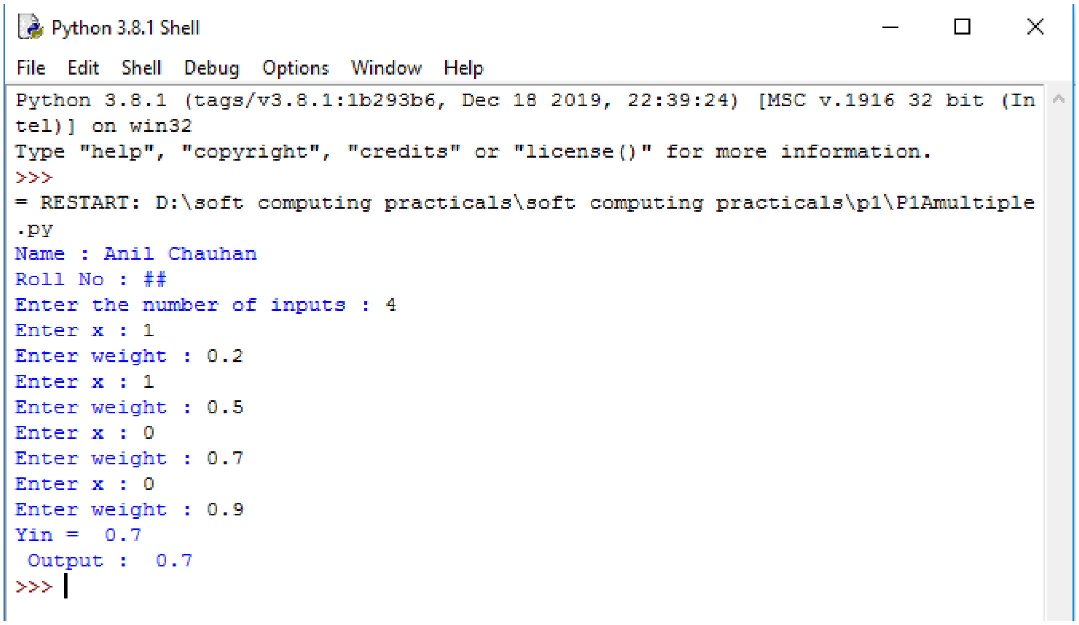
Practical 1-A

Code:-

```
print("Name : Anil Chauhan")
print("Roll No : ##")
n=int(input("Enter the number of inputs : "))
yin=0
for i in range(n):
    x=float(input("Enter x : "))
    w=float(input("Enter weight : "))
    yin=yin + x*w

print("Yin = ", yin)

if(yin<0):
    output=0
elif (yin>1):
    output=1
else:
    output=yin
print (" Output : " , output)
```

Output:



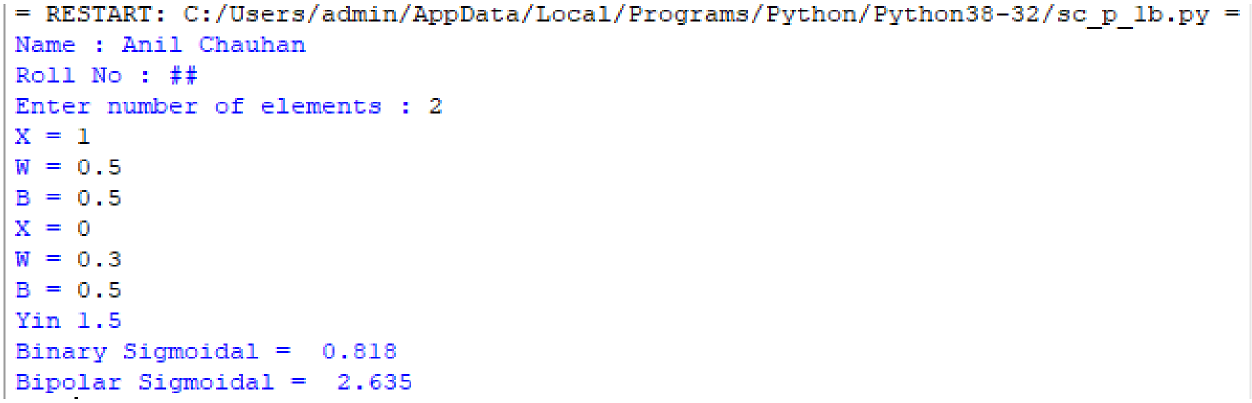
```
Python 3.8.1 (tags/v3.8.1:1b293b6, Dec 18 2019, 22:39:24) [MSC v.1916 32 bit (In
tel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> 
= RESTART: D:\soft computing practicals\soft computing practicals\p1\P1Amultiple
.py
Name : Anil Chauhan
Roll No : ##
Enter the number of inputs : 4
Enter x : 1
Enter weight : 0.2
Enter x : 1
Enter weight : 0.5
Enter x : 0
Enter weight : 0.7
Enter x : 0
Enter weight : 0.9
Yin =  0.7
 Output :  0.7
>>> 
```

Practical 1-B

Code:-

```
import math
print("Name : Anil Chauhan ")
print("Roll No : ##")
n=int(input("Enter number of elements : "))
yin=0
for i in range(0,n):
    x=float(input("X = "))
    w=float(input("W = "))
    b=float(input("B = "))
    yin = yin + x*w +b
print("Yin" , yin)
binary_sigmoidal = (1 / (1 + (math.e**(-yin))))
print("Binary Sigmoidal = " , round(binary_sigmoidal,3))
bipolar_sigmoidal = (2 / (1 + (math.e**(-yin))))+1
print("Bipolar Sigmoidal = " , round(bipolar_sigmoidal,3))
```

Output:



```
= RESTART: C:/Users/admin/AppData/Local/Programs/Python/Python38-32/sc_p_1b.py =
Name : Anil Chauhan
Roll No : ##
Enter number of elements : 2
X = 1
W = 0.5
B = 0.5
X = 0
W = 0.3
B = 0.5
Yin 1.5
Binary Sigmoidal =  0.818
Bipolar Sigmoidal =  2.635
```

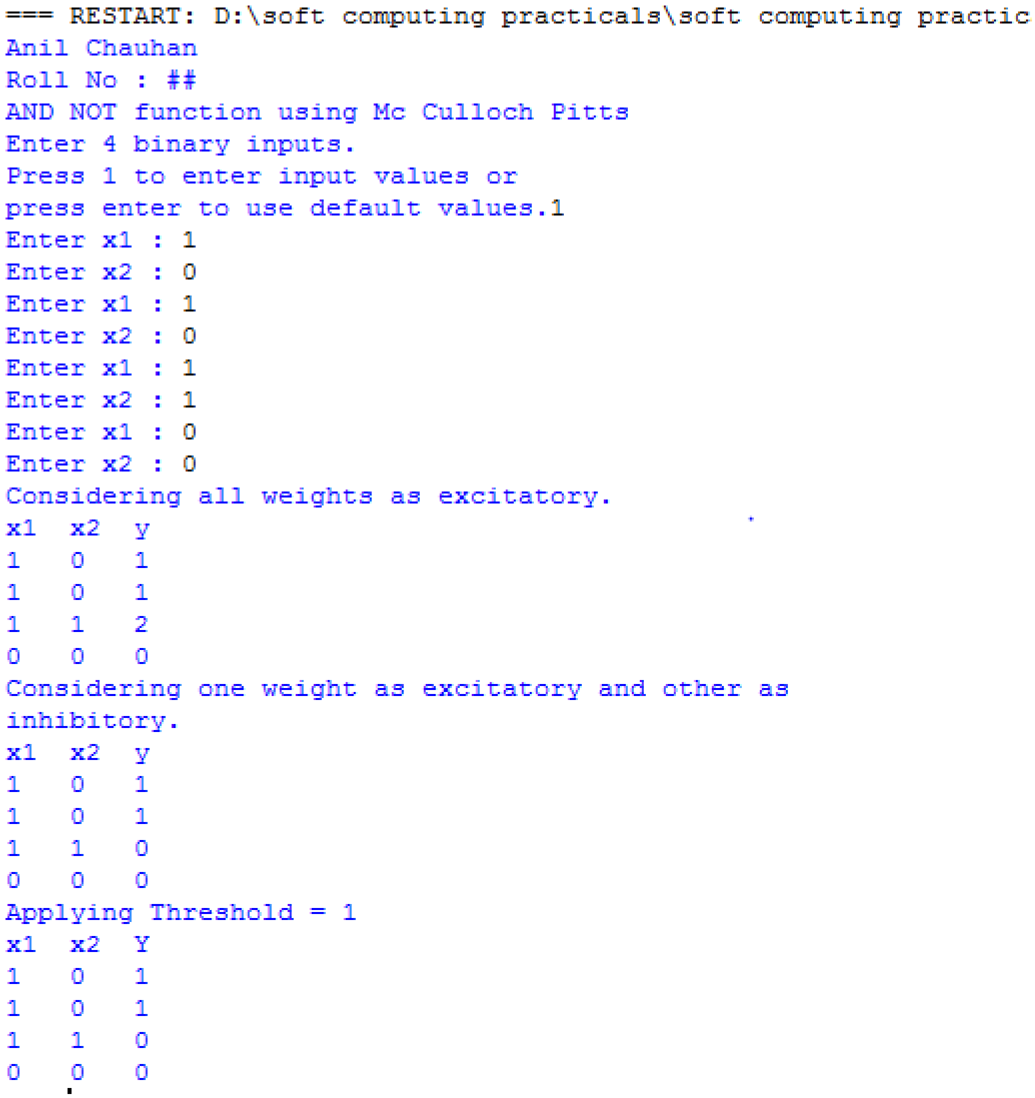
Practical 2-A

Code:-

```
print("Anil Chauhan")
print("Roll No : ##       ")
print("AND NOT function using Mc Culloch Pitts")
print("Enter 4 binary inputs.");
x1inputs=[]     x2inputs=[]
c=input("Press 1 to enter input values or press enter to use default values.")
if(c=="1"):
   for i in range(0,4):
      x1=int(input("Enter x1 : "))
      x1inputs.append(x1)
      x2=int(input("Enter x2 : "))
      x2inputs.append(x2)
else:
   x1inputs=[1,1,0,0]                x2inputs=[1,0,1,0]

print("Considering all weights as excitatory.");
w1 = [1,1,1,1]            w2 = [1,1,1,1]            y=[]
for i in range(0,4):     y.append(x1inputs[i]*w1[i] + x2inputs[i]*w2[i])
print("x1", " x2", " y")
for i in range(0,4):     print(x1inputs[i]," " ,x2inputs[i]," " , y[i])
print("Considering one weight as excitatory and other as inhibitory.");
w1 = [1,1,1,1]            w2 = [-1,-1,-1,-1]        y=[]
for i in range(0,4):     y.append(x1inputs[i]*w1[i] + x2inputs[i]*w2[i])
print("x1", " x2 ", "y")
for i in range(0,4):     print(x1inputs[i]," ",x2inputs[i]," " , y[i])
print("Applying Threshold = 1")
Y=[]
for i in range(0,4):
   if(y[i]>=1):
      value=1
      Y.append(value)
   else:
      value=0
      Y.append(value)
print("x1 ", "x2 ", "Y")
for i in range(0,4):
        print(x1inputs[i]," ", x2inputs[i]," ",Y[i]
```

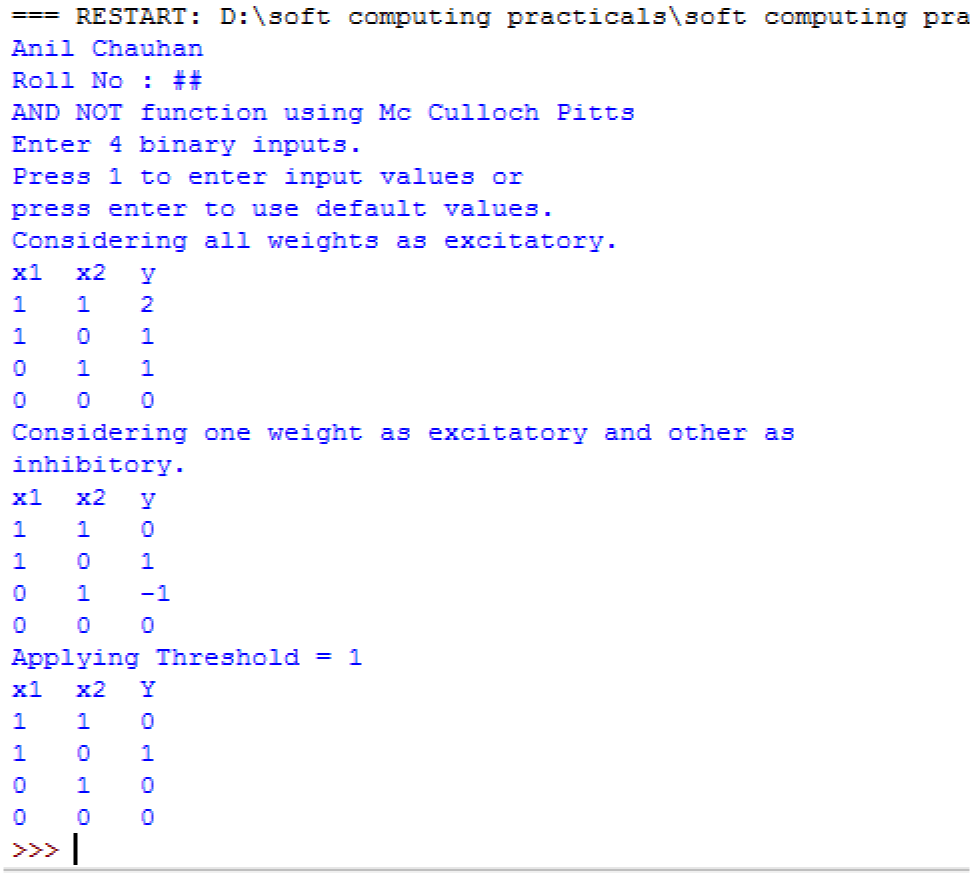
Output:

With User Entered values



```
=== RESTART: D:\soft computing practicals\soft computing practic
Anil Chauhan
Roll No : ##
AND NOT function using Mc Culloch Pitts
Enter 4 binary inputs.
Press 1 to enter input values or
press enter to use default values.1
Enter x1 : 1
Enter x2 : 0
Enter x1 : 1
Enter x2 : 0
Enter x1 : 1
Enter x2 : 1
Enter x1 : 0
Enter x2 : 0
Considering all weights as excitatory.
x1  x2  y
1   0   1
1   0   1
1   1   2
0   0   0
Considering one weight as excitatory and other as
inhibitory.
x1  x2  y
1   0   1
1   0   1
1   1   0
0   0   0
Applying Threshold = 1
x1  x2  Y
1   0   1
1   0   1
1   1   0
0   0   0
```

With default values



```
=== RESTART: D:\soft computing practicals\soft computing pra
Anil Chauhan
Roll No : ##
AND NOT function using Mc Culloch Pitts
Enter 4 binary inputs.
Press 1 to enter input values or
press enter to use default values.
Considering all weights as excitatory.
x1  x2  y
1   1   2
1   0   1
0   1   1
0   0   0
Considering one weight as excitatory and other as
inhibitory.
x1  x2  y
1   1   0
1   0   1
0   1   -1
0   0   0
Applying Threshold = 1
x1  x2  Y
1   1   0
1   0   1
0   1   0
0   0   0
>>> 
```

Practical 2-B

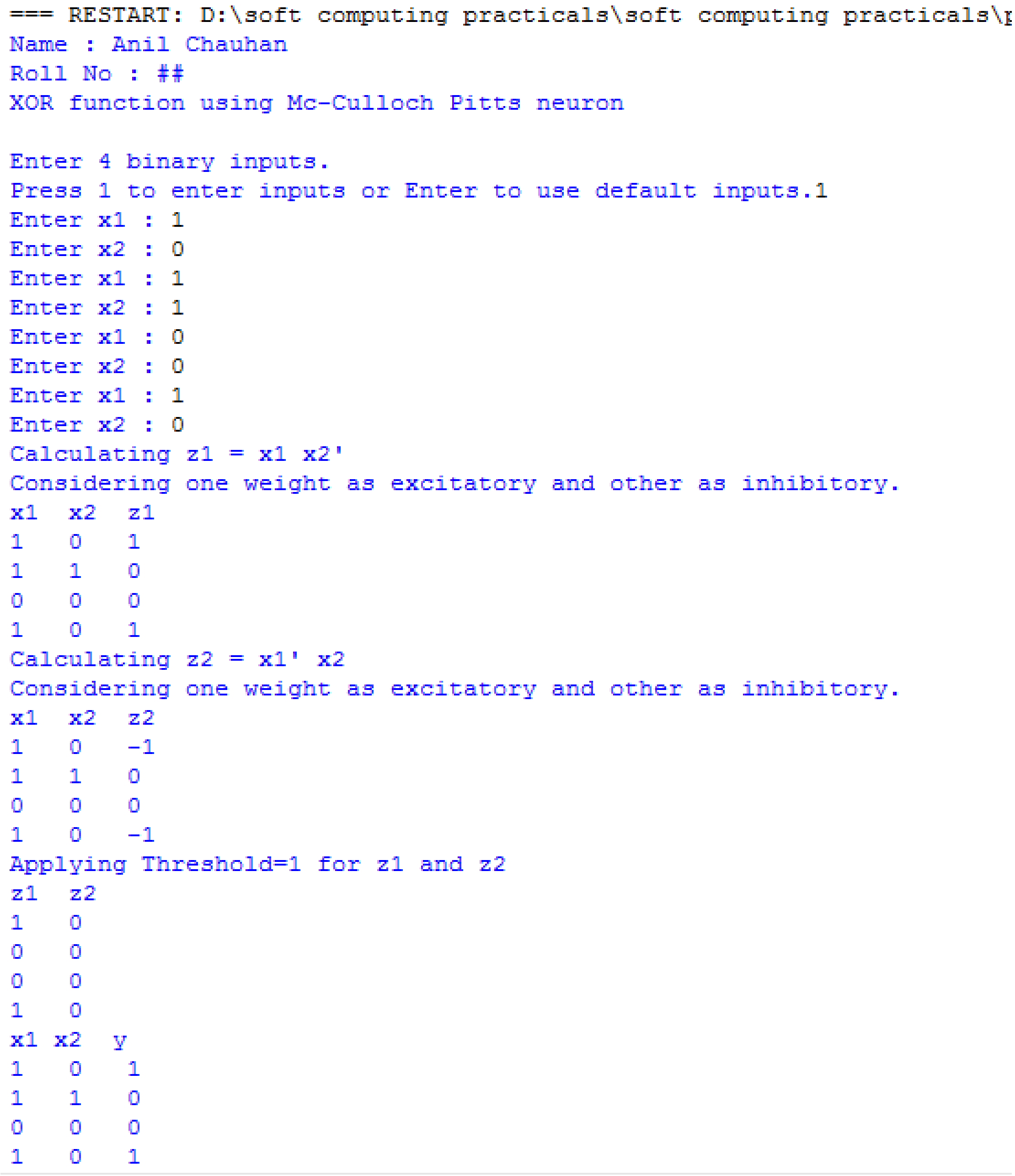
Code:-

```
print("Name : Anil Chauhan")
print("Roll No : ## ")
print("XOR function using Mc-Culloch Pitts neuron")
print()
print("Enter 4 binary inputs.");

x1inputs=[]
x2inputs=[]

c=input("Press 1 to enter inputs or Enter to use default inputs.")

if(c=="1"):
   for i in range(0,4):
         x1=int(input("Enter x1 : "))
         x1inputs.append(x1)
         x2=int(input("Enter x2 : "))
         x2inputs.append(x2)
else:
   x1inputs=[1,1,0,0]
   x2inputs=[1,0,1,0]

print("Calculating z1 = x1 x2'")

print("Considering one weight as excitatory and other as inhibitory.");
```

```
w1 = [1,1,1,1]
w2 = [-1,-1,-1,-1]
z1=[]
for i in range(0,4):
    z1.append(x1inputs[i]*w1[i] + x2inputs[i]*w2[i])
print("x1 " , "x2 " , "z1")
for i in range(0,4):
    print(x1inputs[i] ," ",  x2inputs[i]," " ,  z1[i])
print("Calculating z2 = x1' x2")

print("Considering one weight as excitatory and other as inhibitory.");

w1 = [-1,-1,-1,-1]
w2 = [1,1,1,1]

z2=[]

for i in range(0,4):
    z2.append(x1inputs[i]*w1[i] + x2inputs[i]*w2[i])

print("x1 " , "x2 " , "z2")
for i in range(0,4):
    print(x1inputs[i] ," " ,  x2inputs[i] ," ", z2[i])
print("Applying Threshold=1 for z1 and z2")
for i in range(0,4):
    if(z1[i]>=1):
        z1[i]=1
    else:
        z1[i]=0
```

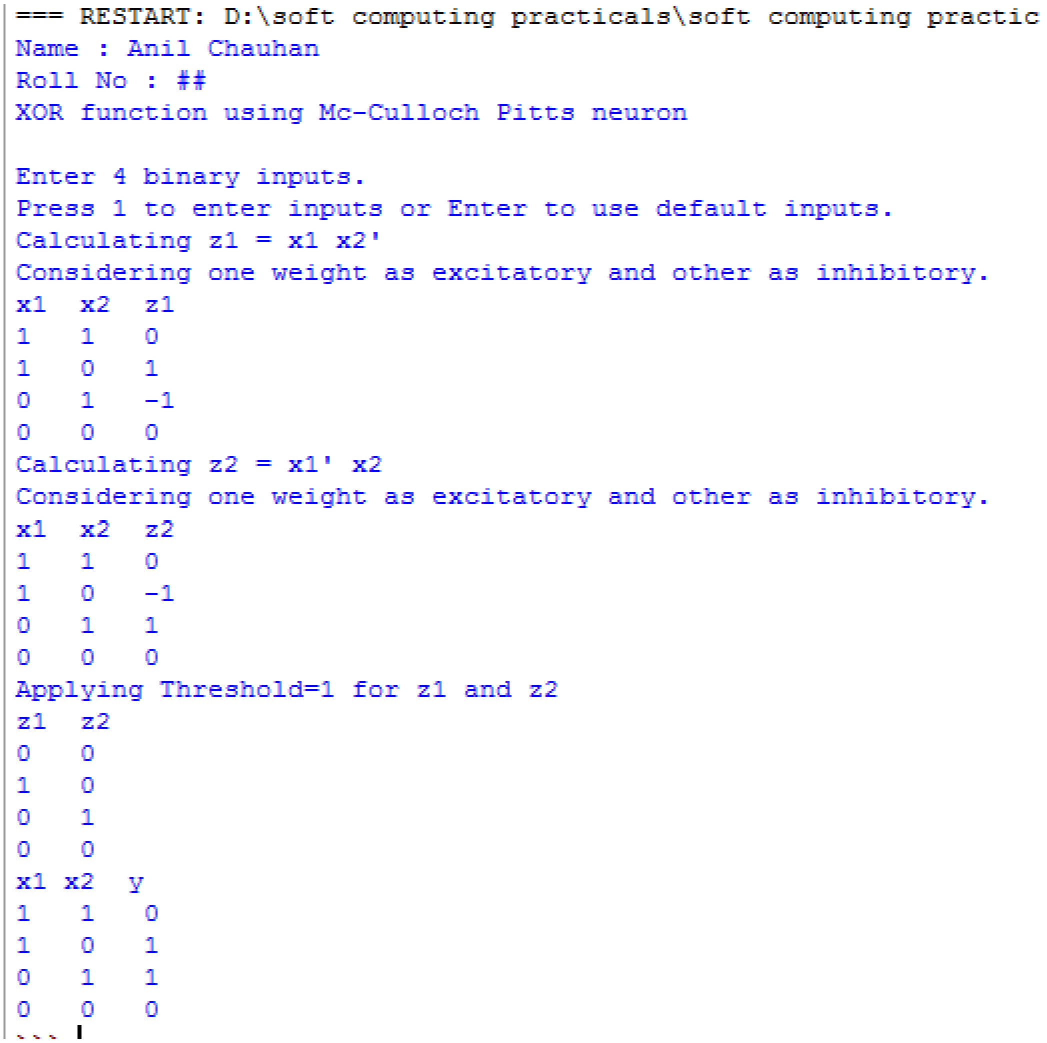
Output:

With user defined input



```
=== RESTART: D:\soft computing practicals\soft computing practicals\
Name : Anil Chauhan
Roll No : ##
XOR function using Mc-Culloch Pitts neuron

Enter 4 binary inputs.
Press 1 to enter inputs or Enter to use default inputs.1
Enter x1 : 1
Enter x2 : 0
Enter x1 : 1
Enter x2 : 1
Enter x1 : 0
Enter x2 : 0
Enter x1 : 1
Enter x2 : 0
Calculating z1 = x1 x2'
Considering one weight as excitatory and other as inhibitory.
x1  x2  z1
1   0   1
1   1   0
0   0   0
1   0   1
Calculating z2 = x1' x2
Considering one weight as excitatory and other as inhibitory.
x1  x2  z2
1   0   -1
1   1   0
0   0   0
1   0   -1
Applying Threshold=1 for z1 and z2
z1  z2
1   0
0   0
0   0
1   0
x1 x2  y
1   0   1
1   1   0
0   0   0
1   0   1
```

With default input values



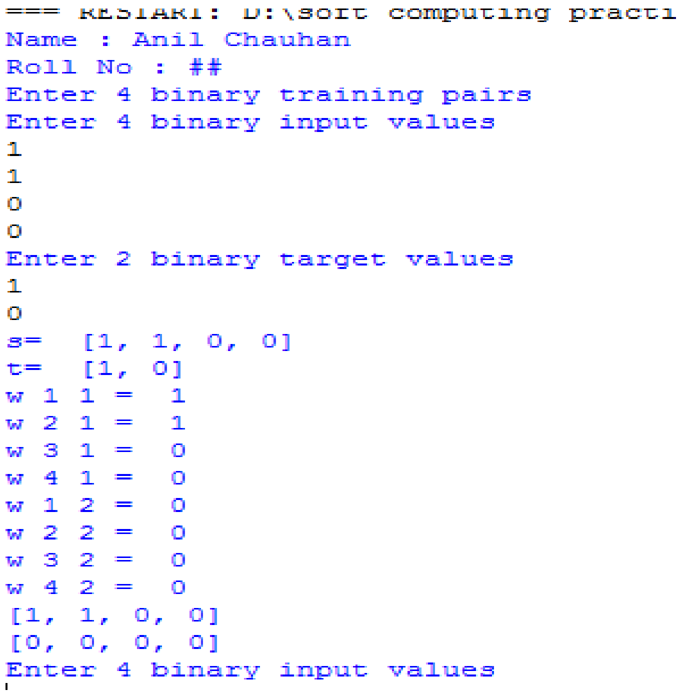
```
=== RESTART: D:\soft computing practicals\soft computing practic
Name : Anil Chauhan
Roll No : ##
XOR function using Mc-Culloch Pitts neuron

Enter 4 binary inputs.
Press 1 to enter inputs or Enter to use default inputs.
Calculating z1 = x1 x2'
Considering one weight as excitatory and other as inhibitory.
x1  x2  z1
1   1   0
1   0   1
0   1   -1
0   0   0
Calculating z2 = x1' x2
Considering one weight as excitatory and other as inhibitory.
x1  x2  z2
1   1   0
1   0   -1
0   1   1
0   0   0
Applying Threshold=1 for z1 and z2
z1  z2
0   0
1   0
0   1
0   0
x1 x2  y
1   1   0
1   0   1
0   1   1
0   0   0
```

Practical 3-A

Code:-

```
print("Name : Anil Chauhan")

print("Roll No : ##")

print("Enter 4 binary training pairs")

w1=[0,0,0,0]

w2=[0,0,0,0]


for m in range(0,4):

    print("Enter 4 binary input values")

    s=[]

    t=[]

    for i in range(0,4):

        x=int(input())

        s.append(x)

    print("Enter 2 binary target values")

    for i in range(0,2):

       y=int(input())

       t.append(y)

    print("s= ",s)

    print("t= ",t)

    w1new=[]

    for i in range(0,4):

       newweight1=w1[i] + s[i]*t[0]

       w1new.append(newweight1)

    '''print new weights'''

    for i in range(0,4):

        print("w",(i+1),"1 = ",w1new[i])
    w2new=[]
    for i in range(0,4):
        newweight2=w2[i] + s[i]*t[1]
        w2new.append(newweight2)
    for i in range(0,4):
         print("w",(i+1),"2 = ",w2new[i])
    w1=w1new
    w2=w2new
    print(w1)
    print(w2)
print("The final weight matrix is : ")
print("W = ")
for i in range(0,4):
    print(w1[i] , w2[i])
print("Done")
```

Output:



```
=== RESTART: D:\soft computing practi
Name : Anil Chauhan
Roll No : ##
Enter 4 binary training pairs
Enter 4 binary input values
1
1
0
0
Enter 2 binary target values
1
0
s=  [1, 1, 0, 0]
t=  [1, 0]
w 1 1 =  1
w 2 1 =  1
w 3 1 =  0
w 4 1 =  0
w 1 2 =  0
w 2 2 =  0
w 3 2 =  0
w 4 2 =  0
[1, 1, 0, 0]
[0, 0, 0, 0]
Enter 4 binary input values
```

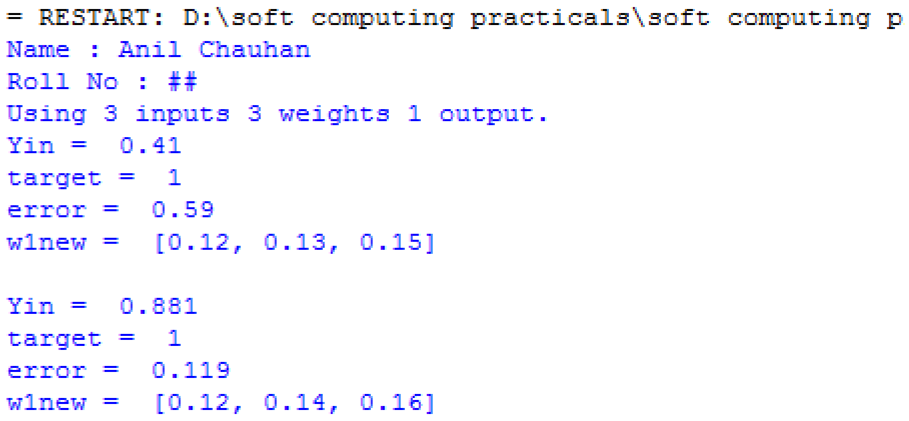
Practical 3-B

Code:-

```
print("Name : Anil Chauhan")

print("Roll No : ##")

import math

print("Using 3 inputs 3 weights 1 output.")

x1=[0.3,0.5,0.8]  #inputs

w1=[0.1,0.1,0.1]  #weights

t=1              #TARGET

a=0.1      #alpha

diff=1     #initial difference

yin=0      #initial net input


while(diff>0.4):


   for i in range(0,3):

      yin = yin + (x1[i]*w1[i])


   yin = yin + 0.25

   yin=round(yin,3)

   print("Yin = ",yin)

   print("target = ",t)

   diff=t-yin


   diff=round(diff,3)

   diff=math.fabs(diff)

   print("error = ",diff)

   neww1=[]
```

```
for i in range(0,3):    #update weights
  w1new=w1[i] + a*diff*x1[i]
  w1new=round(w1new,2)
  neww1.append(w1new)
print("w1new = ",neww1)
w1=neww1
print()
```

output:



```
= RESTART: D:\soft computing practicals\soft computing p
Name : Anil Chauhan
Roll No : ##
Using 3 inputs 3 weights 1 output.
Yin =  0.41
target =  1
error =  0.59
w1new =  [0.12, 0.13, 0.15]

Yin =  0.881
target =  1
error =  0.119
w1new =  [0.12, 0.14, 0.16]
```

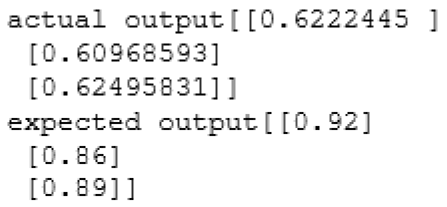
Practical 4-A

Code:-

```
import numpy as np

X=np.array(([2,9],[1,5],[3,6]),dtype=float)

Y=np.array(([92],[86],[89]),dtype=float)

#scale units

X=X/np.amax(X,axis=0)

Y=Y/100;


class NN(object):

        def __init__(self):

                self.inputsize=2

                self.outputsize=1

                self.hiddensize=3


                self.W1=np.random.randn(self.inputsize,self.hiddensize)

                self.W2=np.random.randn(self.hiddensize,self.outputsize)


        def forward(self,X):

                self.z=np.dot(X,self.W1)

                self.z2=self.sigmoidal(self.z)

                self.z3=np.dot(self.z2,self.W2)

                op=self.sigmoidal(self.z3)

                return op;


        def sigmoidal(self,s):

                return 1/(1+np.exp(-s))
```

```
obj=NN()

op=obj.forward(X)

print("actual output"+str(op))

print("expected output"+str(Y))
```

output:



```
actual output[[0.6222445 ]
 [0.60968593]
 [0.62495831]]
expected output[[0.92]
 [0.86]
 [0.89]]
```

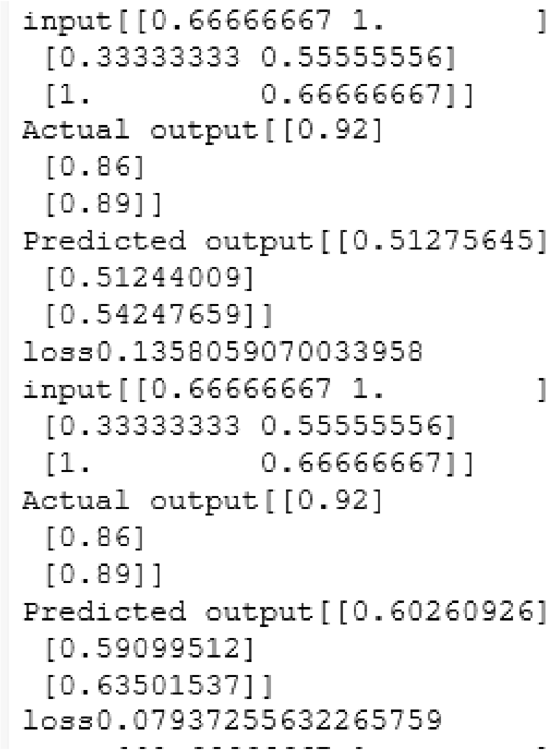
Practical 4-B

Code:-

```
import numpy as np

X=np.array(([2,9],[1,5],[3,6]),dtype=float)

Y=np.array(([92],[86],[89]),dtype=float)

X=X/np.amax(X,axis=0)

Y=Y/100;

class NN(object):

   def __init__(self):

      self.inputsize=2

      self.outputsize=1

      self.hiddensize=3

      self.W1=np.random.randn(self.inputsize,self.hiddensize)

      self.W2=np.random.randn(self.hiddensize,self.outputsize)

   def forward(self,X):

      self.z=np.dot(X,self.W1)

      self.z2=self.sigmoidal(self.z)

      self.z3=np.dot(self.z2,self.W2)

      op=self.sigmoidal(self.z3)

      return op;

   def sigmoidal(self,s):

      return 1/(1+np.exp(-s))

   def sigmoidalprime(self,s):

      return s* (1-s)

   def backward(self,X,Y,o):

      self.o_error=Y-o

      self.o_delta=self.o_error * self.sigmoidalprime(o)

      self.z2_error=self.o_delta.dot(self.W2.T)
```

```
        self.z2_delta=self.z2_error * self.sigmoidalprime(self.z2)
        self.W1 = self.W1 + X.T.dot(self.z2_delta)
        self.W2= self.W2+ self.z2.T.dot(self.o_delta)
    def train(self,X,Y):
        o=self.forward(X)
        self.backward(X,Y,o)
obj=NN()
for i in range(2000):
    print("input"+str(X))
    print("Actual output"+str(Y))
    print("Predicted output"+str(obj.forward(X)))
    print("loss"+str(np.mean(np.square(Y-obj.forward(X)))))
    obj.train(X,Y)
```

output:



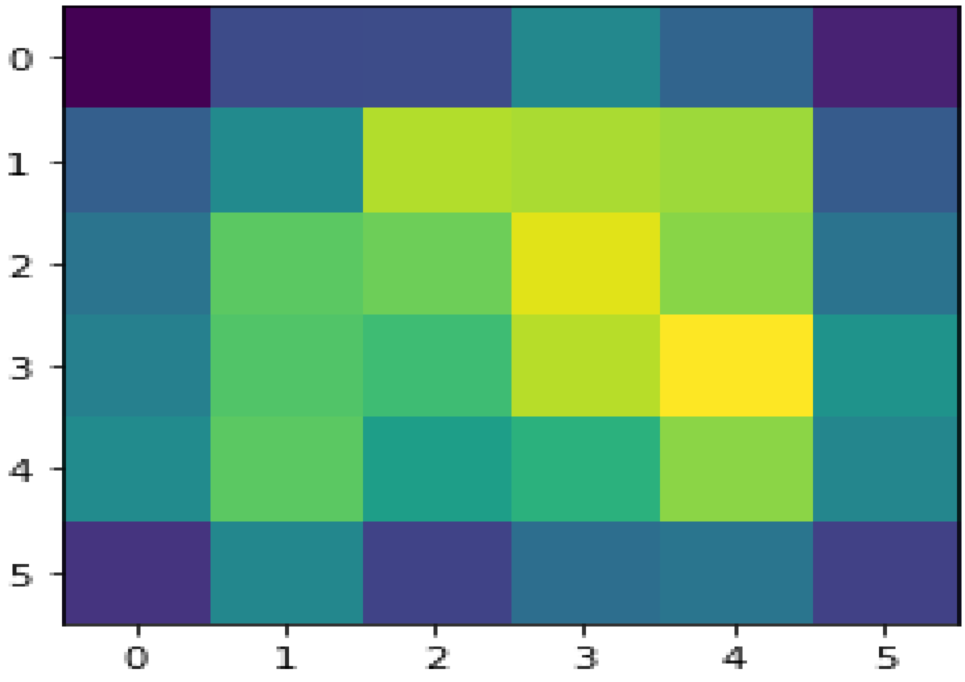
```
input[[0.66666667 1.        ]
 [0.33333333 0.55555556]
 [1.         0.66666667]]
Actual output[[0.92]
 [0.86]
 [0.89]]
Predicted output[[0.51275645]
 [0.51244009]
 [0.54247659]]
loss0.1358059070033958
input[[0.66666667 1.        ]
 [0.33333333 0.55555556]
 [1.         0.66666667]]
Actual output[[0.92]
 [0.86]
 [0.89]]
Predicted output[[0.60260926]
 [0.59099512]
 [0.63501537]]
loss0.07937255632265759
```

Practical 6-A

Code:-

```
from minisom import MiniSom

import matplotlib.pyplot as plt

data = [[ 0.80,  0.55,  0.22,  0.03],

        [ 0.82,  0.50,  0.23,  0.03],

        [ 0.80,  0.54,  0.22,  0.03],

        [ 0.80,  0.53,  0.26,  0.03],

        [ 0.79,  0.56,  0.22,  0.03],

        [ 0.75,  0.60,  0.25,  0.03],

        [ 0.77,  0.59,  0.22,  0.03]]


som = MiniSom(6, 6, 4, sigma=0.3, learning_rate=0.5)

 # initialization of 6x6 SOM

som.train_random(data, 100)

# trains the SOM with 100 iterations

plt.imshow(som.distance_map())
```

Output:



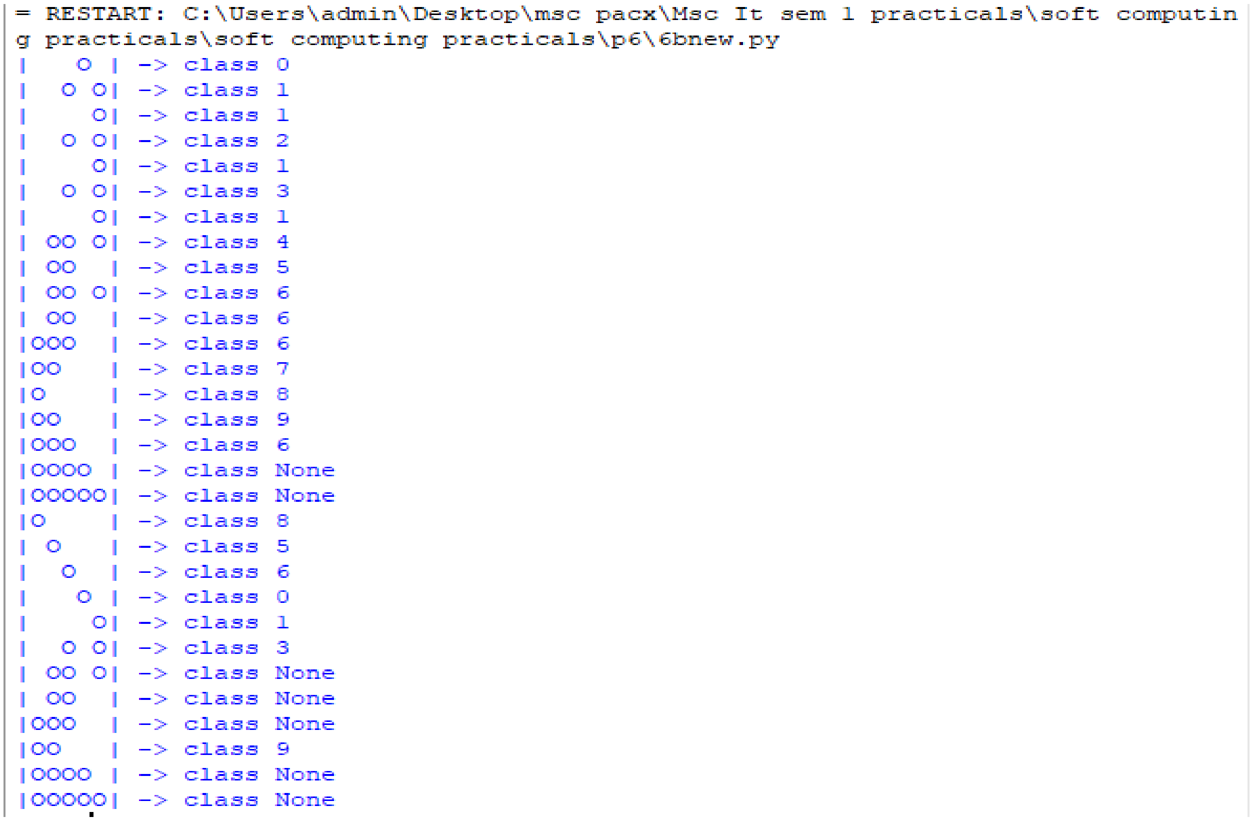
Practical 6-B

Code:-

```
from __future__ import print_function

from __future__ import division

import numpy as np




class ART:

    def __init__(self, n=5, m=10, rho=.5):

        # Comparison layer

        self.F1 = np.ones(n)

        # Recognition layer

        self.F2 = np.ones(m)

        # Feed-forward weights

        self.Wf = np.random.random((m,n))

        # Feed-back weights

        self.Wb = np.random.random((n,m))

        # Vigilance

        self.rho = rho

        # Number of active units in F2

        self.active = 0




    def learn(self, X):
        # Compute F2 output and sort them (I)

        self.F2[...] = np.dot(self.Wf, X)

        I = np.argsort(self.F2[:self.active].ravel())[::-1]


        for i in I:

            # Check if nearest memory is above the vigilance level

            d = (self.Wb[:,i]*X).sum()/X.sum()

            if d >= self.rho:

                # Learn data

                self.Wb[:,i] *= X

                self.Wf[i,:] = self.Wb[:,i]/(0.5+self.Wb[:,i].sum())

                return self.Wb[:,i], i


        # No match found, increase the number of active units

        # and make the newly active unit to learn data

        if self.active < self.F2.size:

            i = self.active

            self.Wb[:,i] *= X

            self.Wf[i,:] = self.Wb[:,i]/(0.5+self.Wb[:,i].sum())

            self.active += 1

            return self.Wb[:,i], i


        return None,None

if __name__ == '__main__':
```

```
np.random.seed(1)

network = ART( 5, 10, rho=0.5)

data = ["   O ",
    "  O O",
    "    O",
    "  O O",
    "    O",
    "  O O",
    "    O",
    " OO O",
    " OO  ",
    " OO O",
    " OO  ",
    "OOO  ",
    "OO   ",
    "O    ",
    "OO   ",
    "OOO  ",
    "OOOO ",
    "OOOOO",
    "O    ",
    " O   ",
    "  O  ",
    "   O ",
    "    O",
    "  O O",
    " OO O",

    " OO  ",

    "OOO  ",

    "OO   ",

    "OOOO ",

    "OOOOO"]

X = np.zeros(len(data[0]))

for i in range(len(data)):

    for j in range(len(data[i])):

    X[j] = (data[i][j] == 'O')

    Z, k = network.learn(X)

    print("|%s|"%data[i],"-> class", k
```

output:



```
= RESTART: C:\Users\admin\Desktop\msc pacx\Msc It sem 1 practicals\soft computin
g practicals\soft computing practicals\p6\6bnew.py
|   O | -> class 0
|  O O| -> class 1
|    O| -> class 1
|  O O| -> class 2
|    O| -> class 1
|  O O| -> class 3
|    O| -> class 1
| OO O| -> class 4
| OO  | -> class 5
| OO O| -> class 6
| OO  | -> class 6
|OOO  | -> class 6
|OO   | -> class 7
|O    | -> class 8
|OO   | -> class 9
|OOO  | -> class 6
|OOOO | -> class None
|OOOOO| -> class None
|O    | -> class 8
| O   | -> class 5
|  O  | -> class 6
|   O | -> class 0
|    O| -> class 1
|  O O| -> class 3
| OO O| -> class None
| OO  | -> class None
|OOO  | -> class None
|OO   | -> class 9
|OOOO | -> class None
|OOOOO| -> class None
```

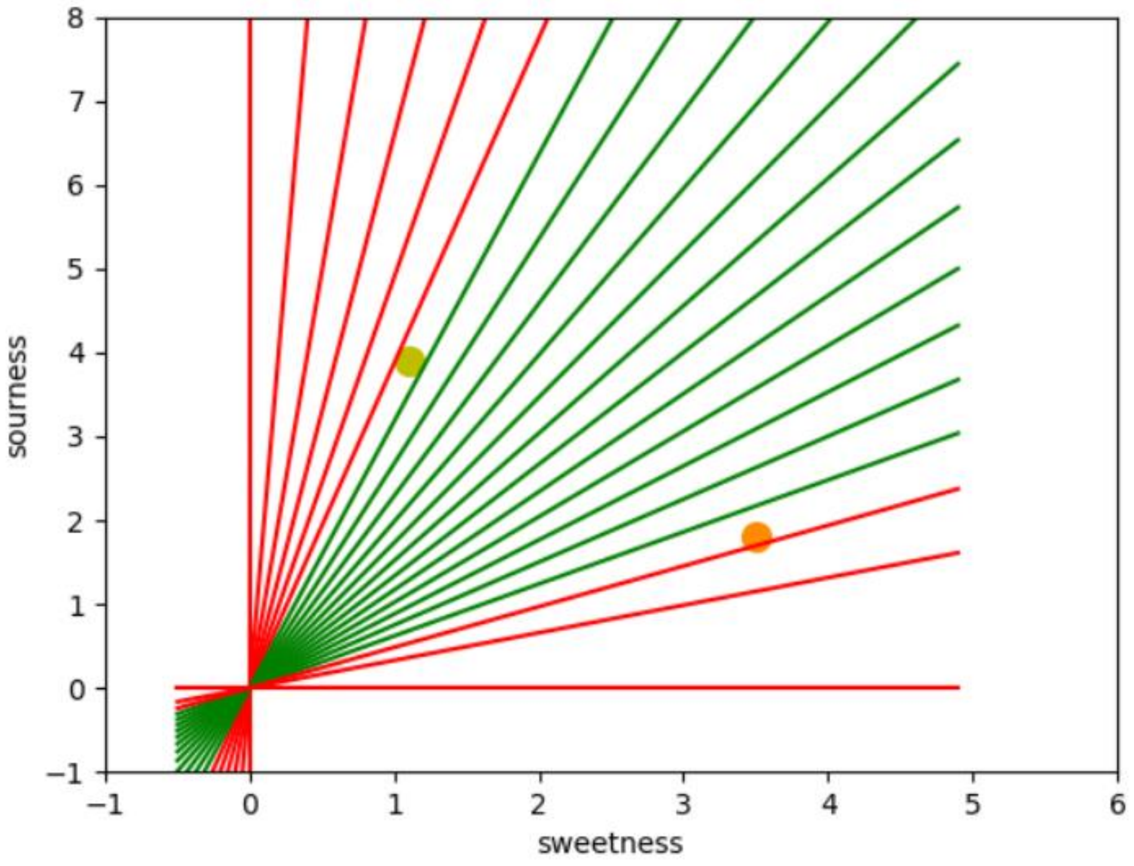
Practical 7-A

Code:-

```
import numpy as np

import matplotlib.pyplot as plt

def create_distance_function(a, b, c):

    """ 0 = ax + by + c """

    def distance(x, y):

        """ returns tuple (d, pos)

            d is the distance

            If pos == -1 point is below the line,

            0 on the line and +1 if above the line

        """

        nom = a * x + b * y + c

        if nom == 0:

            pos = 0

        elif (nom<0 and b<0) or (nom>0 and b>0):

            pos = -1

        else:

            pos = 1

        return (np.absolute(nom) / np.sqrt( a ** 2 + b ** 2), pos)

    return distance
```

```
points = [ (3.5, 1.8), (1.1, 3.9) ]


fig, ax = plt.subplots()
ax.set_xlabel("sweetness")
ax.set_ylabel("sourness")
ax.set_xlim([-1, 6])
ax.set_ylim([-1, 8])
X = np.arange(-0.5, 5, 0.1)
colors = ["r", ""] # for the samples
size = 10
for (index, (x, y)) in enumerate(points):
    if index== 0:
        ax.plot(x, y, "o",
                color="darkorange",
                markersize=size)
    else:
        ax.plot(x, y, "oy",
                markersize=size)
step = 0.05
for x in np.arange(0, 1+step, step):
    slope = np.tan(np.arccos(x))
    dist4line1 = create_distance_function(slope, -1, 0)
    #print("x: ", x, "slope: ", slope)
    Y = slope * X
    results = []

    for point in points:

        results.append(dist4line1(*point))

    #print(slope, results)

    if (results[0][1] != results[1][1]):

        ax.plot(X, Y, "g-")

    else:

        ax.plot(X, Y, "r-")


plt.show()
```

output:



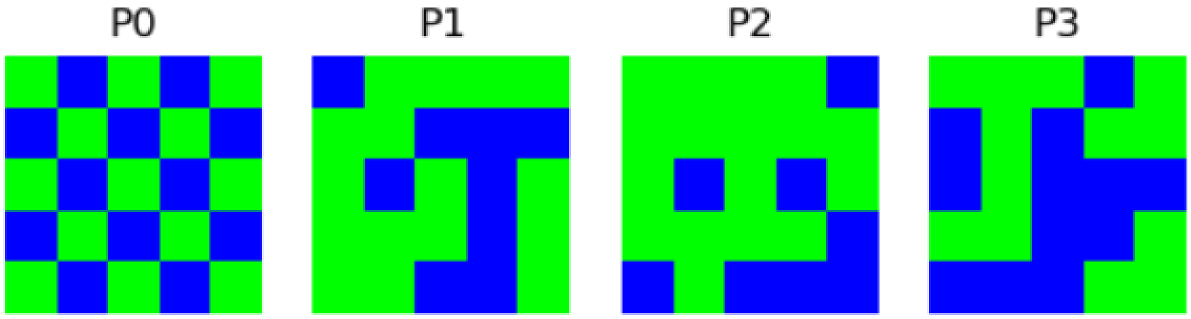
Practical 7-B

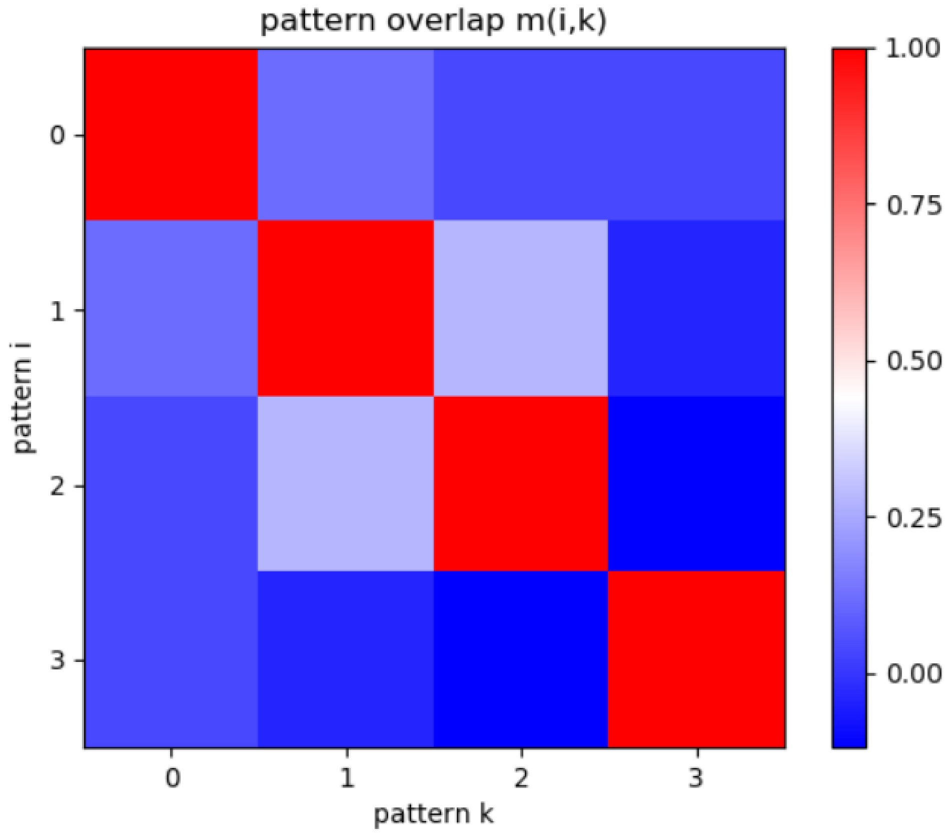
Code:-

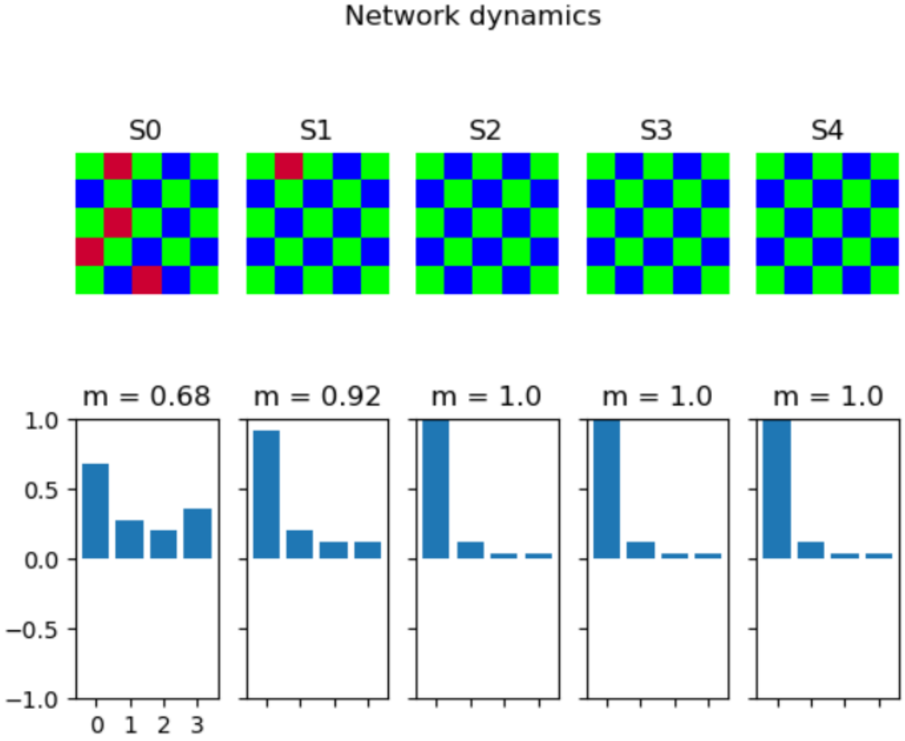
```
import matplotlib.pyplot as plt

from neurodynex.hopfield_network import network, pattern_tools, plot_tools


pattern_size = 5


# create an instance of the class HopfieldNetwork

hopfield_net = network.HopfieldNetwork(nr_neurons= pattern_size**2)

# instantiate a pattern factory

factory = pattern_tools.PatternFactory(pattern_size, pattern_size)

# create a checkerboard pattern and add it to the pattern list

checkerboard = factory.create_checkerboard()

pattern_list = [checkerboard]


# add random patterns to the list

pattern_list.extend(factory.create_random_pattern_list(nr_patterns=3, on_probability=0.5))

plot_tools.plot_pattern_list(pattern_list)

# how similar are the random patterns and the checkerboard? Check the overlaps

overlap_matrix = pattern_tools.compute_overlap_matrix(pattern_list)

plot_tools.plot_overlap_matrix(overlap_matrix)

# let the hopfield network "learn" the patterns. Note: they are not stored

# explicitly but only network weights are updated !
```

```
hopfield_net.store_patterns(pattern_list)


# create a noisy version of a pattern and use that to initialize the network

noisy_init_state = pattern_tools.flip_n(checkerboard, nr_of_flips=4)

hopfield_net.set_state_from_pattern(noisy_init_state)


# from this initial state, let the network dynamics evolve.

states = hopfield_net.run_with_monitoring(nr_steps=4)


# each network state is a vector. reshape it to the same shape used to create the patterns.

states_as_patterns = factory.reshape_patterns(states)

# plot the states of the network

plot_tools.plot_state_sequence_and_overlap(states_as_patterns, pattern_list, reference_idx=0,
suptitle="Network dynamics")
```

output:




pattern overlap m(i,k)
pattern i
pattern k
0
1
2
3
0
1
2
3
1.00
0.75
0.50
0.25
0.00

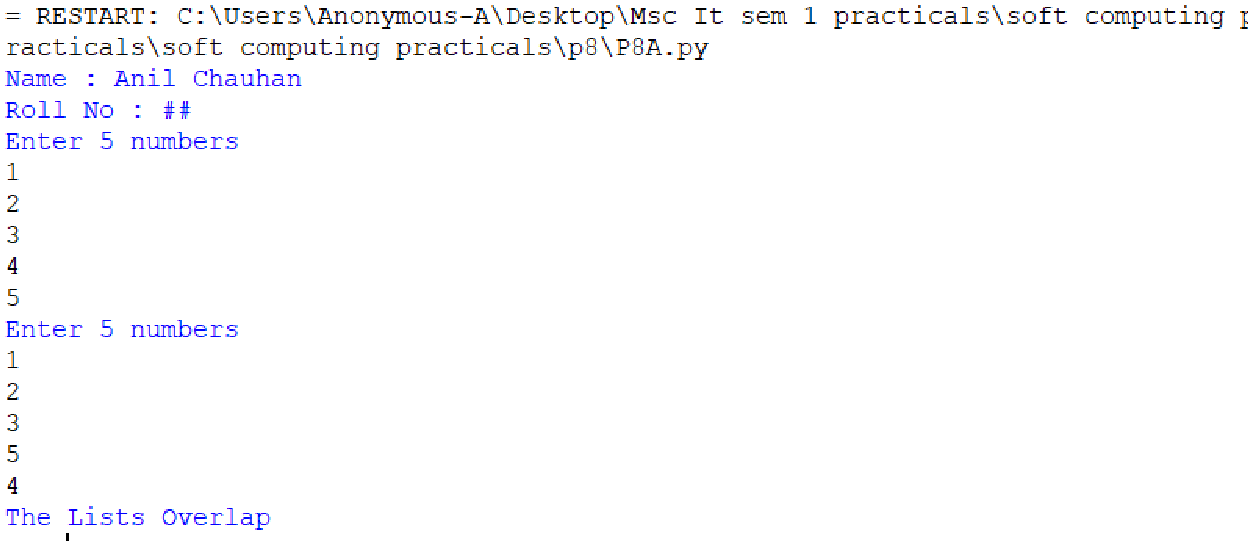

Network dynamics
S0
S1
S2
S3
S4
m = 0.68
m = 0.92
m = 1.0
m = 1.0
m = 1.0
1.0
0.5
0.0
−0.5
−1.0
0
1
2
3

Practical 8-A1

Code:-

```
print("Name : Anil Chauhan")
print("Roll No : ##")
list1=[]
print("Enter 5 numbers")
for i in range(0,5):
    v=input()
    list1.append(v)

list2=[]
print("Enter 5 numbers")
for i in range(0,5):
    v=input()
    list2.append(v)



flag=0
for i in list1:
    if i in list2:
        flag=1

if(flag==1):
   print("The Lists Overlap")
else:
   print("The Lists do Not overlap")
```

output:



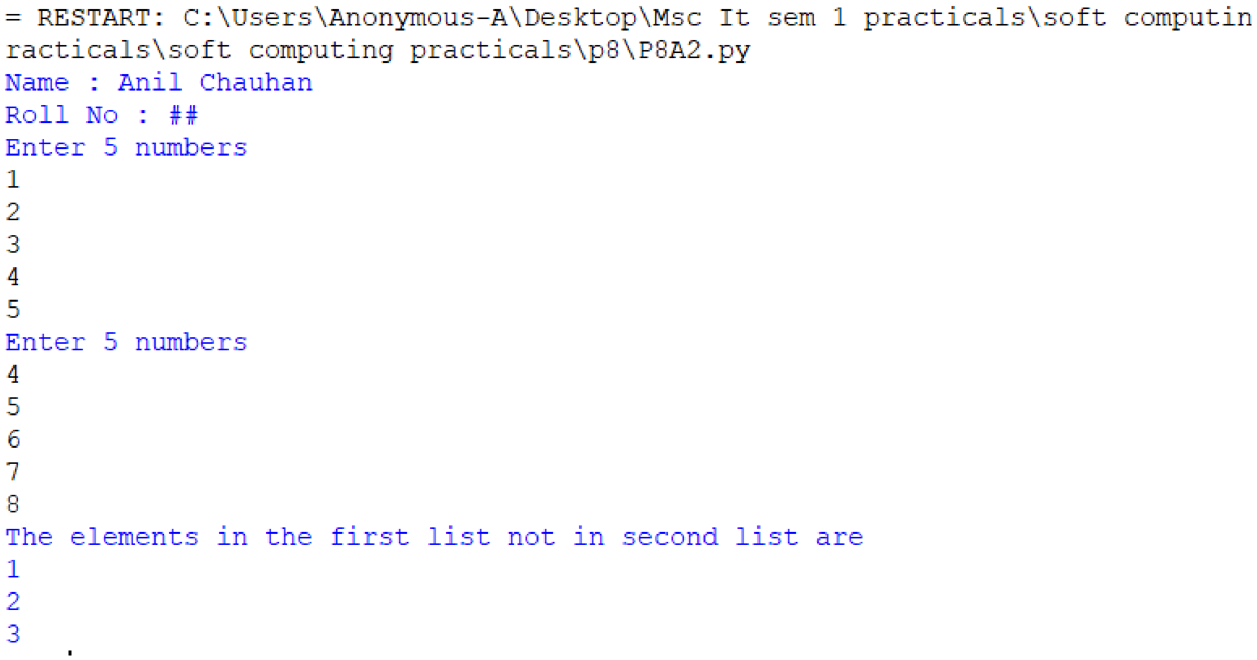
```
= RESTART: C:\Users\Anonymous-A\Desktop\Msc It sem 1 practicals\soft computing p
racticals\soft computing practicals\p8\P8A.py
Name : Anil Chauhan
Roll No : ##
Enter 5 numbers
1
2
3
4
5
Enter 5 numbers
1
2
3
5
4
The Lists Overlap
```

Practical 8-A2

Code:-

```
print("Name : Anil Chauhan")
print("Roll No : ##")
list1=[]
print("Enter 5 numbers")
for i in range(0,5):
   v=input()
   list1.append(v)

list2=[]
print("Enter 5 numbers")
for i in range(0,5):
   v=input()
   list2.append(v)

flag=0
print("The elements in the first list not in second list are")
for i in list1:
   if i not in list2:
       print(i)
```

output:



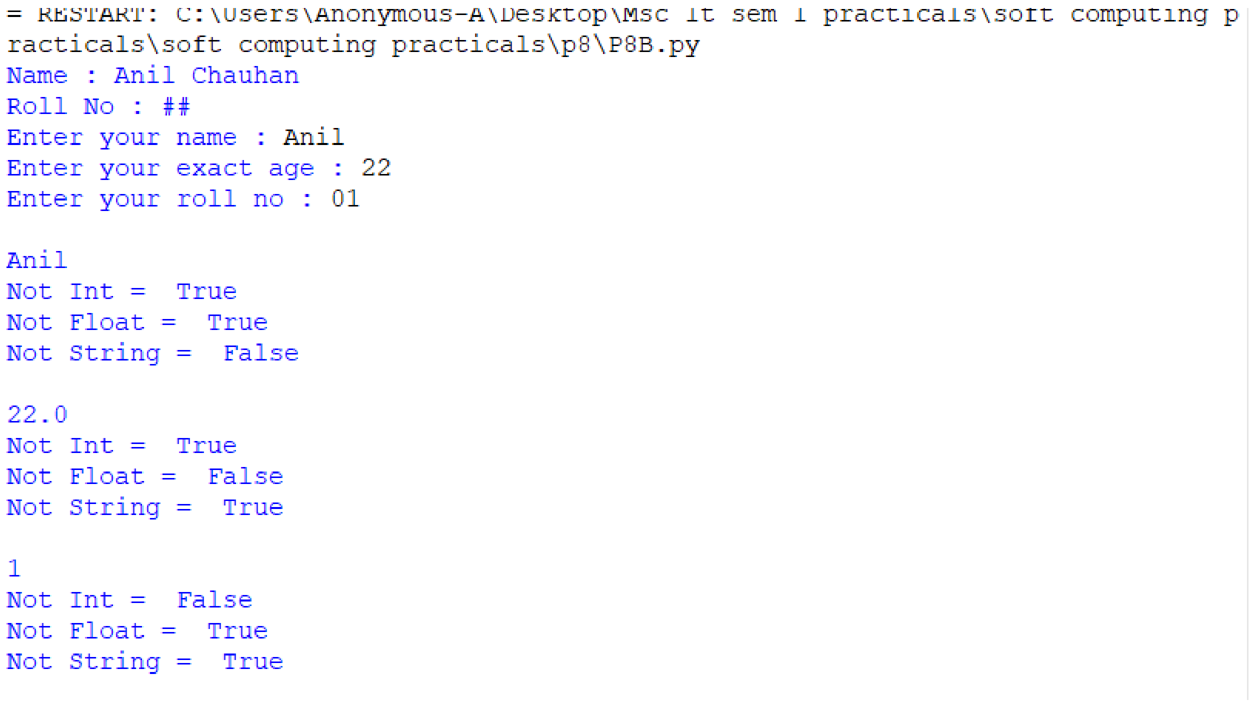
```
= RESTART: C:\Users\Anonymous-A\Desktop\Msc It sem 1 practicals\soft computin
racticals\soft computing practicals\p8\P8A2.py
Name : Anil Chauhan
Roll No : ##
Enter 5 numbers
1
2
3
4
5
Enter 5 numbers
4
5
6
7
8
The elements in the first list not in second list are
1
2
3
```

Practical 8-B

Code:-

```
print("Name : Anil Chauhan")
print("Roll No : ##")
details =[]
name=input("Enter your name : ")
details.append(name)
age=float(input("Enter your exact age : "))
details.append(age)
roll_no=int(input("Enter your roll no : "))
details.append(roll_no)
print()
for i in details:
   print(i)
   print("Not Int = ",type(i) is not int)
   print("Not Float = ",type(i) is not float)
   print("Not String = ",type(i) is not str)
   print()
```

output:



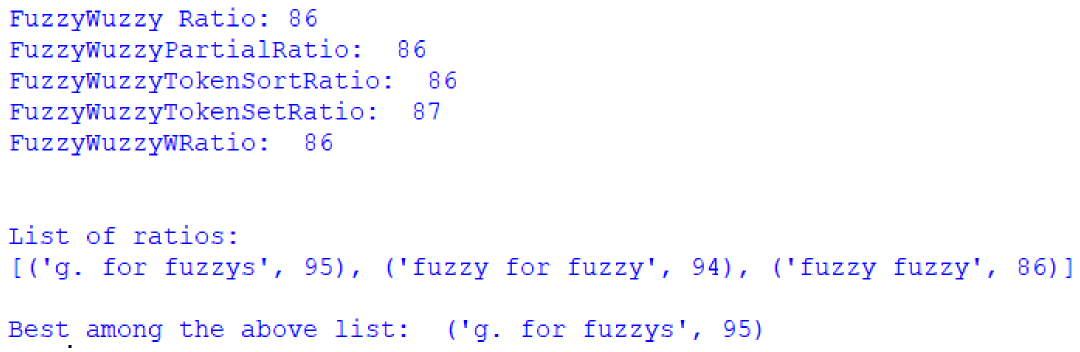
```
= RESTART: C:\Users\Anonymous-A\Desktop\Msc it sem 1 practicals\soft computing p
racticals\soft computing practicals\p8\P8B.py
Name : Anil Chauhan
Roll No : ##
Enter your name : Anil
Enter your exact age : 22
Enter your roll no : 01

Anil
Not Int =  True
Not Float =  True
Not String =  False

22.0
Not Int =  True
Not Float =  False
Not String =  True

1
Not Int =  False
Not Float =  True
Not String =  True
```

Practical 9-A

Code:-

```
from fuzzywuzzy import fuzz

from fuzzywuzzy import process


s1 = "I love fuzzysforfuzzys"

s2 = "I am loving fuzzysforfuzzys"

print ("FuzzyWuzzy Ratio:", fuzz.ratio(s1, s2))

print ("FuzzyWuzzyPartialRatio: ", fuzz.partial_ratio(s1, s2))

print ("FuzzyWuzzyTokenSortRatio: ", fuzz.token_sort_ratio(s1, s2))

print ("FuzzyWuzzyTokenSetRatio: ", fuzz.token_set_ratio(s1, s2))

print ("FuzzyWuzzyWRatio: ", fuzz.WRatio(s1, s2),'\n\n')

# for process library,

query = 'fuzzys for fuzzys'

choices = ['fuzzy for fuzzy', 'fuzzy fuzzy', 'g. for fuzzys']

print ("List of ratios: ")

print (process.extract(query, choices), '\n')

print ("Best among the above list: ",process.extractOne(query, choices))
```

output:



```
FuzzyWuzzy Ratio: 86
FuzzyWuzzyPartialRatio:  86
FuzzyWuzzyTokenSortRatio:  86
FuzzyWuzzyTokenSetRatio:  87
FuzzyWuzzyWRatio:  86


List of ratios:
[('g. for fuzzys', 95), ('fuzzy for fuzzy', 94), ('fuzzy fuzzy', 86)]

Best among the above list:  ('g. for fuzzys', 95)
```

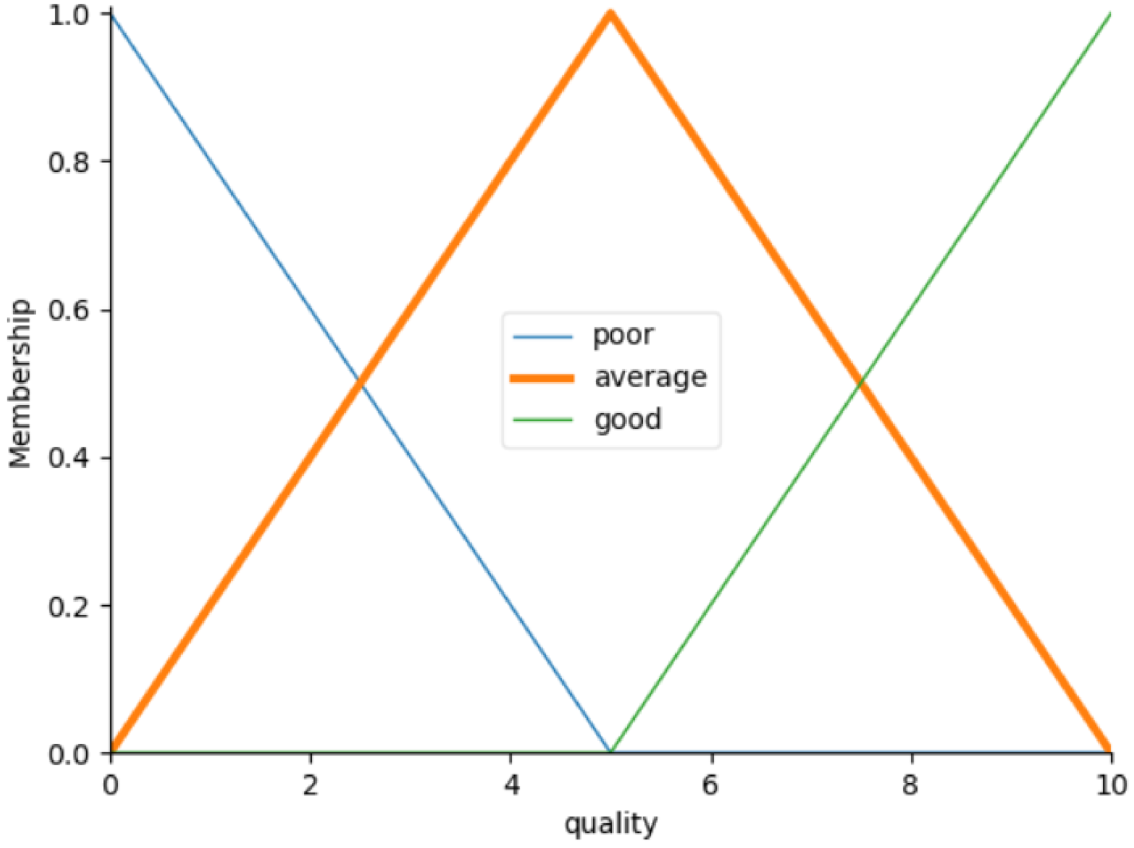
Practical 9-B

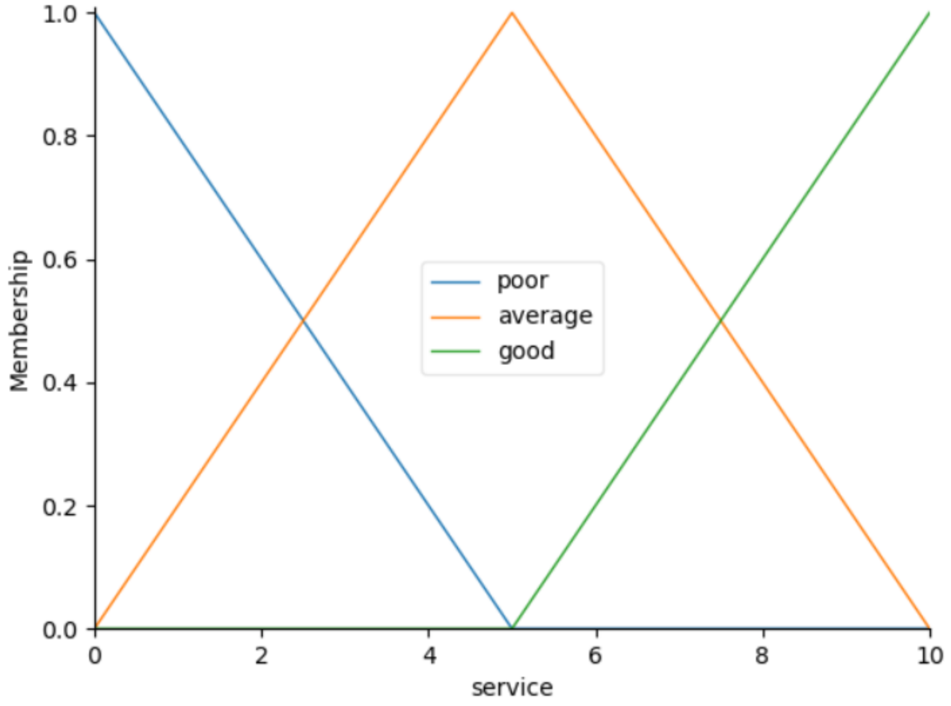
Code:-

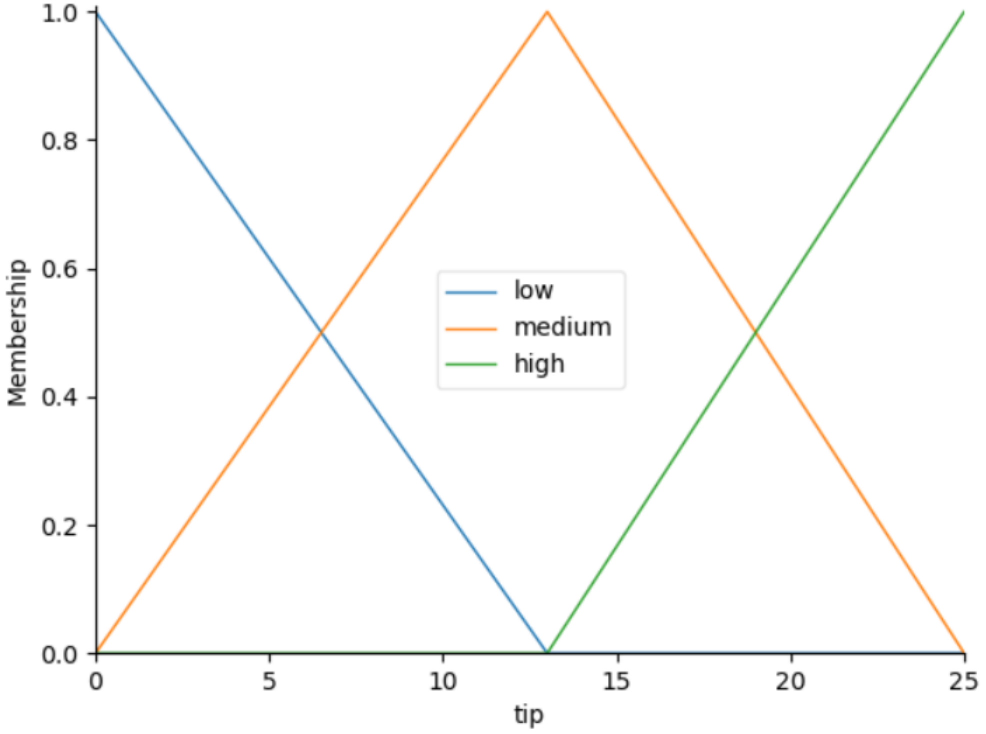
```
import numpy as np

import skfuzzy as fuzz

from skfuzzy import control as ctrl


# New Antecedent/Consequent objects hold universe variables and membership

# functions

quality = ctrl.Antecedent(np.arange(0, 11, 1), 'quality')

service = ctrl.Antecedent(np.arange(0, 11, 1), 'service')

tip = ctrl.Consequent(np.arange(0, 26, 1), 'tip')


# Auto-membership function population is possible with .automf(3, 5, or 7)

quality.automf(3)

service.automf(3)


# Custom membership functions can be built interactively with a familiar,

# Pythonic API

tip['low'] = fuzz.trimf(tip.universe, [0, 0, 13])

tip['medium'] = fuzz.trimf(tip.universe, [0, 13, 25])

tip['high'] = fuzz.trimf(tip.universe, [13, 25, 25])

quality['average'].view()

service.view()

tip.view()


rule1 = ctrl.Rule(quality['poor'] | service['poor'], tip['low'])

rule2 = ctrl.Rule(service['average'], tip['medium'])
```

```
rule3 = ctrl.Rule(service['good'] | quality['good'], tip['high'])

rule1.view()


tipping_ctrl = ctrl.ControlSystem([rule1, rule2, rule3])

tipping = ctrl.ControlSystemSimulation(tipping_ctrl)

tipping.input['quality'] = 6.5

tipping.input['service'] = 9.8


# Crunch the numbers

tipping.compute()

print (tipping.output['tip'])

tip.view(sim=tipping)
```

output:




1.0
0.8
0.6
0.4
0.2
0.0
Membership
0
2
4
6
8
10
service
poor
average
good


1.0
0.8
0.6
0.4
0.2
0.0
Membership
0
5
10
15
20
25
tip
low
medium
high

SOFT COMPUTING

Table of Contents