INDEX

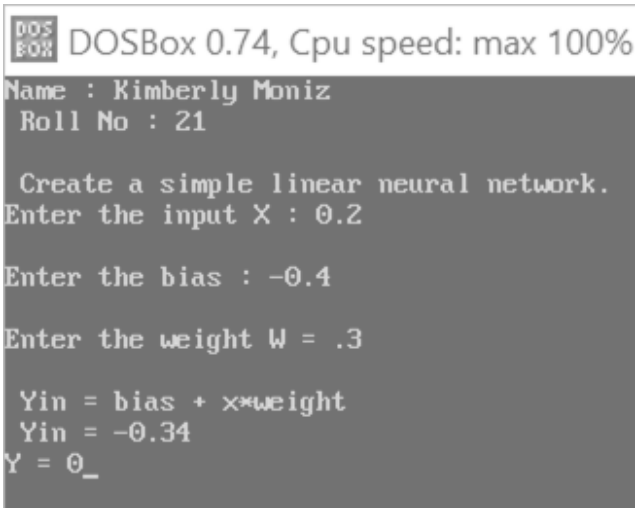
Practical 1

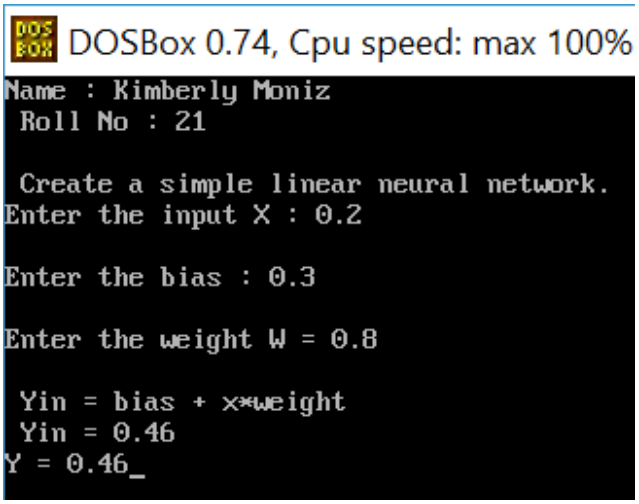
A. Design a simple linear neural network.

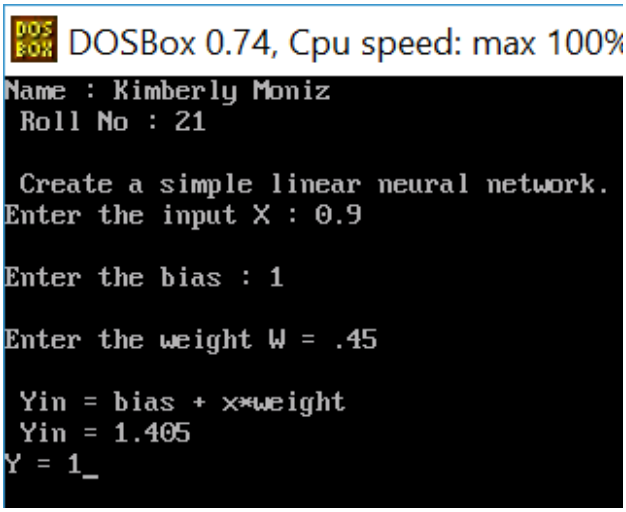
Code :

```
#include<iostream.h>
#include <conio.h>
void main()
{  clrscr();
 cout<<"Name : Kimberly Moniz \n Roll No : 21 \n";
 cout<<"\n Create a simple linear neural network.";

float x,bias,weight,Yin,out;

 cout<<"\nEnter the input X : "; cin>>x;
 cout<<"\nEnter the bias : "; cin>>bias;
 cout<<"\nEnter the weight W = "; cin>>weight;

 Yin = (bias + (x*weight ));

 cout<<"\n Yin = bias + x*weight "<<"\n Yin = "<<Yin;

 if(Yin<0) out=0;
 else if((Yin>=0) && (Yin<1))  out=Yin;
 else out=1;
 cout<<"\nY = "<<out;
 getch();
}
```

Output:


DOSBox 0.74, Cpu speed: max 100%
Name : Kimberly Moniz
Roll No : 21
Create a simple linear neural network.
Enter the input X : 0.2
Enter the bias : -0.4
Enter the weight W = .3
Yin = bias + x*weight
Yin = -0.34
Y = 0_


DOSBox 0.74, Cpu speed: max 100%
Name : Kimberly Moniz
Roll No : 21
Create a simple linear neural network.
Enter the input X : 0.2
Enter the bias : 0.3
Enter the weight W = 0.8
Yin = bias + x*weight
Yin = 0.46
Y = 0.46_

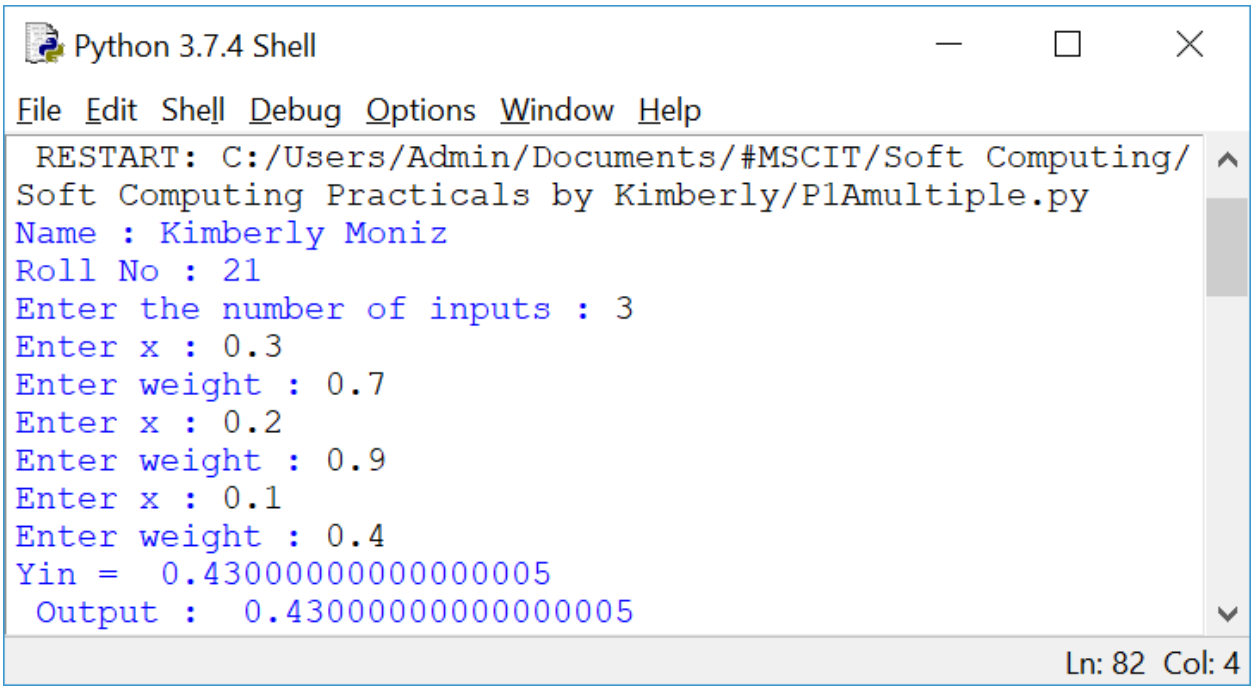

DOSBox 0.74, Cpu speed: max 100%
Name : Kimberly Moniz
Roll No : 21
Create a simple linear neural network.
Enter the input X : 0.9
Enter the bias : 1
Enter the weight W = .45
Yin = bias + x*weight
Yin = 1.405
Y = 1_

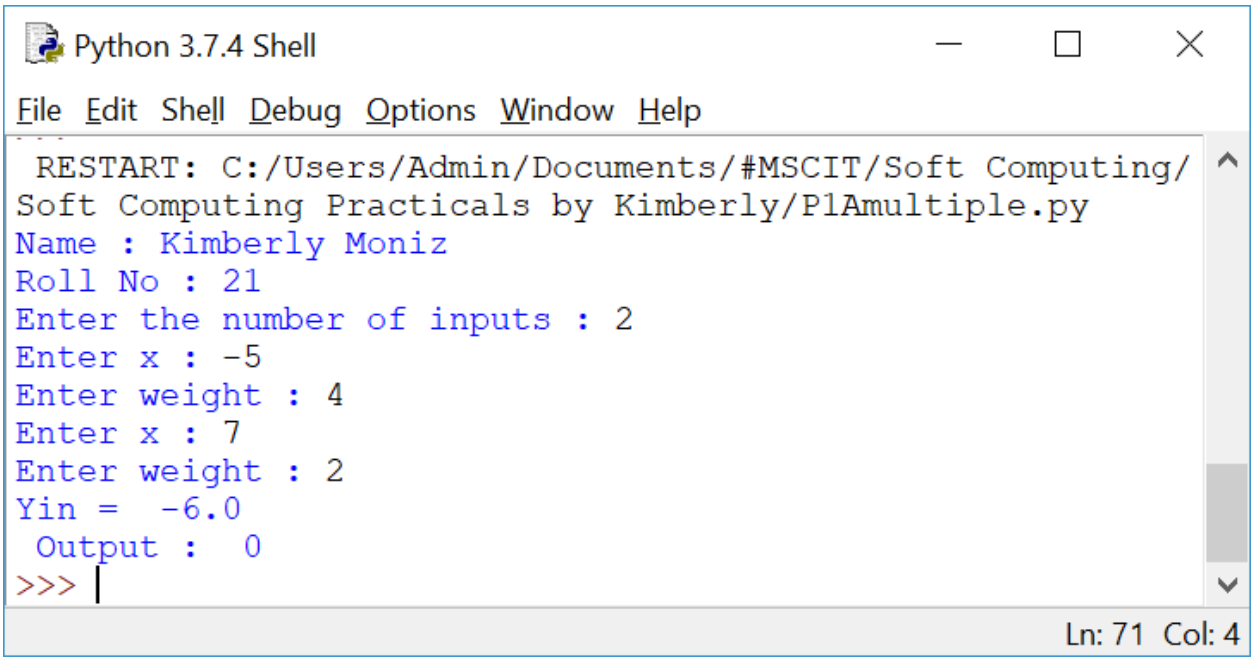
A neural network for multiple inputs.

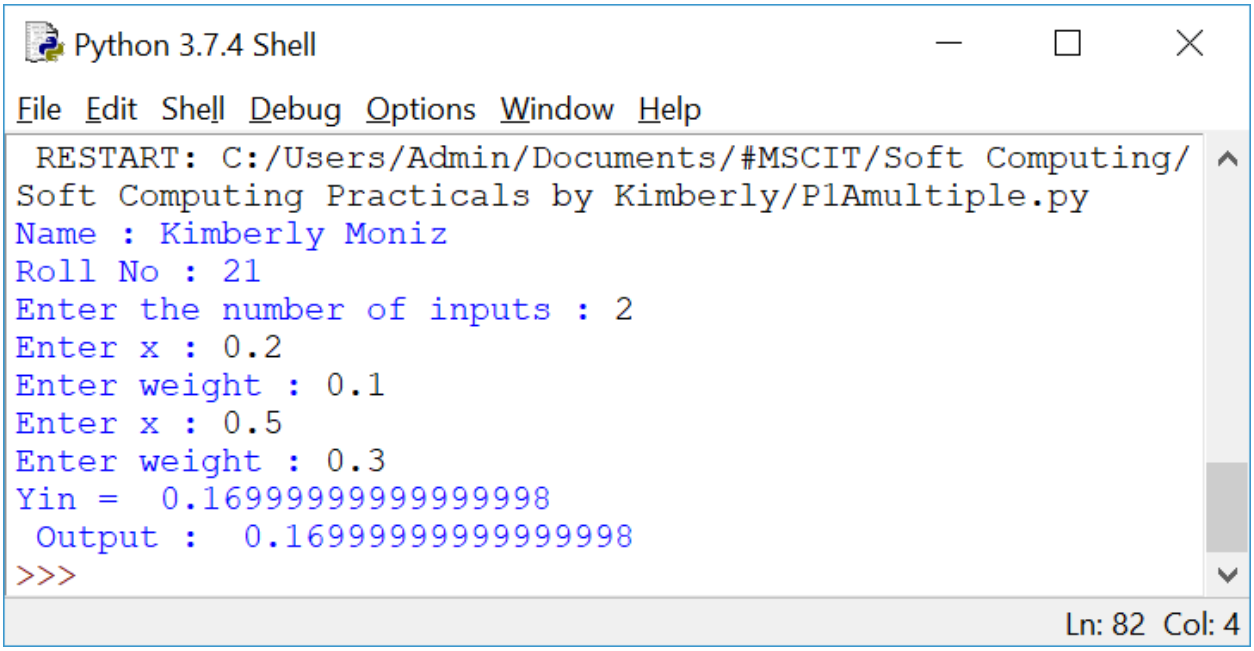
Code :

```
print("Name : Kimberly Moniz")
print("Roll No : 21")
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 3.7.4 Shell
File Edit Shell Debug Options Window Help
RESTART: C:/Users/Admin/Documents/#MSCIT/Soft Computing/Soft Computing Practicals by Kimberly/P1Amultiple.py
Name : Kimberly Moniz
Roll No : 21
Enter the number of inputs : 2
Enter x : -5
Enter weight : 4
Enter x : 7
Enter weight : 2
Yin = -6.0
Output : 0
>>>
Ln: 71 Col: 4

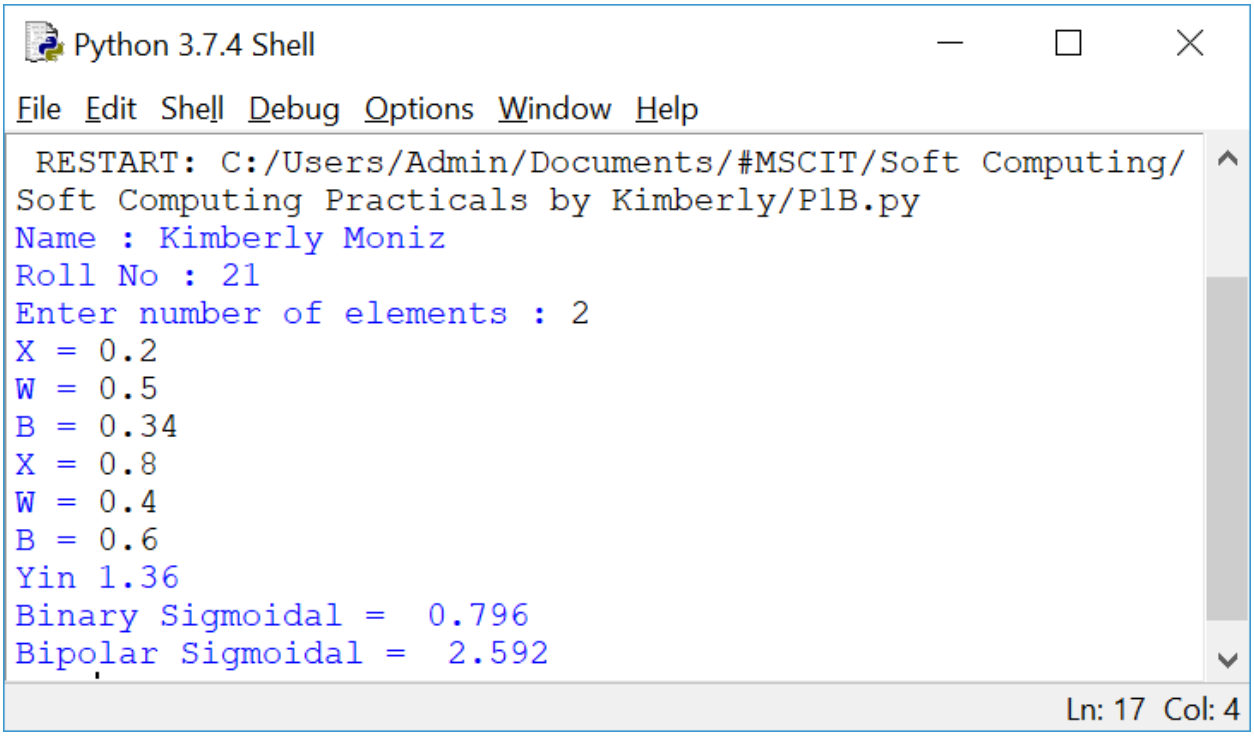

Python 3.7.4 Shell
File Edit Shell Debug Options Window Help
RESTART: C:/Users/Admin/Documents/#MSCIT/Soft Computing/Soft Computing Practicals by Kimberly/P1Amultiple.py
Name : Kimberly Moniz
Roll No : 21
Enter the number of inputs : 2
Enter x : 0.2
Enter weight : 0.1
Enter x : 0.5
Enter weight : 0.3
Yin = 0.16999999999999998
Output : 0.16999999999999998
>>>
Ln: 82 Col: 4

B. Calculate the output of neural net using both binary and bipolar sigmoidal function.

Code :

```
import math
print("Name : Kimberly Moniz ")
print("Roll No : 21")
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 :



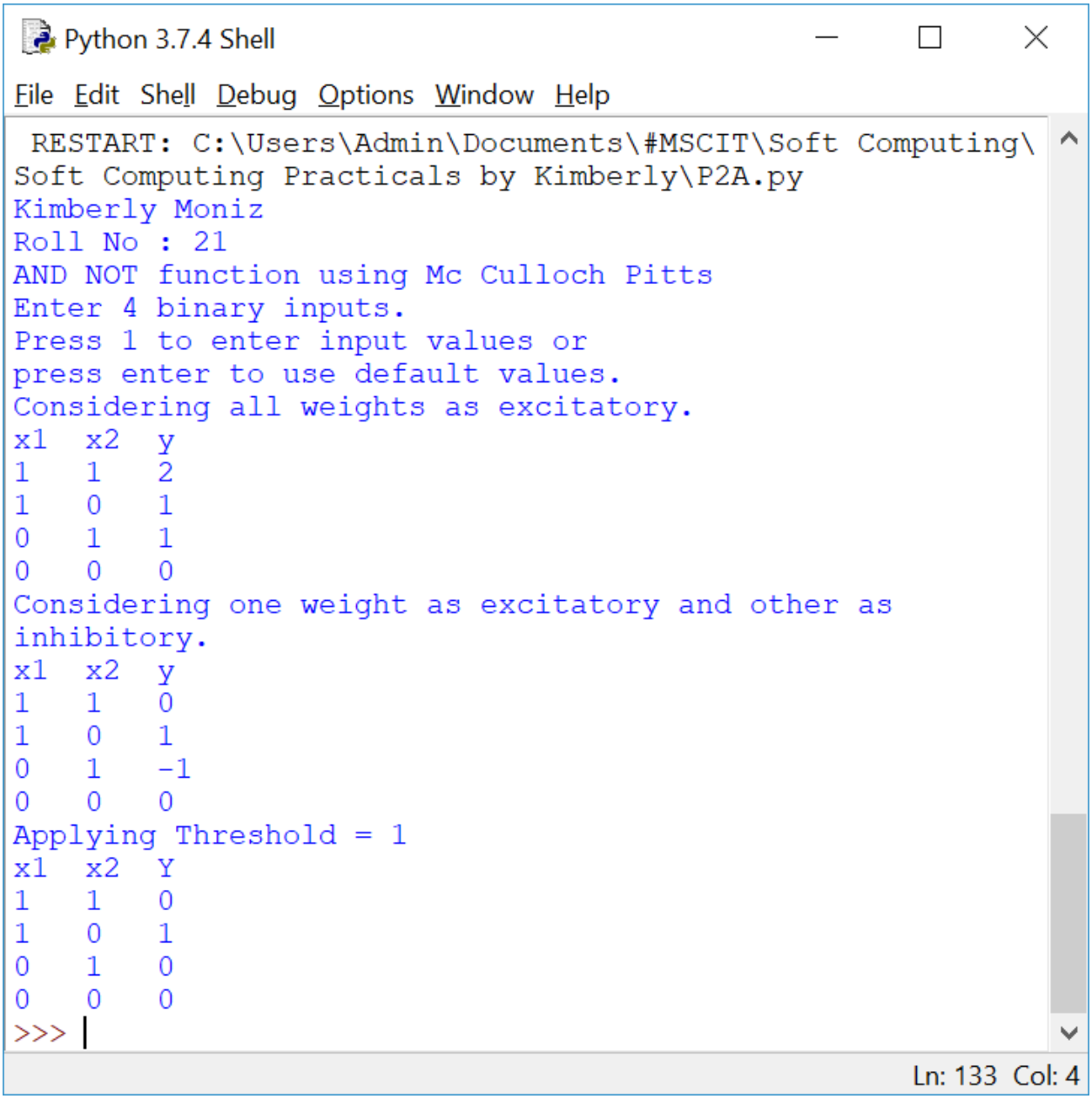
Practical 2

A] Generate AND/NOT function using McCulloch-Pitts neural net.

Code :

```
print("Kimberly Moniz")
print("Roll No : 21 ")
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 :


Python 3.7.4 Shell
File Edit Shell Debug Options Window Help
RESTART: C:\Users\Admin\Documents\#MSCIT\Soft Computing\Soft Computing Practicals by Kimberly\P2A.py
Kimberly Moniz
Roll No : 21
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
>>>
Ln: 133 Col: 4

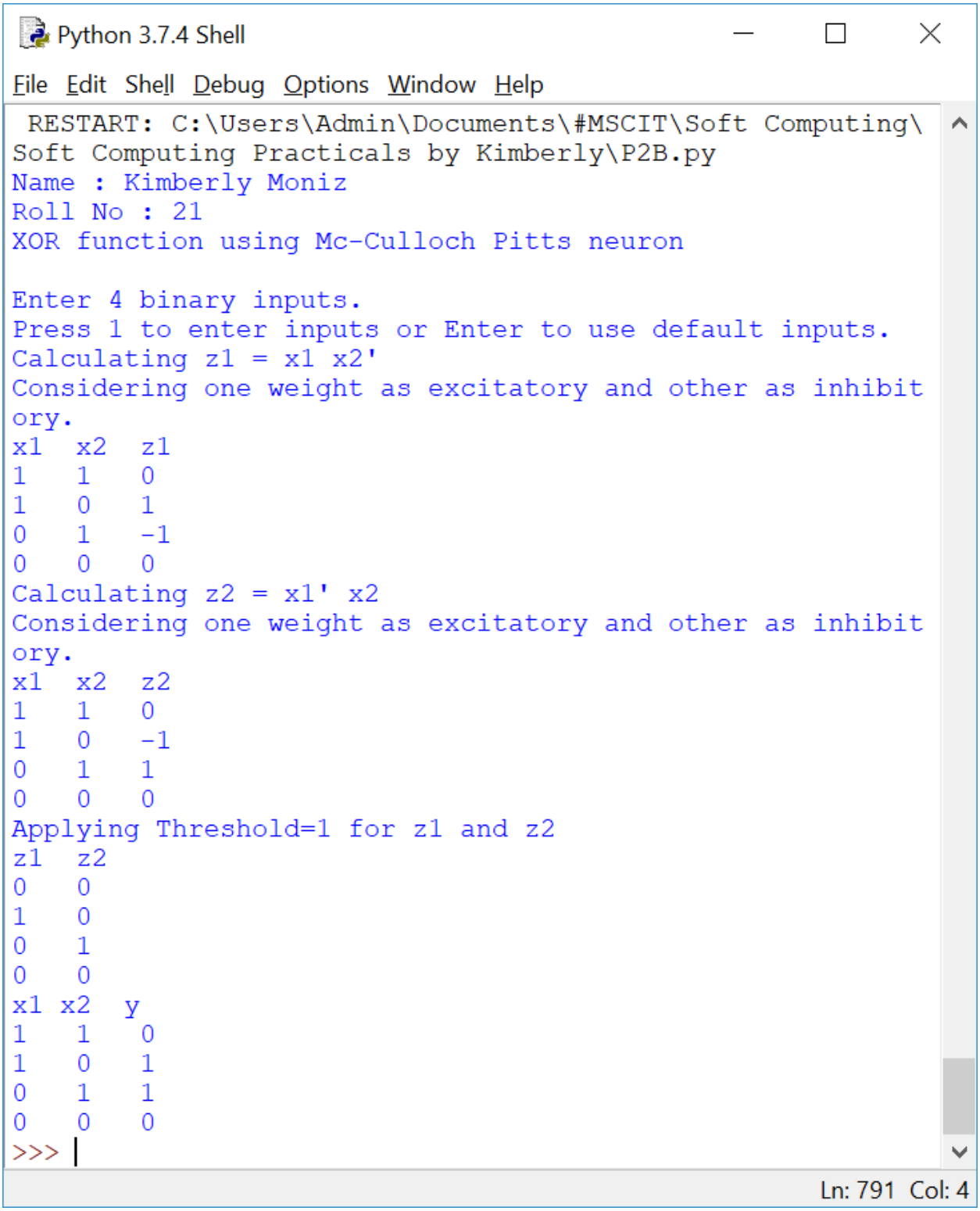
Practical 2

B] Generate XOR function using McCulloch-Pitts neural net.

Code :

```
print("Name : Kimberly Moniz")      print("Roll No : 21 ")
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
   if(z2[i]>=1):       z2[i]=1
   else:       z2[i]=0
print("z1 ","z2")
for i in range(0,4):    print(z1[i] ," ",  z2[i])
y = []    v1=1    v2=1
for i in range(0,4):    y.append( z1[i]*v1 + z2[i]*v2 )
print("x1" , "x2" , " y")
for i in range(0,4):    print(x1inputs[i] ," " ,  x2inputs[i] ," ", y[i])
```

Output :


Python 3.7.4 Shell
File Edit Shell Debug Options Window Help
RESTART: C:\Users\Admin\Documents\#MSCIT\Soft Computing\Soft Computing Practicals by Kimberly\P2B.py
Name : Kimberly Moniz
Roll No : 21
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
>>>
Ln: 791 Col: 4

Practical 3

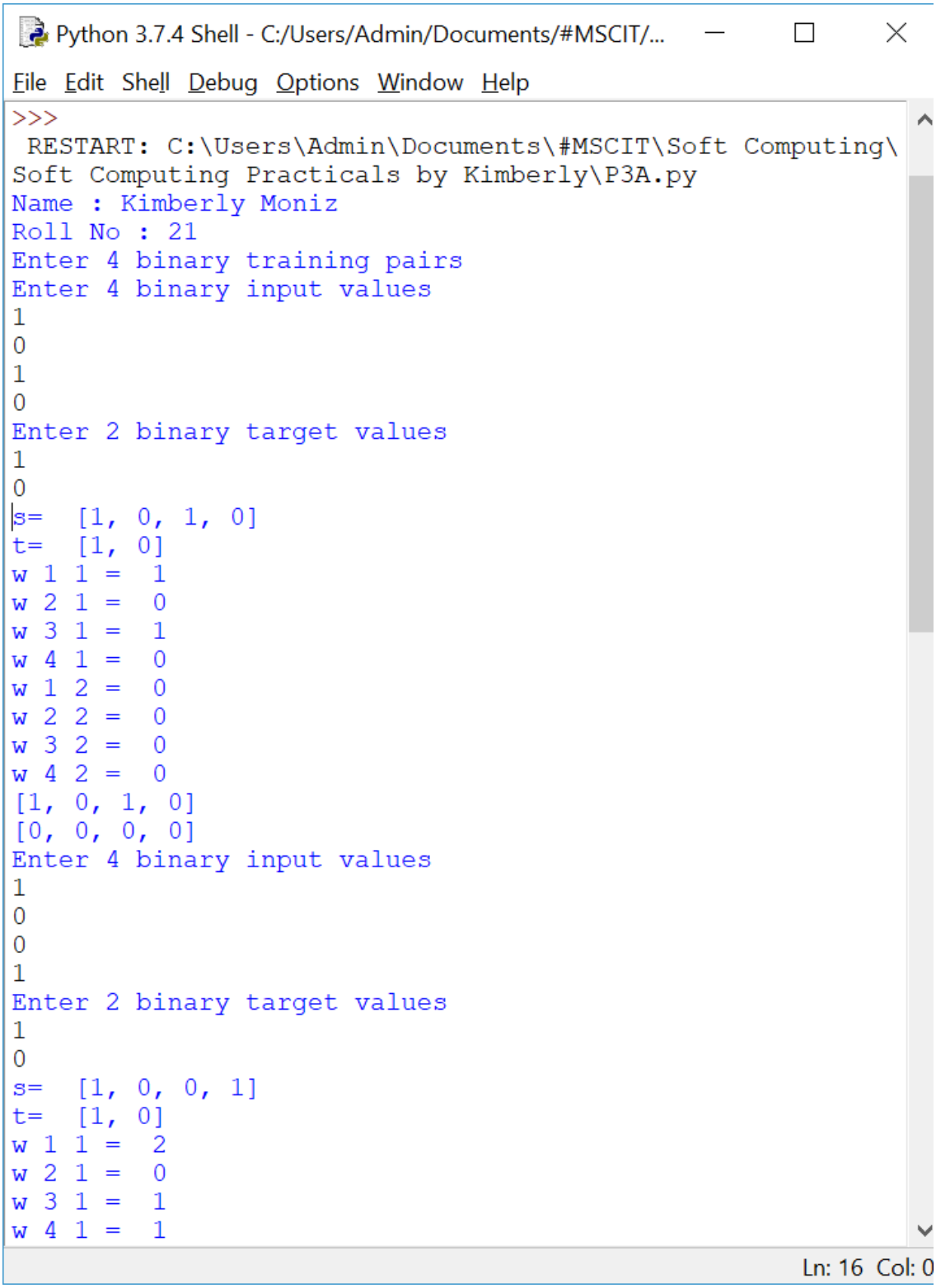
A] Implement Hebb Rule

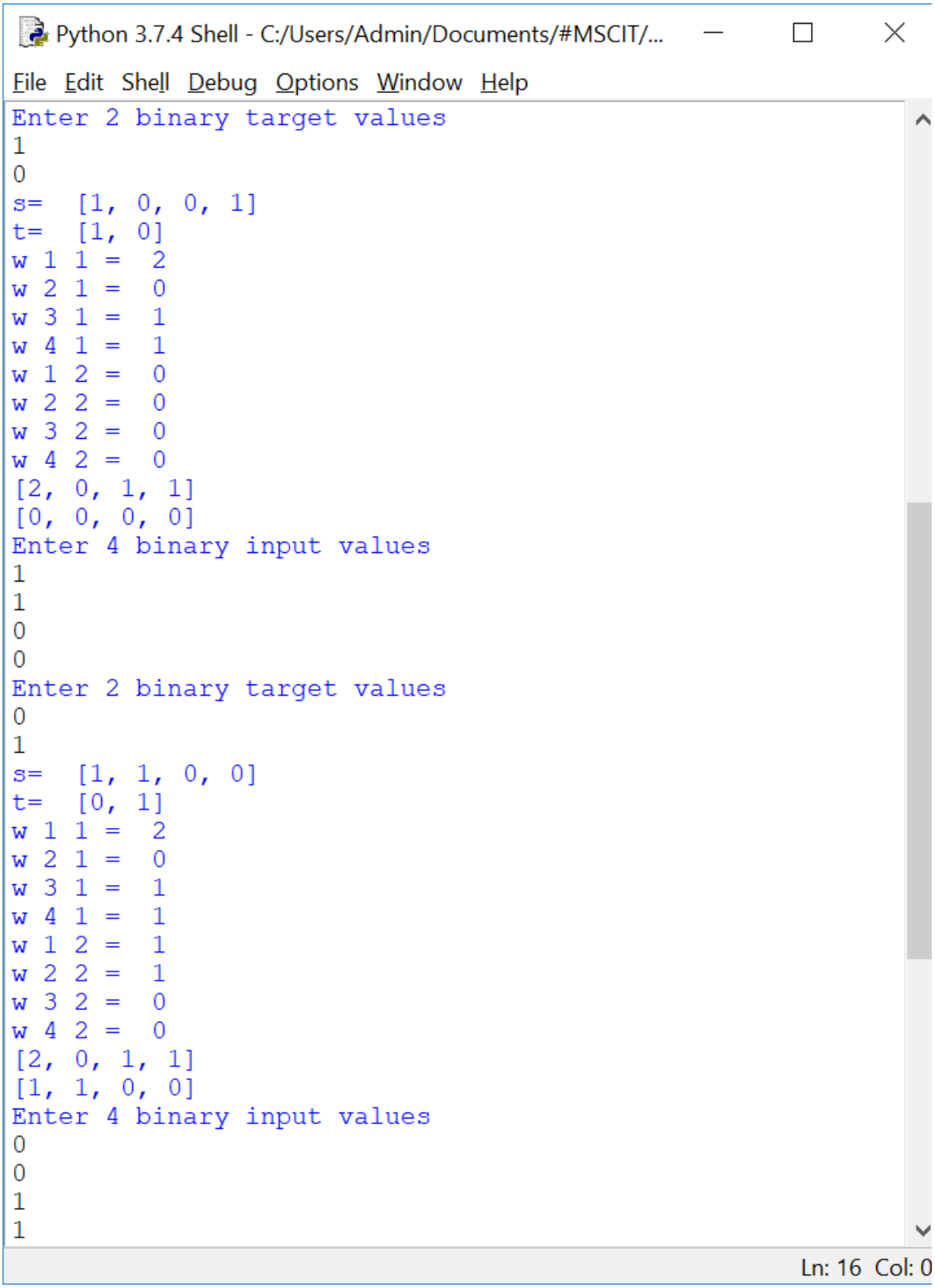
Code:

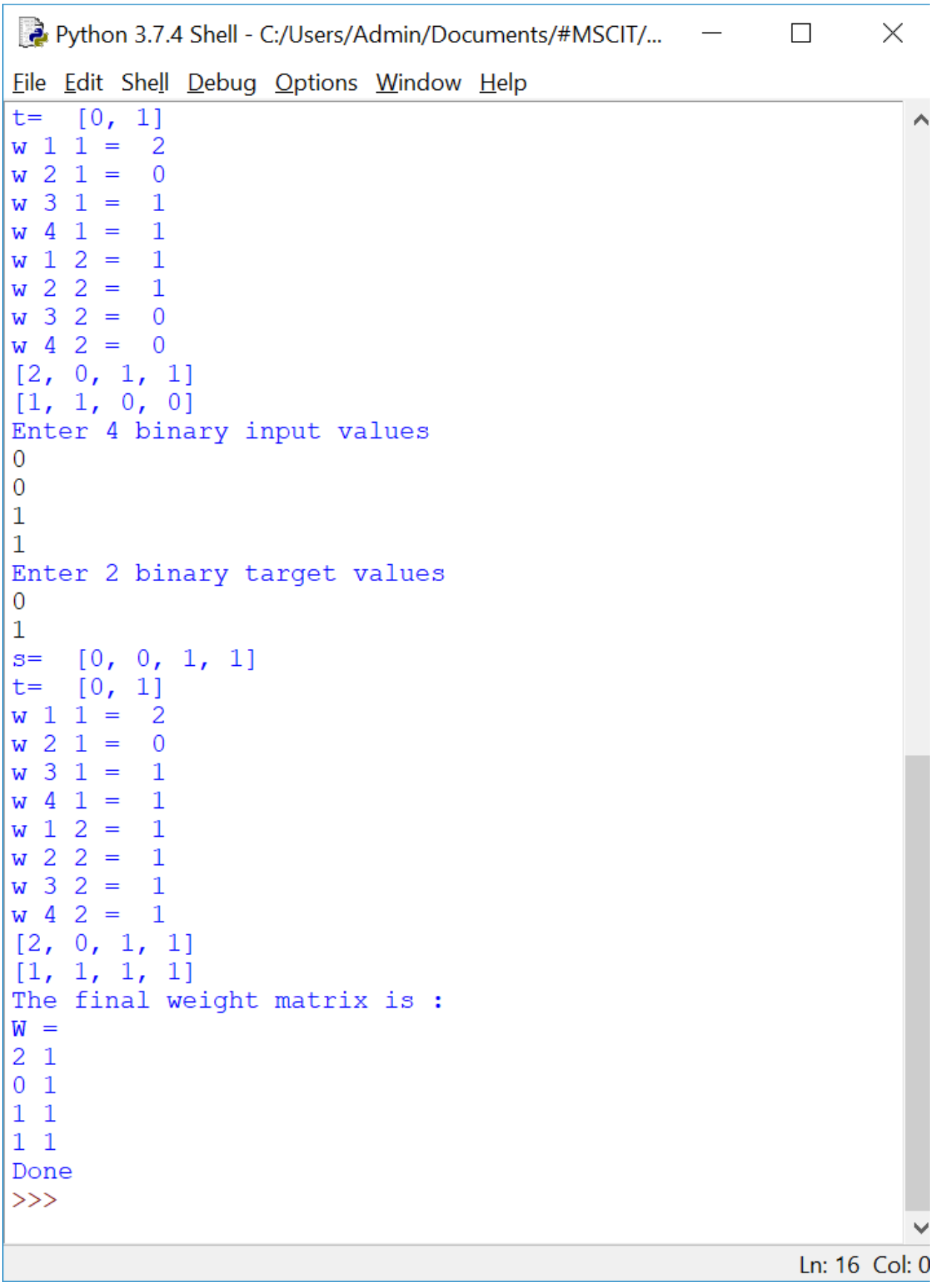
```
print("Name : Kimberly Moniz")
print("Roll No : 21")
print("Enter 4 binary training pairs")
w1=[0,0,0,0]
w2=[0,0,0,0]

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:


Python 3.7.4 Shell - C:/Users/Admin/Documents/#MSCIT/...
File Edit Shell Debug Options Window Help
>>>
RESTART: C:\Users\Admin\Documents\#MSCIT\Soft Computing\Soft Computing Practicals by Kimberly\P3A.py
Name : Kimberly Moniz
Roll No : 21
Enter 4 binary training pairs
Enter 4 binary input values
1
0
1
0
Enter 2 binary target values
1
0
s= [1, 0, 1, 0]
t= [1, 0]
w 1 1 = 1
w 2 1 = 0
w 3 1 = 1
w 4 1 = 0
w 1 2 = 0
w 2 2 = 0
w 3 2 = 0
w 4 2 = 0
[1, 0, 1, 0]
[0, 0, 0, 0]
Enter 4 binary input values
1
0
0
1
Enter 2 binary target values
1
0
s= [1, 0, 0, 1]
t= [1, 0]
w 1 1 = 2
w 2 1 = 0
w 3 1 = 1
w 4 1 = 1
Ln: 16 Col: 0


Python 3.7.4 Shell - C:/Users/Admin/Documents/#MSCIT/...
File Edit Shell Debug Options Window Help
Enter 2 binary target values
1
0
s= [1, 0, 0, 1]
t= [1, 0]
w 1 1 = 2
w 2 1 = 0
w 3 1 = 1
w 4 1 = 1
w 1 2 = 0
w 2 2 = 0
w 3 2 = 0
w 4 2 = 0
[2, 0, 1, 1]
[0, 0, 0, 0]
Enter 4 binary input values
1
1
0
0
Enter 2 binary target values
0
1
s= [1, 1, 0, 0]
t= [0, 1]
w 1 1 = 2
w 2 1 = 0
w 3 1 = 1
w 4 1 = 1
w 1 2 = 1
w 2 2 = 1
w 3 2 = 0
w 4 2 = 0
[2, 0, 1, 1]
[1, 1, 0, 0]
Enter 4 binary input values
0
0
1
1
Ln: 16 Col: 0

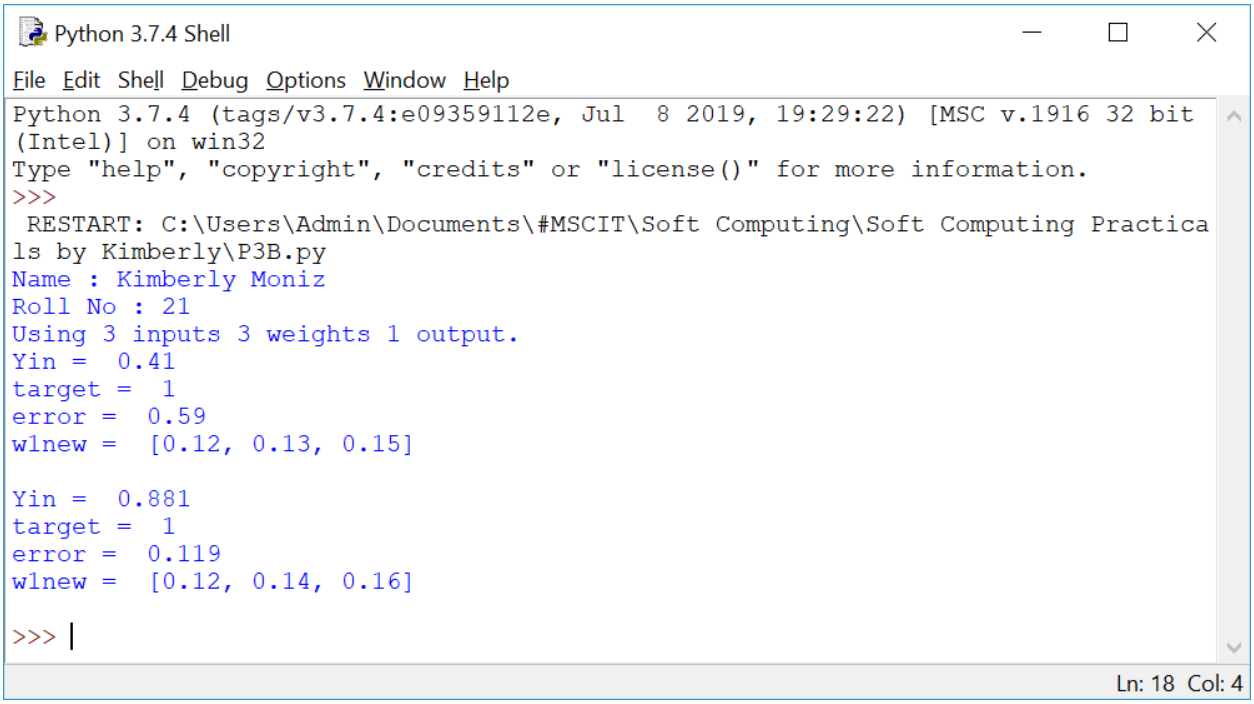

Python 3.7.4 Shell - C:/Users/Admin/Documents/#MSCIT/...
File Edit Shell Debug Options Window Help
t= [0, 1]
w 1 1 = 2
w 2 1 = 0
w 3 1 = 1
w 4 1 = 1
w 1 2 = 1
w 2 2 = 1
w 3 2 = 0
w 4 2 = 0
[2, 0, 1, 1]
[1, 1, 0, 0]
Enter 4 binary input values
0
0
1
1
Enter 2 binary target values
0
1
s= [0, 0, 1, 1]
t= [0, 1]
w 1 1 = 2
w 2 1 = 0
w 3 1 = 1
w 4 1 = 1
w 1 2 = 1
w 2 2 = 1
w 3 2 = 1
w 4 2 = 1
[2, 0, 1, 1]
[1, 1, 1, 1]
The final weight matrix is :
W =
2 1
0 1
1 1
1 1
Done
>>>
Ln: 16 Col: 0

B] Implement Delta Rule

Code :

```
print("Name : Kimberly Moniz")
print("Roll No : 21")
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 :


Python 3.7.4 Shell
File Edit Shell Debug Options Window Help
Python 3.7.4 (tags/v3.7.4:e09359112e, Jul 8 2019, 19:29:22) [MSC v.1916 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
RESTART: C:\Users\Admin\Documents\#MSCIT\Soft Computing\Soft Computing Practicals by Kimberly\P3B.py
Name : Kimberly Moniz
Roll No : 21
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]
>>>
Ln: 18 Col: 4

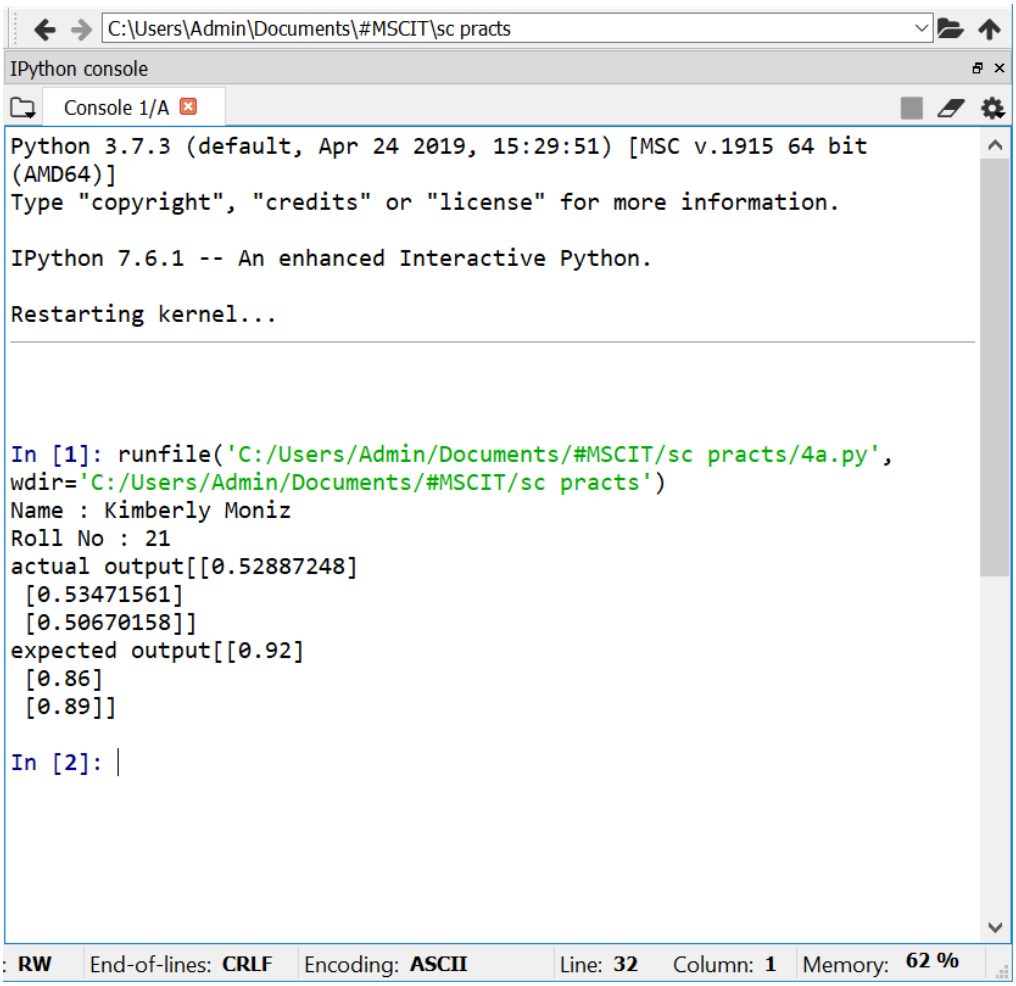
Practical 4

A] Write a program for Back Propogation Algorithm

Code :

```
print("Name : Kimberly Moniz")
print("Roll No : 21")
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 :


C:\Users\Admin\Documents\#MSCIT\sc practs
IPython console
Console 1/A
Python 3.7.3 (default, Apr 24 2019, 15:29:51) [MSC v.1915 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more information.
IPython 7.6.1 -- An enhanced Interactive Python.
Restarting kernel...
In [1]: runfile('C:/Users/Admin/Documents/#MSCIT/sc practs/4a.py', wdir='C:/Users/Admin/Documents/#MSCIT/sc practs')
Name : Kimberly Moniz
Roll No : 21
actual output[[0.52887248]
[0.53471561]
[0.50670158]]
expected output[[0.92]
[0.86]
[0.89]]
In [2]:
RW
End-of-lines: CRLF
Encoding: ASCII
Line: 32
Column: 1
Memory: 62 %

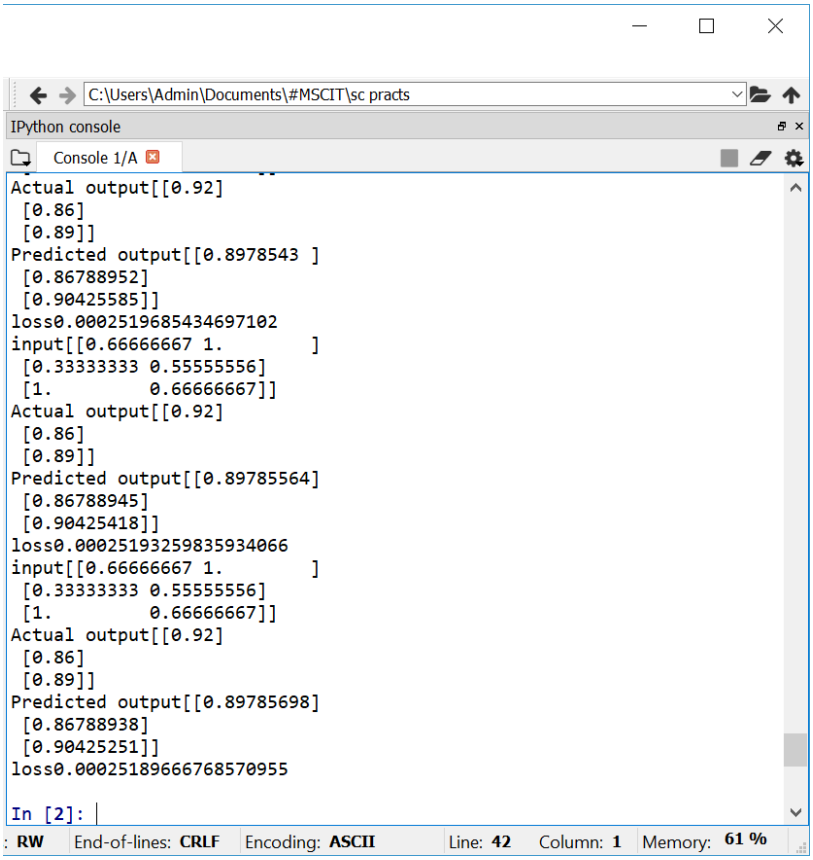
B] Write a program for error Backpropagation algorithm

Code :

```
print("Name : Kimberly Moniz")
print("Roll No : 21")
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 :



Practical 5

A] Hopfield Network

Code : HFN.cpp

```
#include "hop.h"
#include<conio.h>
neuron::neuron(int *j)
{
int i;
for(i=0;i<4;i++)
    {     weightv[i]=*(j+i);     }
}
int neuron::act(int m, int *x)
{ int i; int a=0;
for(i=0;i<m;i++)    {     a += x[i]*weightv[i];     }
return a;
}
int network::threshld(int k)
{
if(k>=0)
    return (1);
else
    return (0);
}
network::network(int a[4],int b[4],int c[4],int d[4])
{
nrn[0] = neuron(a) ;
nrn[1] = neuron(b) ;
nrn[2] = neuron(c) ;
nrn[3] = neuron(d) ;
}



void network::activation(int *patrn)
{
int i,j;
for(i=0;i<4;i++)
    {
    for(j=0;j<4;j++)
```

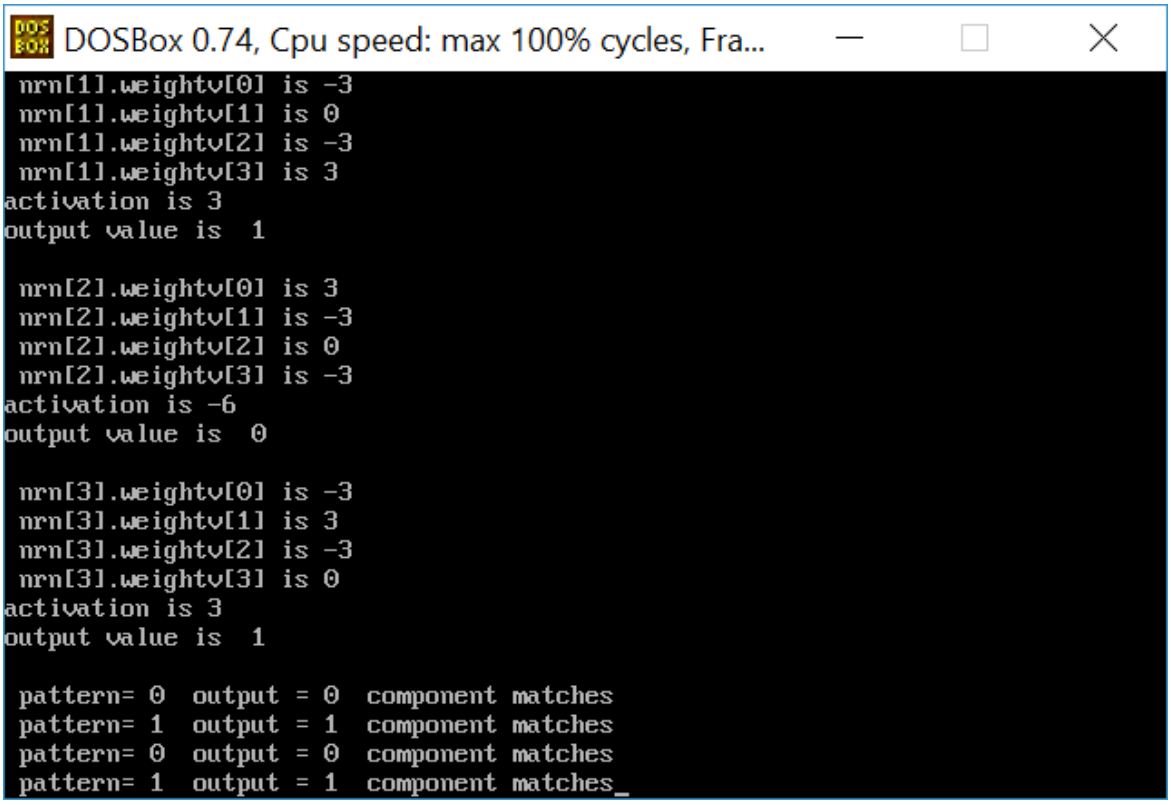
```
        {
        cout<<"\n nrn["<<i<<"].weightv["<<j<<"] is "
            <<nrn[i].weightv[j];
        }
    nrn[i].activation = nrn[i].act(4,patrn);
    cout<<"\nactivation is "<<nrn[i].activation;
    output[i]=threshld(nrn[i].activation);
    cout<<"\noutput value is  "<<output[i]<<"\n";
    }
}
void main ()
{
int patrn1[]= {1,0,1,0},i;
int wt1[]= {0,-3,3,-3};
int wt2[]= {-3,0,-3,3};
int wt3[]= {3,-3,0,-3};
int wt4[]= {-3,3,-3,0};
cout<<"\nTHIS PROGRAM IS FOR A HOPFIELD NETWORK WITH A SINGLE LAYER OF";
cout<<"\n4 FULLY INTERCONNECTED NEURONS. THE NETWORK SHOULD RECALL THE";
cout<<"\nPATTERNS 1010 AND 0101 CORRECTLY.\n";
//create the network by calling its constructor.
// the constructor calls neuron constructor as many times as the number of
// neurons in the network.
network h1(wt1,wt2,wt3,wt4);
//present a pattern to the network and get the activations of the neurons
h1.activation(patrn1);
//check if the pattern given is correctly recalled and give message
for(i=0;i<4;i++)
    {
    if (h1.output[i] == patrn1[i])
        cout<<"\n pattern= "<<patrn1[i]<<
        "  output = "<<h1.output[i]<<"  component matches";
    else
        cout<<"\n pattern= "<<patrn1[i]<<
        "  output = "<<h1.output[i]<<
        "  discrepancy occurred";
    }
cout<<"\n\n";
int patrn2[]= {0,1,0,1};
h1.activation(patrn2);
for(i=0;i<4;i++)
```

```
    {
    if (h1.output[i] == patrn2[i])
        cout<<"\n pattern= "<<patrn2[i]<<
        "  output = "<<h1.output[i]<<"  component matches";
    else
        cout<<"\n pattern= "<<patrn2[i]<<
        "  output = "<<h1.output[i]<<
        "  discrepancy occurred";
     }
getch();
}
```

Code : hop.h

```
#include <stdio.h>
#include <iostream.h>
#include <math.h>
class neuron
{
protected:
    int activation;
    friend class network;
public:
    int weightv[4];
    neuron() {};
    neuron(int *j) ;
    int act(int, int*);
};
class network
{
public:
    neuron   nrn[4];
    int output[4];
    int threshld(int) ;
    void activation(int j[4]);
    network(int*,int*,int*,int*);
};
```

Output :


DOSBox 0.74, Cpu speed: max 100% cycles, Fra...
nrn[1].weightv[0] is -3
nrn[1].weightv[1] is 0
nrn[1].weightv[2] is -3
nrn[1].weightv[3] is 3
activation is 3
output value is 1
nrn[2].weightv[0] is 3
nrn[2].weightv[1] is -3
nrn[2].weightv[2] is 0
nrn[2].weightv[3] is -3
activation is -6
output value is 0
nrn[3].weightv[0] is -3
nrn[3].weightv[1] is 3
nrn[3].weightv[2] is -3
nrn[3].weightv[3] is 0
activation is 3
output value is 1
pattern= 0 output = 0 component matches
pattern= 1 output = 1 component matches
pattern= 0 output = 0 component matches
pattern= 1 output = 1 component matches_

B] Radial Basis

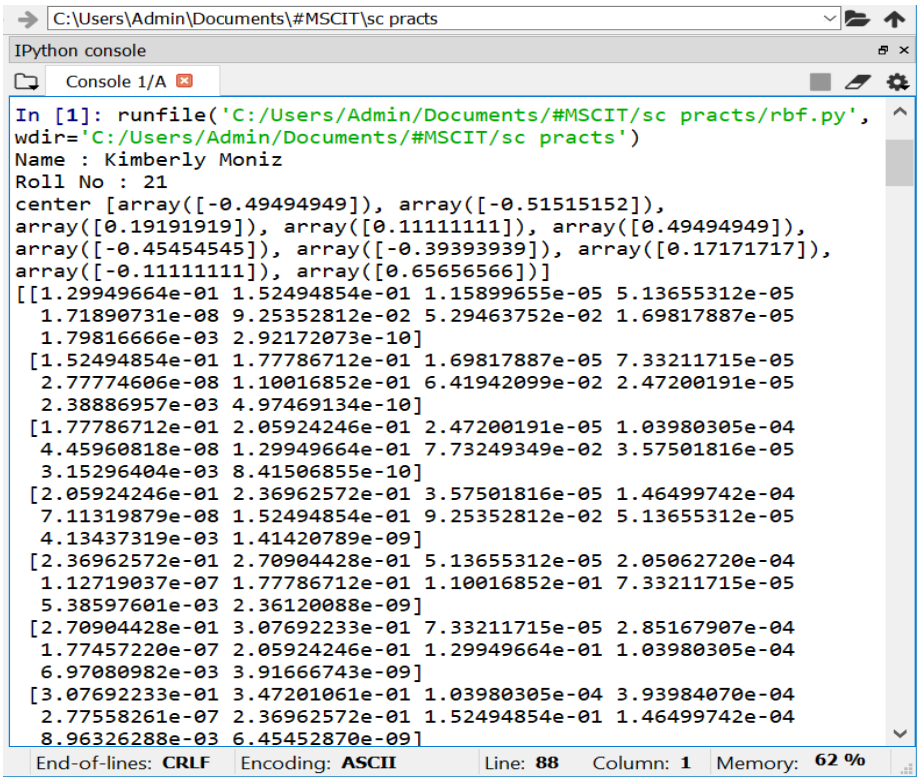
Code :

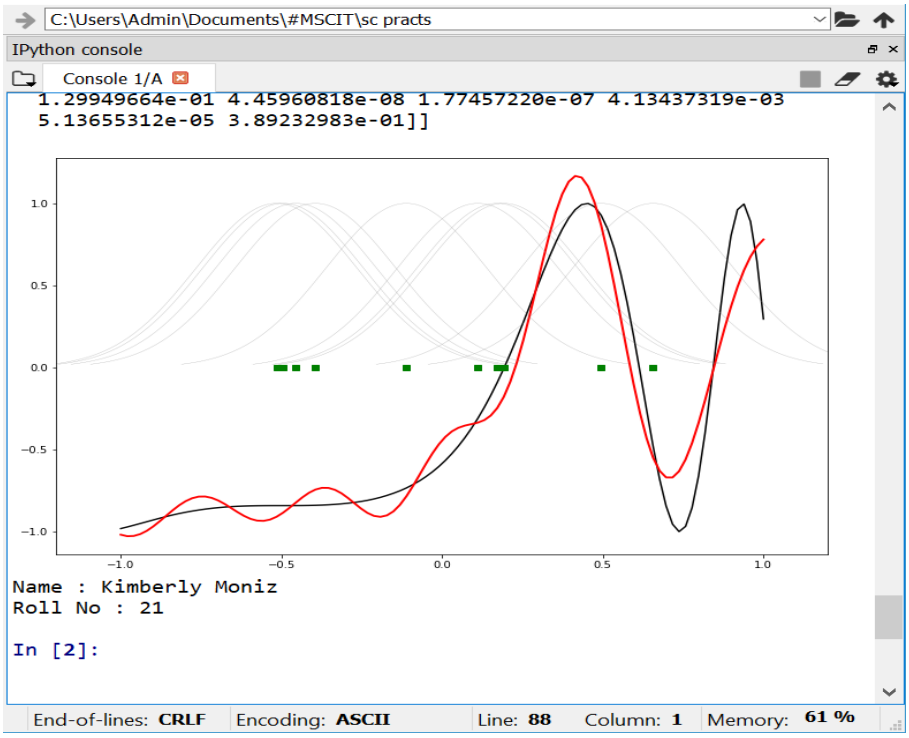
```
from scipy import *
from scipy.linalg import norm, pinv
from matplotlib import pyplot as plt
print("Name : Kimberly Moniz")
print("Roll No : 21")
class RBF:
        def __init__(self, indim, numCenters, outdim):
        self.indim = indim
        self.outdim = outdim
        self.numCenters = numCenters
        self.centers = [random.uniform(-1, 1, indim) for i in range(numCenters)]
        self.beta = 8
        self.W = random.random((self.numCenters, self.outdim))

    def _basisfunc(self, c, d):
        assert len(d) == self.indim
        return exp(-self.beta * norm(c-d)**2)

    def _calcAct(self, X):
        # calculate activations of RBFs
        G = zeros((X.shape[0], self.numCenters), float)
        for ci, c in enumerate(self.centers):
            for xi, x in enumerate(X):
                G[xi,ci] = self._basisfunc(c, x)
        return G
```

```
    def train(self, X, Y):
        """ X: matrix of dimensions n x indim
            y: column vector of dimension n x 1 """

        # choose random center vectors from training set
        rnd_idx = random.permutation(X.shape[0])[:self.numCenters]
        self.centers = [X[i,:] for i in rnd_idx]

        print ("center", self.centers)
        # calculate activations of RBFs
        G = self._calcAct(X)
        print (G)

        # calculate output weights (pseudoinverse)
        self.W = dot(pinv(G), Y)

    def test(self, X):
        """ X: matrix of dimensions n x indim """

        G = self._calcAct(X)
        Y = dot(G, self.W)
        return Y


if __name__ == '__main__':
    # ----- 1D Example ------------------------------------------------
    n = 100

    x = mgrid[-1:1:complex(0,n)].reshape(n, 1)
    # set y and add random noise
```

```
y = sin(3*(x+0.5)**3 - 1)
# y += random.normal(0, 0.1, y.shape)

# rbf regression
rbf = RBF(1, 10, 1)
rbf.train(x, y)
z = rbf.test(x)

# plot original data
plt.figure(figsize=(12, 8))
plt.plot(x, y, 'k-')

# plot learned model
plt.plot(x, z, 'r-', linewidth=2)

# plot rbfs
plt.plot(rbf.centers, zeros(rbf.numCenters), 'gs')

for c in rbf.centers:
    # RF prediction lines
    cx = arange(c-0.7, c+0.7, 0.01)
    cy = [rbf._basisfunc(array([cx_]), array([c])) for cx_ in cx]
    plt.plot(cx, cy, '-', color='gray', linewidth=0.2)

plt.xlim(-1.2, 1.2)
plt.show()
print("Name : Kimberly Moniz")
print("Roll No : 21")
```

Output :


C:\Users\Admin\Documents\#MSCIT\sc practs
IPython console
Console 1/A
In [1]: runfile('C:/Users/Admin/Documents/#MSCIT/sc practs/rbf.py',
wdir='C:/Users/Admin/Documents/#MSCIT/sc practs')
Name : Kimberly Moniz
Roll No : 21
center [array([-0.49494949]), array([-0.51515152]),
array([0.19191919]), array([0.11111111]), array([0.49494949]),
array([-0.45454545]), array([-0.39393939]), array([0.17171717]),
array([-0.11111111]), array([0.65656566])]
[[1.29949664e-01 1.52494854e-01 1.15899655e-05 5.13655312e-05
1.71890731e-08 9.25352812e-02 5.29463752e-02 1.69817887e-05
1.79816666e-03 2.92172073e-10]
[1.52494854e-01 1.77786712e-01 1.69817887e-05 7.33211715e-05
2.77774606e-08 1.10016852e-01 6.41942099e-02 2.47200191e-05
2.38886957e-03 4.97469134e-10]
[1.77786712e-01 2.05924246e-01 2.47200191e-05 1.03980305e-04
4.45960818e-08 1.29949664e-01 7.73249349e-02 3.57501816e-05
3.15296404e-03 8.41506855e-10]
[2.05924246e-01 2.36962572e-01 3.57501816e-05 1.46499742e-04
7.11319879e-08 1.52494854e-01 9.25352812e-02 5.13655312e-05
4.13437319e-03 1.41420789e-09]
[2.36962572e-01 2.70904428e-01 5.13655312e-05 2.05062720e-04
1.12719037e-07 1.77786712e-01 1.10016852e-01 7.33211715e-05
5.38597601e-03 2.36120088e-09]
[2.70904428e-01 3.07692233e-01 7.33211715e-05 2.85167907e-04
1.77457220e-07 2.05924246e-01 1.29949664e-01 1.03980305e-04
6.97080982e-03 3.91666743e-09]
[3.07692233e-01 3.47201061e-01 1.03980305e-04 3.93984070e-04
2.77558261e-07 2.36962572e-01 1.52494854e-01 1.46499742e-04
8.96326288e-03 6.45452870e-09]
End-of-lines: CRLF
Encoding: ASCII
Line: 88
Column: 1
Memory: 62 %


C:\Users\Admin\Documents\#MSCIT\sc practs
IPython console
Console 1/A
1.29949664e-01 4.45960818e-08 1.77457220e-07 4.13437319e-03
5.13655312e-05 3.89232983e-01]]
1.0
0.5
0.0
−0.5
−1.0
−1.0
−0.5
0.0
0.5
1.0
Name : Kimberly Moniz
Roll No : 21
In [2]:
End-of-lines: CRLF
Encoding: ASCII
Line: 88
Column: 1
Memory: 61 %

Practical 6

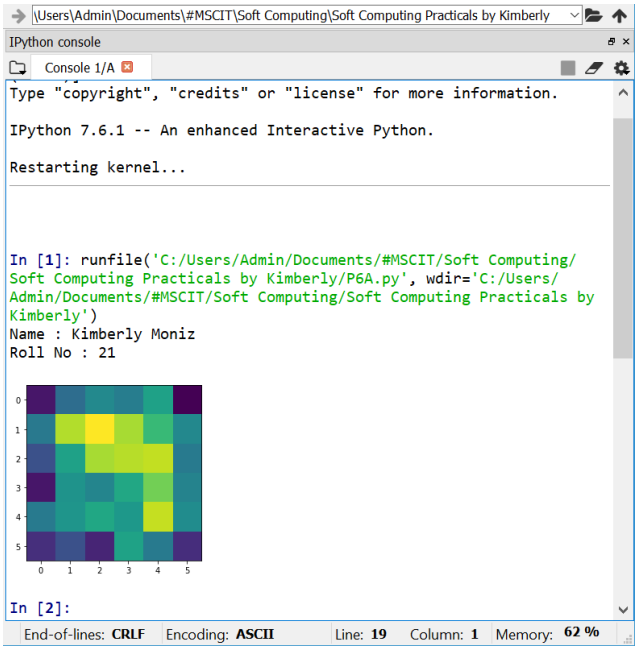
A] Kohonen

Pip install minisom

Code :

```
from minisom import MiniSom
import matplotlib.pyplot as plt
print("Name : Kimberly Moniz")
print("Roll No : 21")
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 :



B] Adaptive Resonance Theory

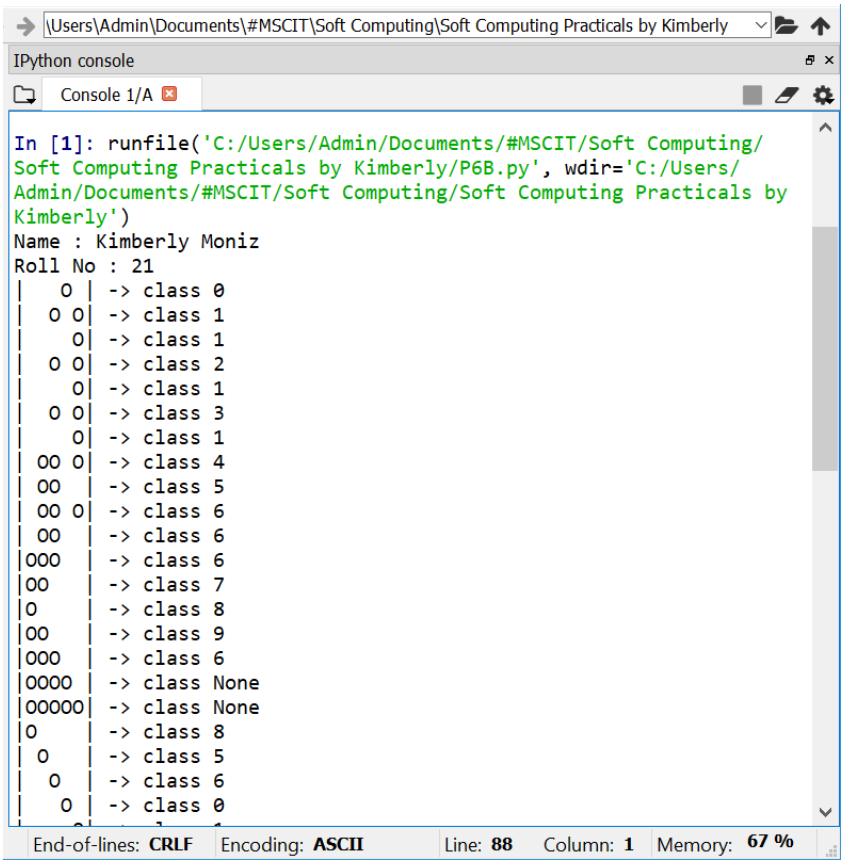
Code :

```
from __future__ import print_function
from __future__ import division
import numpy as np
print("Name : Kimberly Moniz")
print("Roll No : 21")

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 :



Practical 7

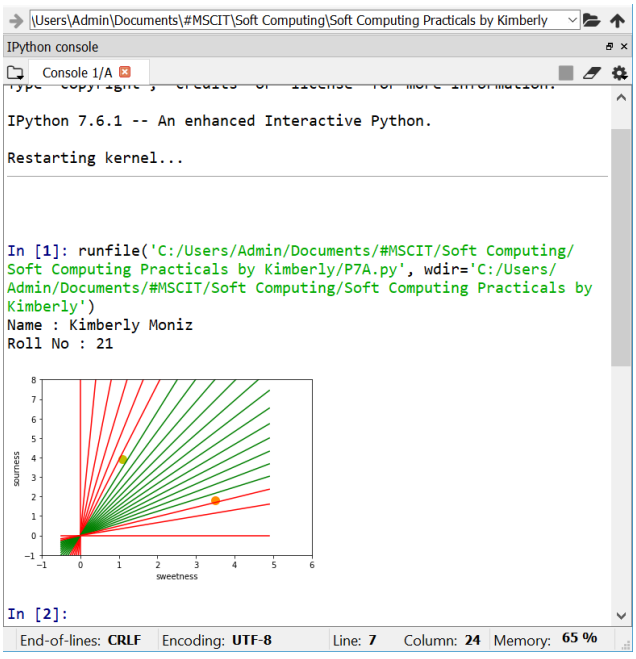
A] Write a program for Linear Separation

Code :

```
import numpy as np
import matplotlib.pyplot as plt
print("Name : Kimberly Moniz")
print("Roll No : 21")
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 :



B] Hopfield for Associative Memory

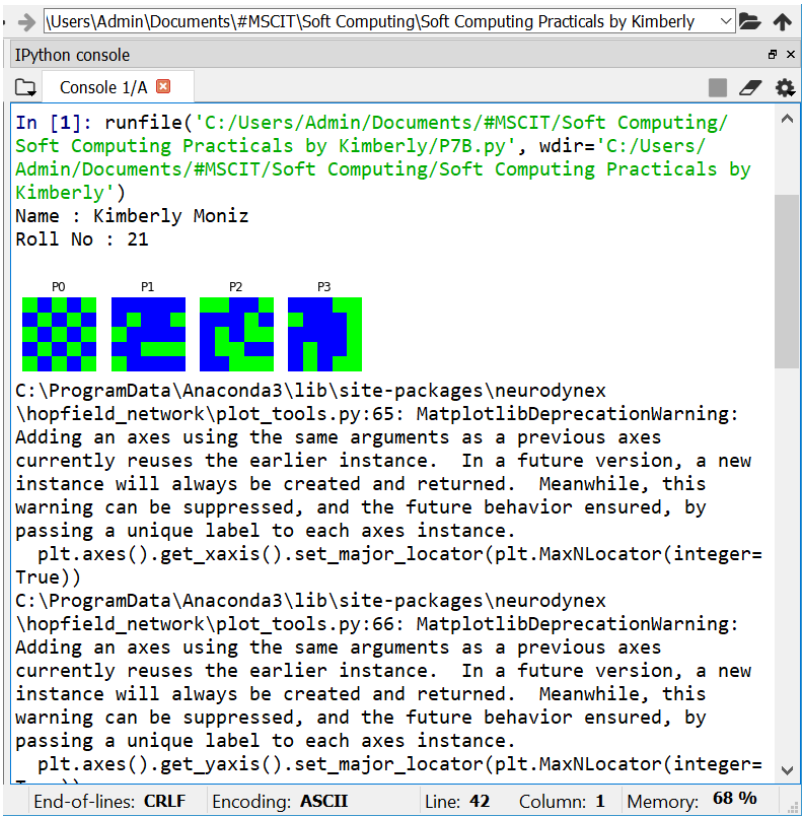
Pip install –upgrade neurodynex
Replace
Import matplotlib.pyplot as plt

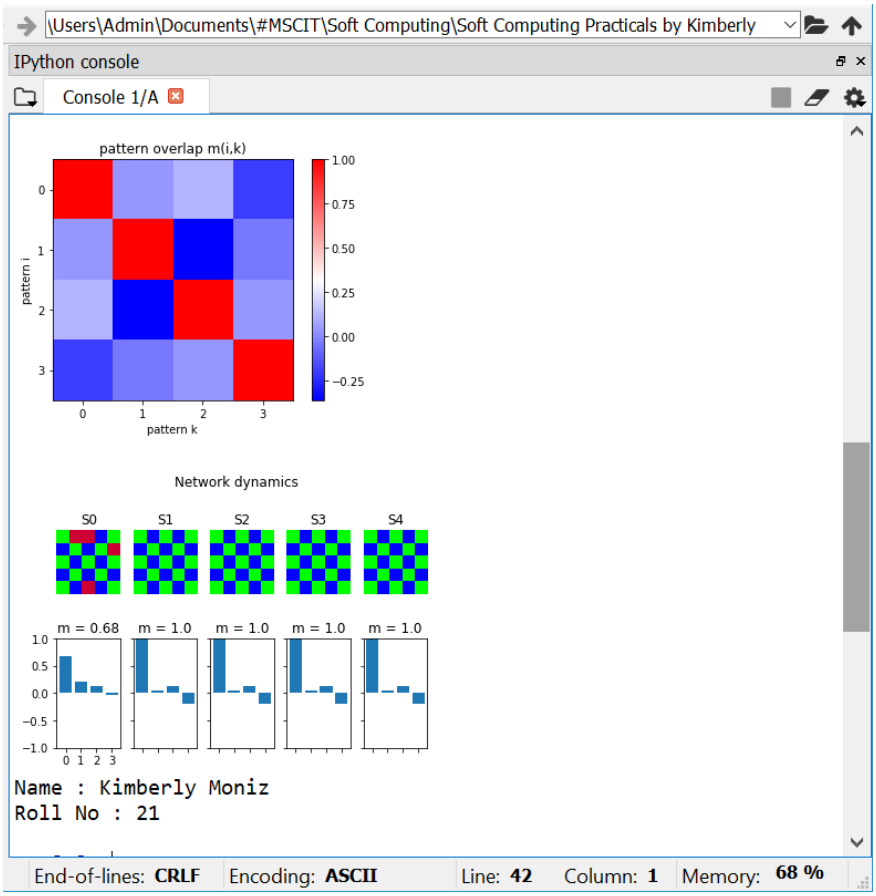
Code :

```
import matplotlib.pyplot as plt
from neurodynex.hopfield_network import network, pattern_tools, plot_tools

print("Name : Kimberly Moniz")
print("Roll No : 21")
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")

print("Name : Kimberly Moniz")
print("Roll No : 21")
```

Output :




\Users\Admin\Documents\#MSCIT\Soft Computing\Soft Computing Practicals by Kimberly
IPython console
Console 1/A
pattern overlap m(i,k)
pattern i
pattern k
Network dynamics
S0
S1
S2
S3
S4
m = 0.68
m = 1.0
m = 1.0
m = 1.0
m = 1.0
Name : Kimberly Moniz
Roll No : 21
End-of-lines: CRLF
Encoding: ASCII
Line: 42
Column: 1
Memory: 68 %

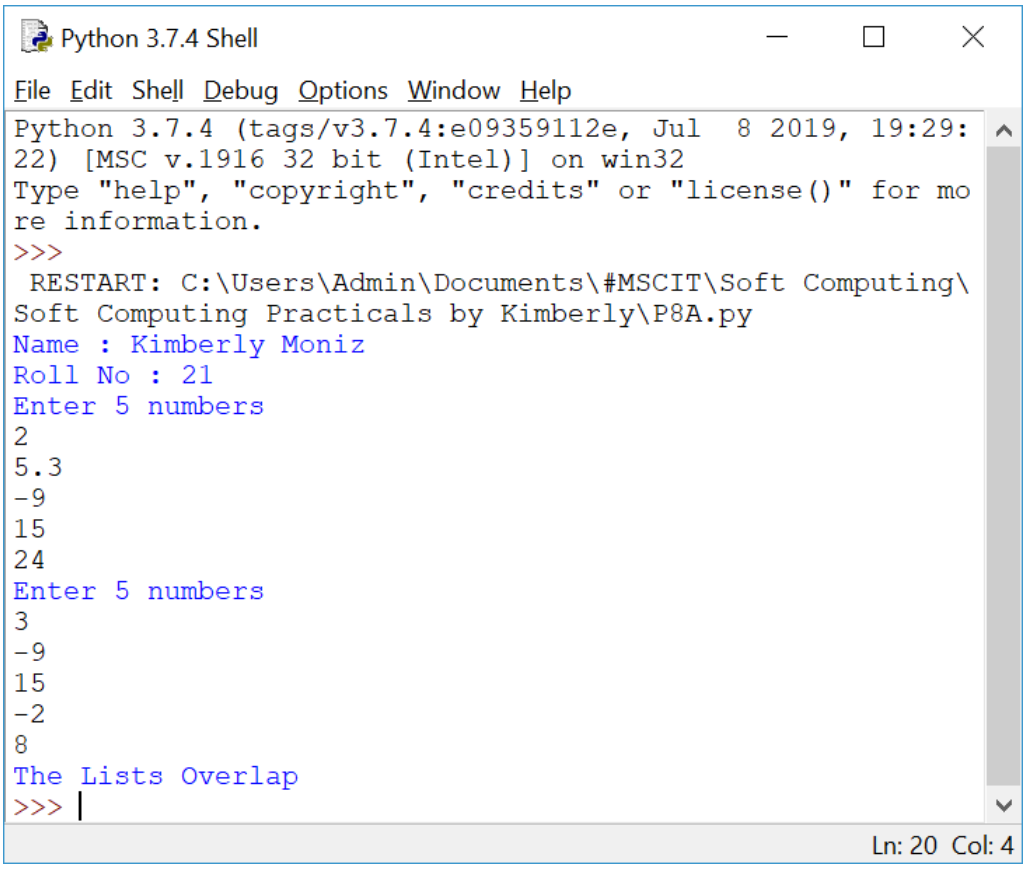
Practical 8

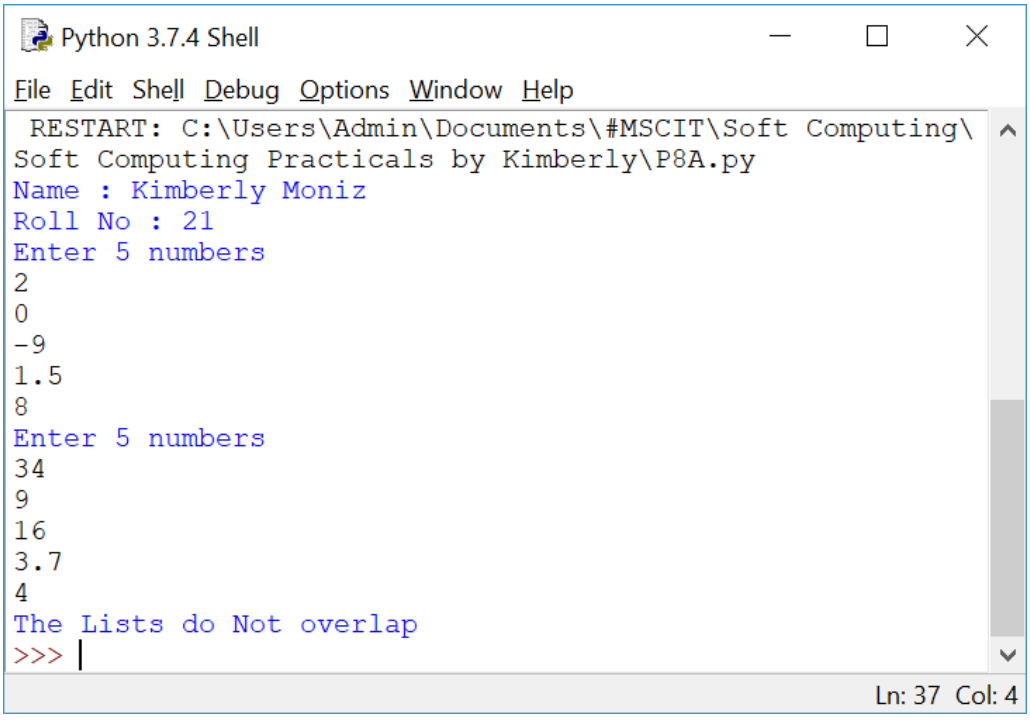
A] Membership and Identity Operators in and not in

Code for IN OPERATOR :

```
print("Name : Kimberly Moniz")
print("Roll No : 21")
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 :

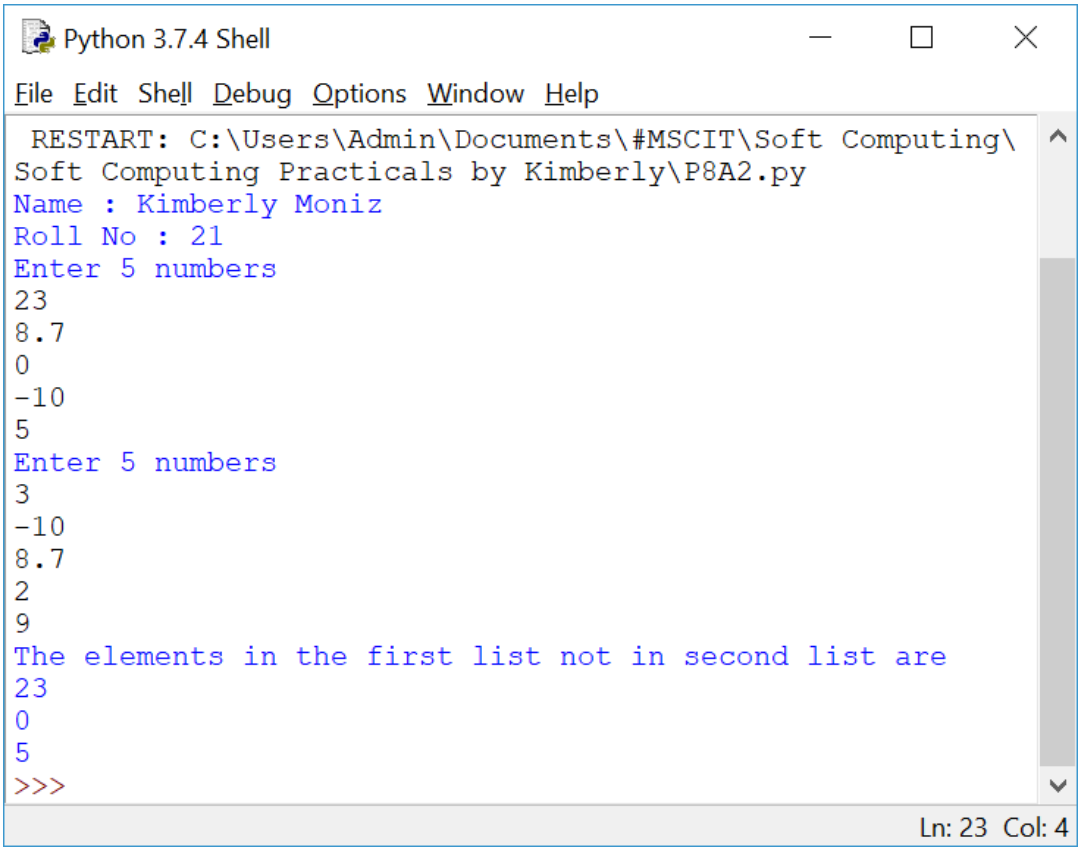

Python 3.7.4 Shell
File Edit Shell Debug Options Window Help
Python 3.7.4 (tags/v3.7.4:e09359112e, Jul 8 2019, 19:29:22) [MSC v.1916 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
RESTART: C:\Users\Admin\Documents\#MSCIT\Soft Computing\Soft Computing Practicals by Kimberly\P8A.py
Name : Kimberly Moniz
Roll No : 21
Enter 5 numbers
2
5.3
-9
15
24
Enter 5 numbers
3
-9
15
-2
8
The Lists Overlap
>>>
Ln: 20 Col: 4


Python 3.7.4 Shell
File Edit Shell Debug Options Window Help
RESTART: C:\Users\Admin\Documents\#MSCIT\Soft Computing\Soft Computing Practicals by Kimberly\P8A.py
Name : Kimberly Moniz
Roll No : 21
Enter 5 numbers
2
0
-9
1.5
8
Enter 5 numbers
34
9
16
3.7
4
The Lists do Not overlap
>>>
Ln: 37 Col: 4

Code for NOT IN operator:

```
print("Name : Kimberly Moniz")
print("Roll No : 21")
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 :

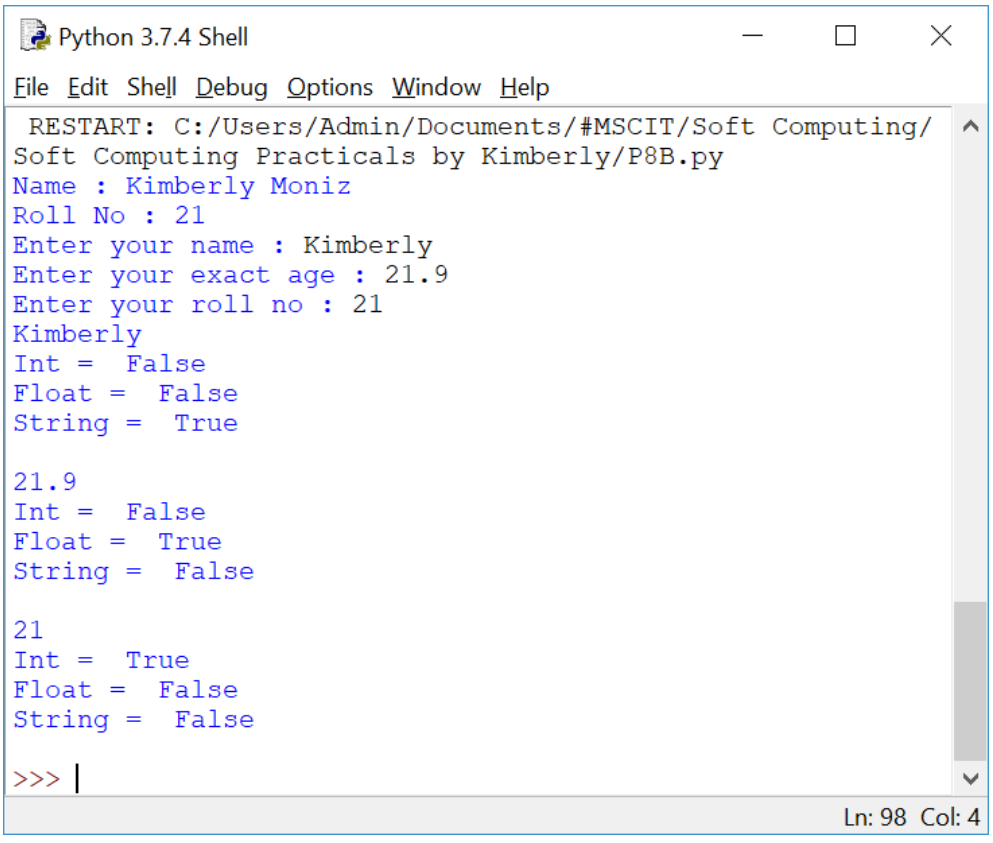


B] Membership and Identity Operators IS and IS NOT

Code for IS OPERATOR :

```
print("Name : Kimberly Moniz")
print("Roll No : 21")
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)

for i in details:
    print(i)
    print("Int = ",type(i) is int)
    print("Float = ",type(i) is float)
    print("String = ",type(i) is str)
    print()
```

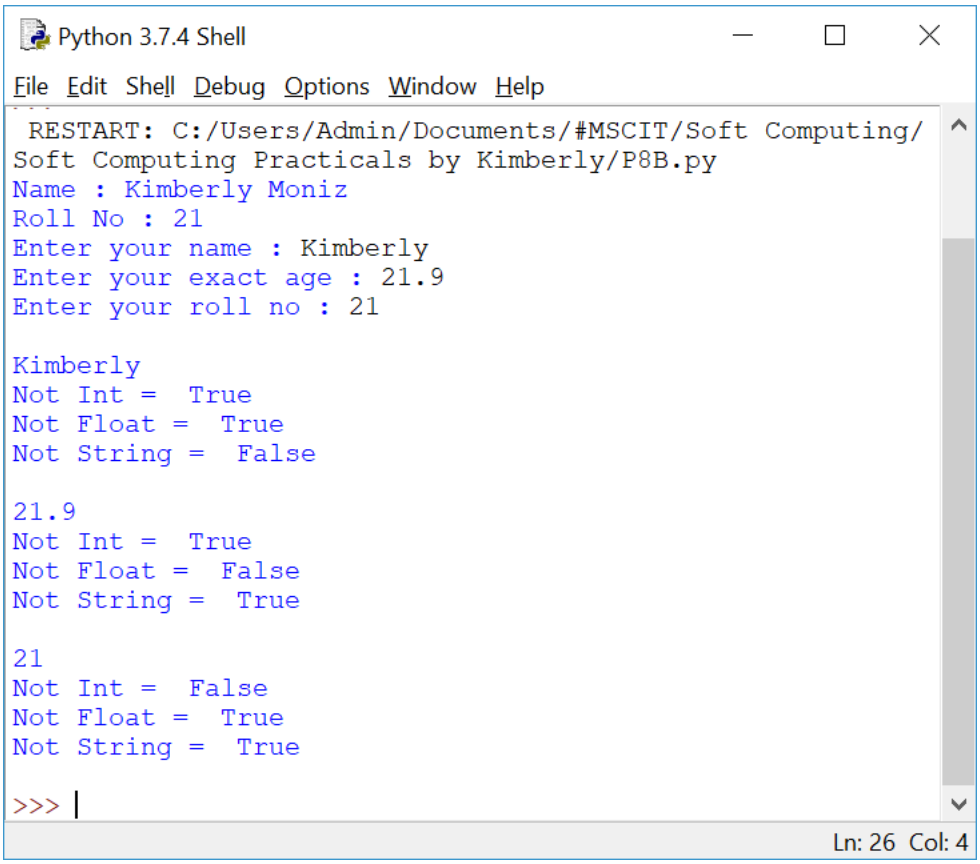
Output :



Code for IS NOT OPERATOR :

```
print("Name : Kimberly Moniz")
print("Roll No : 21")
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 :



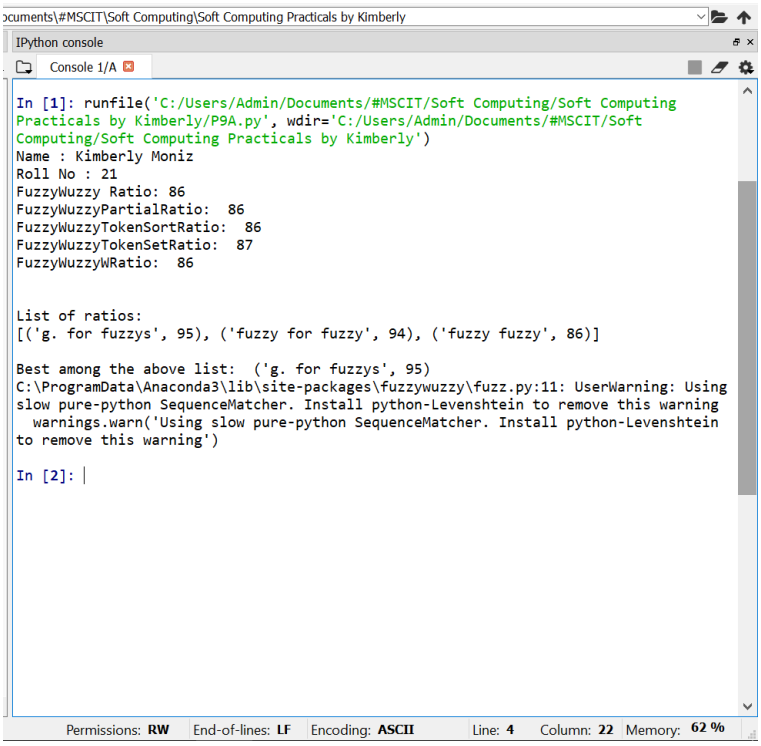
Practical 9

A] Find ratios using fuzzy logic

Code :

```
from fuzzywuzzy import fuzz
from fuzzywuzzy import process
print("Name : Kimberly Moniz")
print("Roll No : 21")
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 :

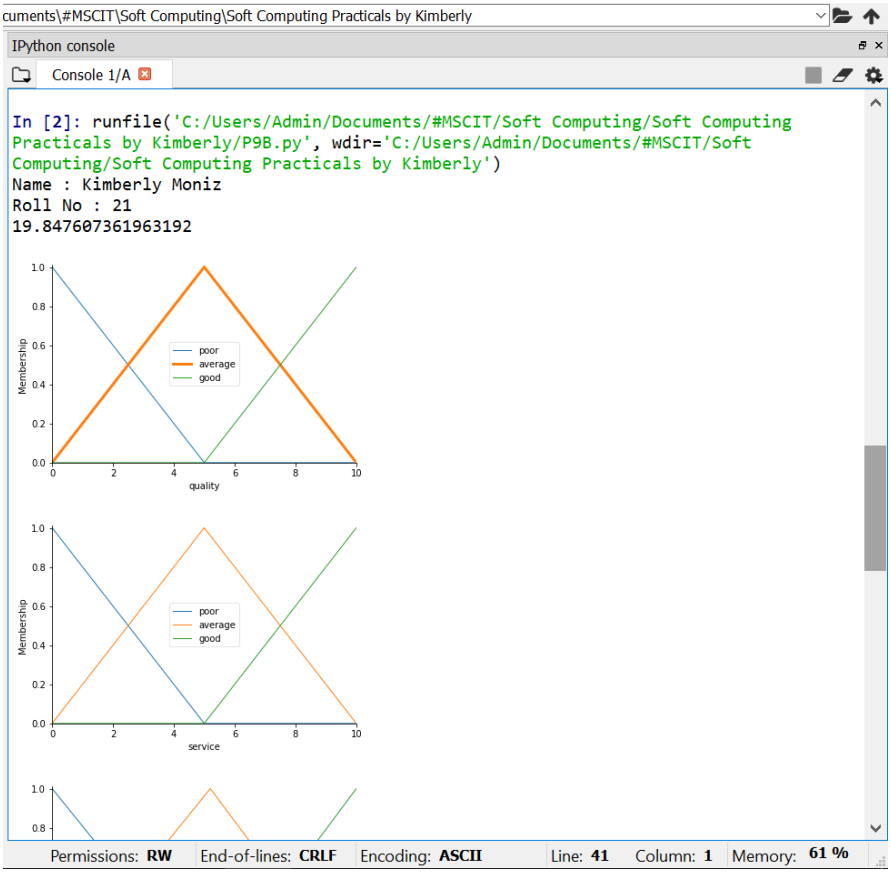


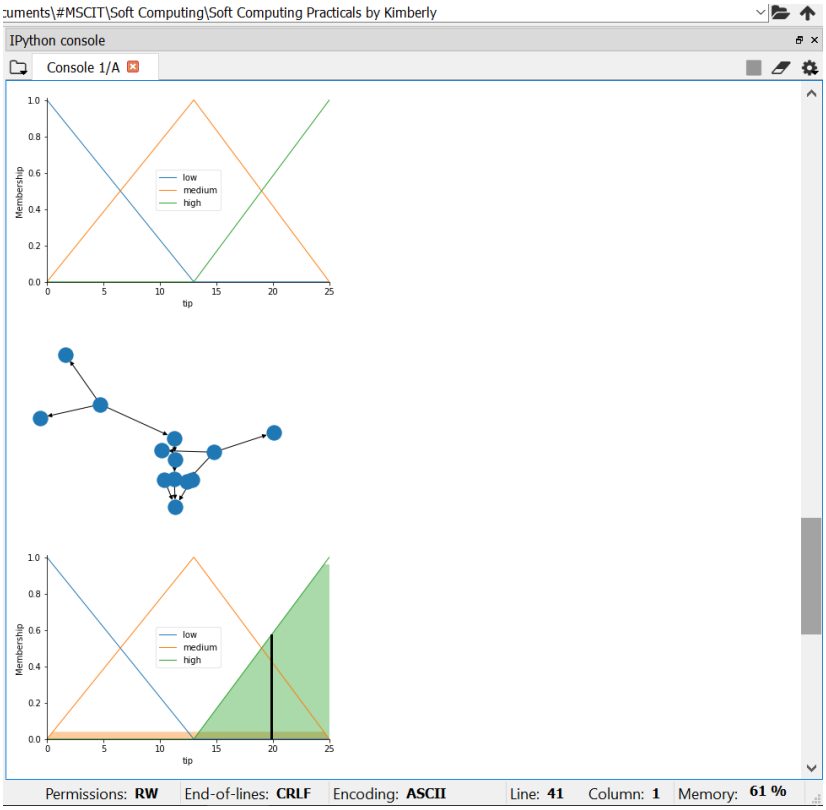
B] Solve Tipping problem using fuzzy logic
Code :

```
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
print("Name : Kimberly Moniz")
print("Roll No : 21")
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 :




cuments\#MSCIT\Soft Computing\Soft Computing Practicals by Kimberly
IPython console
Console 1/A
Membership
low
medium
high
tip
Permissions: RW
End-of-lines: CRLF
Encoding: ASCII
Line: 41
Column: 1
Memory: 61 %

Practical 10

A] Implementation of Simple genetic algorithm

Code :

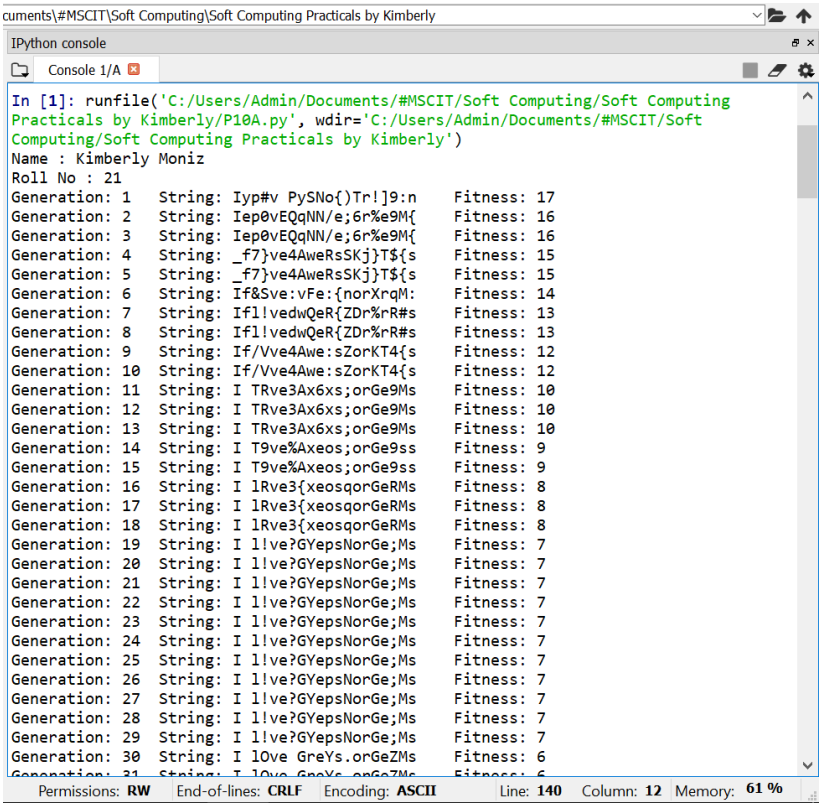
```
import random
print("Name : Kimberly Moniz")
print("Roll No : 21")
# Number of individuals in each generation
POPULATION_SIZE = 100
 # Valid genes
GENES = '''abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOP
QRSTUVWXYZ 1234567890, .-;:_!"#%&/()=?@${[]}'''
 # Target string to be generated
TARGET = "I love GeeksforGeeks"
 class Individual(object):
  '''
  Class representing individual in population
  '''
  def __init__(self, chromosome):
    self.chromosome = chromosome
    self.fitness = self.cal_fitness()

  @classmethod
  def mutated_genes(self):
    '''
    create random genes for mutation
    '''
    global GENES
    gene = random.choice(GENES)
    return gene

  @classmethod
  def create_gnome(self):
    '''
    create chromosome or string of genes
    '''
    global TARGET
    gnome_len = len(TARGET)
    return [self.mutated_genes() for _ in range(gnome_len)]

  def mate(self, par2):
```

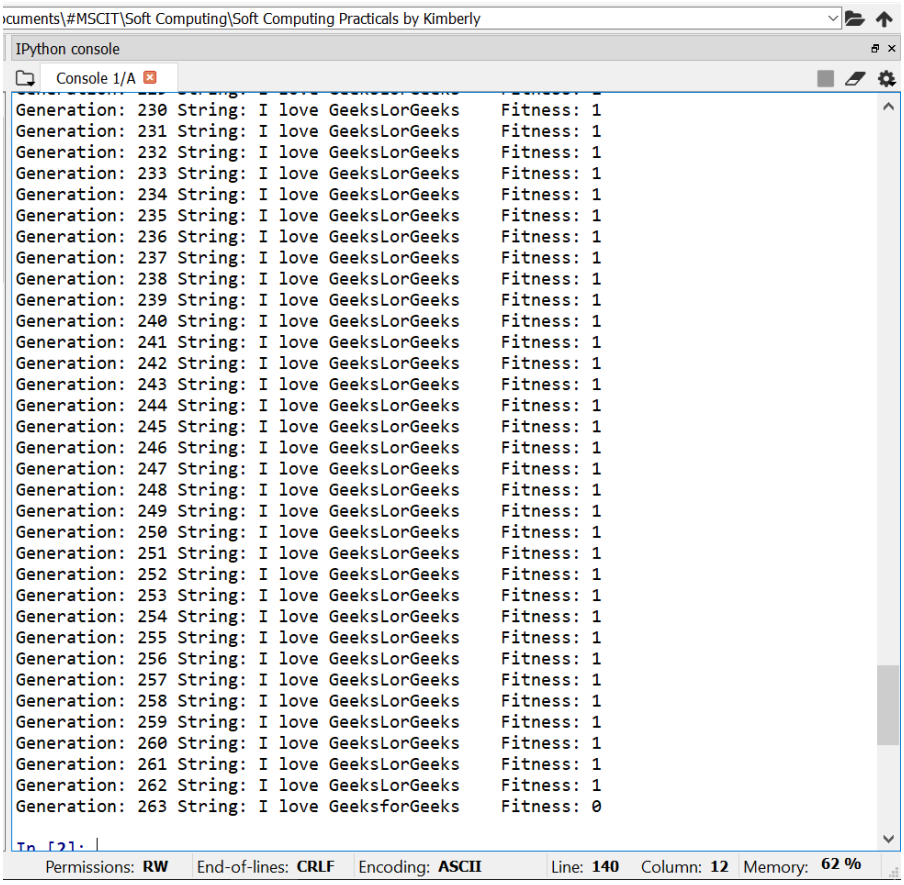
```
    '''
    Perform mating and produce new offspring
    '''

    # chromosome for offspring
    child_chromosome = []
    for gp1, gp2 in zip(self.chromosome, par2.chromosome):

        # random probability
        prob = random.random()

        # if prob is less than 0.45, insert gene
        # from parent 1
        if prob < 0.45:
            child_chromosome.append(gp1)

        # if prob is between 0.45 and 0.90, insert
        # gene from parent 2
        elif prob < 0.90:
            child_chromosome.append(gp2)

        # otherwise insert random gene(mutate),
        # for maintaining diversity
        else:
            child_chromosome.append(self.mutated_genes())

    # create new Individual(offspring) using
    # generated chromosome for offspring
    return Individual(child_chromosome)

def cal_fitness(self):
    '''
    Calculate fittness score, it is the number of
    characters in string which differ from target
    string.
    '''
    global TARGET
    fitness = 0
    for gs, gt in zip(self.chromosome, TARGET):
        if gs != gt: fitness+= 1
    return fitness
```

```
# Driver code
def main():
    global POPULATION_SIZE

    #current generation
    generation = 1

    found = False
    population = []

    # create initial population
    for _ in range(POPULATION_SIZE):
                gnome = Individual.create_gnome()
                population.append(Individual(gnome))

    while not found:
        # sort the population in increasing order of fitness score
        population = sorted(population, key = lambda x:x.fitness)

        # if the individual having lowest fitness score ie.
        # 0 then we know that we have reached to the target
        # and break the loop
        if population[0].fitness <= 0:
            found = True
            break

        # Otherwise generate new offsprings for new generation
        new_generation = []

        # Perform Elitism, that mean 10% of fittest population
        # goes to the next generation
        s = int((10*POPULATION_SIZE)/100)
        new_generation.extend(population[:s])

        # From 50% of fittest population, Individuals
        # will mate to produce offspring
        s = int((90*POPULATION_SIZE)/100)
        for _ in range(s):
            parent1 = random.choice(population[:50])
            parent2 = random.choice(population[:50])
```

```
            child = parent1.mate(parent2)
            new_generation.append(child)

        population = new_generation

        print("Generation: {}\tString: {}\tFitness: {}". format(generation,
"".join(population[0].chromosome),
              population[0].fitness))

        generation += 1

print("Generation: {}\tString: {}\tFitness: {}". format(generation,
          "".join(population[0].chromosome),
          population[0].fitness))

if __name__ == '__main__':
    main()
```

Output :




ɔcuments\#MSCIT\Soft Computing\Soft Computing Practicals by Kimberly
IPython console
Console 1/A
Generation: 230 String: I love GeeksLorGeeks Fitness: 1
Generation: 231 String: I love GeeksLorGeeks Fitness: 1
Generation: 232 String: I love GeeksLorGeeks Fitness: 1
Generation: 233 String: I love GeeksLorGeeks Fitness: 1
Generation: 234 String: I love GeeksLorGeeks Fitness: 1
Generation: 235 String: I love GeeksLorGeeks Fitness: 1
Generation: 236 String: I love GeeksLorGeeks Fitness: 1
Generation: 237 String: I love GeeksLorGeeks Fitness: 1
Generation: 238 String: I love GeeksLorGeeks Fitness: 1
Generation: 239 String: I love GeeksLorGeeks Fitness: 1
Generation: 240 String: I love GeeksLorGeeks Fitness: 1
Generation: 241 String: I love GeeksLorGeeks Fitness: 1
Generation: 242 String: I love GeeksLorGeeks Fitness: 1
Generation: 243 String: I love GeeksLorGeeks Fitness: 1
Generation: 244 String: I love GeeksLorGeeks Fitness: 1
Generation: 245 String: I love GeeksLorGeeks Fitness: 1
Generation: 246 String: I love GeeksLorGeeks Fitness: 1
Generation: 247 String: I love GeeksLorGeeks Fitness: 1
Generation: 248 String: I love GeeksLorGeeks Fitness: 1
Generation: 249 String: I love GeeksLorGeeks Fitness: 1
Generation: 250 String: I love GeeksLorGeeks Fitness: 1
Generation: 251 String: I love GeeksLorGeeks Fitness: 1
Generation: 252 String: I love GeeksLorGeeks Fitness: 1
Generation: 253 String: I love GeeksLorGeeks Fitness: 1
Generation: 254 String: I love GeeksLorGeeks Fitness: 1
Generation: 255 String: I love GeeksLorGeeks Fitness: 1
Generation: 256 String: I love GeeksLorGeeks Fitness: 1
Generation: 257 String: I love GeeksLorGeeks Fitness: 1
Generation: 258 String: I love GeeksLorGeeks Fitness: 1
Generation: 259 String: I love GeeksLorGeeks Fitness: 1
Generation: 260 String: I love GeeksLorGeeks Fitness: 1
Generation: 261 String: I love GeeksLorGeeks Fitness: 1
Generation: 262 String: I love GeeksLorGeeks Fitness: 1
Generation: 263 String: I love GeeksforGeeks Fitness: 0
In [2]:
Permissions: RW
End-of-lines: CRLF
Encoding: ASCII
Line: 140
Column: 12
Memory: 62 %

B] Create two classes: City and Fitness using Genetic algorithm

Code :

```
import numpy as np, random, operator, pandas as pd, matplotlib.pyplot as plt
print("Name : Kimberly Moniz")
print("Roll No : 21")
class City:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def distance(self, city):
        xDis = abs(self.x - city.x)
        yDis = abs(self.y - city.y)
        distance = np.sqrt((xDis ** 2) + (yDis ** 2))
        return distance

    def __repr__(self):
        return "(" + str(self.x) + "," + str(self.y) + ")"
class Fitness:
    def __init__(self, route):
        self.route = route
        self.distance = 0
        self.fitness= 0.0

    def routeDistance(self):
        if self.distance ==0:
            pathDistance = 0
            for i in range(0, len(self.route)):
                fromCity = self.route[i]
                toCity = None
                if i + 1 < len(self.route):
                    toCity = self.route[i + 1]
                else:
                    toCity = self.route[0]
                pathDistance += fromCity.distance(toCity)
            self.distance = pathDistance
        return self.distance

    def routeFitness(self):
        if self.fitness == 0:
```

```
            self.fitness = 1 / float(self.routeDistance())
        return self.fitness
def createRoute(cityList):
    route = random.sample(cityList, len(cityList))
    return route
def initialPopulation(popSize, cityList):
    population = []


    for i in range(0, popSize):
        population.append(createRoute(cityList))
    return population
def rankRoutes(population):
    fitnessResults = {}
    for i in range(0,len(population)):
        fitnessResults[i] = Fitness(population[i]).routeFitness()
    return sorted(fitnessResults.items(), key = operator.itemgetter(1), reverse = True)
def selection(popRanked, eliteSize):
    selectionResults = []
    df = pd.DataFrame(np.array(popRanked), columns=["Index","Fitness"])
    df['cum_sum'] = df.Fitness.cumsum()
    df['cum_perc'] = 100*df.cum_sum/df.Fitness.sum()

    for i in range(0, eliteSize):
        selectionResults.append(popRanked[i][0])
    for i in range(0, len(popRanked) - eliteSize):
        pick = 100*random.random()
        for i in range(0, len(popRanked)):
            if pick <= df.iat[i,3]:
                selectionResults.append(popRanked[i][0])
                break
    return selectionResults
def matingPool(population, selectionResults):
    matingpool = []
    for i in range(0, len(selectionResults)):
        index = selectionResults[i]
        matingpool.append(population[index])
    return matingpool
def breed(parent1, parent2):
    child = []
    childP1 = []
```

```
    childP2 = []

    geneA = int(random.random() * len(parent1))
    geneB = int(random.random() * len(parent1))

    startGene = min(geneA, geneB)
    endGene = max(geneA, geneB)


    for i in range(startGene, endGene):
        childP1.append(parent1[i])

    childP2 = [item for item in parent2 if item not in childP1]


    child = childP1 + childP2
    return child
def breedPopulation(matingpool, eliteSize):
    children = []
    length = len(matingpool) - eliteSize
    pool = random.sample(matingpool, len(matingpool))


    for i in range(0,eliteSize):
        children.append(matingpool[i])

    for i in range(0, length):
        child = breed(pool[i], pool[len(matingpool)-i-1])
        children.append(child)
    return children
def mutate(individual, mutationRate):
    for swapped in range(len(individual)):
        if(random.random() < mutationRate):
            swapWith = int(random.random() * len(individual))

            city1 = individual[swapped]
            city2 = individual[swapWith]

            individual[swapped] = city2
            individual[swapWith] = city1
    return individual
```

```
def mutatePopulation(population, mutationRate):
    mutatedPop = []

    for ind in range(0, len(population)):
        mutatedInd = mutate(population[ind], mutationRate)
        mutatedPop.append(mutatedInd)
    return mutatedPop
def nextGeneration(currentGen, eliteSize, mutationRate):
    popRanked = rankRoutes(currentGen)
    selectionResults = selection(popRanked, eliteSize)
    matingpool = matingPool(currentGen, selectionResults)
    children = breedPopulation(matingpool, eliteSize)
    nextGeneration = mutatePopulation(children, mutationRate)
    return nextGeneration
def geneticAlgorithm(population, popSize, eliteSize, mutationRate, generations):
    pop = initialPopulation(popSize, population)
    print("Initial distance: " + str(1 / rankRoutes(pop)[0][1]))

    for i in range(0, generations):
        pop = nextGeneration(pop, eliteSize, mutationRate)

    print("Final distance: " + str(1 / rankRoutes(pop)[0][1]))
    bestRouteIndex = rankRoutes(pop)[0][0]
    bestRoute = pop[bestRouteIndex]
    return bestRoute
cityList = []
for i in range(0,25):
    cityList.append(City(x=int(random.random() * 200), y=int(random.random() * 200)))
geneticAlgorithm(population=cityList, popSize=100, eliteSize=20, mutationRate=0.01,
generations=500)

def geneticAlgorithmPlot(population, popSize, eliteSize, mutationRate, generations):
    pop = initialPopulation(popSize, population)
    progress = []
    progress.append(1 / rankRoutes(pop)[0][1])

    for i in range(0, generations):
        pop = nextGeneration(pop, eliteSize, mutationRate)
        progress.append(1 / rankRoutes(pop)[0][1])

    plt.plot(progress)
```

```
    plt.ylabel('Distance')
    plt.xlabel('Generation')
    plt.show()
geneticAlgorithmPlot(population=cityList, popSize=100, eliteSize=20, mutationRate=0.01,
generations=500)
```

Output :