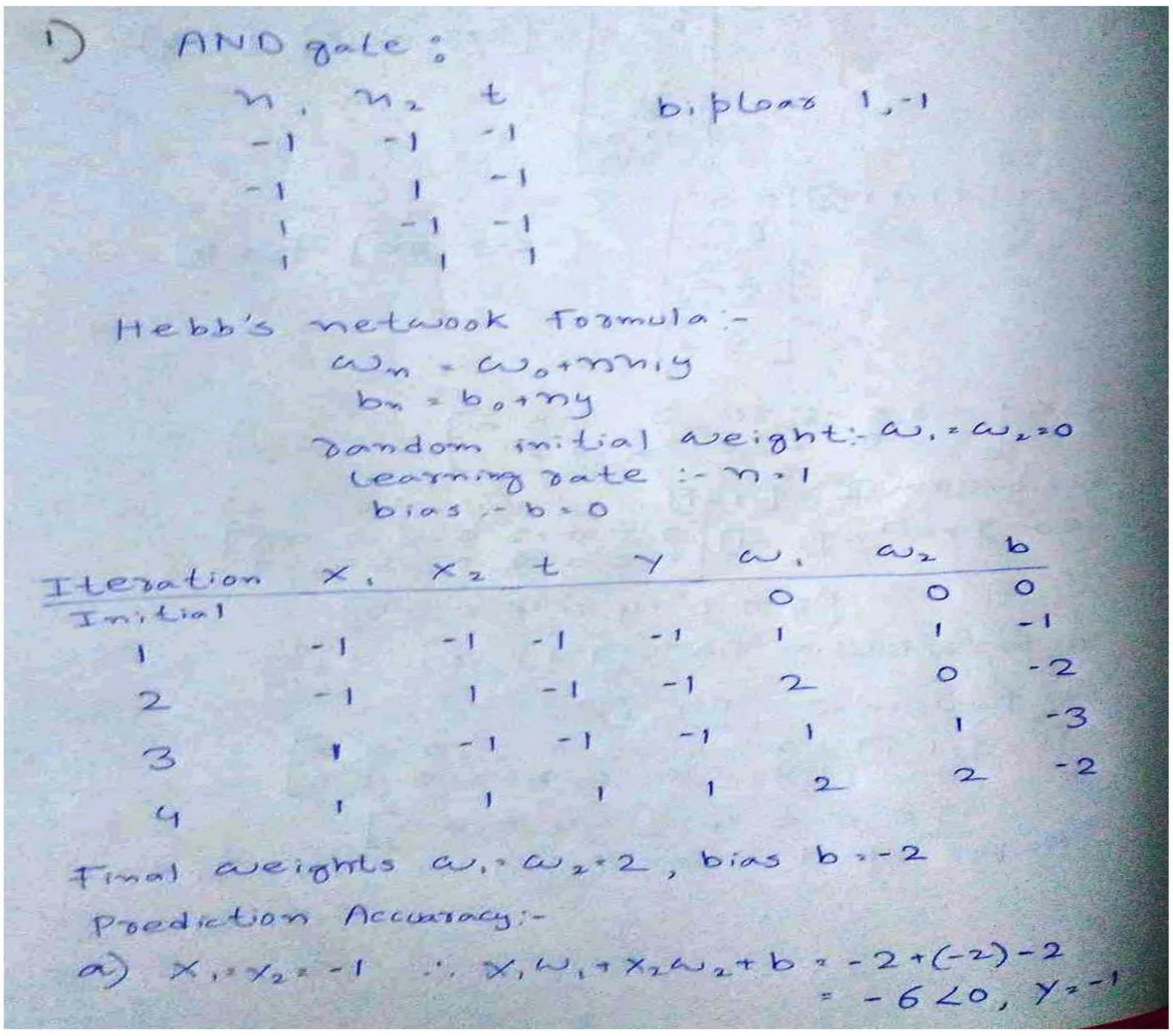
Output:



Practical-2
Hebb's Network

1] Design a Hebb's network to implement logical AND function (use bipolar
inputs and targets).
a. Predict the accuracy.
b. Construct the decision boundary
c. Draw the network.

Solution:



1) AND gate :

x_1	x_2	t
-1	-1	-1
-1	1	-1
1	-1	-1
1	1	1

bipolar 1, -1

Hebb's network formula :-

$w_n = w_0 + x_n y$

$b_n = b_0 + y$

Random initial weight :- $w_1 = w_2 = 0$

Learning rate :- $\eta = 1$

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) $x_1 = x_2 = -1$ $\therefore x_1 w_1 + x_2 w_2 + b = -2 + (-2) - 2$

$= -6 < 0, \; y = -1$

b) $X_1 = -1 \quad X_2 = 1$

$\therefore X_1\omega_1 + X_2\omega_2 + b = -2+2-2 = -2 < 0 \; ; Y = -1$

c) $X_1 = 1, X_2 = -1$

$\therefore X_1\omega_1 + X_2\omega_2 + b = 2-2-2 = -2 < 0 \; ; Y = -1$

d) $X_1 = 1, X_2 = 1$

$\therefore X_1\omega_1 + X_2\omega_2 + b = 2+2-2 = 2 > 0 \; ; Y = 1$

So, accuracy is 100%

Decision Line:

$$\omega_1 X_1 + \omega_2 X_2 + b \geq 0$$

$$2X_1 + 2X_2 - 2 \geq 0$$

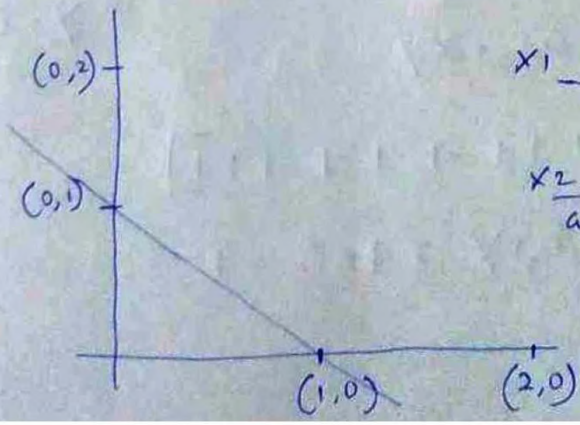
$$\therefore 2X_1 + 2X_2 - 2 = 0$$

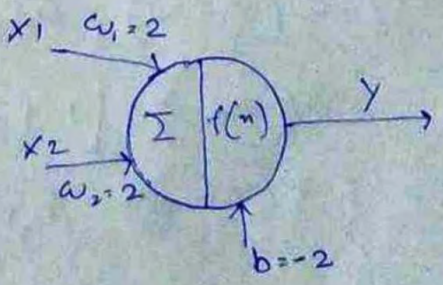
$$(X_1 + X_2) = 1$$

$$X_1 = 1 - X_2$$

if, $X_2 = 0, X_1 = 1$ if $X_1 = 0 ; X_2 = 1$

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

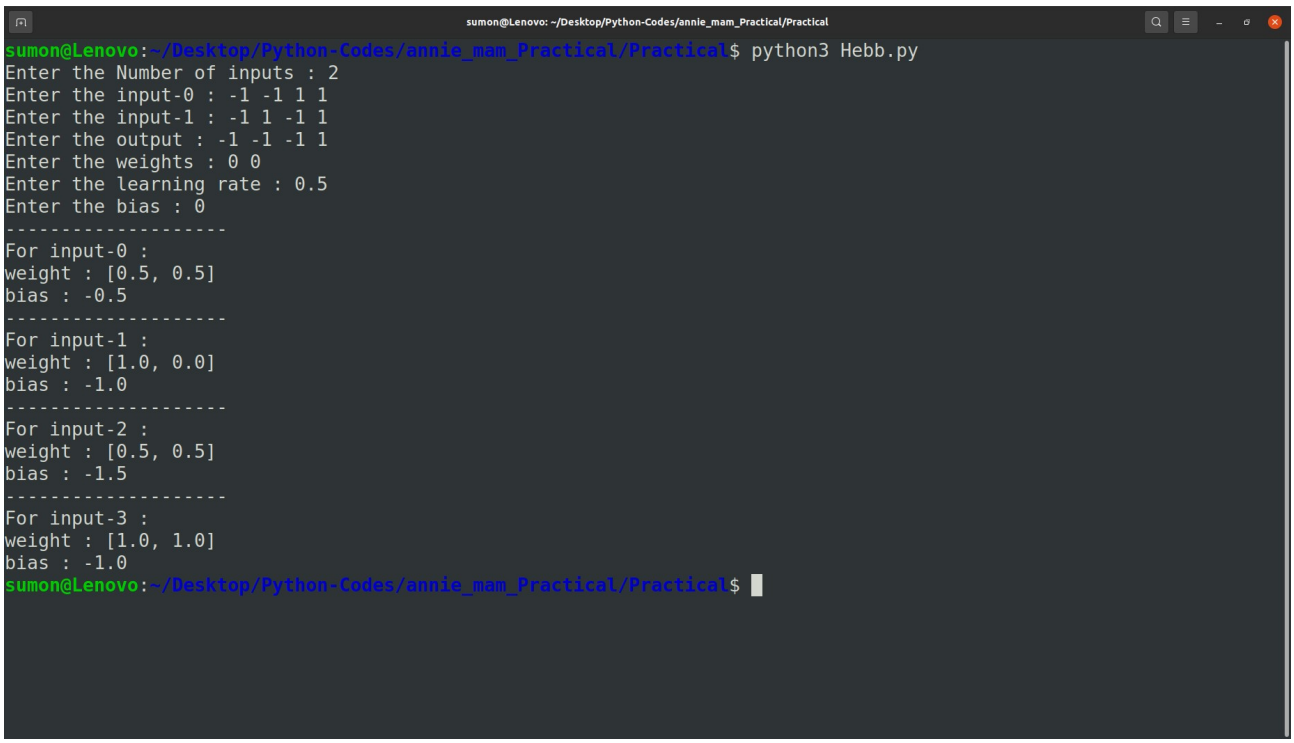




Code:

```
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
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

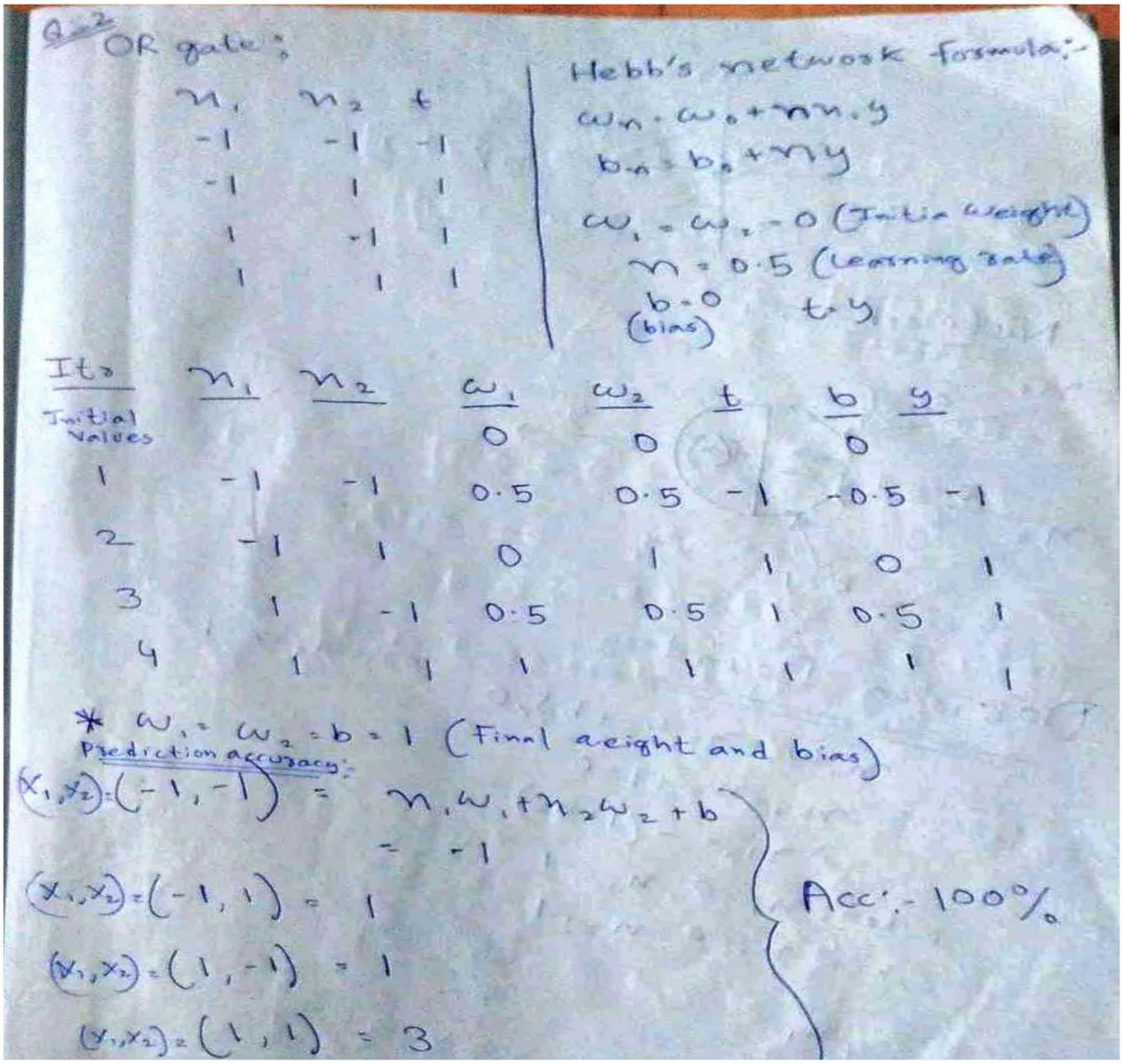
For input-1 :
weight : [1.0, 0.0]
bias : -1.0

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.
a. Predict the accuracy.
b. Construct the decision boundary
c. Draw the network.

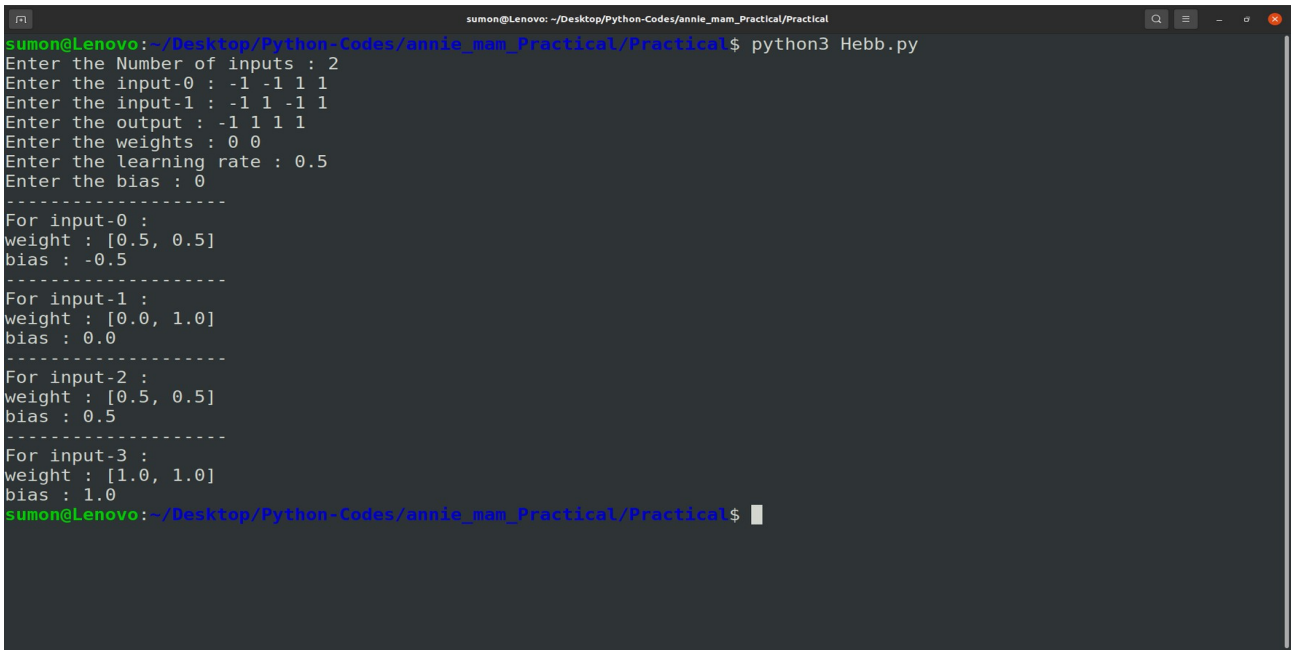
Solution:



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
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

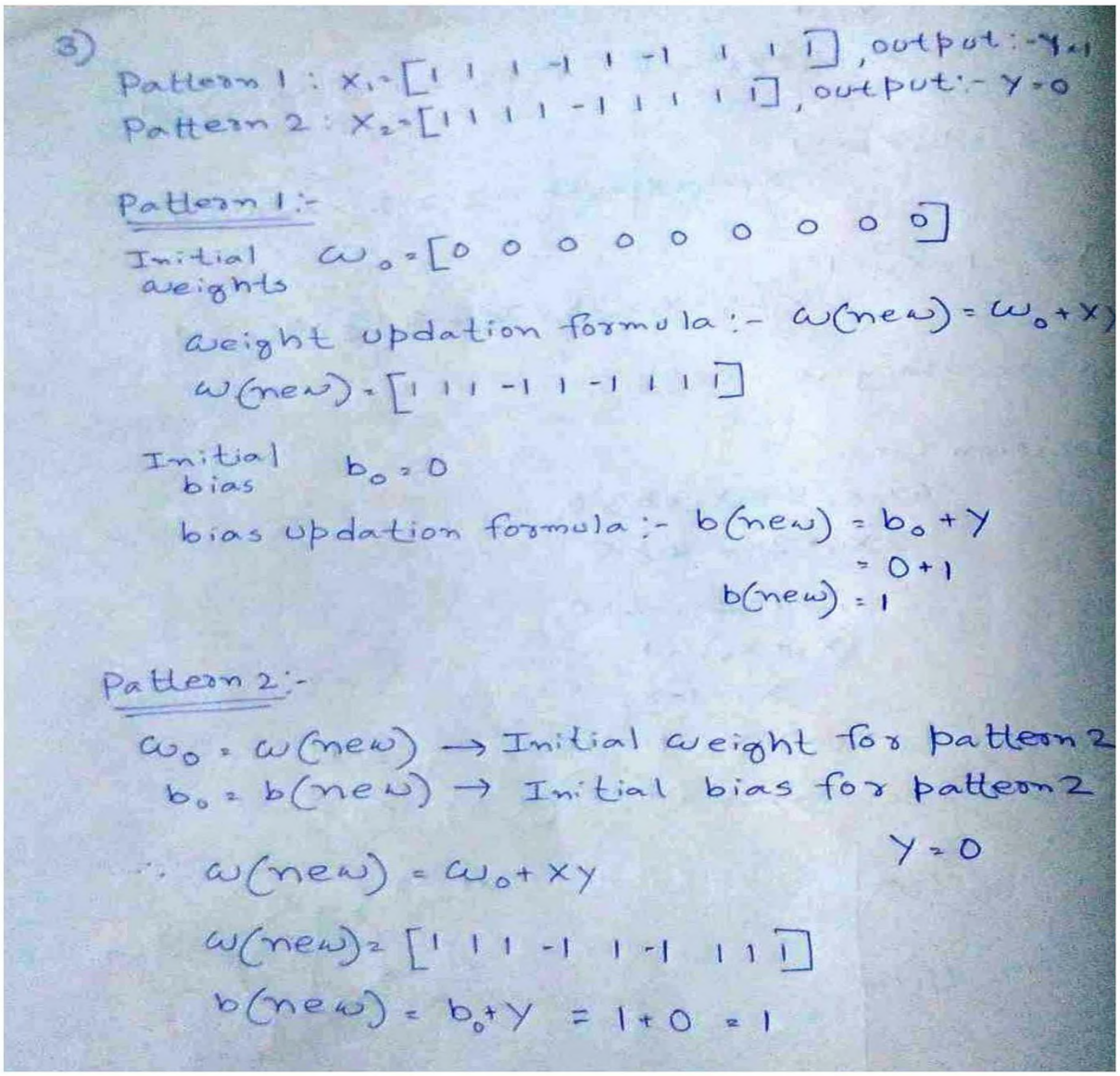
For input-1 :
weight : [0.0, 1.0]
bias : 0.0

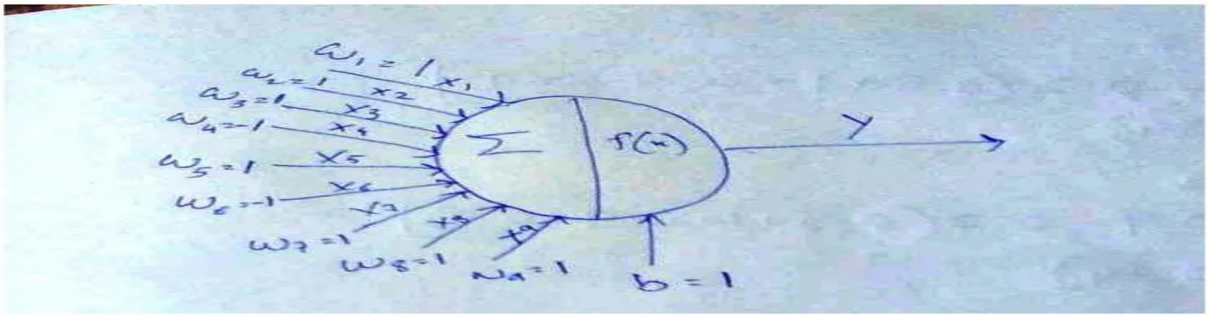
For input-2 :
weight : [0.5, 0.5]
bias : 0.5

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 patter “+” 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:



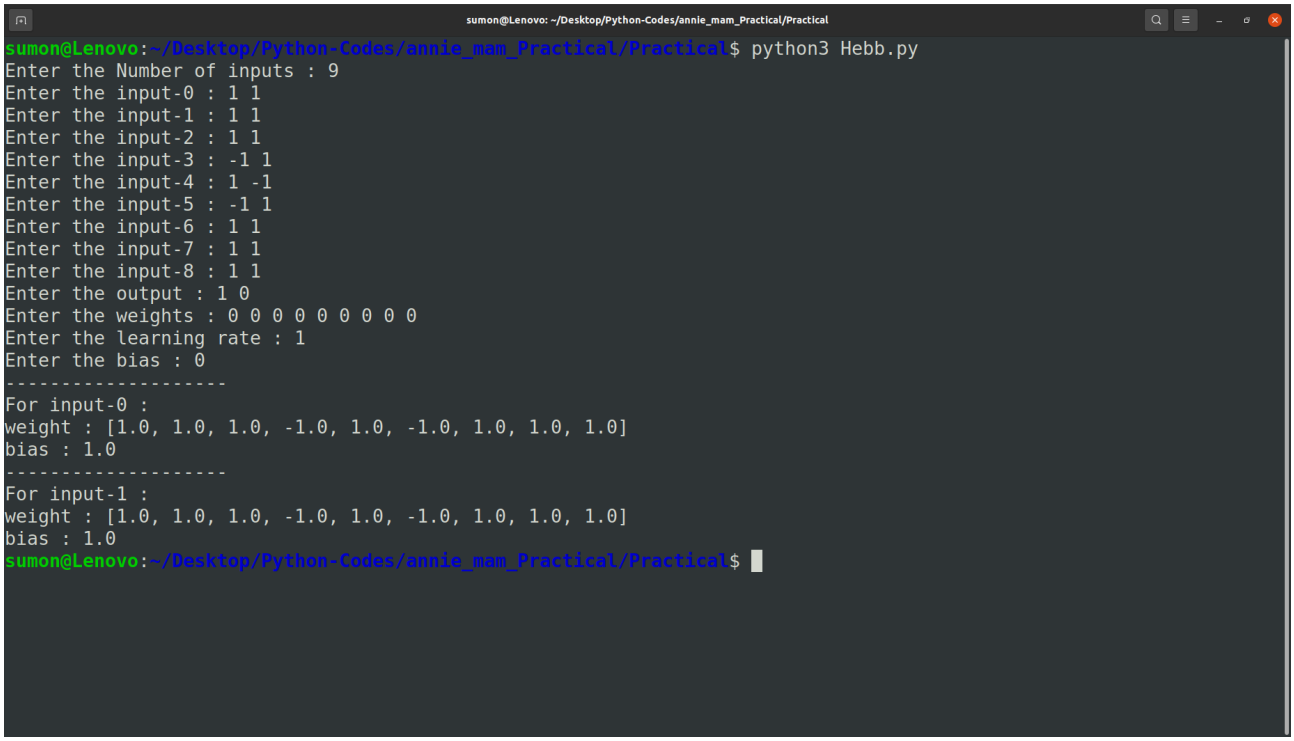


Code:

```
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]
```

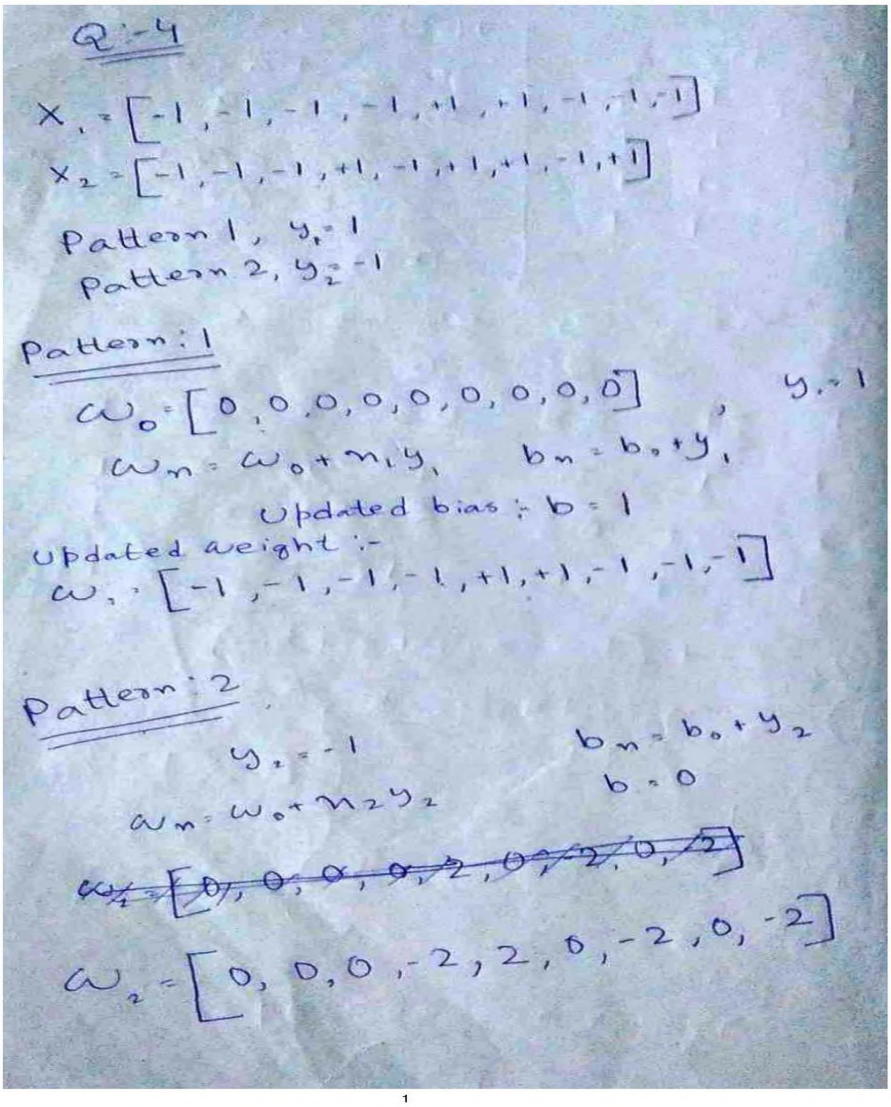
```
bias = bias + n*output[i]
print('----------------------')
print(f'For input-{i} : ')
print(f'weight : {weight}')
print(f'bias : {bias} ')
```

Output:

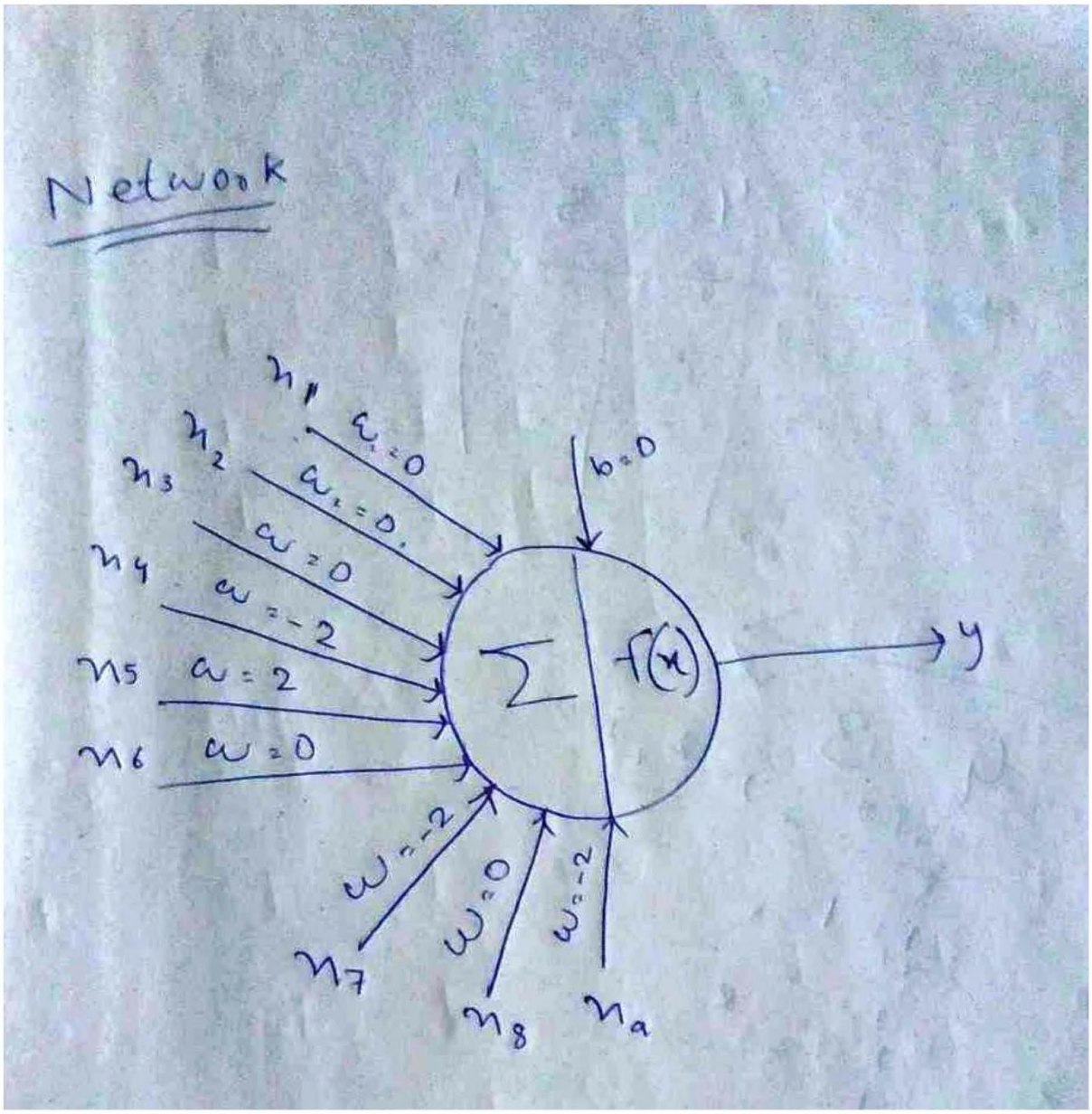


4] Using HEBB rule, classify the two-dimensional input pattern. Draw the
network architecture. Pattern 1 output value is 1, Pattern 2 output value is -1

Solution:



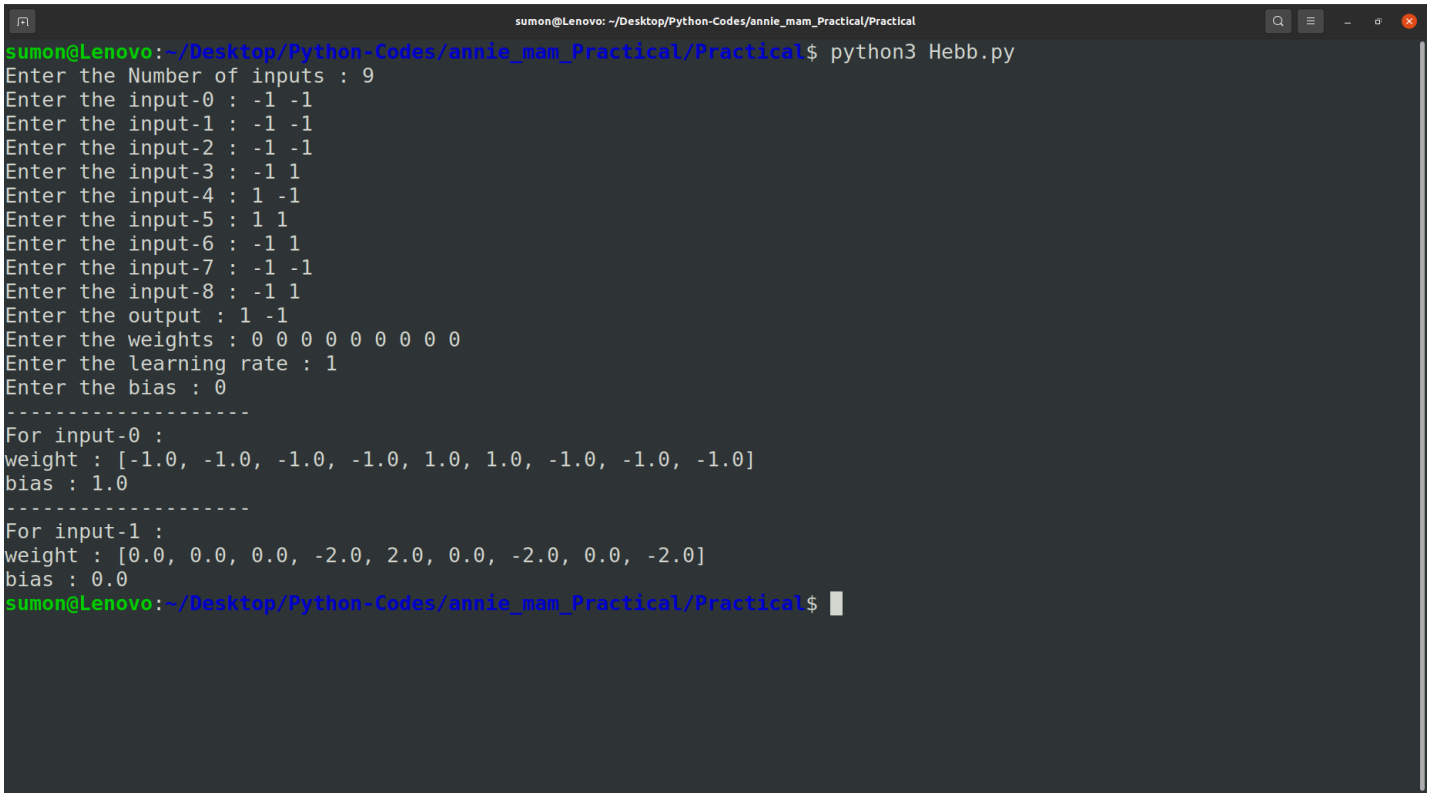
Network



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
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 € = 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 ≥ 0
0 if x < 0
Find the winning neuron.

Solution:

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

Condition 1:-

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

$a_i = f[a_i(t) - \epsilon \Sigma(a_i(t))]$ → Formula

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

$= f(-0.2) < 0$

$\therefore a_1 = 0$

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

$= f(0.05) \quad \therefore a_2 = 0.05$

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

$= f(0.8)$

$\therefore a_3 = 0.8$

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 $[0, 0, 0.7875]$

∴ The winning neuron is 3rd neuron

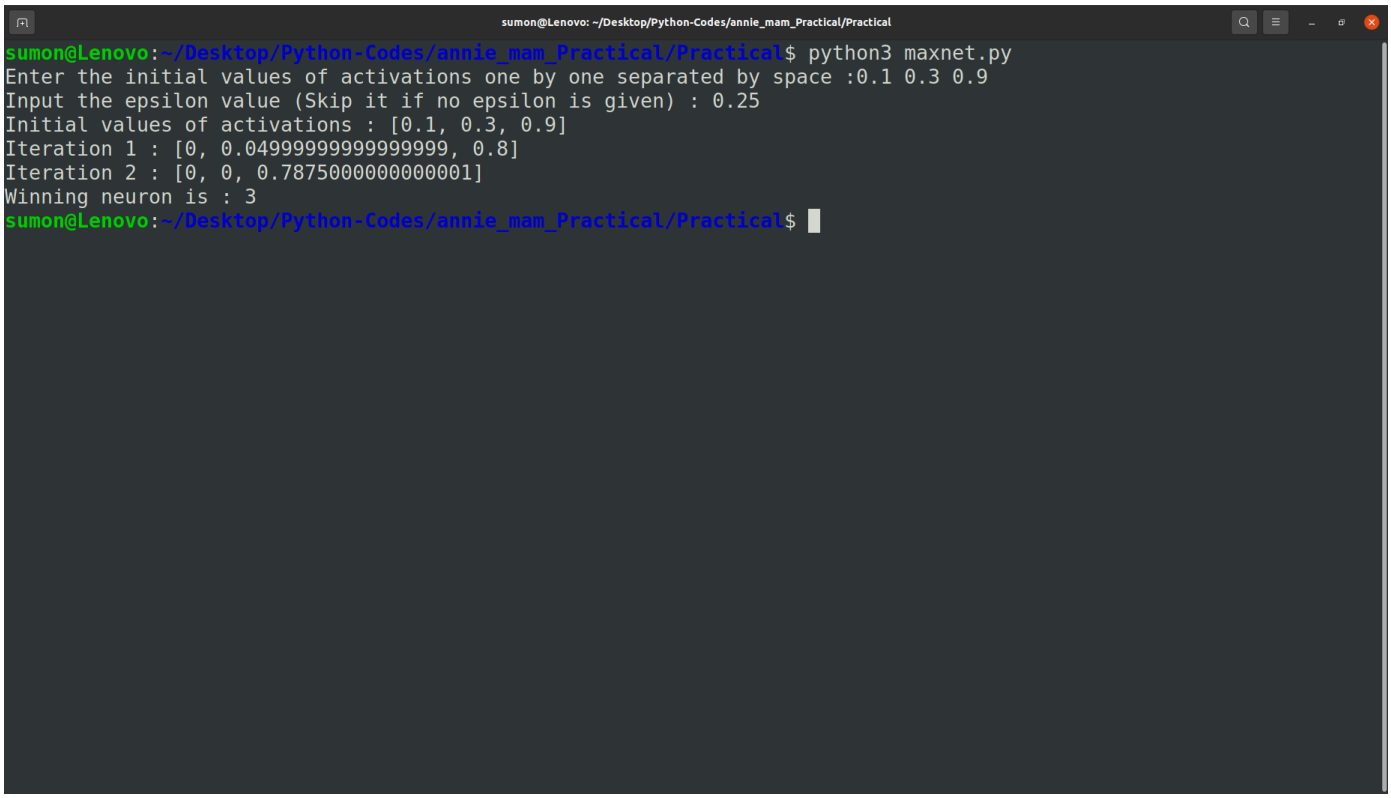
Code:

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

f (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:



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) = x if $x \geq 0$
0 if x< 0

Solution:

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

Condition 1:

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

$a_1 = f[a_i(t) - \epsilon \Sigma a_i(t)]$ → Formula

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

$= f(-0.24) = 0$

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

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

$= f(0.24)$

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

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

$= f(0.36)$

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

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

$= f(0.24)$

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

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

$= f(0.24)$

Condition 2:-

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

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

$= (-0.216) = 0$

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

$= (0.072)$

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

$= (0.216)$

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

$= (0.072)$

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

$= (0.072)$

Condition 3:-

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

$a_1(3) = [0 - 0.2(0.072 + 0.216 + 0.072 + 0.072)]$

$= (-0.0864) = 0$

$a_2(3) = [0.072 - 0.2(0 + 0.216 + 0.072 + 0.072)]$

$= 0$

$a_3(3) = [0.216 - 0.2(0 + 0.072 + 0.072 + 0.072)]$

$= 0.1728$

$a_4(3) = 0$

$a_5(3) = 0$

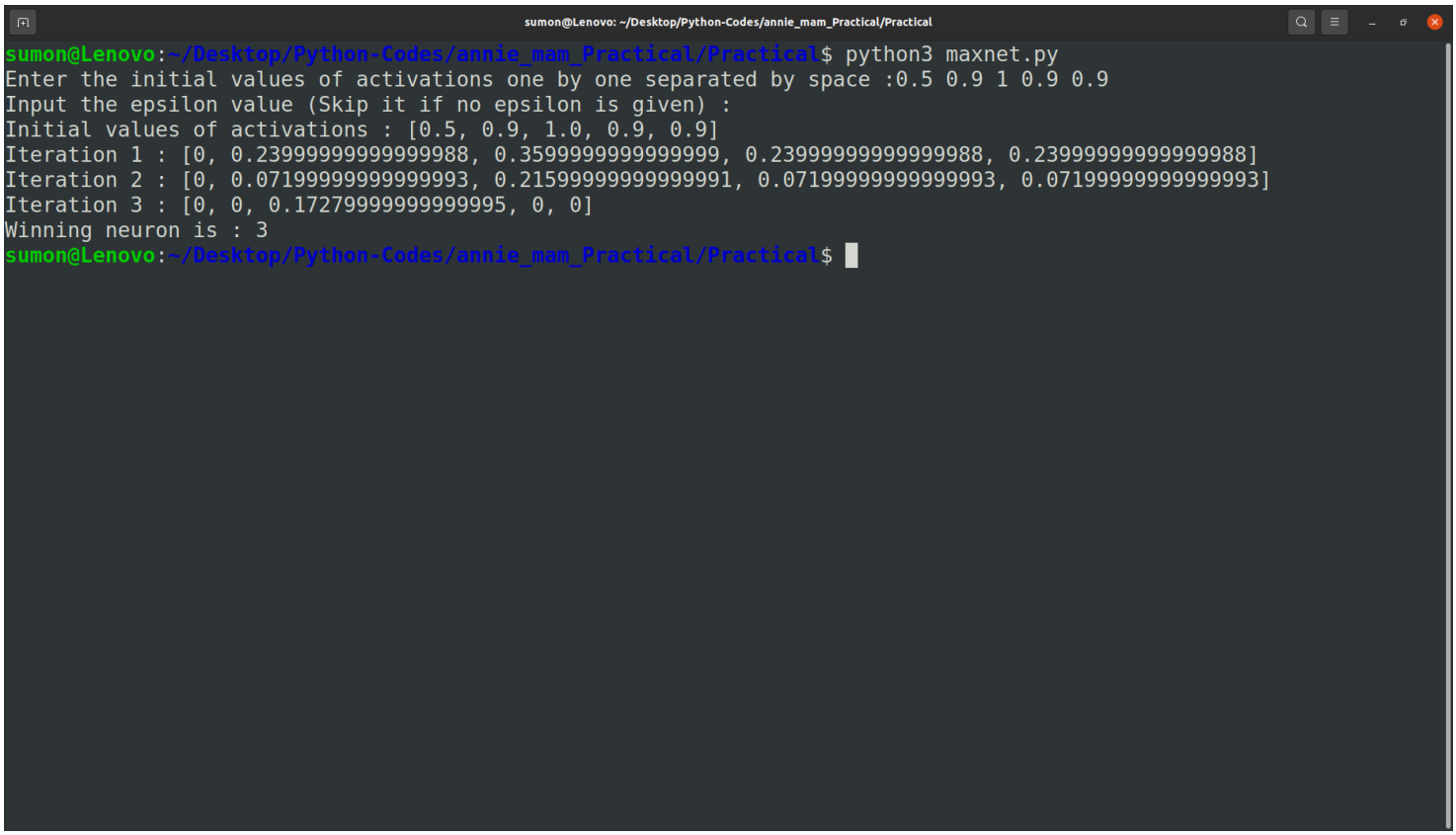
∴ The winning neuron is 3

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:



3] Construct the action of Maxnet with 4 neurons and inhibitory weight € = 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:

$a_1 = 0.3 \quad \epsilon = 0.3$

$a_i = f[a_j(t) - \sum_{i \neq j} a_i(t)]$ → Formula

$a_1 = 0.3 - 0.3[0.3 + 0.5 + 0.7]$

$= f(-0.15) = 0$

$a_2 = 0.3, \epsilon = 0.3$

$a_2 = 0.3 - 0.3[0.3 + 0.5 + 0.7]$

$= f(-0.15) = 0$

$a_3 = 0.5 \quad \epsilon = 0.3$

$a_3 = 0.5 - 0.3[0.3 + 0.3 + 0.7]$

$= f(0.11)$

$a_4 = 0.7$, $\epsilon = 0.3$

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

$= f(0.37)$

Condition 2:-

$[0, 0, 0.11, 0.37]$

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

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

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

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

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

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

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

$= 0.337$

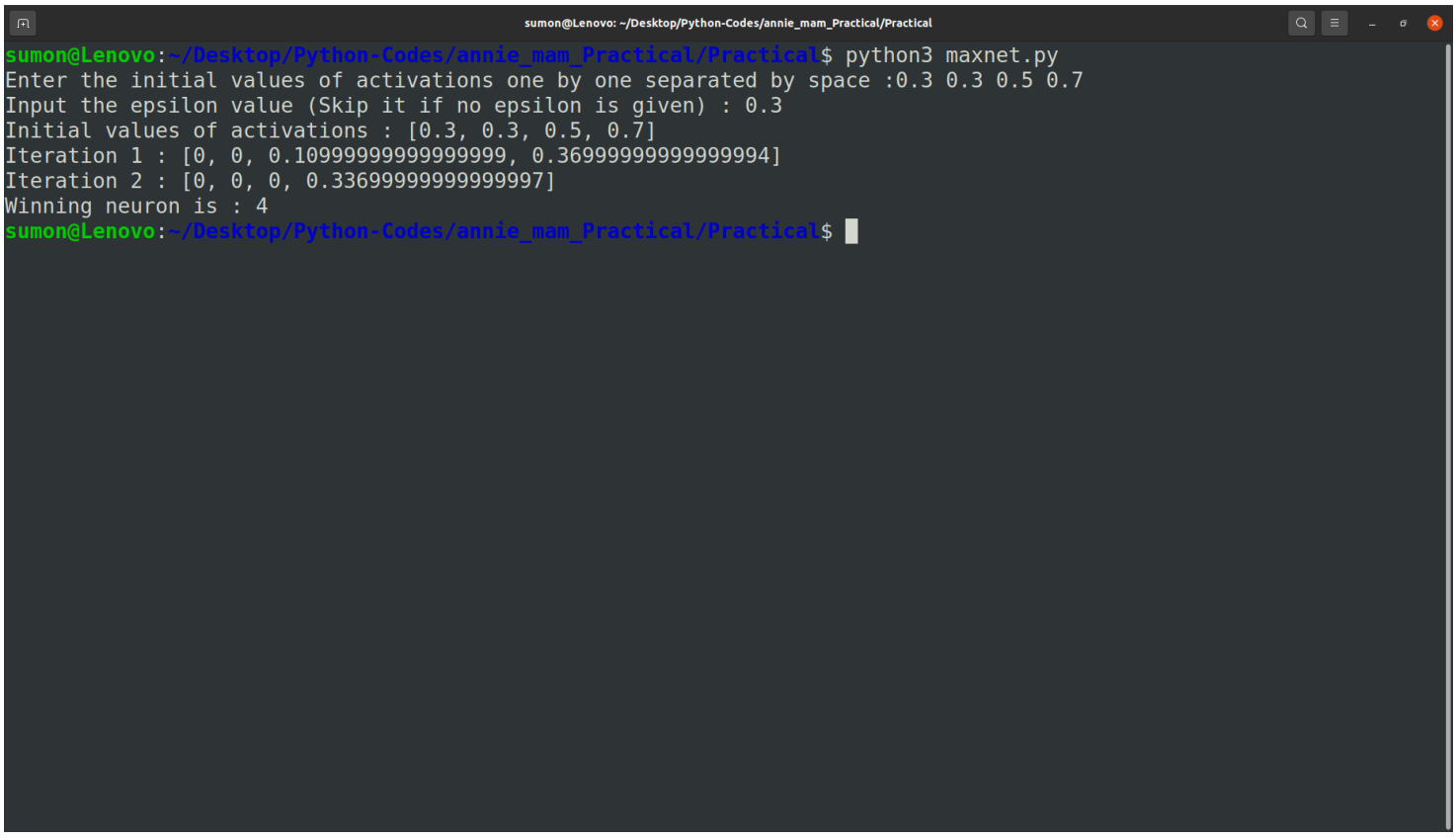
The winning neuron is 4

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:



4] Construct a Maxnet with four neurons and inhibitory weight € =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) = x if x > 0
0 if x ≤ 0

Solution:

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

Condition 1:

$a_1(1) = f[a_i(t) - \epsilon[\Sigma(a_i(t))]]$

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

$= f[-0.12] < 0$

$\therefore a_1(1) < 0$

$a_3(1) = f[a_i(t) - \epsilon[\Sigma(a_i(t))]]$

$= f[0.7 - 0.2(0.3+0.5+0.9)]$

$= f[0.36] > 0$

$\therefore a_3(1) = 0.36$

$a_2(1) = f[a_i(t) - \epsilon[\Sigma(a_i(t))]]$

$= f[0.5 - 0.2(0.3+0.7+0.9)]$

$= f[0.12] > 0$

$\therefore a_2(1) = 0.12$

$a_4(1) = f[a_i(t) - \epsilon[\Sigma(a_i(t))]]$

$= f[0.9 - 0.2(0.3+0.5+0.7)]$

$= f(0.6) > 0$

$\therefore a_4 = 0.6$

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

Condition 2:

$a_1(2) = 0$

$a_2(2) = f[0.12 - 0.2(0.36 + 0.6)]$

$= f(-0.072) < 0$

$\therefore a_2(2) = 0$

$a_3(2) = f[0.36 - 0.2(0.12 + 0.6)]$

$= f(0.216) > 0$

$\therefore a_3(2) = 0.216$

$a_4(2) = f[0.6 - 0.2(0.36 + 0.12)]$

$= f[0.504] > 0$

$\therefore a_4(2) = 0.504$

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

Condition 3:

$a_1(3) = 0$

$a_2(3) = 0$

$a_3(3) = f[0.216 - 0.2(0.504)]$

$= f(0.116) > 0$

$\therefore a_3(3) = 0.116$

$a_4(3) = f[0.504 - 0.2 \times 0.216]$

$= f(0.4608) > 0$

$\therefore a_4(3) = 0.4608$

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

Condition 4:-

$a_1(4) = 0$

$a_2(4) = 0$

$a_3(4) = f[0.116 - 0.2(0.468)]$

$= f(0.02304) > 0$

$\therefore a_3(4) = 0.02304$

$a_4(4) = f[0.4608 - 0.2 \times 0.116]$

$= f[0.4376] > 0$

$\therefore a_4(4) = 0.4376$

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

Condition 5:-

$a_1(5) = 0$

$a_2(5) = 0$

$a_3(5) = f[0.024 - 0.2(0.4376)$

$= f(-0.06) < 0$

$\therefore a_3(5) = 0$

$a_4(5) = f[0.438 - 0.2(0.024)]$

$= f(0.4332) > 0$

$\therefore a_4(5) = 0.4332$

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

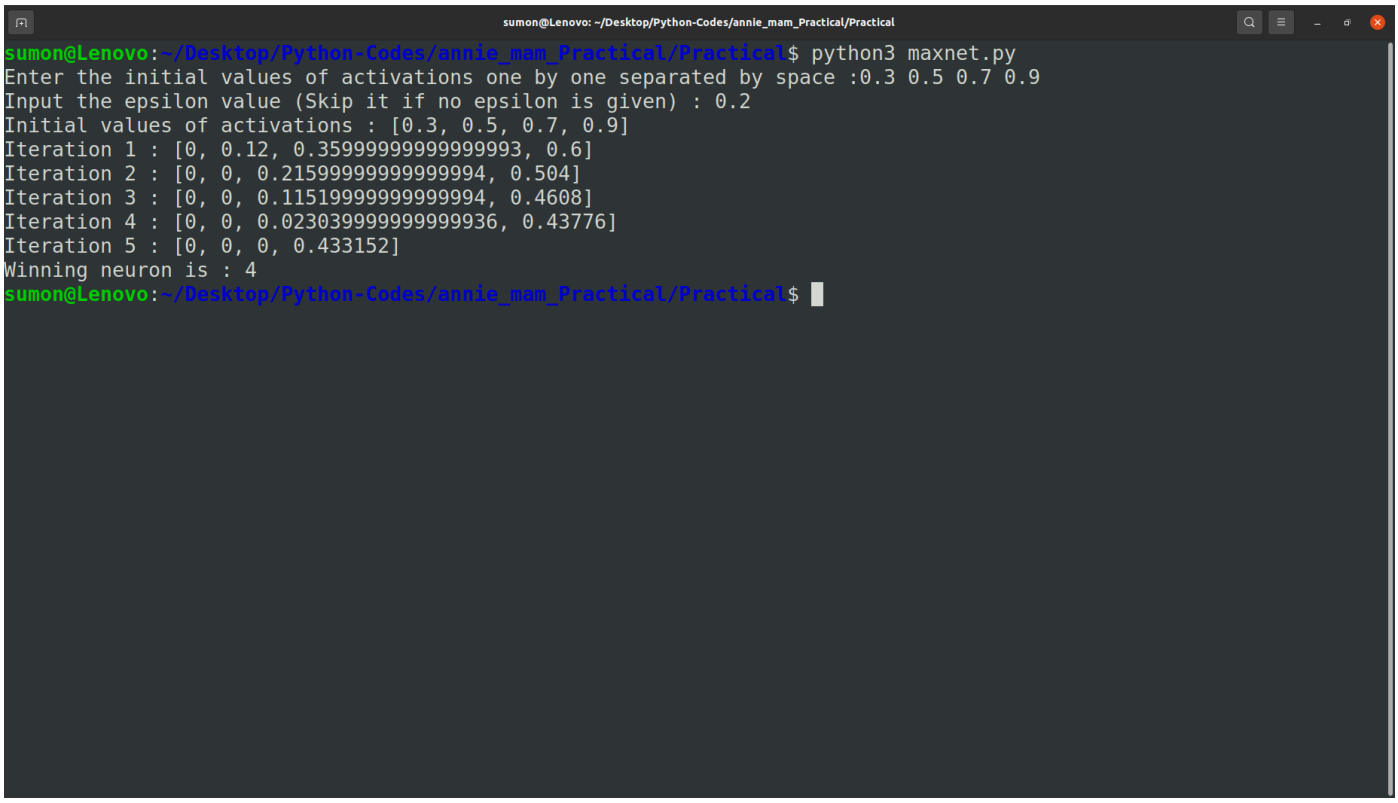
$\therefore$ The winning neuron is the 4th neuron.

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:



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

$X_1 = [-1\ -1\ 1\ -1]$, $X_2 = [-1\ -1\ 1\ 1]$, $X_3 = [-1\ -1\ -1\ 1]$,

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

* $n_y \rightarrow$ exemplar vector

$$\omega = \begin{bmatrix} 1 & -1 \\ -1 & -1 \\ -1 & -1 \\ -1 & 1 \end{bmatrix}$$ initial weight is: $\frac{1}{2} \times$ exemplar vector

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

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

1st iteration :-

$X_1 = [-1\ -1\ 1\ -1]$ with ω_1

$y(1) = b + \sum n_i \omega_1$

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

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

$= 2 + [0]$

$y(1) = 2$

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

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

$= 2 + [0]$

$y(2) = 2$

As $y(1) = y(2)$ so any one of them has best match ~~ba~~ exemplar for X_1.

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

$\therefore y(2)$ has best match exemplar for X_2

3rd iteration :-

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

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

$$= 2 + [0] = 2$$

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

$$= 2 + [2] = 4$$

As, $y(2) > y(1)$ so $y(2)$ has best match exemplar for X_3

4th iteration:-

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

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

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

$$= 2 + 1 = 3$$

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

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

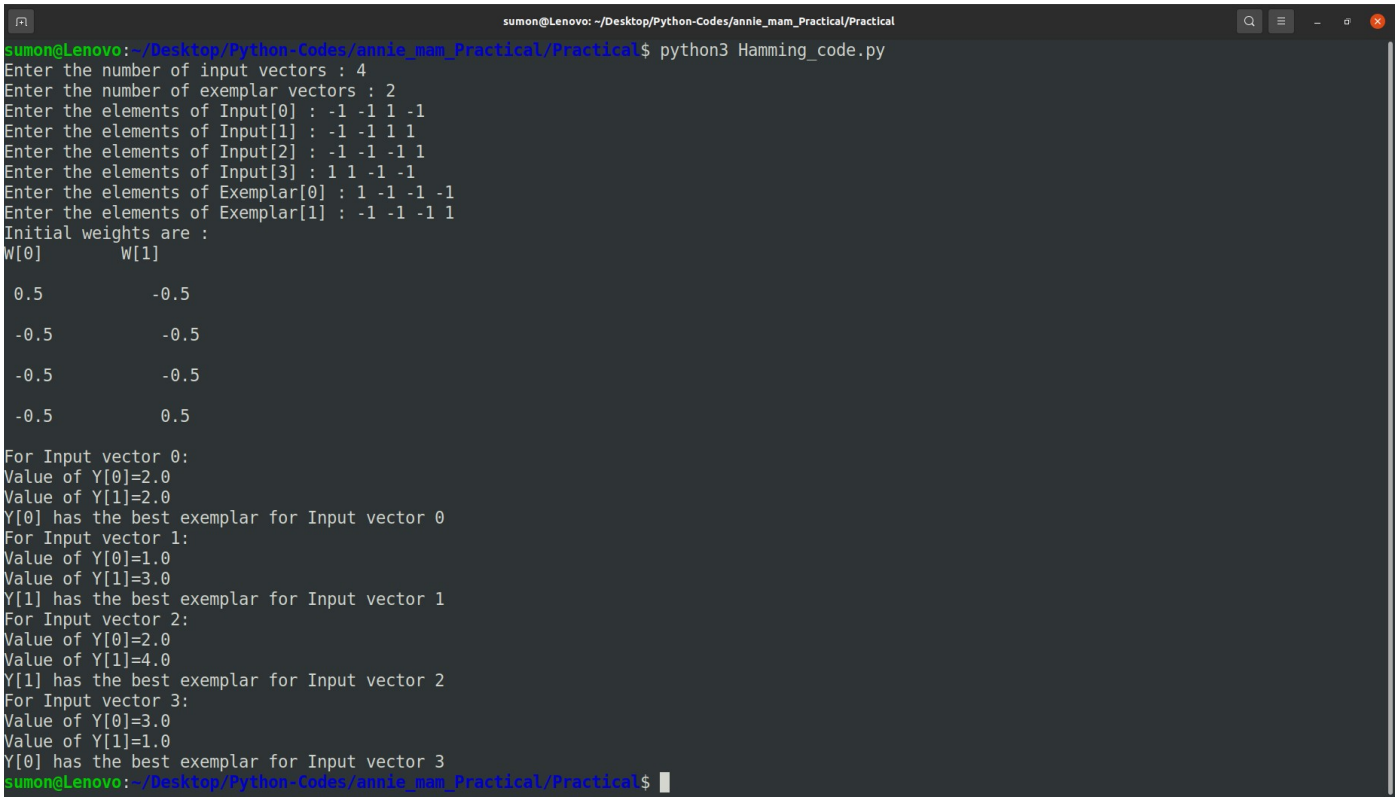
As, $y(1) > y(2)$ so $y(1)$ has best match exemplar for x_4

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:



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:

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

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

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

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

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

Now, bias is $b = \frac{n}{2} = \frac{4}{2} = 2$

1st iteration:-

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

$$= 2 + 1$$

$$= 3$$

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

$$= 2 + (-1)$$

$$= 1$$

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

2nd iteration :-

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

$$= 2 + 2 = 4$$

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

$$= 2$$

As, $y(1) > y(2)$ so $y(1)$ is best match exemples for X_2

3rd iteration :-

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

$$= 2$$

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

As, $y(2) > y(1)$ so $y(2)$ is best match exemples for X_3

4th iteration :-

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

$$= 2 + (-1) = 1$$

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

$$= 3$$

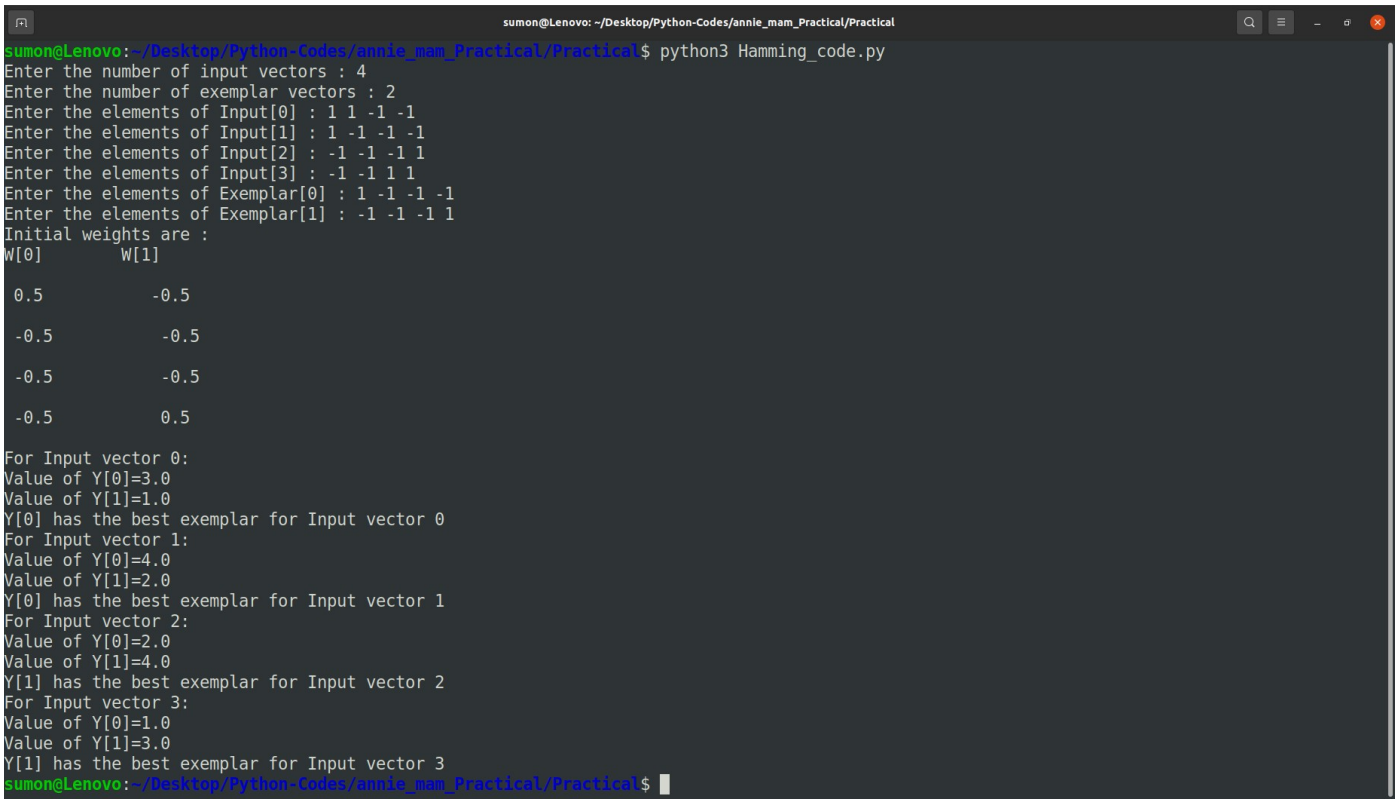
As, $y(2) > y(1)$ so $y(2)$ is best match exemples for X_4

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:



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:

1) Kohonen

1) $W = \begin{bmatrix} w_1 & w_2 \\ 1.0 & 0.3 \\ 0.9 & 0.5 \\ 0.7 & 0.7 \\ 0.5 & 0.9 \\ 0.3 & 1 \end{bmatrix}$

$X = [0.0 \quad 0.5 \quad 1.0 \quad 0.5 \quad 0.0]$

$\alpha = 0.25$

$D(i) = \sum (w_{ij} - x_i)^2 \rightarrow$ Square of euclidean distance

$i = 1$ to 5

For w_1 $\leftarrow D(1) = (1-0)^2 + (0.9-0.5)^2 + (0.7-1)^2 + (0.5-0.5)^2 + (0.3-0.0)^2$

$= 1 + 0.16 + 0.09 + 0.09$

$= 1.34$

For w_2 $\leftarrow D(2) = (0.3-0.0)^2 + (0.5-0.5)^2 + (0.7-1)^2 + (0.9-0.5)^2 + (1-0.0)^2$

$= 0.09 + 0.09 + 0.16 + 1 = 1.34$

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

$\alpha = 0.25$

$w_{ij}(new) = w_{ij}(old) + \alpha(x_i - w_{ij}(old))$

$w_{11}(new) = 1 + 0.25(0-1)$

$w_{11}=1$
$x_1=0$

$w_{11}(new) = 0.75$

$w_{21}(new) = 0.9 + 0.25(0.5-0.9)$

$= 0.8$

$w_{31}(new) = 0.7 + 0.25(1-0.7)$

$= 0.775$

$w_{41}(new) = 0.5 + 0.25(0.5-0.5)$

$= 0.5$

$w_{51}(new) = 0.3 + 0.25(0.0-0.3)$

$= 0.225$

The updated weight is

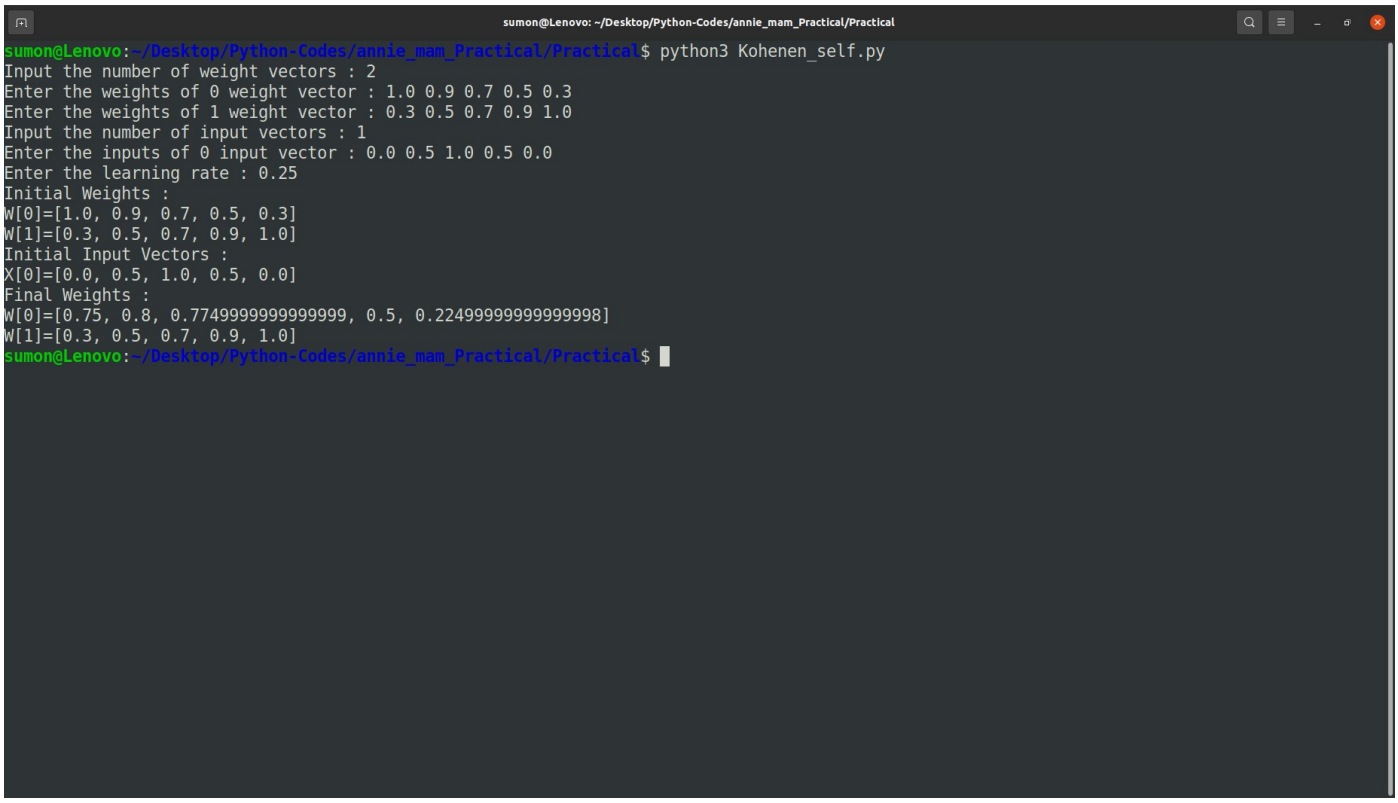
$$W = \begin{bmatrix} 0.75 & 0.3 \\ 0.8 & 0.5 \\ 0.77 & 0.7 \\ 0.5 & 0.9 \\ 0.22 & 1 \end{bmatrix}$$

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
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 : 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, 0.22499999999999998]
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 w2 = [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.

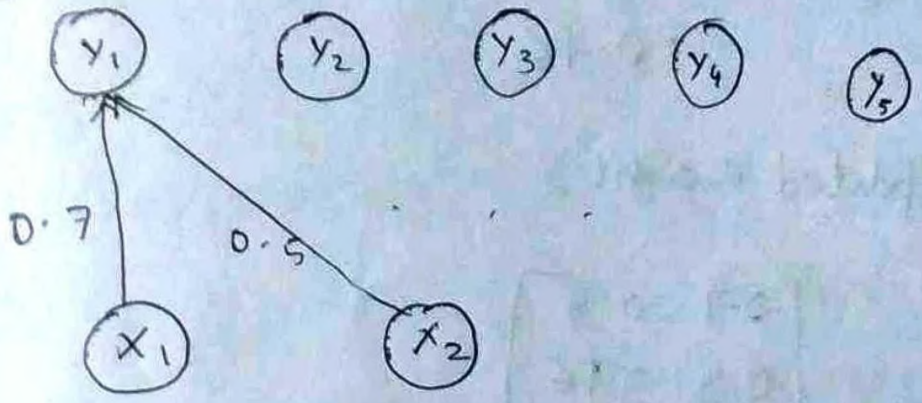
Solution:

2) $W = [w_1, w_2]$

	0	1
0	0.7	0.5
1	0.3	0.2
2	0.1	0.4
3	0.6	0.9
4	1.0	0.5

$X = (0.35, 0.05)$ (x_1, x_2)

$\alpha = 0.25$



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

$= 0.325$

$D(2) = (0.3 - 0.35)^2 + (0.2 - 0.05)^2$

$= 0.025$

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

$= 0.185$

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

$= 0.785$

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

Min :- $D(2) = 0.025$, so update the weight of $(\omega_{21}, \omega_{22})$

$$\omega_{12}(new) = 0.3 + 0.25(0.35 - 0.3)$$
$$= 0.31$$

$$\omega_{22}(new) = 0.2 + 0.25(0.05 - 0.2)$$
$$= 0.16$$

Updated weight :-

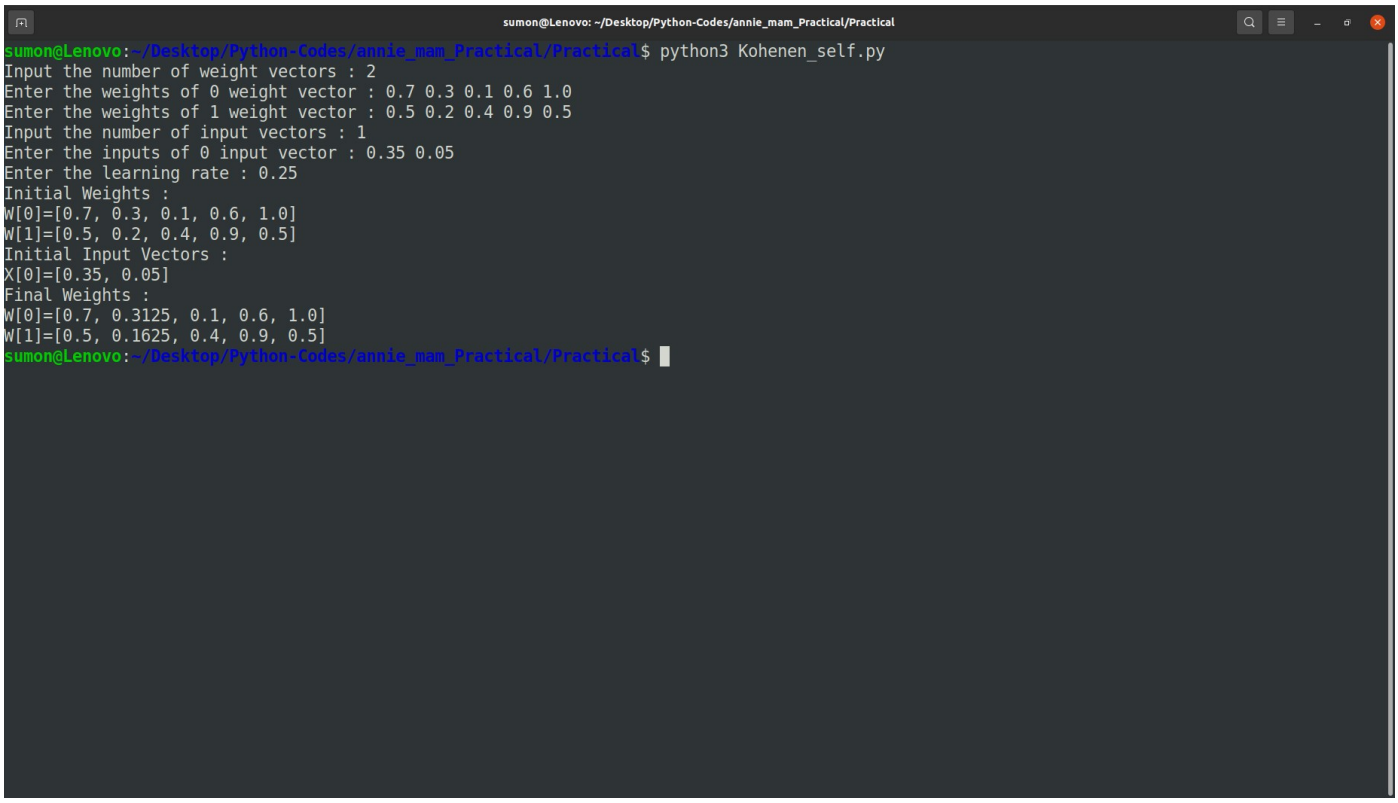
$$\begin{bmatrix} 0.7 & 0.5 \\ 0.3 & 0.16 \\ 0.1 & 0.4 \\ 0.6 & 0.9 \\ 1.0 & 0.5 \end{bmatrix}$$

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)
eturn 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
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 : 0.7 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:

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

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

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

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

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

$\alpha = 0.5$

Condition (1) :-

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

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

$$= 0.4$$

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

$$= 2.04$$

As $D(2) > D(1)$ so update weights w_1 col

new $w_0 = \begin{bmatrix} 0.1 & 0.9 \\ 0.2 & 0.7 \\ 0.8 & 0.5 \\ 0.9 & 0.3 \end{bmatrix}$

for 2nd input: $X_2 = [1\ 0\ 0\ 0]$

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

$= 2.5$

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

$= 0.84$

$D(1) > D(2)$ so update w_2 col

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

$= 0.95$

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

$= 0.35$

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

$= 0.55$

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

$= \cancel{0.45}\ 0.15$

Updated weight :-

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

For 3rd. input :- $X_3 = [0\ 1\ 1\ 0]$

$$D(1) = (0.1)^2 + (1-0.2)^2 + \cancel{(0.8)^2 + (0.9)^2}$$
$$+ (1-0.8)^2 + (0.9)^2$$
$$= 1.5$$

$$D(2) = (0.95)^2 + (1-0.35)^2 + (1-0.55)^2$$
$$+ (0.15)^2$$

$$= 1.55$$

$D(1) < D(2)$ update col 1 (w_1)

$a_{1n}(new) = 0.25 + 0.5(0 - 0.25)$

$= 0.125$

Updated weights :-

$$W_2 = \begin{bmatrix} 0.375 & 0 \\ 0.5 & 1 \\ 0.75 & 0 \\ 0.125 & 1 \end{bmatrix}$$

For 4th input :-

$X_4 = [0 \quad 0 \quad 0 \quad 1]$

$D(1) = 1.71$

$D(2) = 1$

$D(1) > D(2)$

update W_2 col

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

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

$w_{23}(new) = 0 + 0.5(0 - 0) = 0$

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

updated weights :-

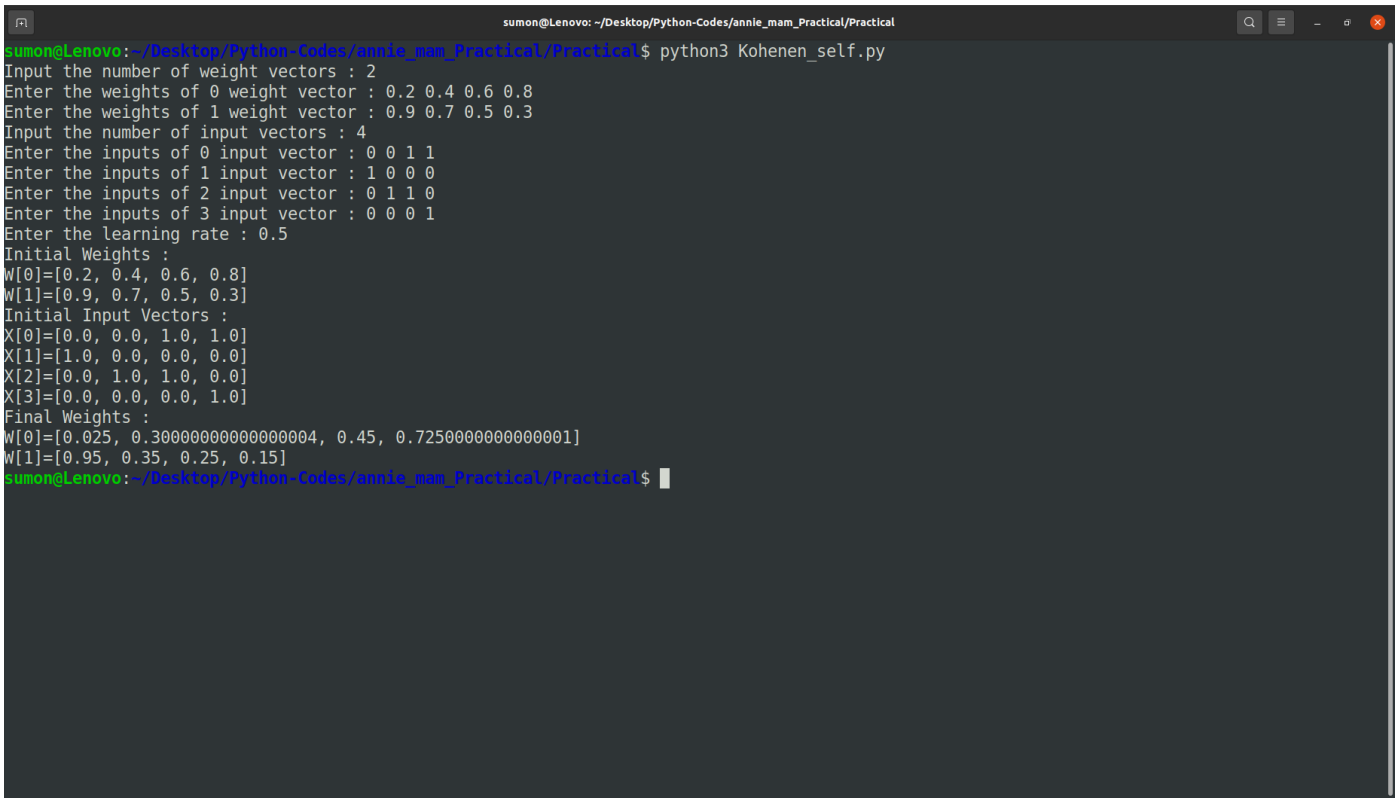
$$W = \begin{bmatrix} 0.375 & 0 \\ 0.5 & 0.5 \\ 0.75 & 0 \\ 0.125 & 1 \end{bmatrix}$$

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
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, 0.30000000000000004, 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:

4) Initial weights:-

$$W = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \qquad \alpha = 0.5$$

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 w_1 col;

$$w_{11}(new) = 1 + 0.5(0 - 1)$$
$$= 0.5$$

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

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

$w_{14} = 0 + 0.5(1-0)$

$= 0.5$

Updated weight

$$w_0 = \begin{bmatrix} 0.5 & 0 \\ 0 & 1 \\ 1 & 0 \\ 0.5 & 1 \end{bmatrix}$$

For 2nd :- $x_2 = [1\ 0\ 0\ 0]$

Ans

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

$= 1.5$

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

$D(1) < D(2)$ update w_1 col

$w_{11}(new) = 0.5 + 0.5(1-0.5)$

$= 0.75$

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

$w_{13}(new) = 1 + 0.5(0-1) = 0.5$

$w_{14}(new) = 0.5 + 0.5(-0.5) = 0.25$

Updated weights:-

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

For 3rd. input :- $x_3 = [0 \quad 1 \quad 1 \quad 0]$

$D(1) = 1.875$

$D(2) = 2$

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

update col ω_i

$\omega_{11}(new) = 0.75 + 0.5(0 - 0.75)$

$= 0.375$

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

$= 0.5$

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

$= 0.75$

$a_{1n}(new) = 0.25 + 0.5(0 - 0.25)$

$= 0.125$

Updated weights :-

$$W_2 = \begin{bmatrix} 0.375 & 0 \\ 0.5 & 1 \\ 0.75 & 0 \\ 0.125 & 1 \end{bmatrix}$$

For 4th input :-

$X_4 = [0 \quad 0 \quad 0 \quad 1]$

$D(1) = 1.71$

$D(2) = 1$

$D(1) > D(2)$

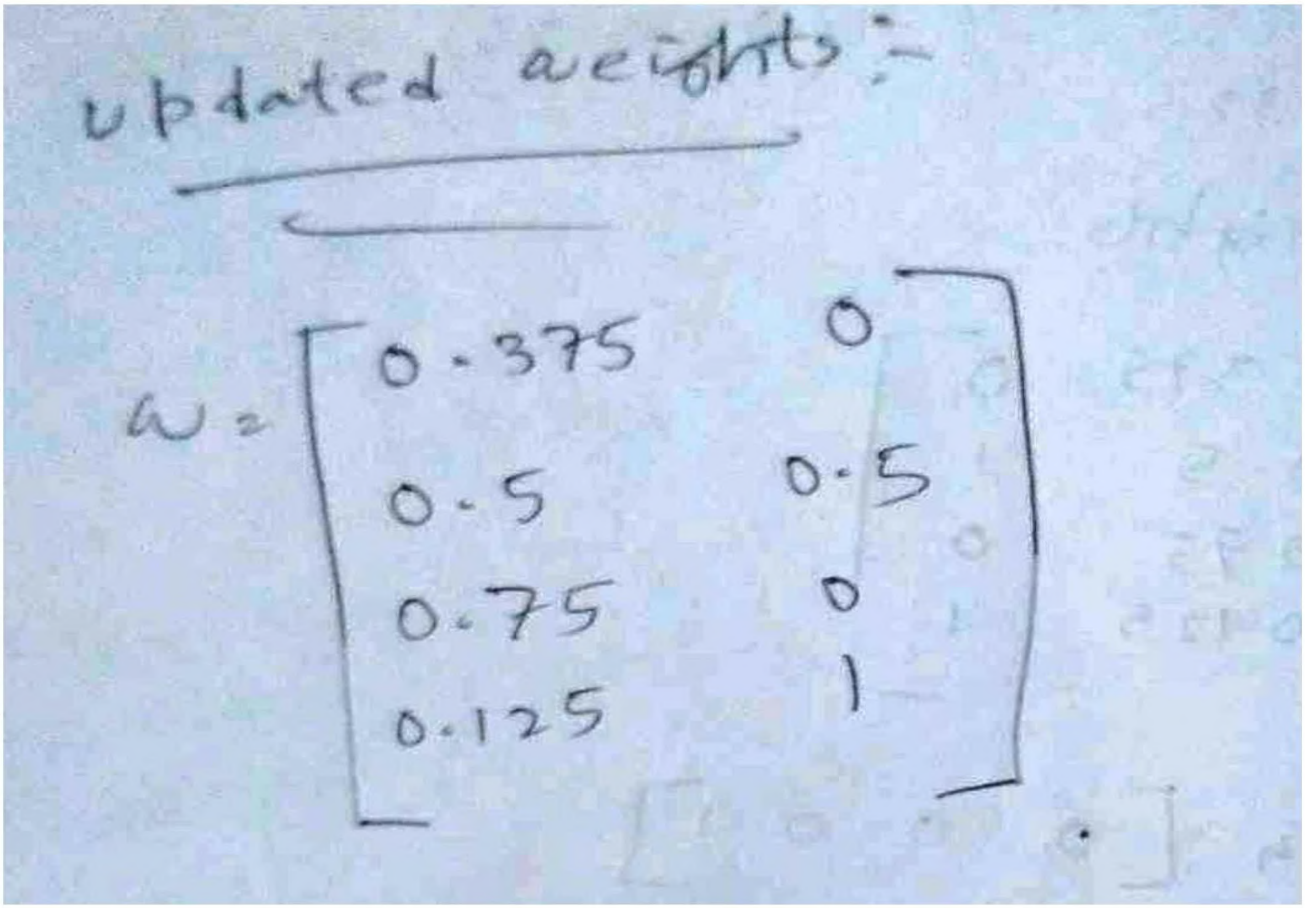
update W_2 col

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

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

$w_{23}(new) = 0 + 0.5(0 - 0) = 0$

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

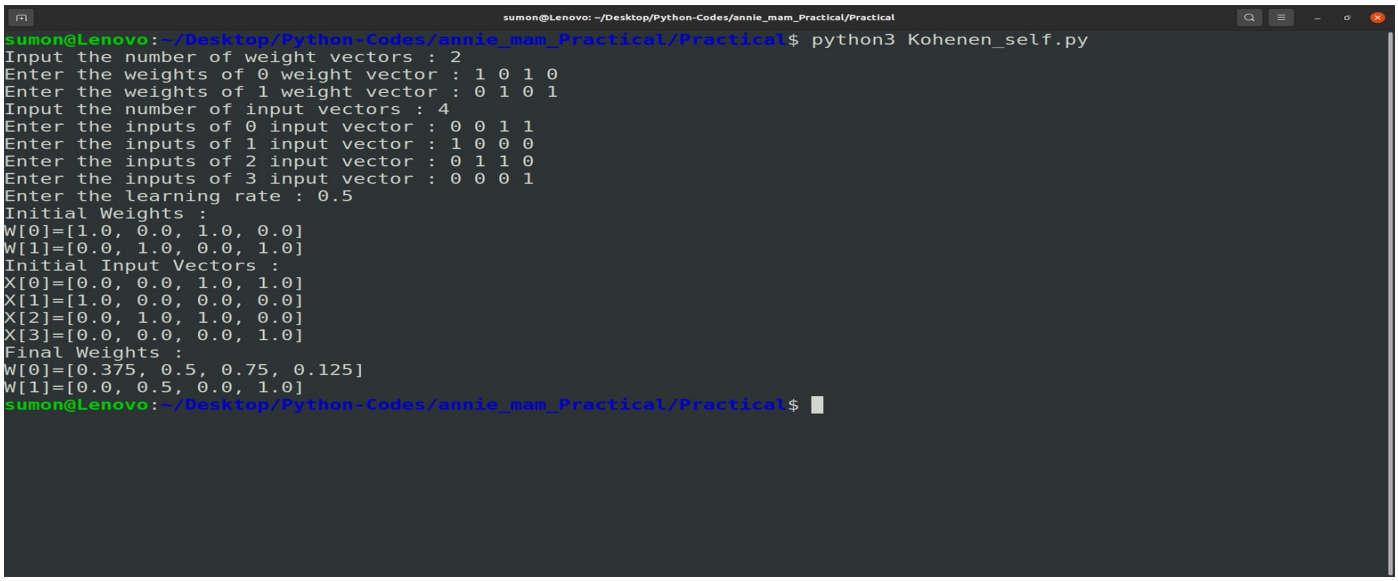

updated weights :-
w =
0.375 0
0.5 0.5
0.75 0
0.125 1

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
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 : 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	Input(s)	Target(t)
E	[1 1 1 1 -1 -1 1 1 1 1 -1 -1 1 1 1]	[-1 1]
f	[1 1 1 1 1 1 1 -1 -1 1 -1 -1 1 -1 -1]	[1 1]

$$w = \sum s^T t$$

$$w_1 = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ -1 \\ -1 \\ 1 \\ 1 \\ 1 \\ 1 \\ -1 \\ -1 \\ 1 \\ 1 \\ 1 \end{bmatrix} \begin{bmatrix} -1 & 1 \end{bmatrix} = \begin{bmatrix} -1 & 1 \\ -1 & 1 \\ -1 & 1 \\ -1 & 1 \\ 1 & -1 \\ 1 & -1 \\ -1 & 1 \\ -1 & 1 \\ -1 & 1 \\ -1 & 1 \\ 1 & -1 \\ 1 & -1 \\ -1 & 1 \\ -1 & 1 \\ -1 & 1 \end{bmatrix} \rightarrow w_1$$

$$w_2 = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ -1 \\ -1 \\ 1 \\ -1 \\ -1 \\ 1 \\ -1 \\ -1 \end{bmatrix} \begin{bmatrix} 1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \\ 1 & 1 \\ 1 & 1 \\ 1 & 1 \\ 1 & 1 \\ 1 & 1 \\ -1 & -1 \\ -1 & -1 \\ 1 & 1 \\ -1 & -1 \\ -1 & -1 \\ 1 & 1 \\ -1 & -1 \\ -1 & -1 \end{bmatrix} \rightarrow w_2$$

$\therefore W = W_1 \cdot W_2 =$

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

Now, $X_{in} \propto W$

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

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

$= [-12, 18]$

$= (-1 \quad 1)$

Now, $f * \omega$

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

$= [12, 16]$

$= (1, 1)$

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

Now, for pattern E (-1 1)

$$X * \omega^T = \begin{bmatrix} 1&1&1&1&-1&-1&1&1&1&1&-1&-1&1&1&1 \end{bmatrix}$$

Now, for pattern F (1, 1)

$$X * \omega^T = \begin{bmatrix} 1&1&1&1&1&1&1&1&-1&-1&1&-1&-1&1&-1&-1 \end{bmatrix}$$

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

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

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

target $I = [1, -1]$ $C = [-1, 1]$

weight $= \sum S^T \cdot t = \sum I^T t$

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

$$\omega_2 = \begin{bmatrix} \sum C^T t \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ -1 \\ 1 \\ 1 \end{bmatrix} [-1\ 1] = \begin{bmatrix} -1 & 1 \\ -1 & 1 \\ -1 & 1 \\ -1 & 1 \\ -1 & 1 \\ 1 & -1 \\ -1 & 1 \\ -1 & 1 \end{bmatrix}$$

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

$x_1 = [1\ 1\ 1\ -1\ 1\ 1\ 1\ 1]$

$x_1 \times W = [1\ 1\ 1\ -1\ 1\ 1\ 1\ 1] \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ -2 & 2 \\ 0 & 0 \\ 2 & -2 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} = [2+2 \quad -2-2] = [4\ -4]$

$x_2 \times W = [1\ 1\ 1\ 1\ 1\ -1\ 1\ 1] \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ -2 & 2 \\ 0 & 0 \\ 2 & -2 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} = [-2-2 \quad 2+2] = [-4\ 4]$

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 :- $[1, -1]$

So $I \times W^T = [1\ -1] \begin{bmatrix} 0 & 0 & 0 & -2 & 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2 & 0 & -2 & 0 & 0 & 0 \end{bmatrix}$

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

As per activation func^n :- $[1\ 1\ 1\ -1\ 1\ 1\ 1\ 1]$

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

So, $[-1\ 1] \begin{bmatrix} 0 & 0 & 0 & -2 & 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2 & 0 & -2 & 0 & 0 & 0 \end{bmatrix}$

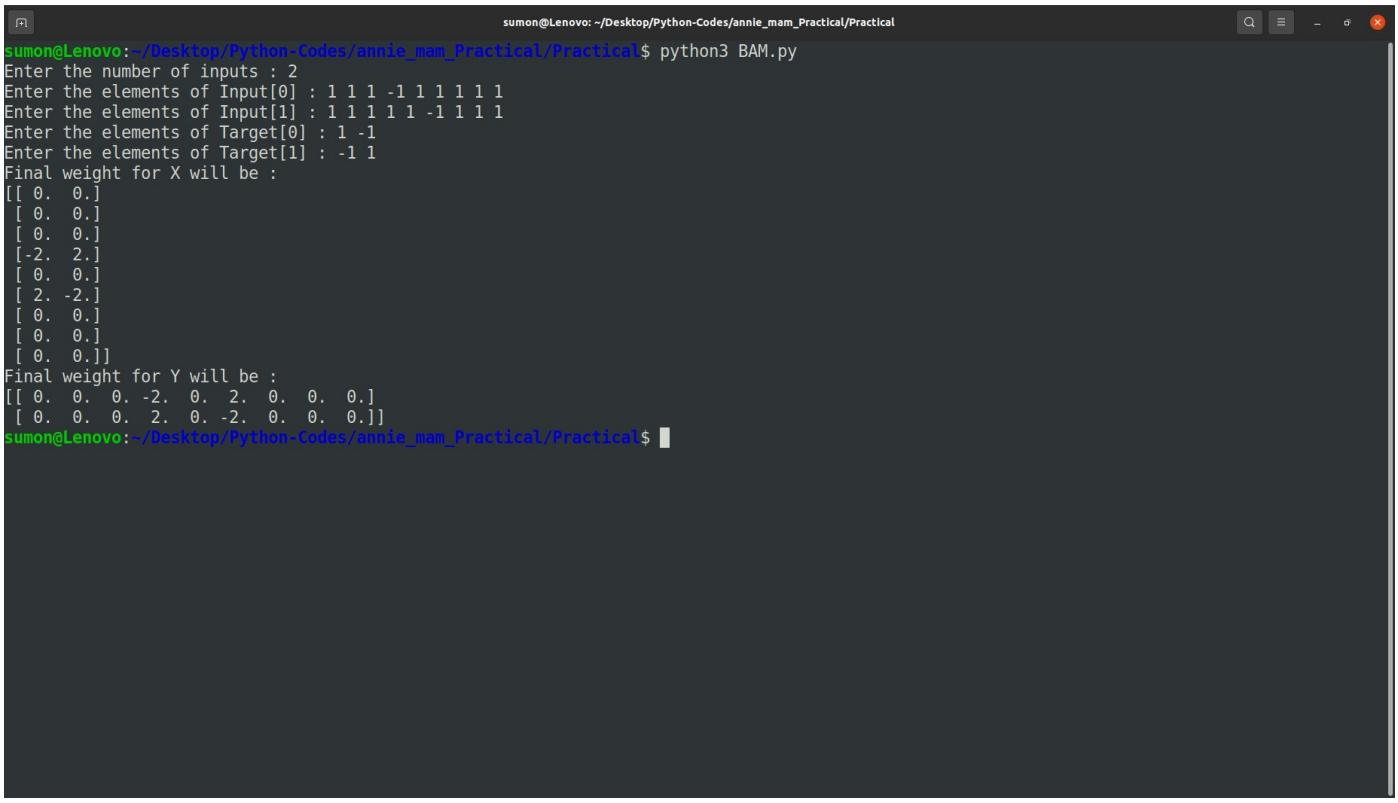
$= [0\ 0\ 0\ 4\ 0\ -4\ 0\ 0\ 0]$

As per activation func^n :- $[1\ 1\ 1\ 1\ 1\ -1\ 1\ 1] = C$

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
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. 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:

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

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

union :- $\underline{L} \cup \underline{M} = \mu_L(x) \cup \mu_M(x) = \max\left(\mu_L(x), \mu_M(x)\right)$

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

Intersection :- $\mu_L(x) \cap \mu_M(x) = \min\left(\mu_L(x), \mu_M(x)\right)$

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

Complement :-

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

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

Difference :-

$\underline{L}/\underline{M} = \underline{L} \cap \overline{\underline{M}} = \underline{\mu_L(x)} \cap \overline{\underline{\mu_M(x)}}$

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

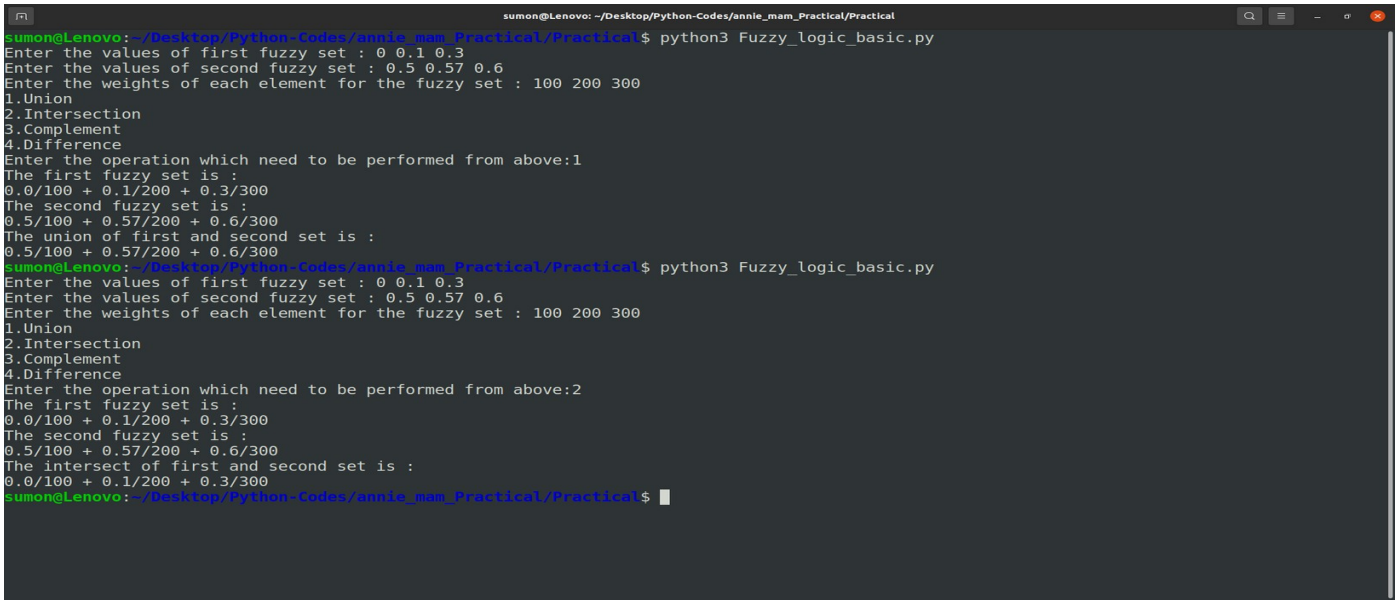
Code:

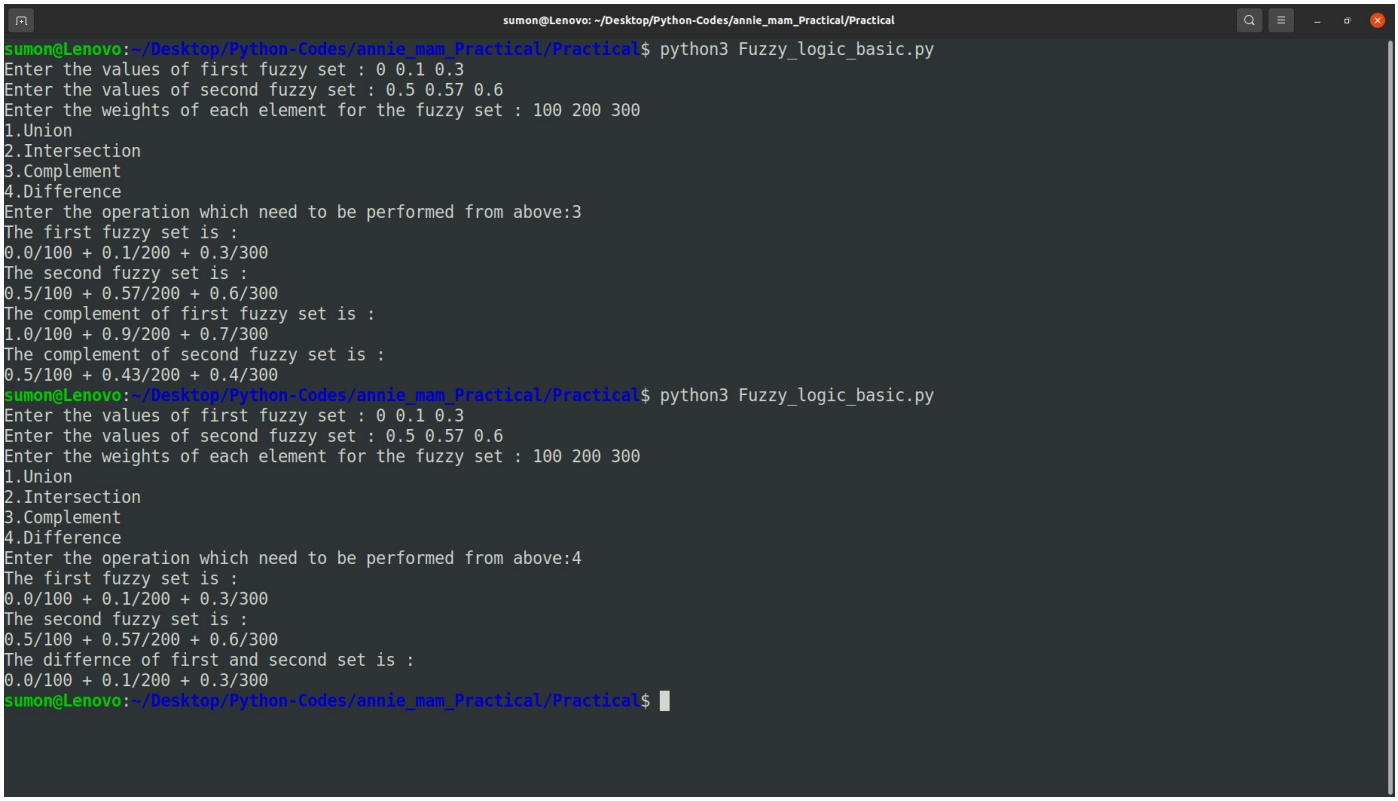
```
def union(A,B):
R = []
for i in range(0, len(A)):
if(A[i] > B[i]):
R.append(A[i])
else:
R.append(B[i])
return R
def intersect(A,B):
R = []
for i in range(0, len(A)):
if(A[i] < B[i]):
R.append(A[i])
else:
R.append(B[i])
return R
def complement(A):
R = []
for i in range(0,len(A)):
V = 1 - A[i]
R.append(V)
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:





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 :- $\underline{A} \cup \underline{B} = \mu_{\underline{A}}(x) \cup \mu_{\underline{B}}(x)$

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

Intersection :- $\underline{A} \cap \underline{B} = \mu_{\underline{A}}(x) \cap \mu_{\underline{B}}(x)$

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

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 :-

$\underline{A} \mid \underline{B} = \mu_{\underline{A}}(x) \cap \overline{\mu_{\underline{B}}(x)}$

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

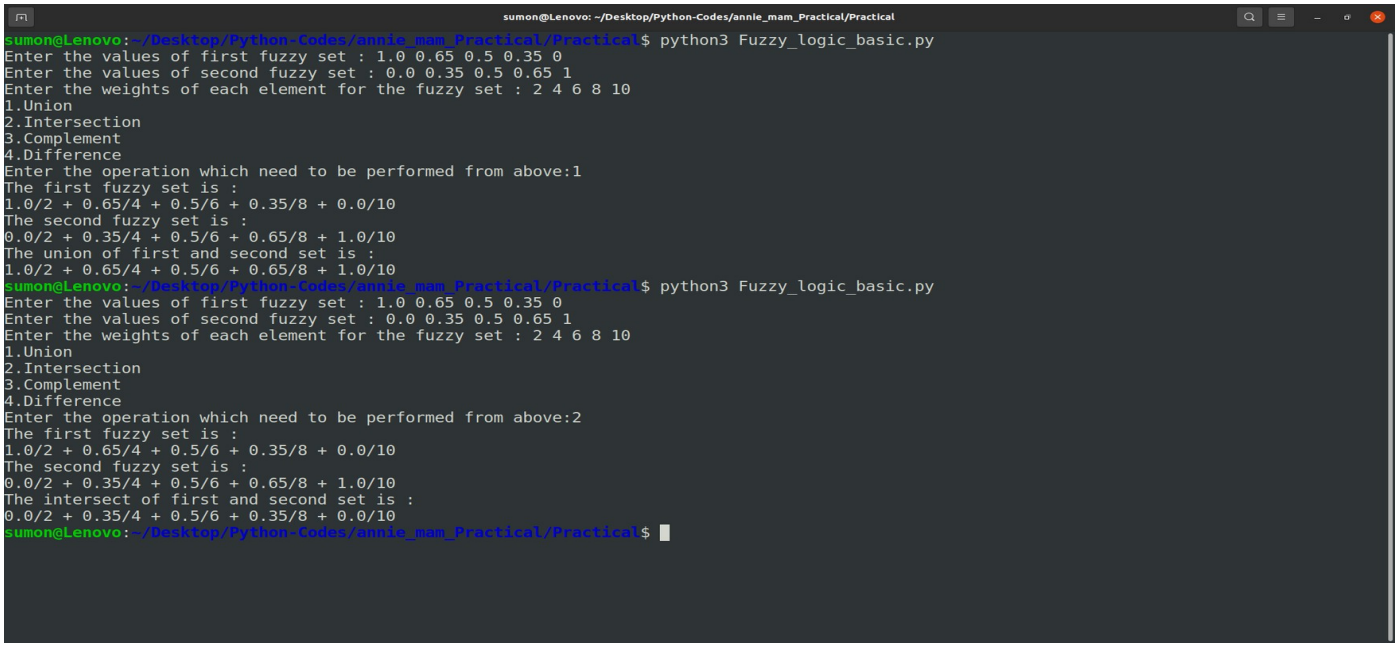
Code:

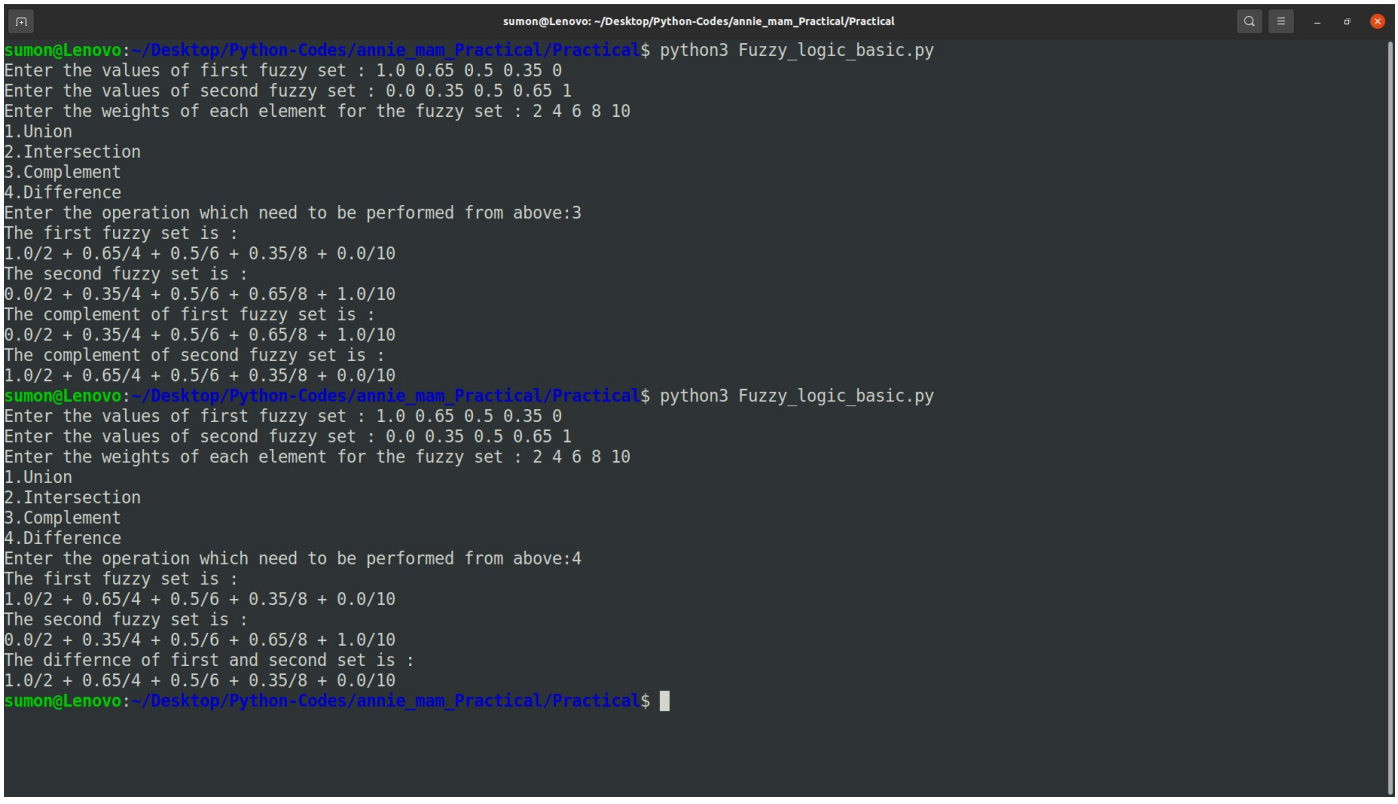
```
def union(A,B):
R = []
for i in range(0, len(A)):
if(A[i] > B[i]):
R.append(A[i])
else:
R.append(B[i])
return R
def intersect(A,B):
R = []
for i in range(0, len(A)):
if(A[i] < B[i]):
R.append(A[i])
else:
R.append(B[i])
return R
def complement(A):
R = []
for i in range(0,len(A)):
V = 1 - A[i]
R.append(V)
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:





3] Implement Union, Intersection, Complement and Difference on Fuzzy set.
I1= {0/0 + 0.5/20 + 0.35/40 + 0.75/60 + 0.95/80 + 1/100}
I2 = {0/0 + 0.45/20 + 0.55/40 + 0.65/60 + 0.9/80 +1/100}

Solution:

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

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

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

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

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

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

Complement :-

$\overline{I_1} = \left\{ \frac{1}{0} + \frac{0.5}{20} + \frac{0.65}{40} + \frac{0.25}{60} + \frac{0.05}{80} + \frac{0}{100} \right\}$

$\overline{I_2} = \left\{ \frac{1}{0} + \frac{0.55}{20} + \frac{0.45}{40} + \frac{0.35}{60} + \frac{0.1}{80} + \frac{0}{100} \right\}$

Difference :-

$I_1 | I_2 = \mu_{I_1}(x) \cap \overline{\mu_{I_2}(x)}$

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

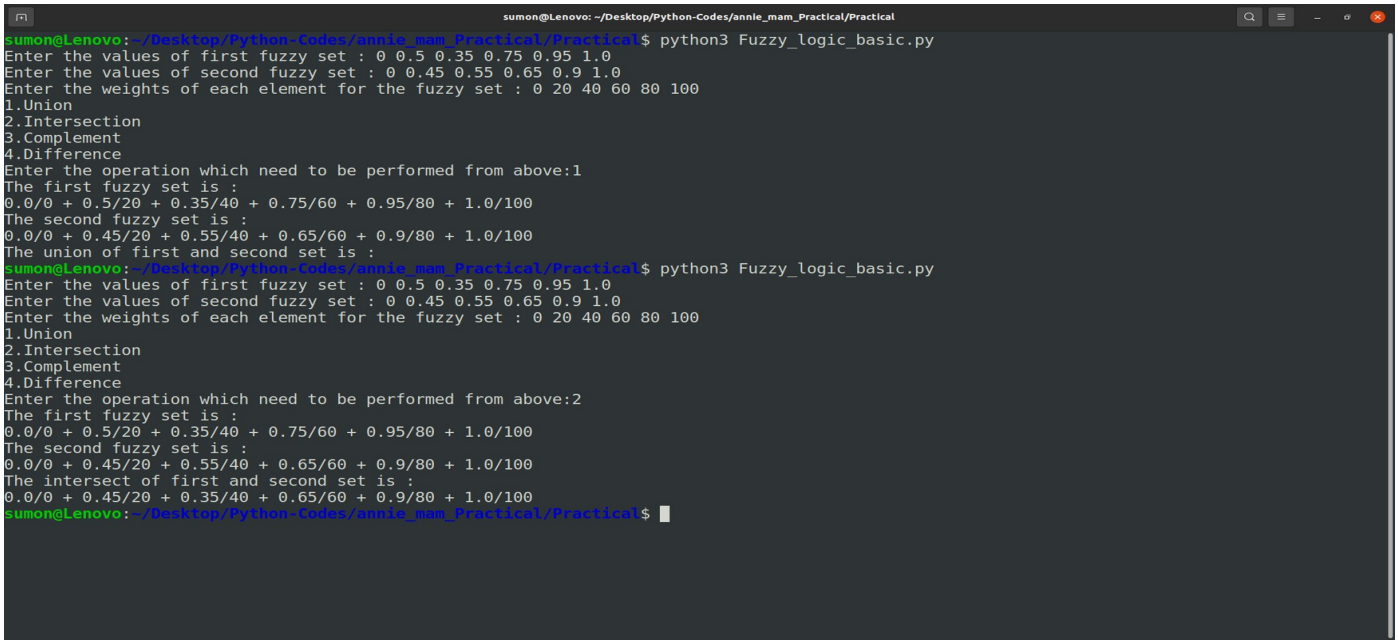
Code:

```
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: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.
T1 = {0/0 + 0.2/1 + 0.3/2 + 0.6/3 + 0.9/4 + 1/5}
T2 = {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 :- $T_1 \cup T_2 = \mu_{T_1}(x) \cup \mu_{T_2}(x)$

$= \left\{ \frac{0}{0} + \frac{0.1}{1} + \frac{0.2}{2} \right\}$

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

Intersection :- $T_1 \cap T_2 = \mu_{T_1}(x) \cap \mu_{T_2}(x)$

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

Complement :

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

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

Difference :

$T_1 | T_2 = \mu_{T_1}(x) \cap \overline{\mu_{T_2}(x)}$

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

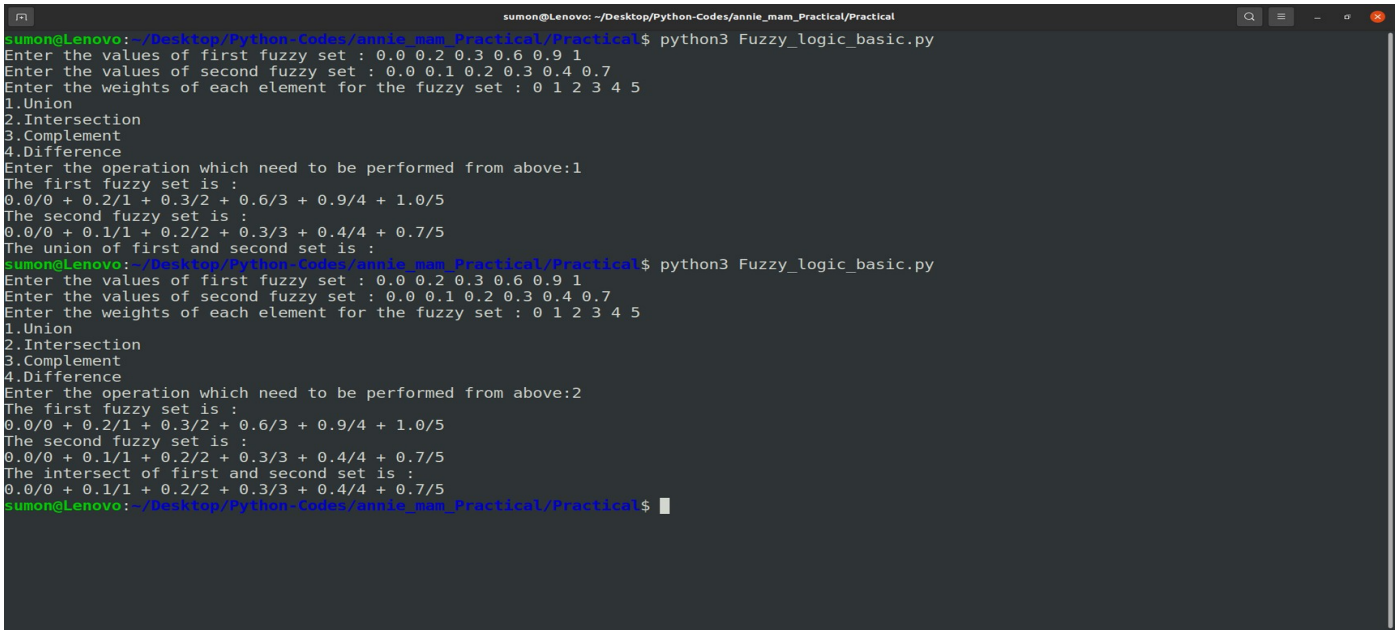
Code:

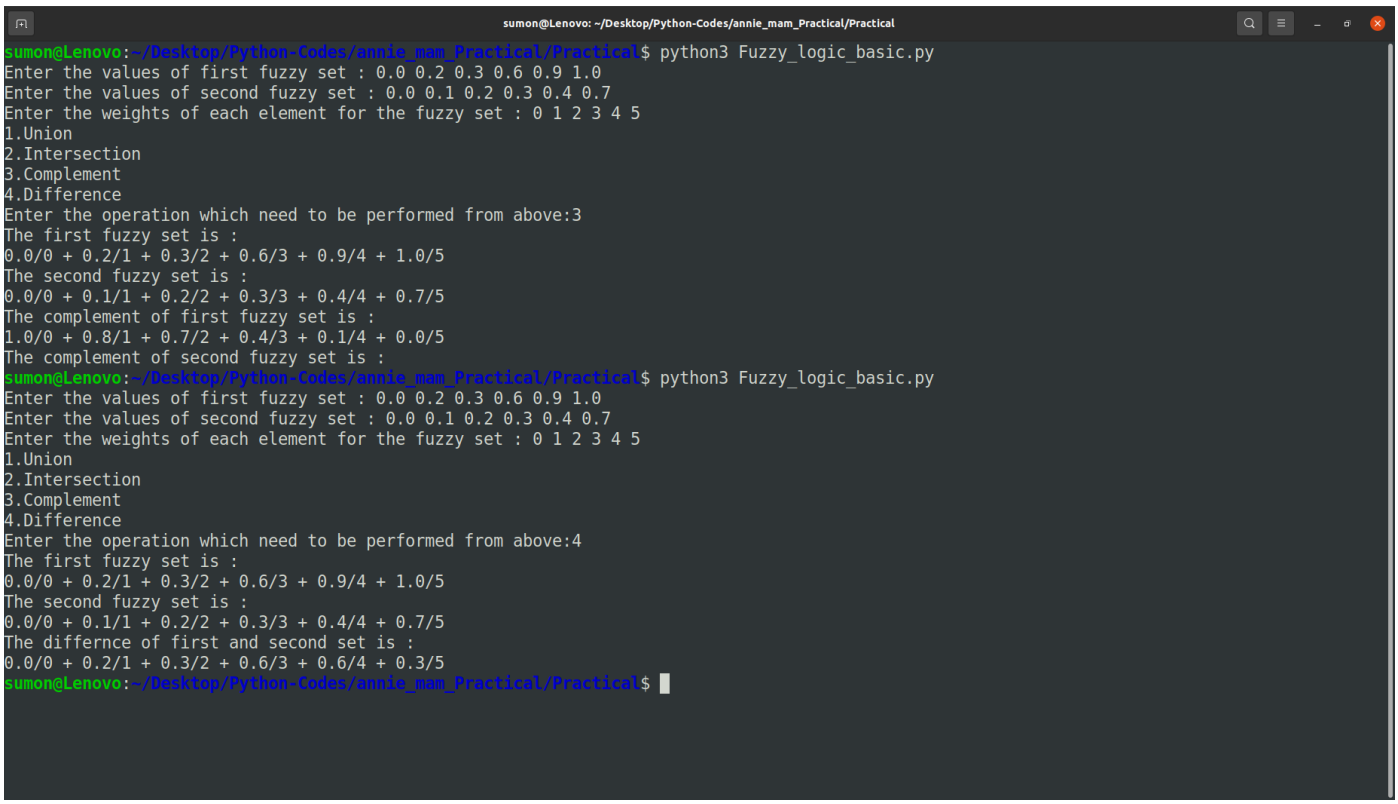
```
def union(A,B):
R = []
for i in range(0, len(A)):
if(A[i] > B[i]):
R.append(A[i])
else:
R.append(B[i])
return R
def intersect(A,B):
R = []
for i in range(0, len(A)):
if(A[i] < B[i]):
R.append(A[i])
else:
R.append(B[i])
return R
def complement(A):
R = []
for i in range(0,len(A)):
V = 1 - A[i]
R.append(V)
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:





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

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

$R = A \times B$

$x_i \rightarrow A$
$y_j \rightarrow B$

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

$\mu(x_1, y_1) = \min(0.1, 1) = 0.1$

$\mu(x_1, y_2) = \min(0.1, 0.2) = 0.1$

$\mu(x_1, y_3) = \min(0.1, 0.5) = 0.1$

$\mu(x_1, y_4) = \min(0.1, 0.7) = 0.1$

$\mu(x_1, y_5) = \min(0.1, 0.3) = 0.1$

$\mu(x_1, y_6) = \min(0.1, 0) = 0$

$\mu(x_2, y_1) = \min(0.2, 1) = 0.2$

$\mu(x_2, y_2) = \min(0.2, 0.2) = 0.2$

$\mu(x_2, y_3) = \min(0.2, 0.5) = 0.2$

$\mu(x_2, y_4) = \min(0.2, 0.7) = 0.2$

$\mu(x_2, y_5) = \min(0.2, 0.3) = 0.2$

$\mu(x_2, y_6) = \min(0.2, 0) = 0$

$\mu(x_3, y_1) = \min(0.3, 1) = 0.3$

$\mu(x_3, y_2) = \min(0.3, 0.2) = 0.2$

$\mu(x_3, y_5) = \min(0.3, 0.5) = 0.3$

$\mu(x_3, y_4) = \min(0.3, 0.7) = 0.3$

$\mu(x_3, y_3) = \min(0.3, 0.3) = 0.3$

$\mu(x_3, y_6) = \min(0.3, 0) = 0$

$\mu(x_4, y_1) = \min(0.4, 1) = 0.4$

$\mu(x_4, y_2) = \min(0.4, 0.2) = 0.2$

$\mu(x_4, y_3) = \min(0.4, 0.5) = 0.4$

$\mu(x_4, y_4) = \min(0.4, 0.7) = 0.4$

$\mu(x_4, y_5) = \min(0.4, 0.3) = 0.3$

$\mu(x_4, y_6) = \min(0.4, 0) = 0$

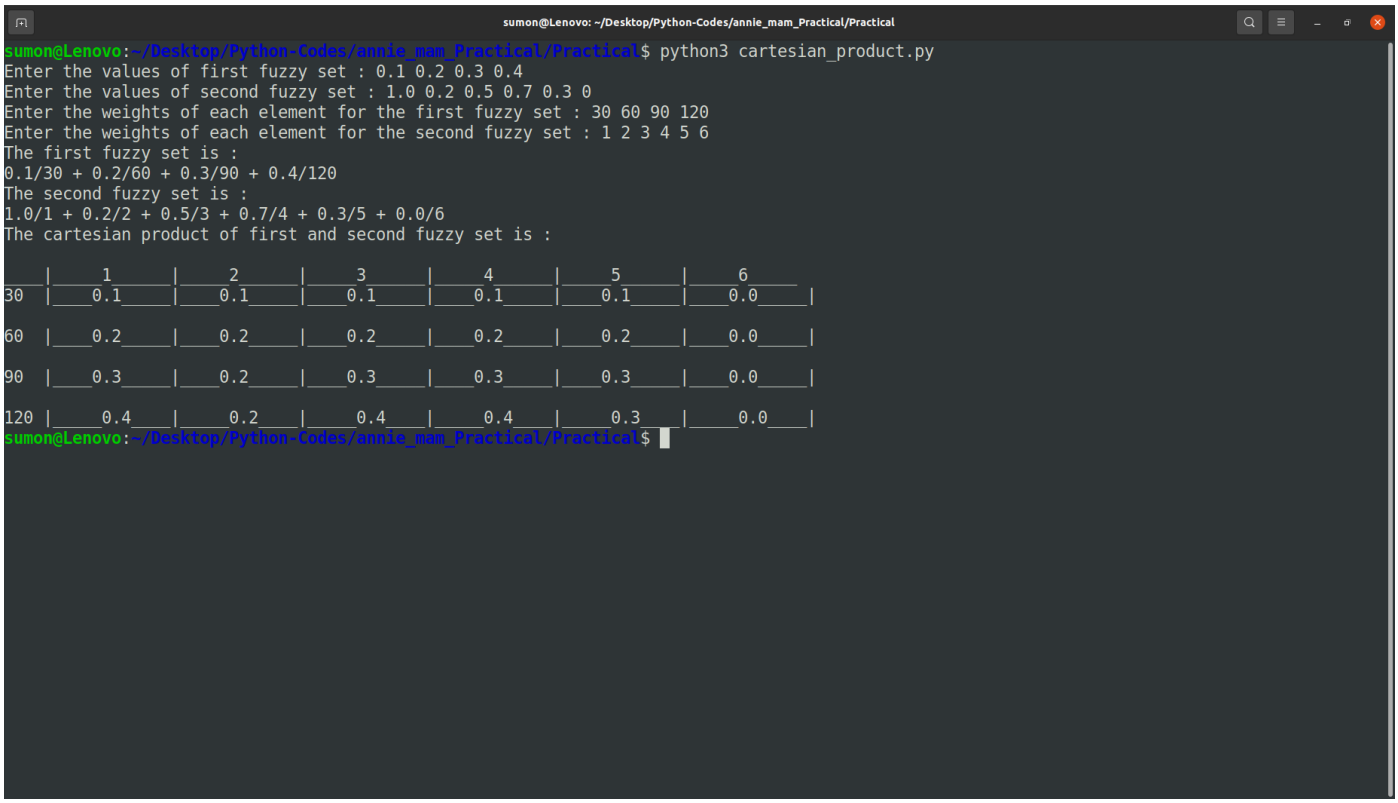
$$\therefore R = \begin{bmatrix} 0.1 & 0.1 & 0.1 & 0.1 & 0.1 & 0 \\ 0.2 & 0.2 & 0.2 & 0.2 & 0.2 & 0 \\ 0.3 & 0.2 & 0.3 & 0.3 & 0.3 & 0 \\ 0.4 & 0.2 & 0.4 & 0.4 & 0.3 & 0 \end{bmatrix}$$

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]}_____', 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:



2] Cartesian product of two fuzzy sets and perform max-min composition and max product composition.
R1 = {0.1/2 + 0.3/4 + 0.7/6 + 0.4/8}
R2 = {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) $R_1 = \left\{ \frac{0.1}{2} + \frac{0.3}{4} + \frac{0.7}{6} + \frac{0.4}{8} + \frac{0.2}{10} \right\}$

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

$x_i \to R_1$

$y_j \to R_2$

$$R_1 \times R_2 = \left[\mu(x_i, y_j) \right]_{i=1..5,\ j=1..6}$$

$\mu(x_1, y_1) = \min(0.1, 0.1) = 0.1$

$\mu(x_1, y_2) = \min(0.1, 0.3) = 0.1$

$\mu(x_1, y_3) = \min(0.1, 0.3) = 0.1$

$\mu(x_1, y_4) = \min(0.1, 0.4) = 0.1$

$\mu(x_1, y_5) = \min(0.1, 0.5) = 0.1$

$\mu(x_1, y_6) = \min(0.1, 0.2) = 0.1$

$\mu(x_2, y_1) = \min(0.3, 0.1) = 0.1$

$\mu(x_2, y_2) = \min(0.3, 0.3) = 0.3$

$\mu(x_2, y_3) = \min(0.3, 0.3) = 0.3$

$\mu(x_2, y_4) = \min(0.3, 0.4) = 0.3$

$\mu(x_2, y_5) = \min(0.3, 0.5) = 0.3$

$\mu(x_2, y_6) = \min(0.3, 0.2) = 0.2$

$\mu(x_3, y_1) = \min(0.7, 0.1) = 0.1$

$\mu(x_3, y_2) = \min(0.7, 0.3) = 0.3$

$\mu(x_3, y_3) = \min(0.7, 0.3) = 0.3$

$\mu(x_3, y_4) = \min(0.7, 0.4) = 0.4$

$\mu(x_3, y_5) = \min(0.7, 0.5) = 0.5$

$\mu(x_3, y_6) = \min(0.7, 0.2) = 0.2$

$\mu(x_4, y_1) = \min(0.4, 0.1) = 0.1$

$\mu(x_4, y_2) = \min(0.4, 0.3) = 0.3$

$\mu(x_4, y_3) = \min(0.4, 0.3) = 0.3$

$\mu(x_4, y_4) = \min(0.4, 0.4) = 0.4$

$\mu(x_4, y_5) = \min(0.4, 0.5) = 0.4$

$\mu(x_4, y_6) = \min(0.4, 0.2) = 0.2$

$\mu(x_5, y_1) = \min(0.2, 0.1) = 0.1$

$\mu(x_5, y_2) = \min(0.2, 0.3) = 0.2$

$\mu(x_5, y_3) = \min(0.2, 0.3) = 0.2$

$\mu(x_5, y_4) = \min(0.2, 0.4) = 0.2$

$\mu(x_5, y_5) = \min(0.2, 0.5) = 0.2$

$\mu(x_5, y_6) = \min(0.2, 0.2) = 0.2$

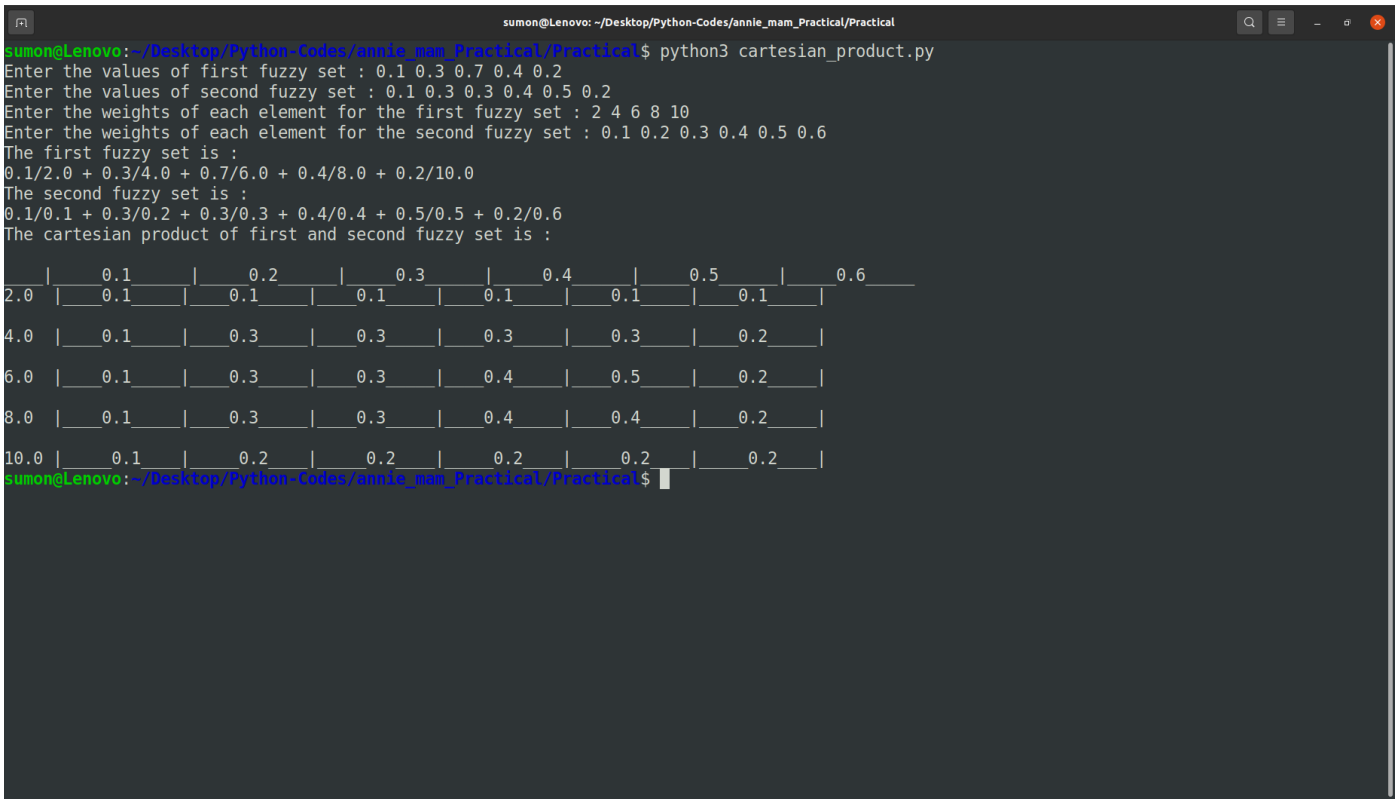
$$R_1 \times R_2 = \begin{bmatrix} 0.1 & 0.1 & 0.1 & 0.1 & 0.1 & 0.1 \\ 0.1 & 0.3 & 0.3 & 0.3 & 0.3 & 0.2 \\ 0.1 & 0.3 & 0.3 & 0.4 & 0.5 & 0.2 \\ 0.1 & 0.3 & 0.3 & 0.4 & 0.4 & 0.2 \\ 0.1 & 0.3 & 0.2 & 0.2 & 0.2 & 0.2 \end{bmatrix}$$

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:



3] Cartesian product of two fuzzy sets and perform max-min composition and max product composition.
M1 = {0/0.72 + 0.8/0.725 + 1/0.75 + 0.8/0.775 + 0/0.78}
M2 = {0/21k + 0.2/22k + 0.7/23k + 1/24k + 0.7/25k + 0.2/26k + 0/27k}

Solution:

3)

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

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

$$M_1 \times M_2 = \left[\mu(x_i, y_j) \right]_{i=1\ldots5,\ j=1\ldots7}$$

$x_i \rightarrow M_1$

$y_j = M_2$

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

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:

4] Cartesian product of two fuzzy sets and perform max-min composition and max product composition.
I1 = {0.2/20 + 0.3/40 + 0.6/60 + 0.8/80 + 1.0/100 + 0.2/120}
I2 = {0.35/500 + 0.67/1000 + 0.97/1500 + 0.25/1800}

Solution:

4) $I_1 = \left\{ \frac{0.2}{20} + \frac{0.3}{40} + \frac{0.6}{60} + \frac{0.8}{80} + \frac{1.0}{100} + \frac{02}{120} \right\}$

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

$I_1 \times I_2 = [\mu(x_i, y_j)]$ $\quad i = 1 \ldots 6$, $j = 1 \ldots 4$

$x_i \rightarrow I_1$

$y_j \rightarrow I_2$

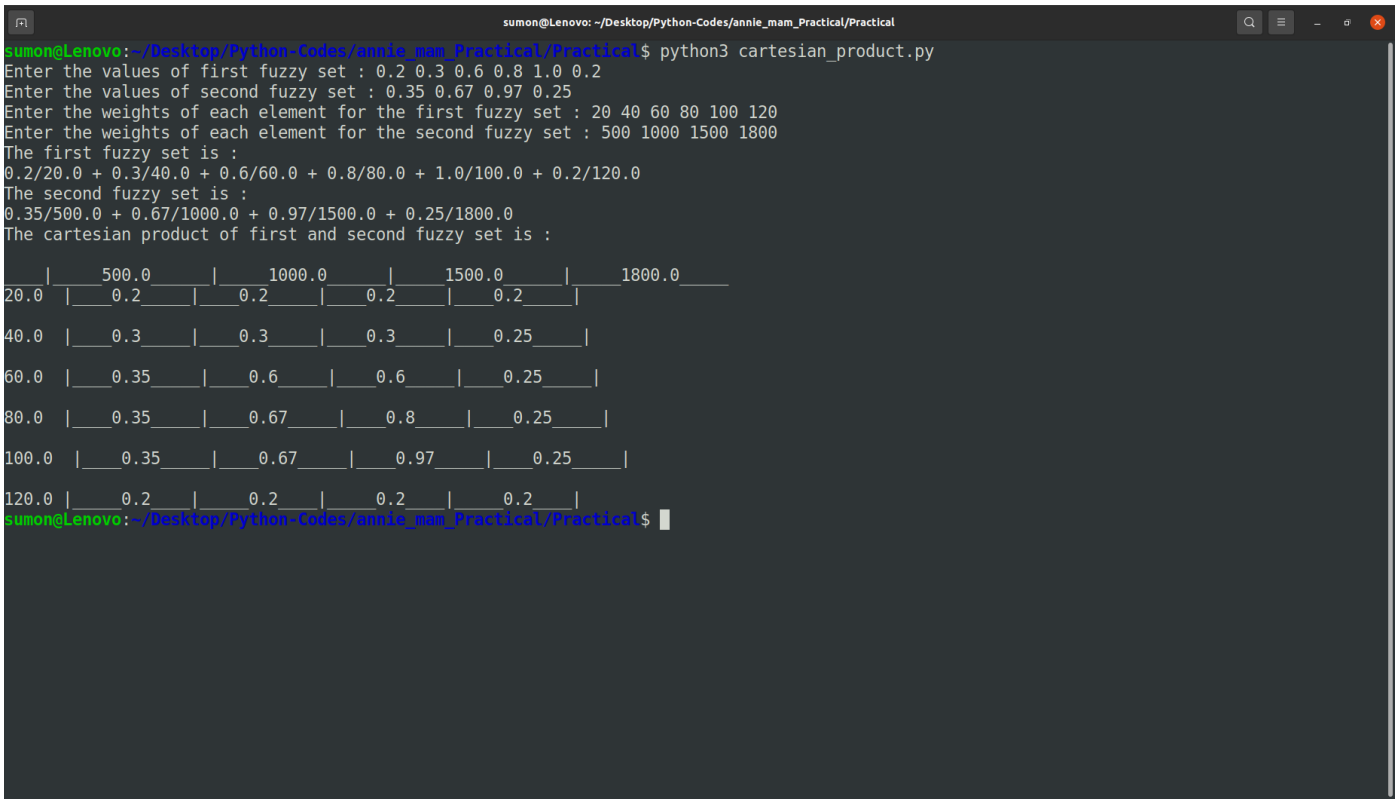
$$I_1 \times I_2 = \begin{bmatrix} 0.2 & 0.2 & 0.2 & 0.2 \\ 0.3 & 0.3 & 0.3 & 0.25 \\ 0.35 & 0.6 & 0.6 & 0.25 \\ 0.35 & 0.67 & 0.8 & 0.25 \\ 0.35 & 0.67 & 0.97 & 0.25 \\ 0.2 & 0.2 & 0.2 & 0.2 \end{bmatrix}$$

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:



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]

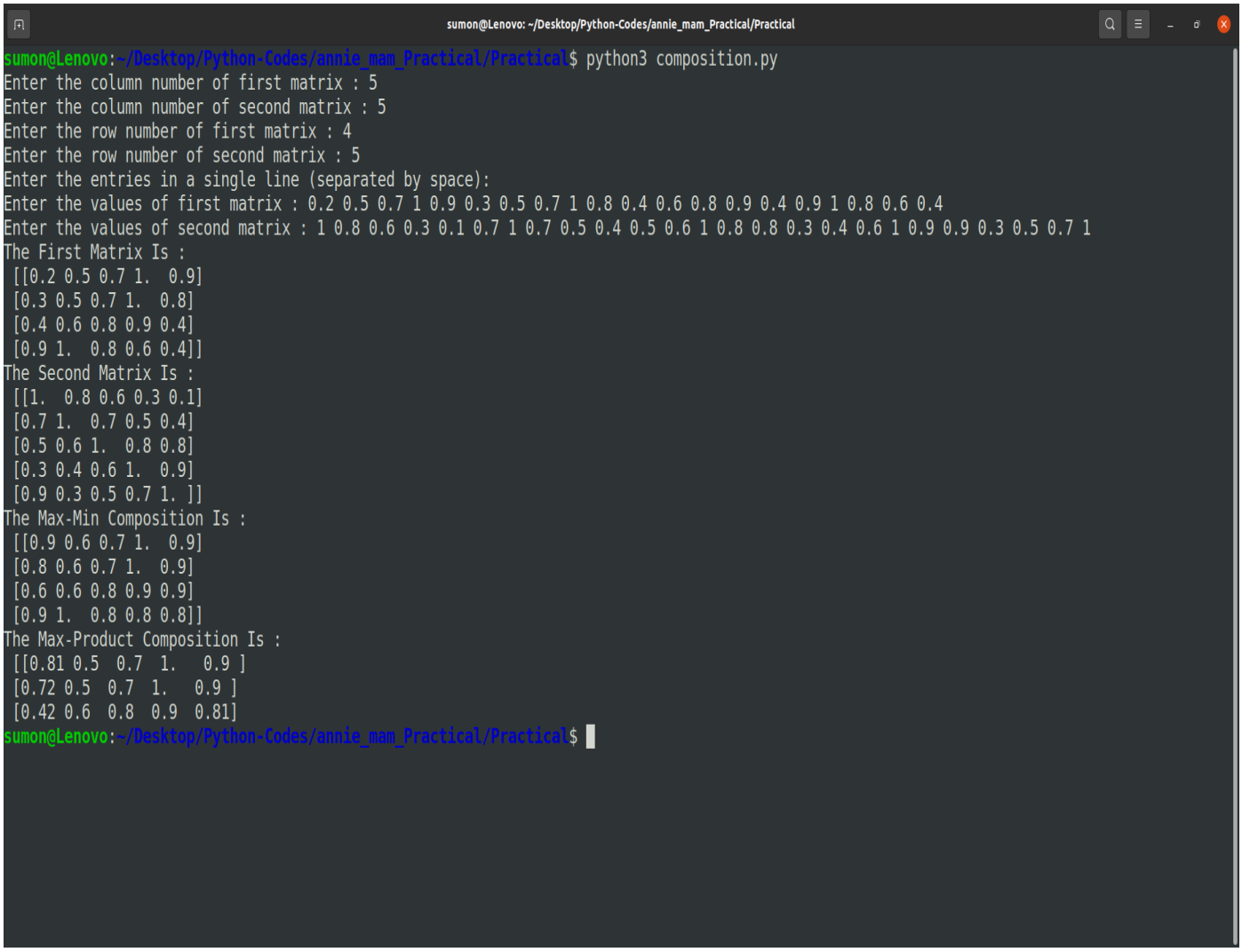
S = [1 0.8 0.6 0.3 0.1
0.7 1 0.7 0.5 0.4
0.5 0.6 1 0.8 0.8
0.3 0.4 0.6 1 0.9
0.9 0.3 0.5 0.7 1]

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
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]

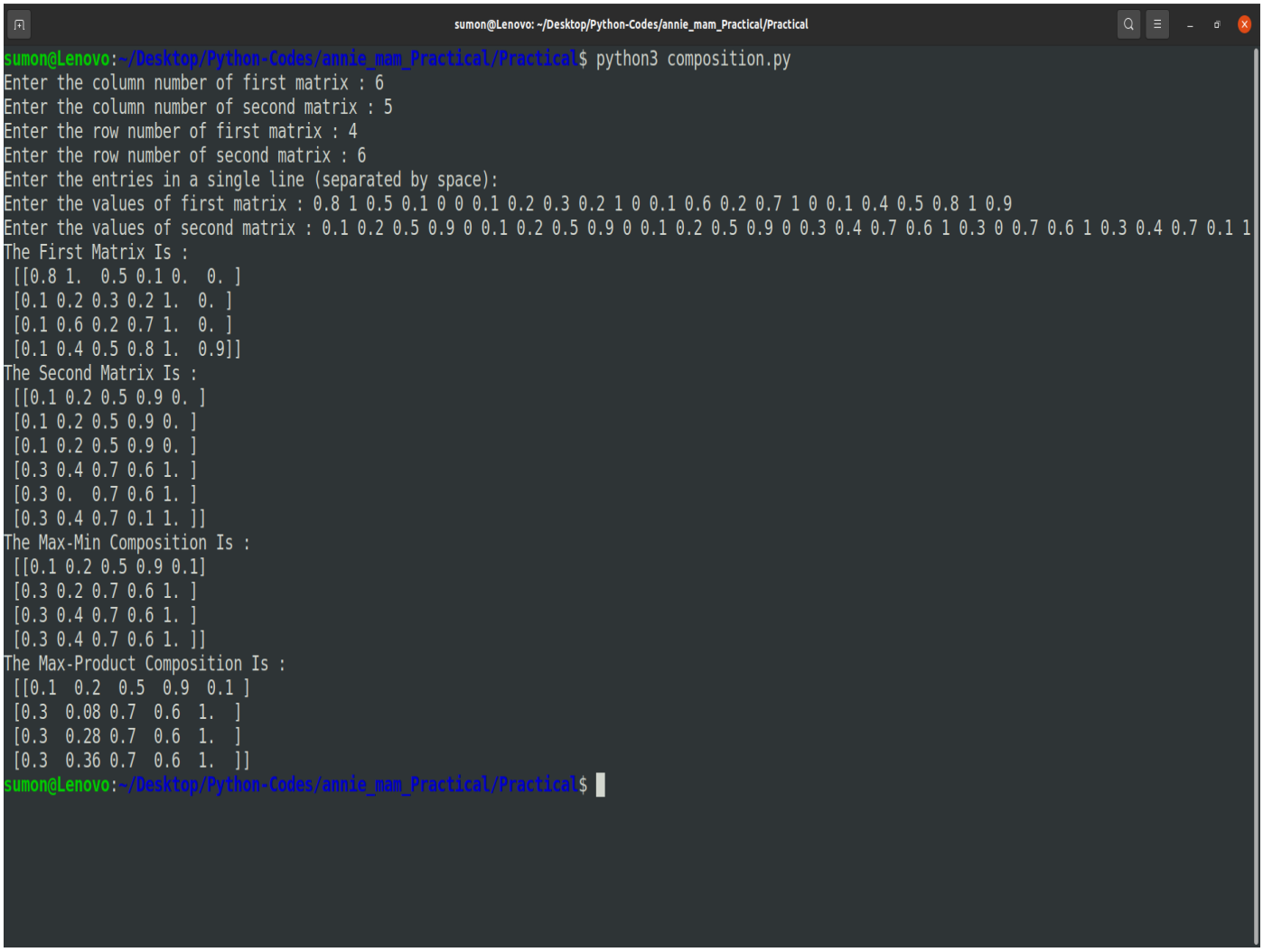
A2 = [0.1 0.2 0.5 0.9 0
0.1 0.2 0.5 0.9 0
0.1 0.2 0.5 0.9 0
0.3 0.4 0.7 0.6 1
0.3 0 0.7 0.6 1
0.3 0.4 0.7 0.1 1]

Code:

```
mport 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
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

μ(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) $\underline{A} = \left\{ \frac{0.2}{x_1} + \frac{0.3}{x_2} + \frac{0.4}{x_3} + \frac{0.7}{x_4} + \frac{0.1}{x_5} \right\}$

$\underline{B} = \left\{ \frac{0.4}{x_1} + \frac{0.5}{x_2} + \frac{0.6}{x_3} + \frac{0.8}{x_4} + \frac{0.9}{x_5} \right\}$

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

a) $(\bar{A})_{0.7} = \left\{ \frac{0.8}{x_1} + \frac{0.7}{x_2} + \frac{0.6}{x_3} + \frac{0.3}{x_4} + \frac{0.9}{x_5} \right\}_{0.7}$

$= \left\{ \frac{0.8}{x_1} + \frac{0.7}{x_2} + \frac{0.9}{x_5} \right\}$

b) $(\bar{B})_{0.2} = \left\{ \frac{0.4}{x_1} + \frac{0.5}{x_2} + \frac{0.6}{x_3} + \frac{0.8}{x_4} + \frac{0.9}{x_5} \right\}$

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

$= \left\{ \frac{0.4}{x_1} + \frac{0.5}{x_2} + \frac{0.6}{x_3} + \frac{0.8}{x_4} + \frac{0.9}{x_5} \right\}_{0.6}$

$= \left\{ \frac{0.6}{x_3} + \frac{0.8}{x_4} + \frac{0.9}{x_5} \right\}$

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

$= \left\{ \frac{0.2}{x_1} + \frac{0.3}{x_2} + \frac{0.4}{x_3} + \frac{0.7}{x_4} + \frac{0.1}{x_5} \right\}_{0.5}$

$= \left\{ \frac{0.7}{x_4} \right\}$

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

$= \left\{ \frac{0.8}{x_1} + \frac{0.7}{x_2} + \frac{0.6}{x_3} + \frac{0.7}{x_4} + \frac{0.9}{x_5} \right\}_{0.7}$

$= \left\{ \frac{0.8}{x_1} + \frac{0.7}{x_2} + \frac{0.7}{x_4} + \frac{0.9}{x_5} \right\}$

f) $(\underline{B} \cap \underline{\bar{B}})_{0.3} = \left\{ \frac{0.4}{x_1} + \frac{0.5}{x_2} + \frac{0.4}{x_3} + \frac{0.2}{x_4} + \frac{0.1}{x_5} \right\}_{0.3}$

$= \left\{ \frac{0.4}{x_1} + \frac{0.5}{x_2} + \frac{0.4}{x_3} \right\}$

g) $(\overline{\underline{A} \cap \underline{B}})_{0.6} = \left\{ \frac{0.8}{x_1} + \frac{0.7}{x_2} + \frac{0.6}{x_3} + \frac{0.3}{x_4} + \frac{0.9}{x_5} \right\}_{0.6}$

$= \left\{ \frac{0.8}{x_1} + \frac{0.7}{x_2} + \frac{0.6}{x_3} + \frac{0.9}{x_5} \right\}$

h) $(\underline{\bar{A}} \cup \underline{\bar{B}})_{0.8} = \left\{ \frac{0.8}{x_1} + \frac{0.7}{x_2} + \frac{0.6}{x_3} + \frac{0.3}{x_4} + \frac{0.9}{x_5} \right\}_{0.6}$

$= \left\{ \frac{0.8}{x_1} + \frac{0.9}{x_5} \right\}$

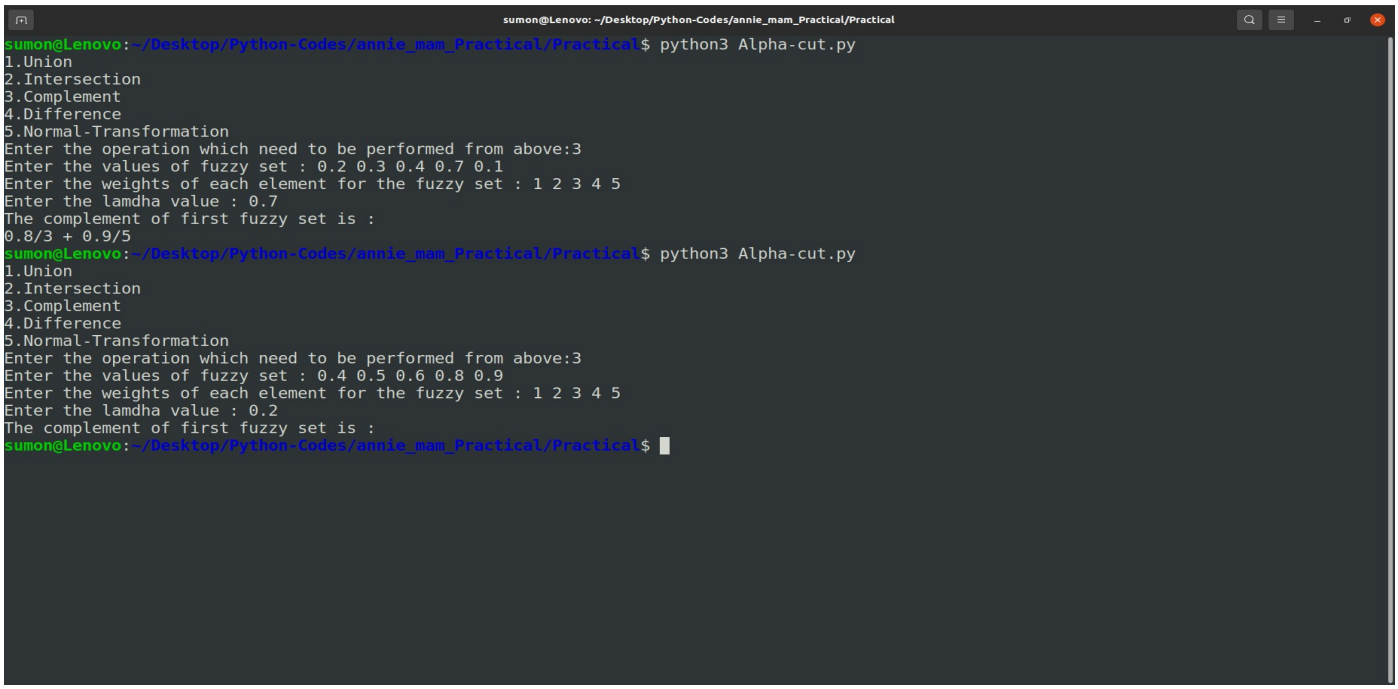
Code:

```
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 = " + ")
```

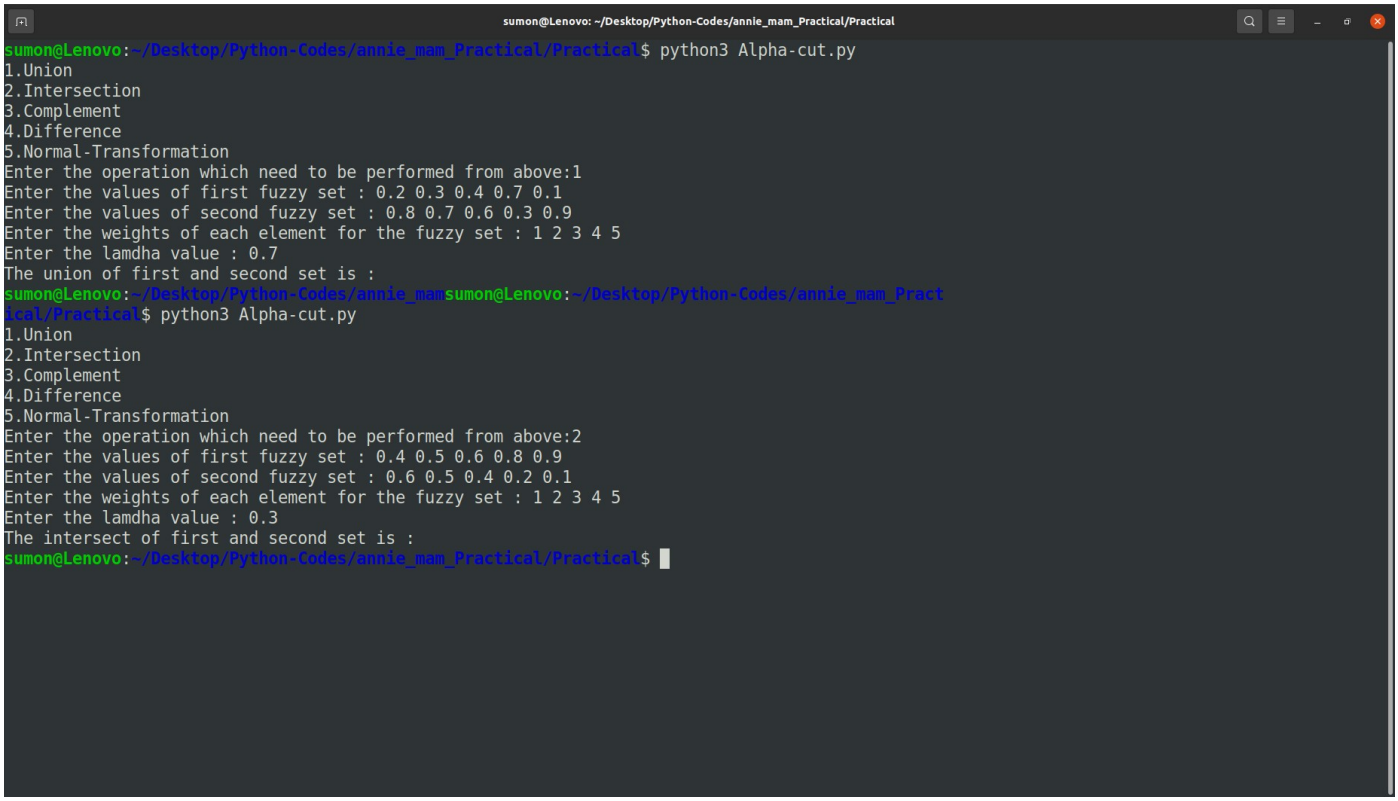
```
print(f'{round(A[-1], 2)}/ {round(W[-1], 1)}')
OP =
OP=float(input('1.Union\n2.Intersection\n3.Complement\
n4.Difference\n5.No
rmal-Transformation\nEnter the operation which need to be
performed from
above:'))
if (OP==1):
A=list(map(float,input('Enter the values of first fuzzy set :
').strip().split()))
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 :
').strip().split()))
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 :
').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=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 :
').strip().split()))
W=list(input('Enter the weights of each element for the fuzzy set :
').strip().split())
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
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
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
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_mamsumon@Lenovo:~/Desktop/Python-Codes/annie_mam_Pract
ical/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
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 U S_2) b) (S_1 ∩ S_2) c) c(S_1)
d) c(S_2) e) c(S_1 U S_2) f) c(S_1 ∩ S_2)

Solution:

f) $(B \cap \bar{B})_{0.3} = \left\{\frac{0.4}{x_1} + \frac{0.7}{x_2} + \frac{0.4}{x_3} + \frac{0.2}{x_4} + \frac{0.1}{x_5}\right\}_{0.3}$

$= \left\{\frac{0.4}{x_1} + \frac{0.7}{x_2} + \frac{0.4}{x_3}\right\}$

g) $(\overline{B \cap B})_{0.6} = \left\{\frac{0.8}{x_1} + \frac{0.7}{x_2} + \frac{0.6}{x_3} + \frac{0.3}{x_4} + \frac{0.9}{x_5}\right\}_{0.6}$

$= \left\{\frac{0.8}{x_1} + \frac{0.7}{x_2} + \frac{0.6}{x_3} + \frac{0.9}{x_5}\right\}$

h) $(\bar{A} \cup \bar{B})_{0.8} = \left\{\frac{0.8}{x_1} + \frac{0.7}{x_2} + \frac{0.6}{x_3} + \frac{0.3}{x_4} + \frac{0.9}{x_5}\right\}_{0.6}$

$= \left\{\frac{0.8}{x_1} + \frac{0.9}{x_5}\right\}$

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

$= \left\{\frac{0.5}{20} + \frac{0.65}{40} + \frac{0.85}{60} + \frac{1.0}{80} + \frac{1.0}{100}\right\}$

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

$= \left\{\frac{0.6}{40} + \frac{0.8}{60} + \frac{0.95}{80} + \frac{1.0}{100}\right\}$

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

$= \left\{\frac{0.5}{20} + \frac{1}{0}\right\}$

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

$= \left\{\frac{1}{0} + \frac{0.55}{20}\right\}$

f) $\left(\underline{S}_1 \cap \overline{\underline{S}_2}\right)_{0.5} = \left\{\frac{1}{0} + \frac{0.55}{20} + \frac{0.4}{40} + \frac{0.2}{60} + \frac{0.05}{80} + \frac{0}{100}\right\}_{0.5}$

$= \left\{\frac{1}{0} + \frac{0.55}{20}\right\}$

e) $\left(\overline{\underline{S}_1 \cup \underline{S}_2}\right)_{0.5} = \left\{\frac{1}{0} + \frac{0.5}{20} + \frac{0.35}{40} + \frac{0.15}{60} + \frac{0}{80} + \frac{0}{100}\right\}_{0.5}$

$= \left\{\frac{1}{0} + \frac{0.5}{20}\right\}$

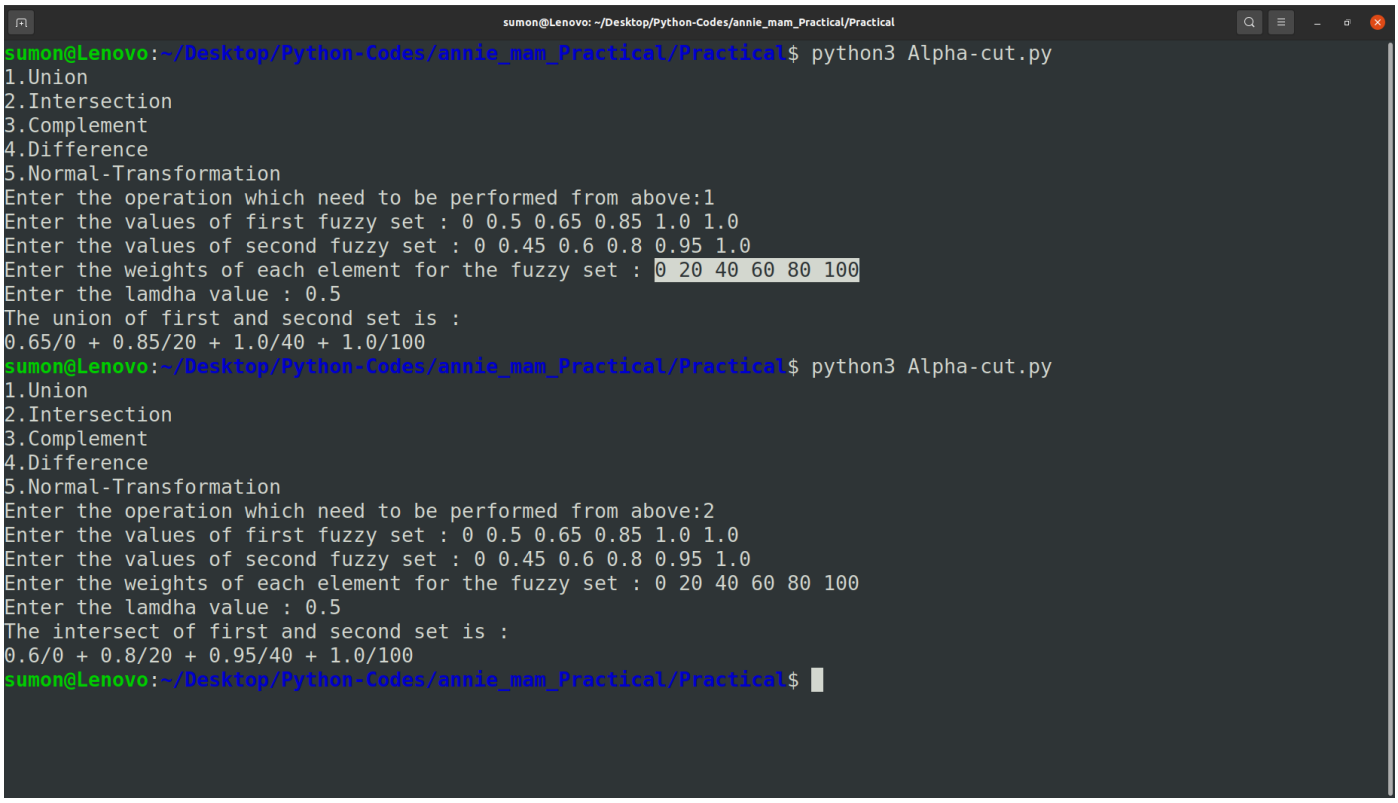
Code:

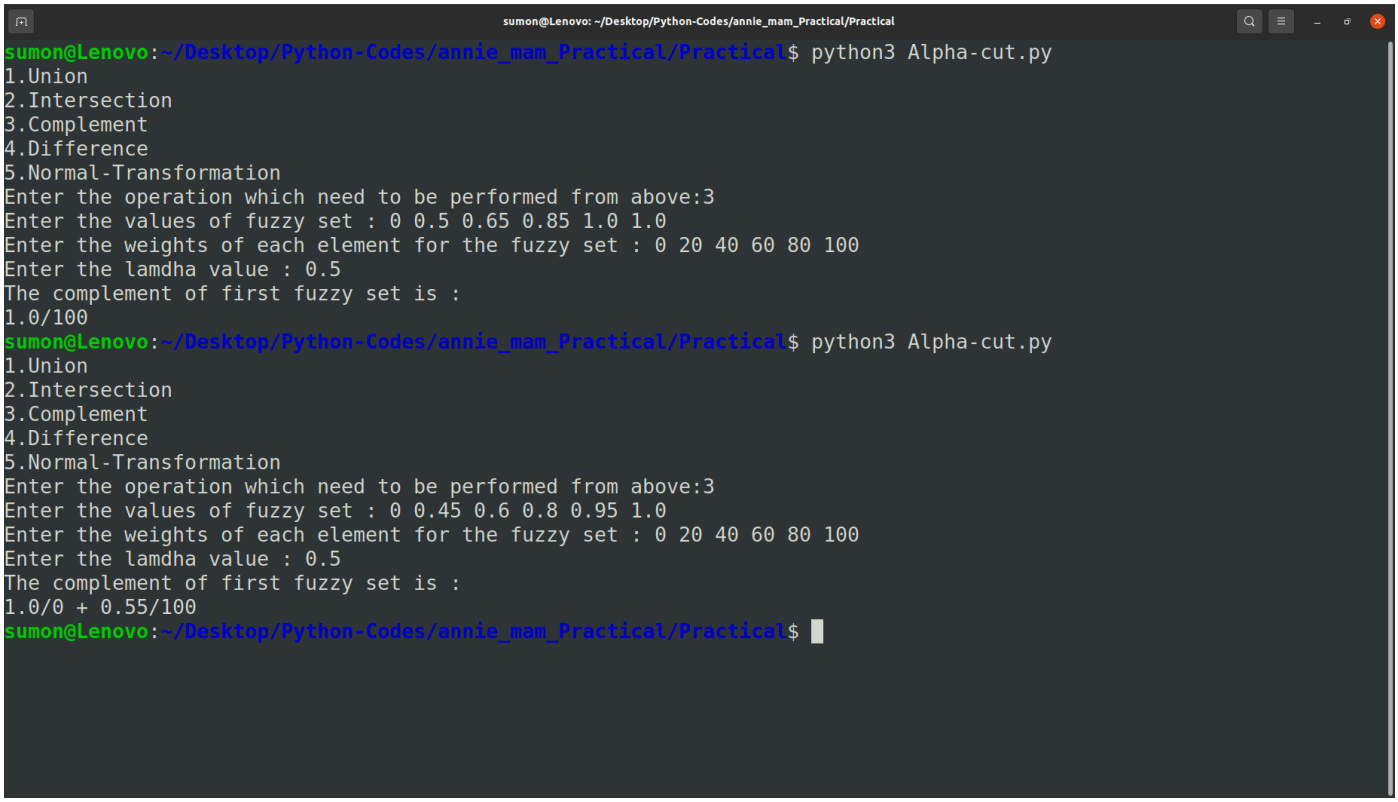
```
def union(A,B):
R = []
for i in range(0, len(A)):
if (A[i]>B[i]):
R.append(A[i])
else:
R.append(B[i])
return R
def intersect(A,B):
R = []
for i in range(0, len(A)):
if (A[i]<B[i]):
R.append(A[i])
else:
R.append(B[i])
return R
def complement(A):
R = []
for i in range(0, len(A)):
V = 1- A[i]
R.append(V)
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 = " + ")
```

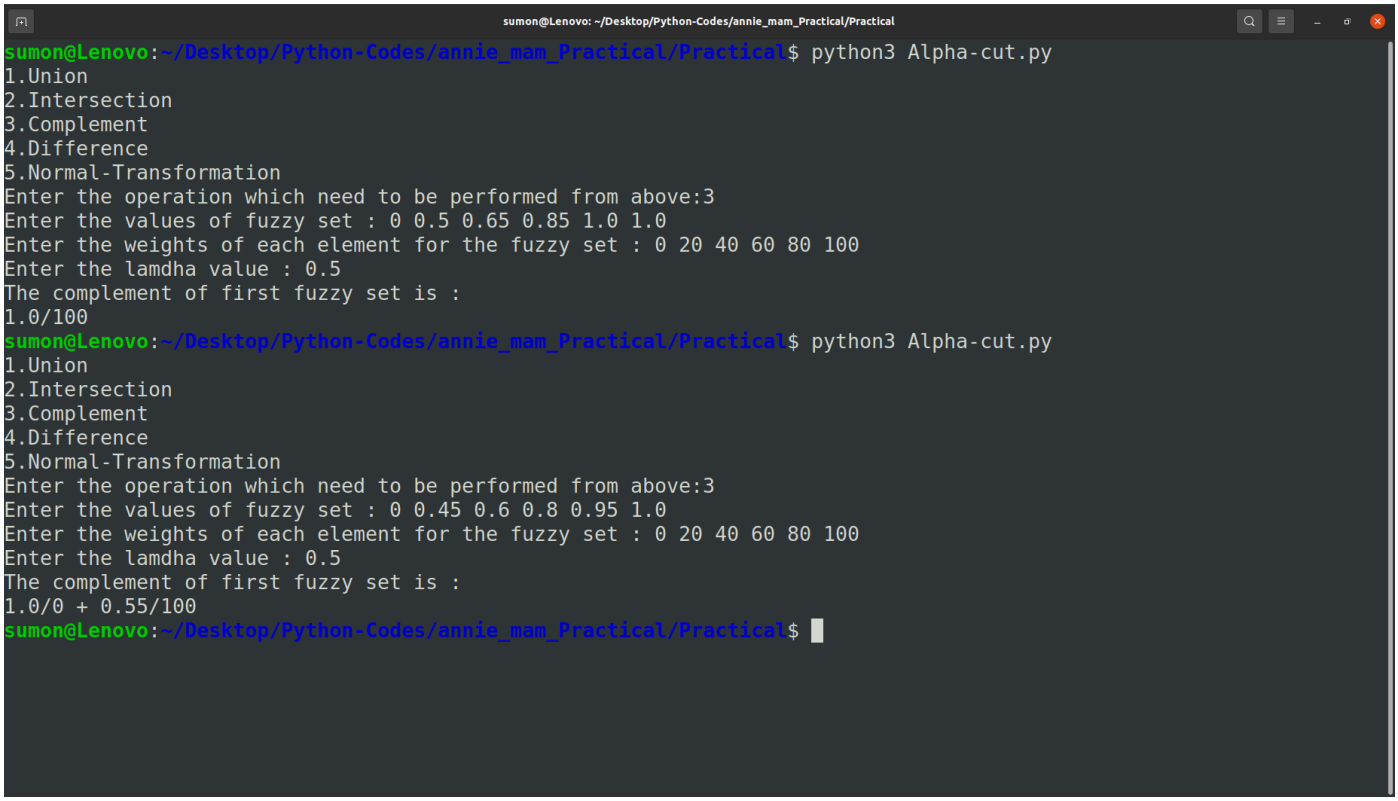
```
print(f'{round(A[-1], 2)}/ {round(W[-1], 1)}')
OP =
OP=float(input('1.Union\n2.Intersection\n3.Complement\
n4.Difference\n5.No
rmal-Transformation\nEnter the operation which need to be
performed from
above:'))
if (OP==1):
A=list(map(float,input('Enter the values of first fuzzy set :
').strip().split()))
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 :
').strip().split()))
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 :
').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=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 :
').strip().split()))
W=list(input('Enter the weights of each element for the fuzzy set :
').strip().split())
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
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
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
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 U B d) A ∩ B
e) $\bar{A}$ U c(B) f) $\bar{A}$ ∩ c(B)

Solution:

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

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

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

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

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

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

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

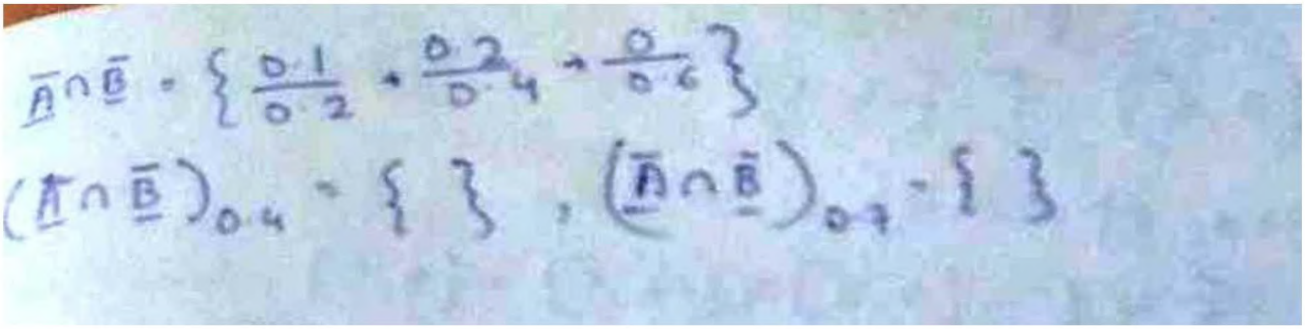
$(\underline{A} \cup \underline{B})_{0.7} = \left\{ \frac{0.9}{0.2} + \frac{0.8}{0.4} + \frac{1}{0.6} \right\}$

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

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

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

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


$\bar{A} \cap \bar{B} = \{\frac{0.1}{0.2}, \frac{0.2}{0.4}, \frac{0}{0.6}\}$
$(\bar{A} \cap \bar{B})_{0.4} = \{\}$, $(\bar{A} \cap \bar{B})_{0.7} = \{\}$

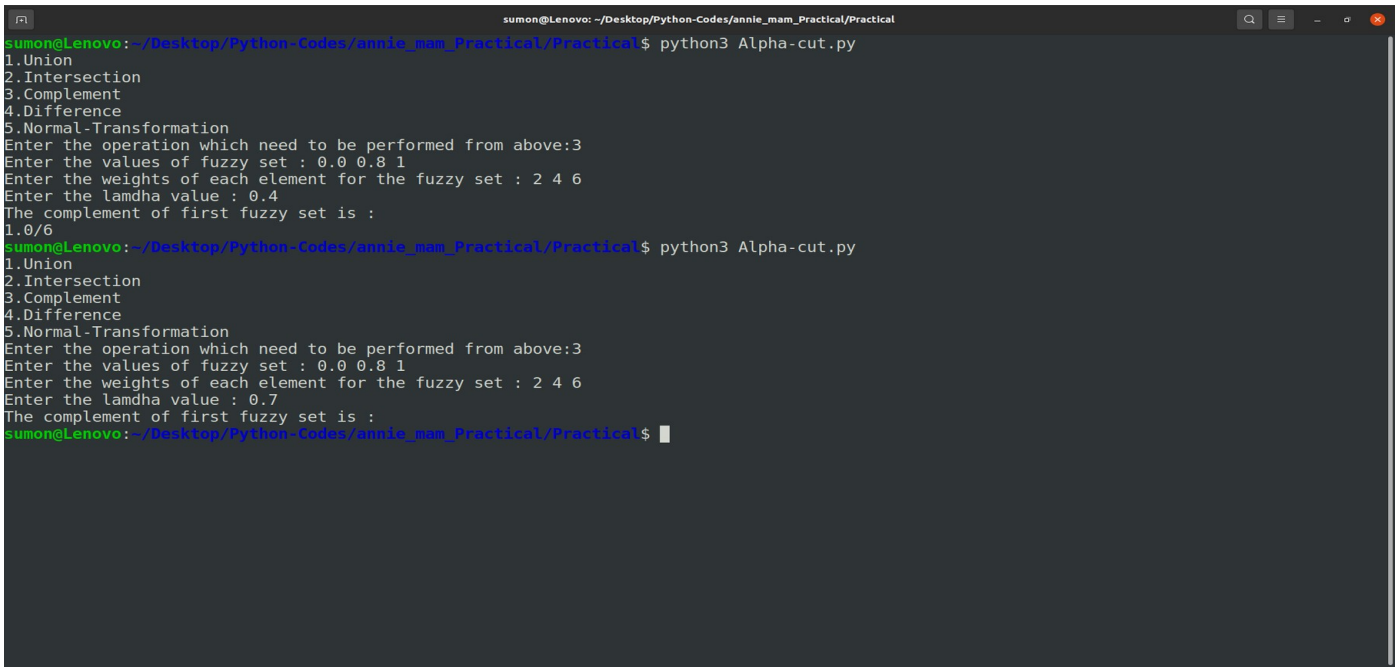
Code:

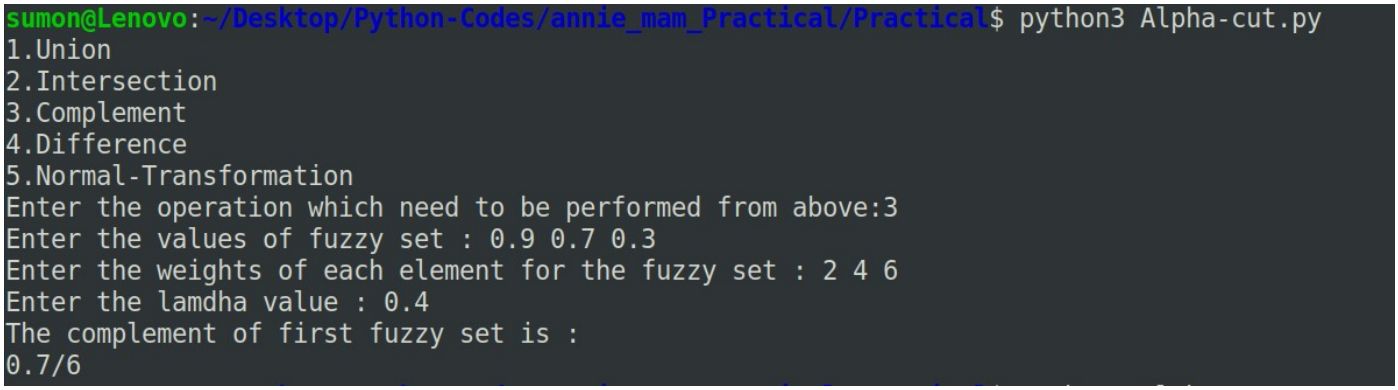
```
def union(A,B):
R = []
for i in range(0, len(A)):
if (A[i]>B[i]):
R.append(A[i])
else:
R.append(B[i])
return R
def intersect(A,B):
R = []
for i in range(0, len(A)):
if (A[i]<B[i]):
R.append(A[i])
else:
R.append(B[i])
return R
def complement(A):
R = []
for i in range(0, len(A)):
V = 1- A[i]
R.append(V)
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 = " + ")
```

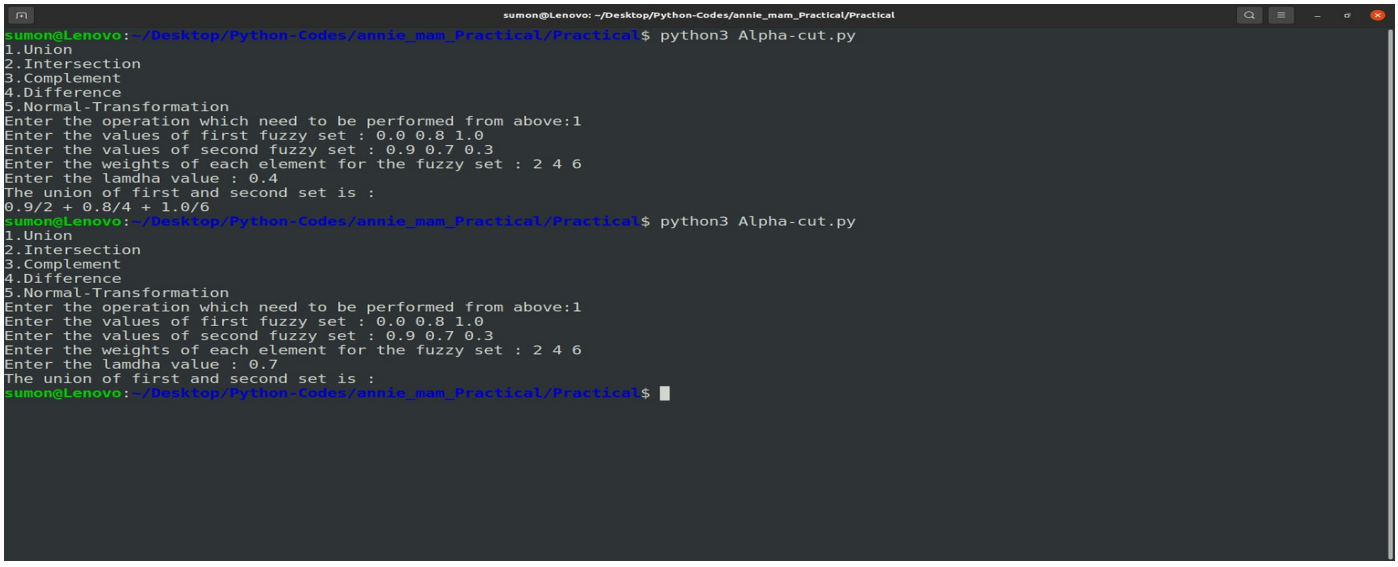
```
print(f'{round(A[-1], 2)}/ {round(W[-1], 1)}')
OP =
OP=float(input('1.Union\n2.Intersection\n3.Complement\
n4.Difference\n5.No
rmal-Transformation\nEnter the operation which need to be
performed from
above:'))
if (OP==1):
A=list(map(float,input('Enter the values of first fuzzy set :
').strip().split()))
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()))
```

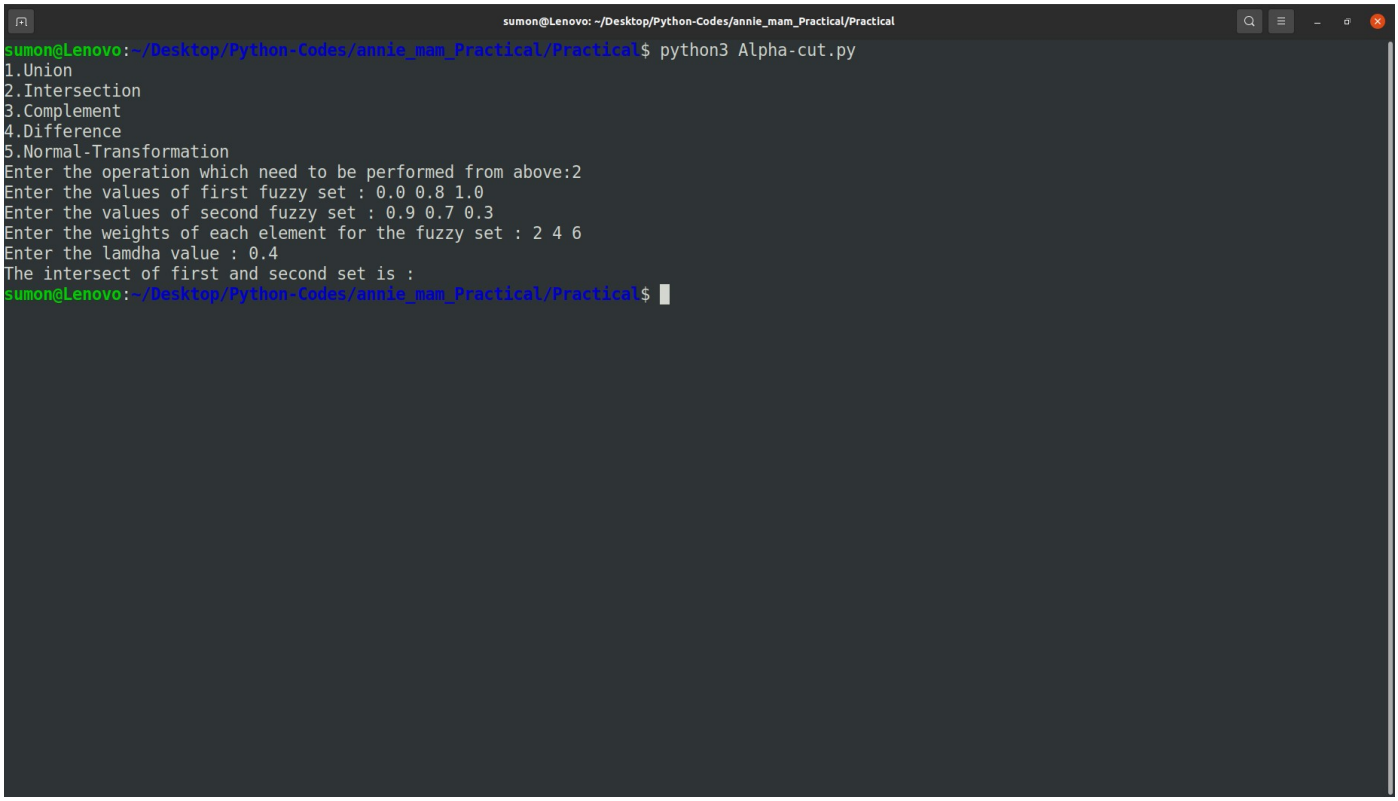
```
W=list(map(int,input('Enter the weights of each element for the fuzzy set :
').strip().split()))
lam=float(input('Enter the lamdha value : '))
R=complement(A)
R=[x for x in R if x>lam]
print('The complement of first fuzzy set is :')
print_fuzzy_set(R,W)
elif (OP==4):
A=list(map(float,input('Enter the values of first fuzzy set :
').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=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 :
').strip().split()))
W=list(input('Enter the weights of each element for the fuzzy set :
').strip().split())
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
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

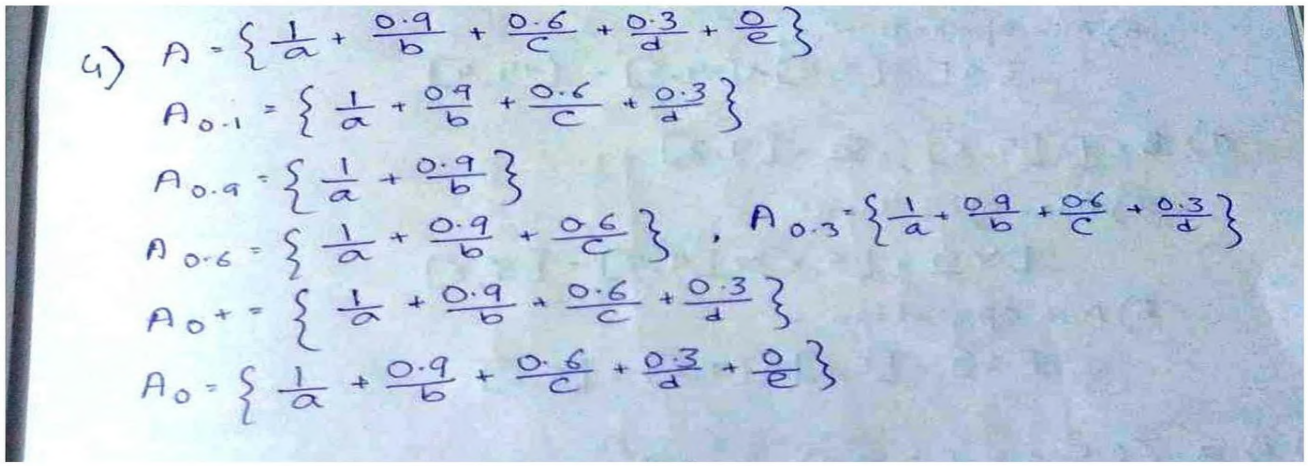

sumon@Lenovo: ~/Desktop/Python-Codes/annie_mam_Practical/Practical
sumon@Lenovo:~/Desktop/Python-Codes/annie_mam_Practical/Practical$ python3 Alpha-cut.py
1.Union
2.Intersection
3.Complement
4.Difference
5.Normal-Transformation
Enter the operation which need to be performed from above:1
Enter the values 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
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
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:



Code:

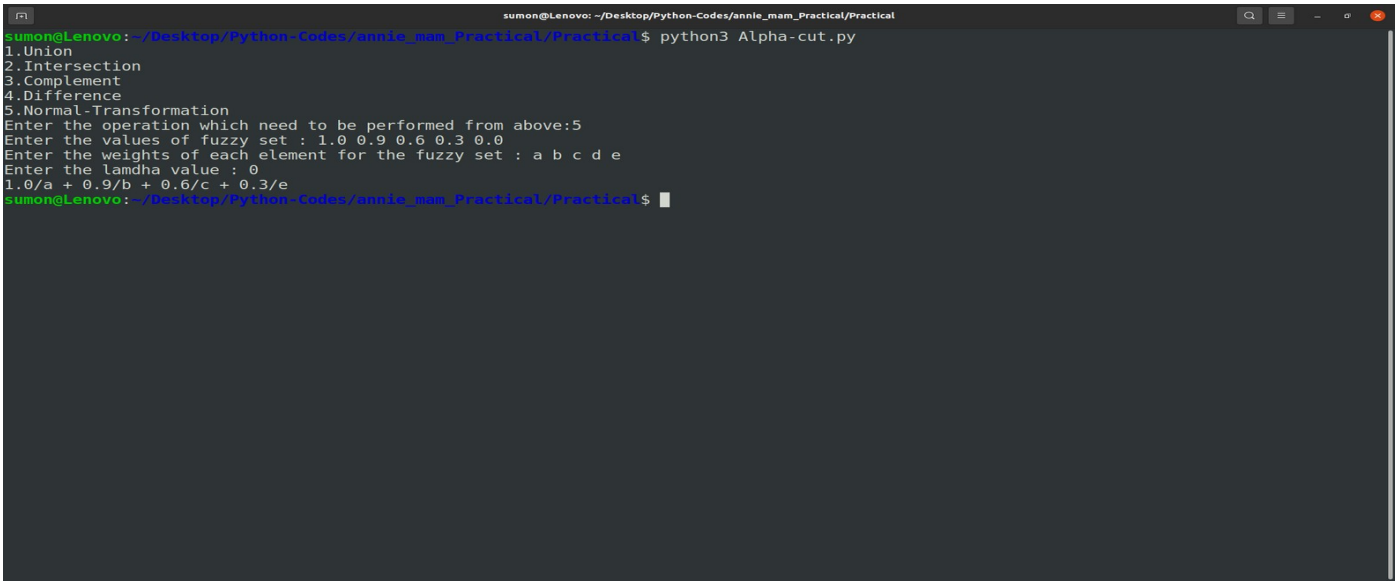
```
def union(A,B):
R = []
for i in range(0, len(A)):
if (A[i]>B[i]):
R.append(A[i])
else:
R.append(B[i])
return R
def intersect(A,B):
R = []
for i in range(0, len(A)):
if (A[i]<B[i]):
R.append(A[i])
else:
R.append(B[i])
return R
def complement(A):
R = []
for i in range(0, len(A)):
V = 1- A[i]
R.append(V)
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 :
').strip().split()))
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 :
').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=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 :
').strip().split()))
W=list(input('Enter the weights of each element for the fuzzy set :
').strip().split())
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
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
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] x [1,3] ii) [1,2] x [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) $[3,5]-[4,5] = [3-5, 5-4] = [-2,1]$

ii) $[6,9]-[2,4] = [6-4, 9-2] = [2,7]$

c) i) $[2,1]\times[1,3] = [2\times1, 1\times3] = [2,3]$

ii) $[1,2]\times[5,3] = [1\times5, 2\times3] = [5,6]$

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:

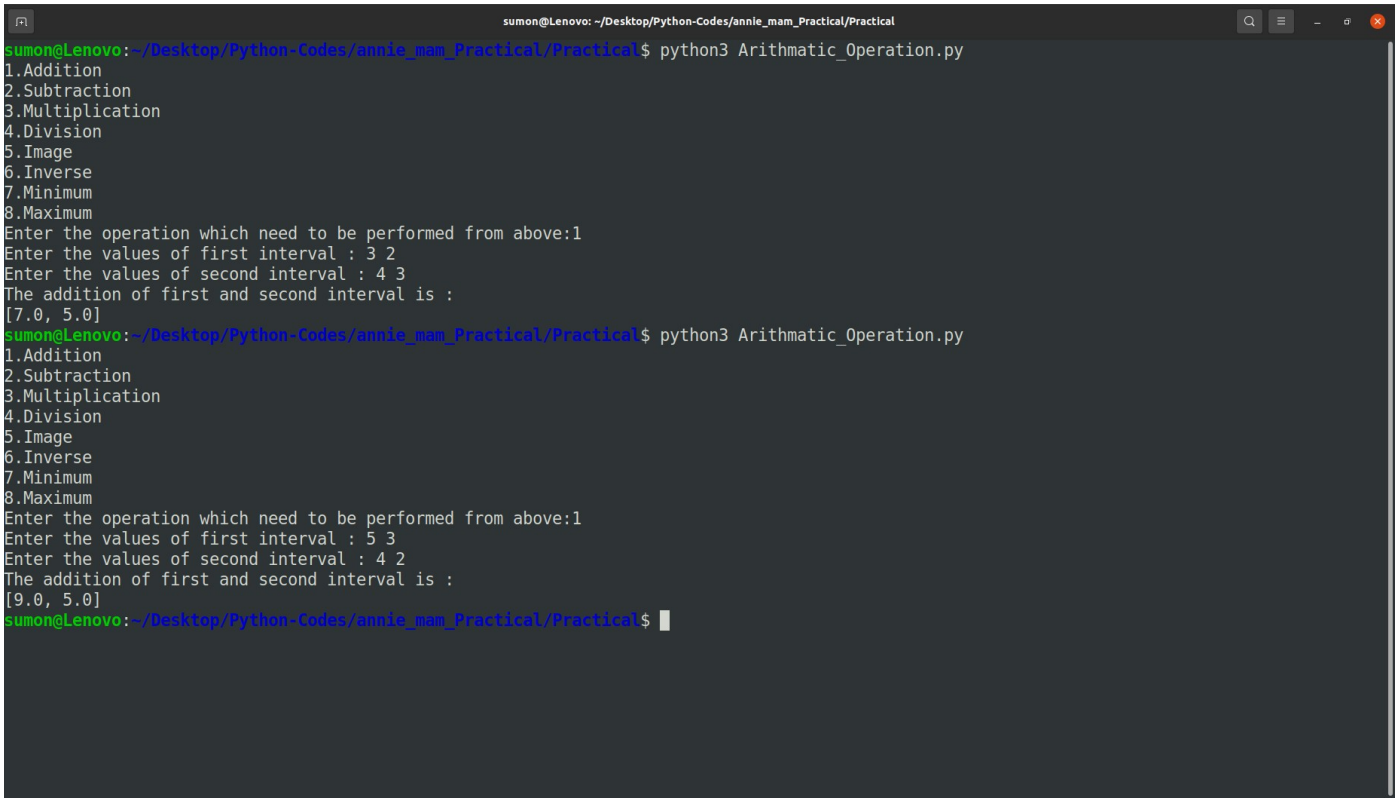
```
def ADD(x,y):
r=[]
r.append(x[0]+y[0])
r.append(x[1]+y[1])
return r
def SUB(x,y):
r=[]
r.append(x[0]-y[1])
r.append(x[1]-y[0])
return r
def MUL(x,y):
r=[]
r.append(x[0]*y[0])
r.append(x[1]*y[1])
return r
def DIV(x,y):
r=[]
r.append(x[0]/y[1])
r.append(x[1]/y[0])
return r
def IMG(x):
r=[]
r.append(-x[1])
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]:
```

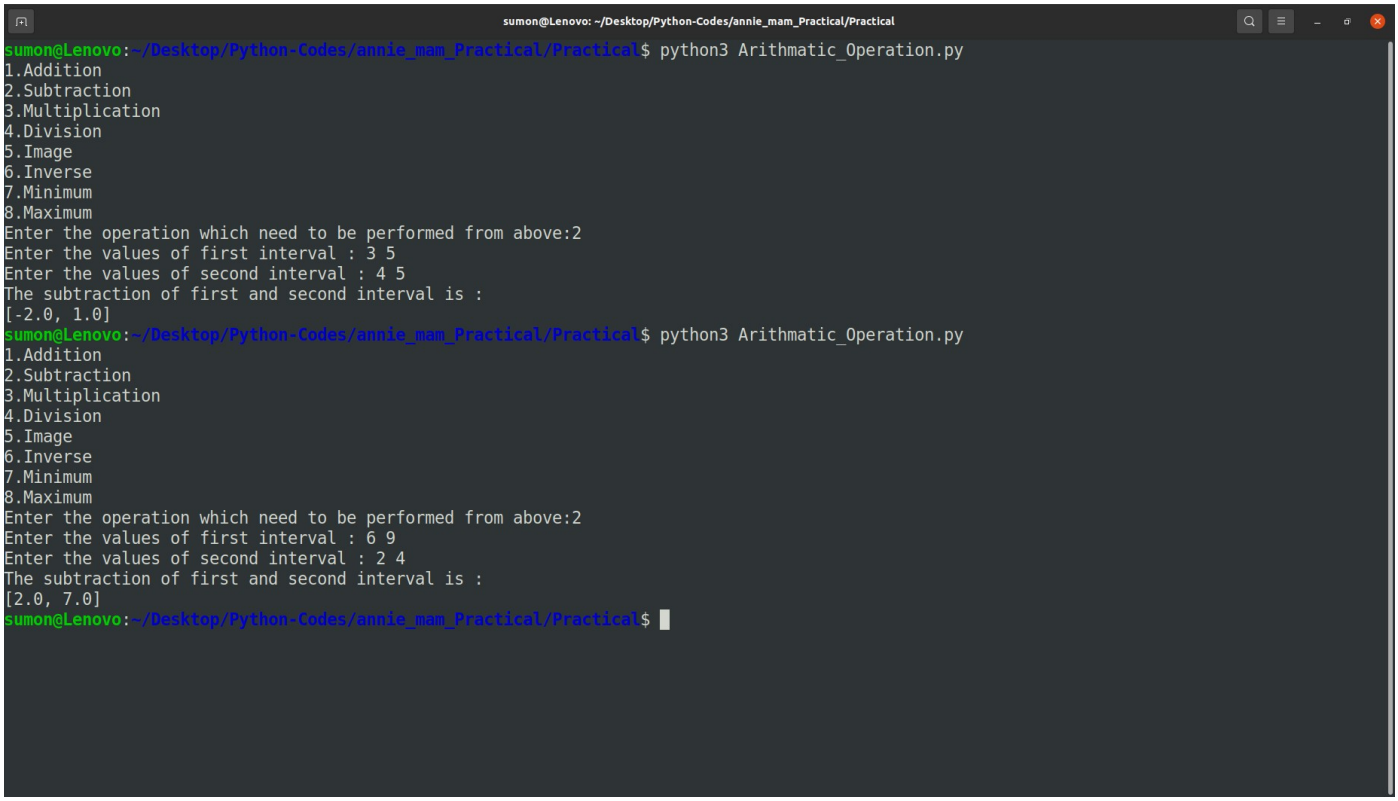
```
r.append(y[i])
else:
r.append(x[i])
return r

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 :
```

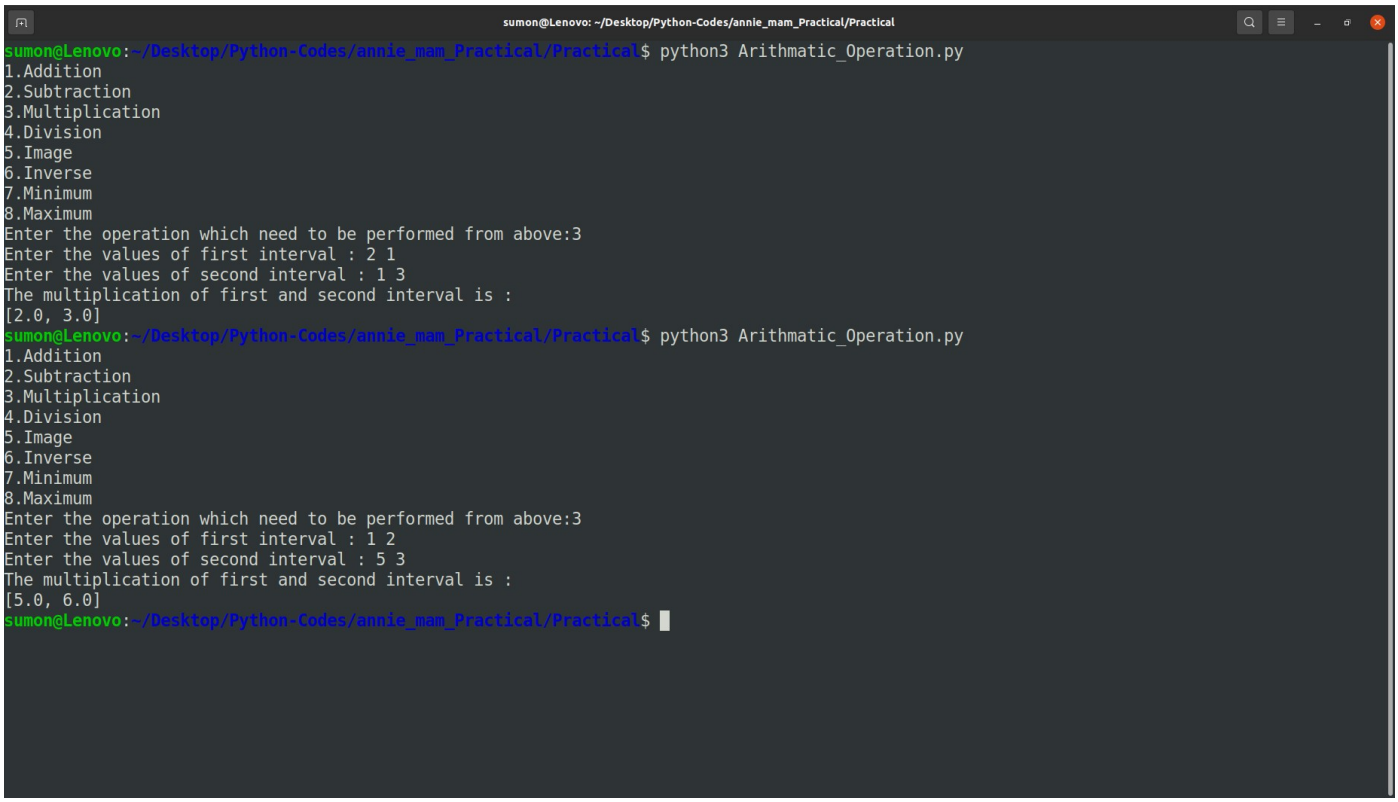
```
').strip().split()))
B=list(map(float,input('Enter the values of second interval :
').strip().split()))
R=MUL(A,B)
print('The multiplication of first and second interval is :')
print(R)
elif (OP==4):
A=list(map(float,input('Enter the values of first interval :
').strip().split()))
B=list(map(float,input('Enter 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 :
```

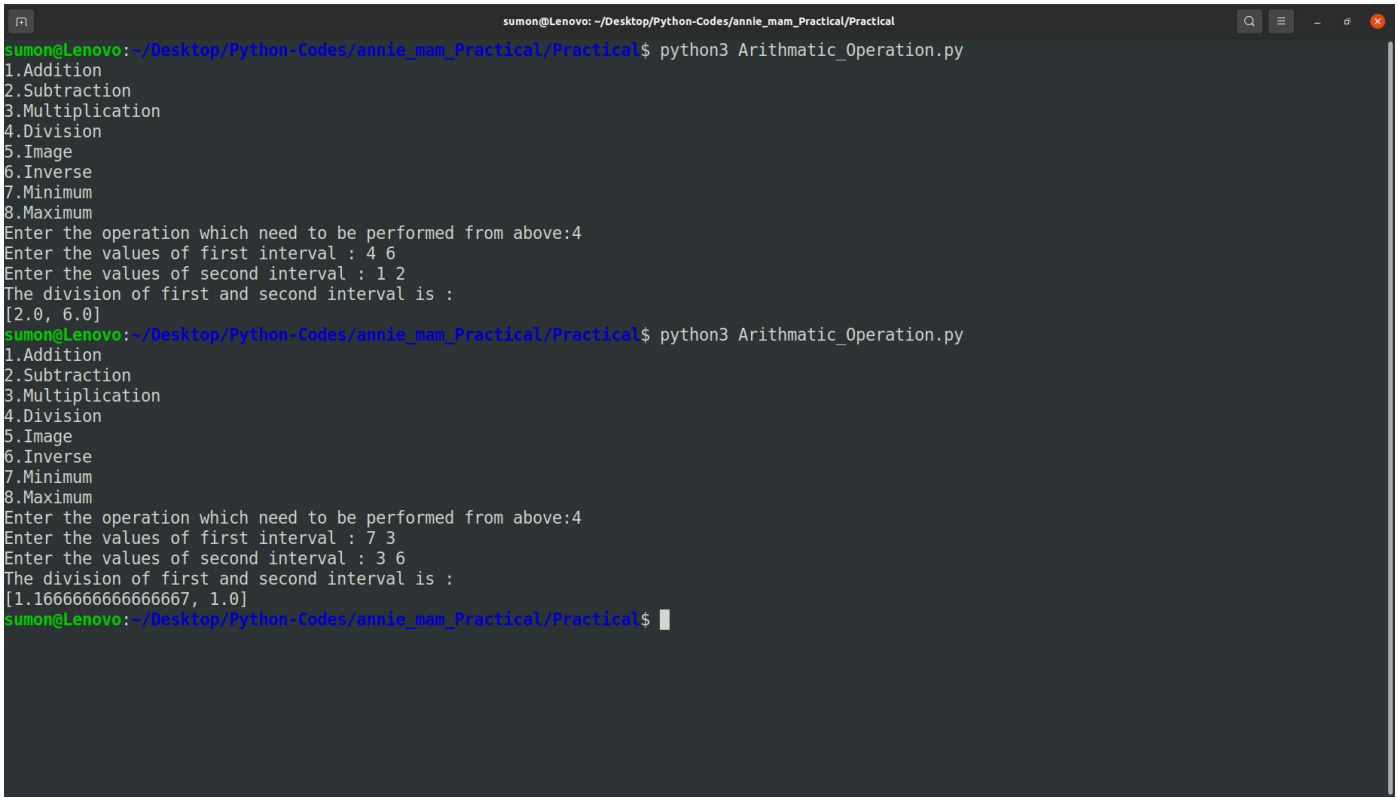
').strip().split()))
B=list(map(float,input('Enter the values of second interval :
').strip().split()))
R=MAX(A,B)
print('The maximum of first and second interval is :')
print(R)
else:
print('Please select any valid number : 1,2,3,4')
exit()

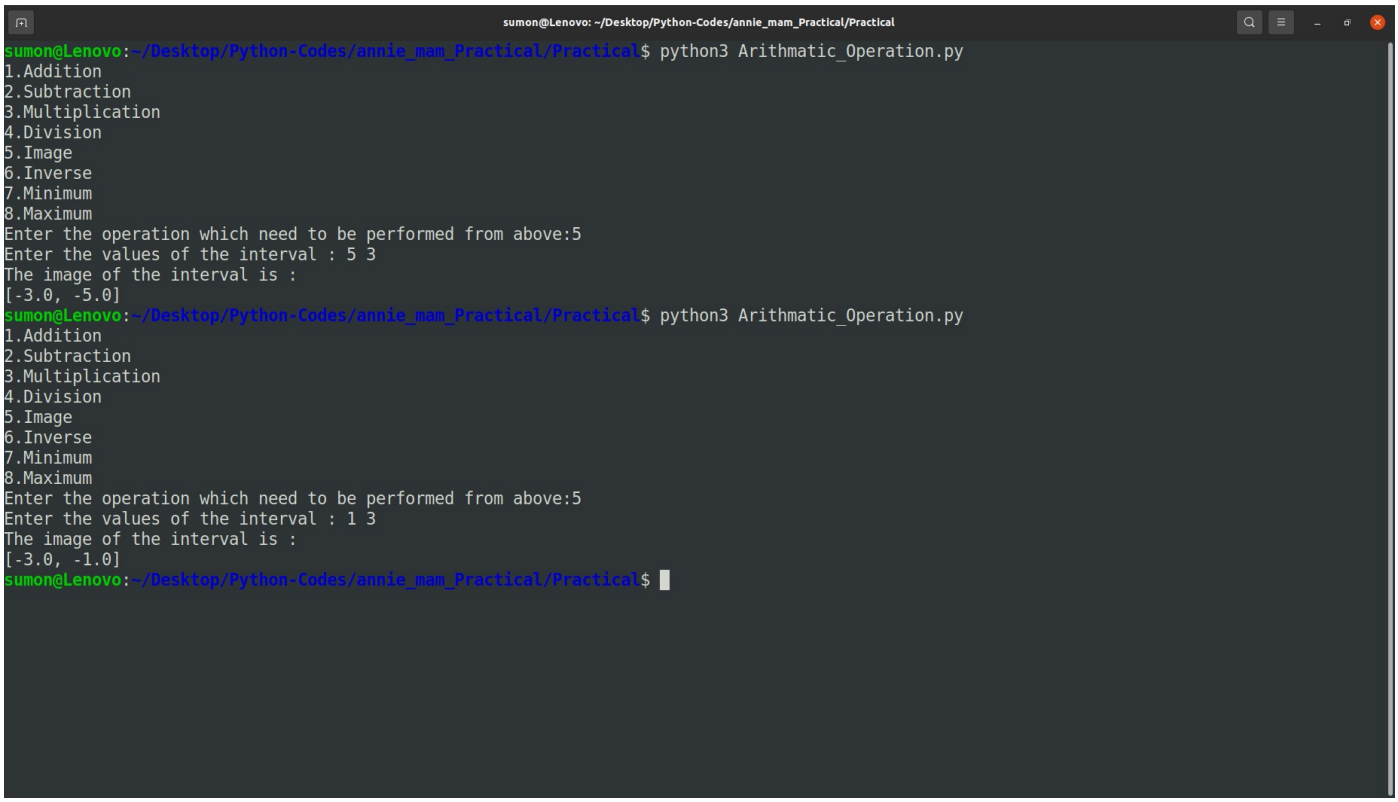
Output:

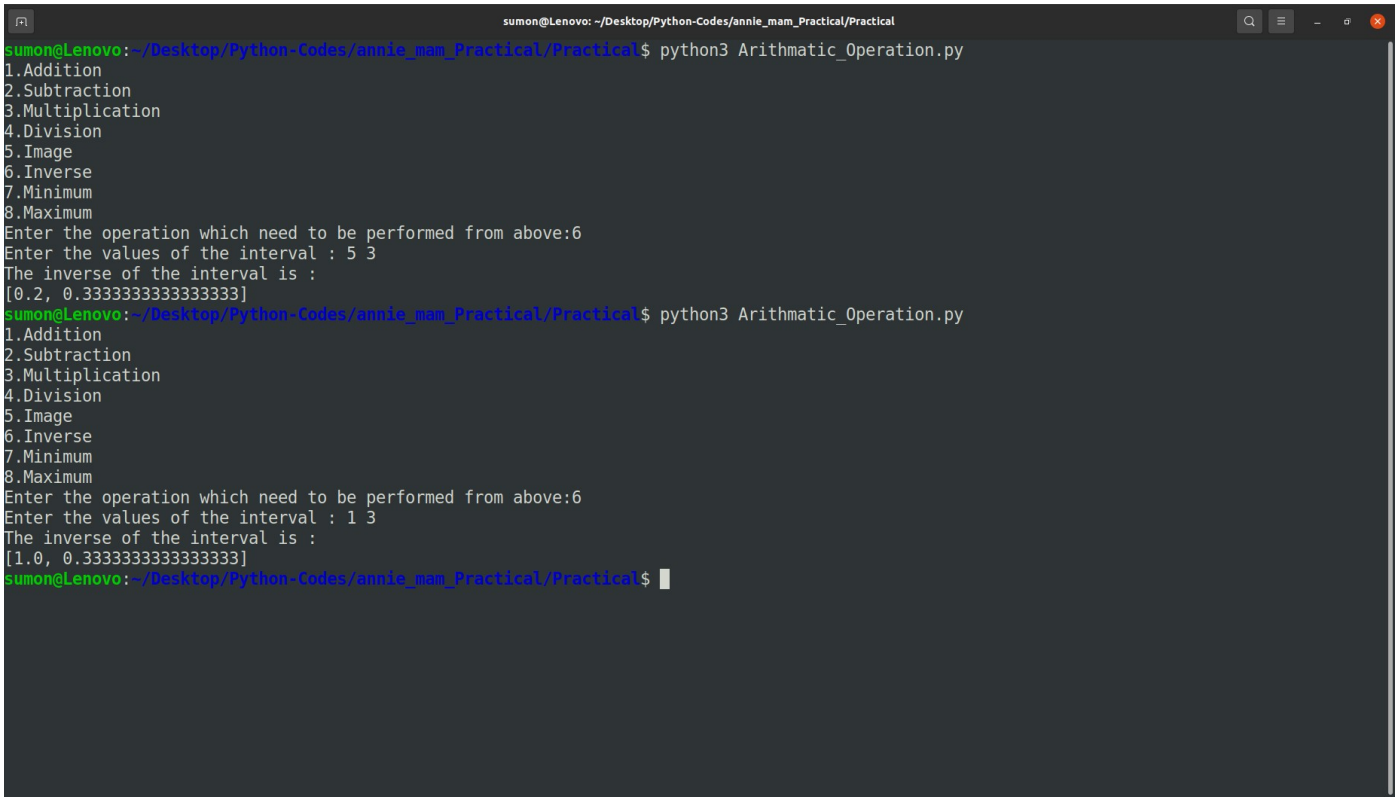



sumon@Lenovo: ~/Desktop/Python-Codes/annie_mam_Practical/Practical
sumon@Lenovo:~/Desktop/Python-Codes/annie_mam_Practical/Practical$ python3 Arithmatic_Operation.py
1.Addition
2.Subtraction
3.Multiplication
4.Division
5.Image
6.Inverse
7.Minimum
8.Maximum
Enter the operation which need to be performed 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
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
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
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$



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) i) $\underline{E} = [2,4]$, $\underline{I} = [-4,5]$

a) Max operation :-

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

b) min operation :-

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

ii) $\underline{F} = [5,6]$, $\underline{G} = [9,2]$

a) Max operation :-

$$\underline{F} \vee \underline{G} = [5,6] \vee [9,2] = [9,6]$$

b) min operation :-

$$\underline{F} \wedge \underline{G} = [5,6] \wedge [9,2] = [5,2]$$

Code:

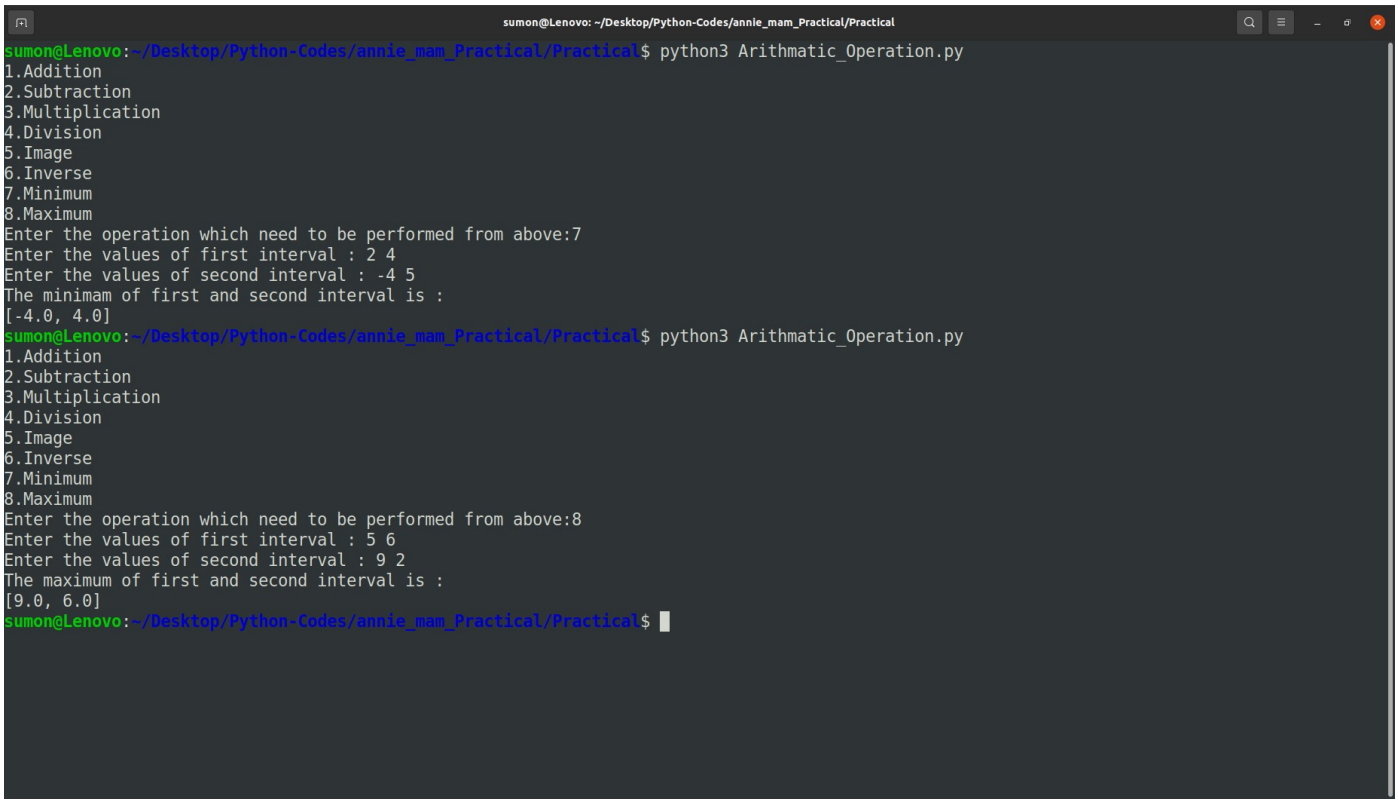
```
def ADD(x,y):
r=[]
r.append(x[0]+y[0])
r.append(x[1]+y[1])
return r
def SUB(x,y):
r=[]
r.append(x[0]-y[1])
r.append(x[1]-y[0])
return r
def MUL(x,y):
r=[]
r.append(x[0]*y[0])
r.append(x[1]*y[1])
return r
def DIV(x,y):
r=[]
r.append(x[0]/y[1])
r.append(x[1]/y[0])
return r
def IMG(x):
r=[]
r.append(-x[1])
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:



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 :-

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

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

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

* outer product :-

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

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

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

$$= 1.0$$

ii) $\underline{c} = (0.5, 0.7, 0.2, 0.3, 1, 0.8)$

$\underline{d} = (0, 0.2, 0.1, 0.4, 0.6, 1.0)$

inner product :-

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

$= (0.5 \wedge 0) \vee (0.7 \wedge 0.2) \vee (0.2 \wedge 0.1) \vee (0.3 \wedge 0.4) \vee (1 \wedge 0.6)$
$\vee (0.8 \wedge 1.0)$

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

outer product :-

~~$(0.8 \wedge 0)$~~

$(0.5 \vee 0) \wedge (0.7 \vee 0.2) \wedge (0.2 \vee 0.1) \wedge (0.3 \vee 0.4) \wedge (1 \vee 0.6)$
$\wedge (0.8 \vee 1.0)$

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

$= 0.2$

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: