**AIM:** Write a program in Python to implement single layer perceptron for ANDNOT function.

**CODE:**

import numpy as np

x=np.array([[1,1],[1,-1],[-1,1],[-1,-1]])

t=np.array([[-1],[1],[-1],[-1]])

w=np.array([[0],[0]])

b=0

theta=float(input("Enter new theta:"))

alpha=float(input("Enter new alpha:"))

yin=np.zeros(shape=(4,1))

y=np.zeros(shape=(4,1))

i=0

found=0

while(found==0):

    yin=x[i][0]\*w[0]+x[i][1]\*w[1]

    yin = yin+b

    if(yin>theta):

        y[i] = 1

    elif(yin<=theta and yin>=-theta):

        y[i]=0

    else:

        y[i]=-1

    if (y[i]==t[i]):

        print("NO UPDATION REQUIRED")

        print(y[i])

        if(i<3):

            i=i+1

        else:

            i=0

    else:

        print("MODEL IS NOT TRAINED")

        print("The value of output is")

        print(y)

        w[0]=w[0]+alpha\*x[i][0]\*t[i]

        w[1]=w[1]+alpha\*x[i][1]\*t[i]

        b = b+alpha\*t[i]

        if(i<3):

            i=i+1

        else:

            i=0

    if(y==t).all():

        found=1

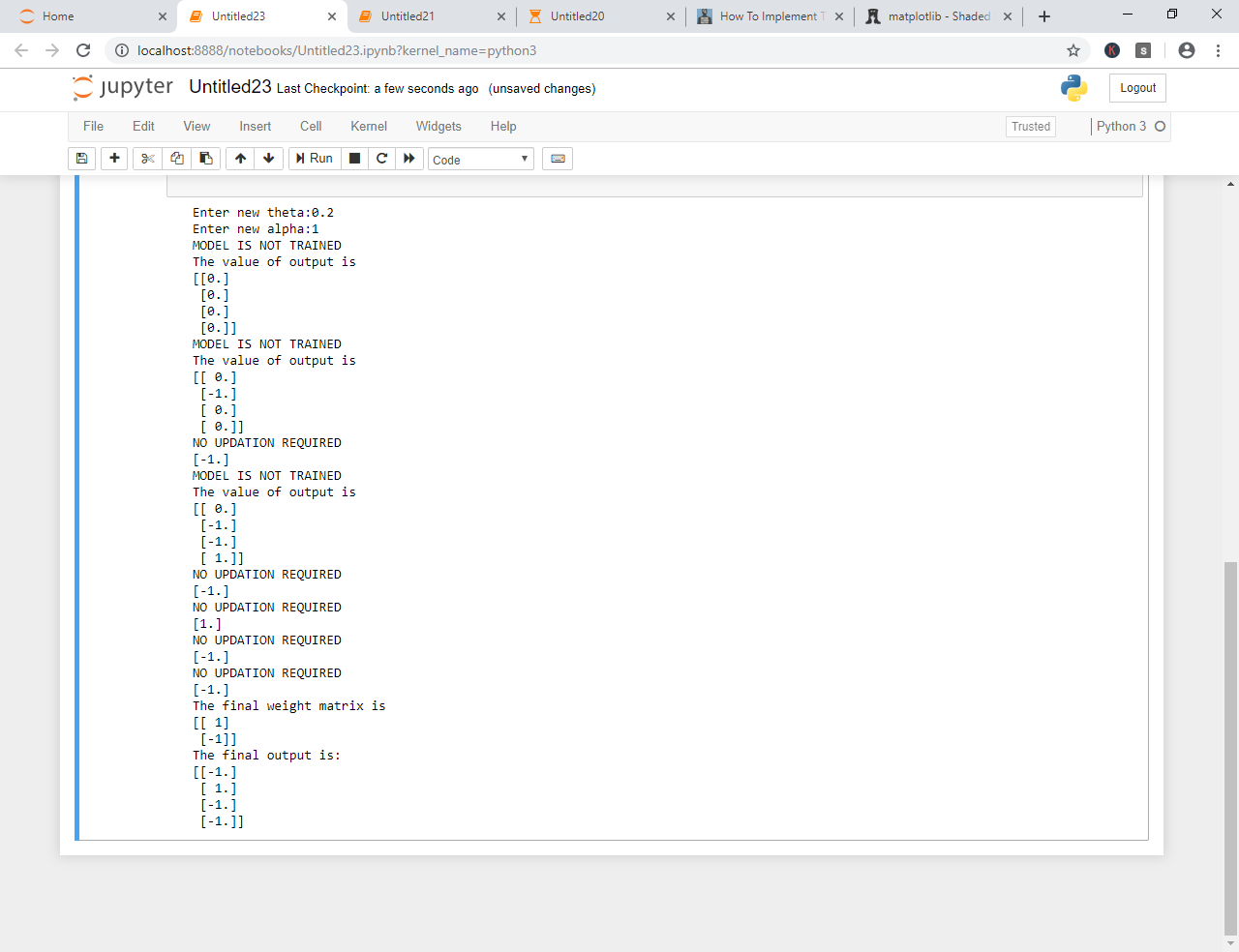
print("The final weight matrix is ")

print(w)

print("The final output is:")

print(y)

**OUTPUT:**



**AIM:** Write a program in Python to implement single layer perceptron for AND function.

**CODE:**

import numpy as np

x=np.array([[1,1],[1,-1],[-1,1],[-1,-1]])

t=np.array([[1],[1],[1],[-1]])

w=np.array([[0],[0]])

b=0

theta=float(input("Enter new theta:"))

alpha=float(input("Enter new alpha:"))

yin=np.zeros(shape=(4,1))

y=np.zeros(shape=(4,1))

i=0

found=0

while(found==0):

    yin=x[i][0]\*w[0]+x[i][1]\*w[1]

    yin = yin+b

    if(yin>theta):

        y[i] = 1

    elif(yin<=theta and yin>=-theta):

        y[i]=0

    else:

        y[i]=-1

    if (y[i]==t[i]):

        print("NO UPDATION REQUIRED")

        print(y[i])

        if(i<3):

            i=i+1

        else:

            i=0

    else:

        print("MODEL IS NOT TRAINED")

        print("The value of output is")

        print(y)

        w[0]=w[0]+alpha\*x[i][0]\*t[i]

        w[1]=w[1]+alpha\*x[i][1]\*t[i]

        b = b+alpha\*t[i]

        if(i<3):

            i=i+1

        else:

            i=0

    if(y==t).all():

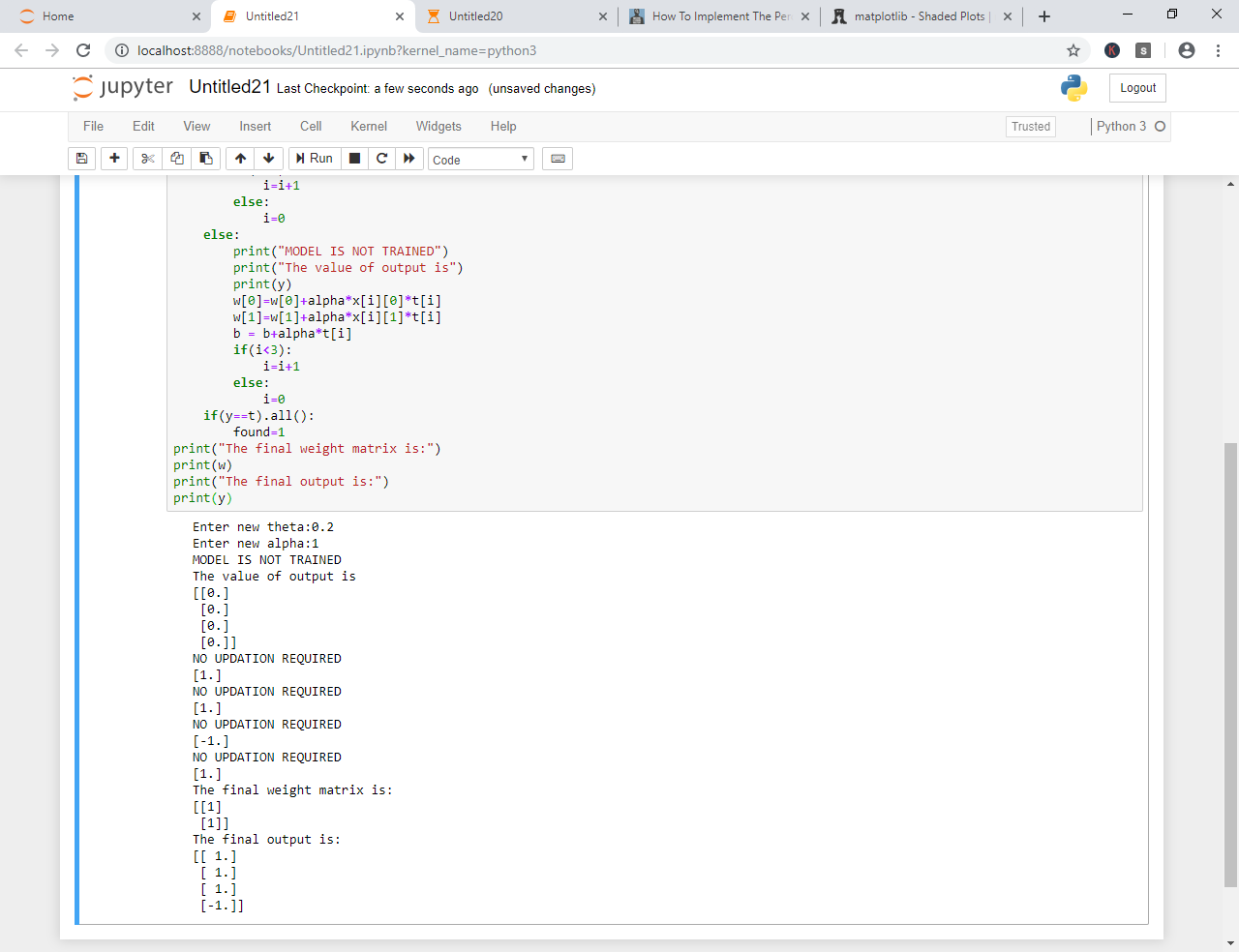
        found=1

print("The final weight matrix is:")

print(w)

print("The final output is:")

print(y)

**OUTPUT:**  


**Aim:**Write a program to implement the following logic functions using single layer Perceptron OR logic functions.

**Code:**

#OR

importnumpy as np

x=np.array([[1,1],[1,-1],[-1,1],[-1,-1]])

t=np.array([[1],[1],[1],[-1]])

w=np.array([[0],[0]])

b=0

theta=float(input("enter new theta"))

alpha=float(input("enter new alpha"))

yin=np.zeros(shape=(4,1))

y=np.zeros(shape=(4,1))

i=0

found=0

while(found==0):

yin=x[i][0]\*w[0]+x[i][1]\*w[1]

yin = yin+b

if(yin>theta):

y[i] = 1

elif(yin<=theta and yin>=-theta):

y[i]=0

else:

y[i]=-1

if (y[i]==t[i]):

print("NO UPDATION REQUIRED")

print(y[i])

if(i<3):

i=i+1

else:

i=0

else:

print("MODEL IS NOT TRAINED")

print("The value of output is")

print(y)

w[0]=w[0]+alpha\*x[i][0]\*t[i]

w[1]=w[1]+alpha\*x[i][1]\*t[i]

        b = b+alpha\*t[i]

if(i<3):

i=i+1

else:

i=0

if(y==t).all():

found=1

print("The final weight matrix is ")

print(w)

print("The final output is:")

print(y)

**Output:**

