**Practical 3**

**Aim:-Decision tree learning**

**import numpy as np**

**import pandas as pd**

**PlayTennis= pd.read\_csv("D:\AI practical\PT\PTcsv\PlayTennis.csv")**

**print(PlayTennis,"\n\n")**

**from sklearn.preprocessing import LabelEncoder**

**Le= LabelEncoder()**

**PlayTennis['Outlook']=Le.fit\_transform(PlayTennis['Outlook'])**

**PlayTennis['Temperature']=Le.fit\_transform(PlayTennis['Temperature'])**

**PlayTennis['Humidity']=Le.fit\_transform(PlayTennis['Humidity'])**

**PlayTennis['Wind']=Le.fit\_transform(PlayTennis['Wind'])**

**PlayTennis['Play Tennis']=Le.fit\_transform(PlayTennis['Play Tennis'])**

**print(PlayTennis,"\n\n")**

**y=PlayTennis['Play Tennis']**

**X=PlayTennis.drop(['Play Tennis'], axis=1)**

**from sklearn import tree**

**clf=tree.DecisionTreeClassifier(criterion='entropy')**

**clf=clf.fit(X, y)**

**print(tree.plot\_tree(clf),"\n\n")**

**import graphviz**

**dot\_data=tree.export\_graphviz(clf, out\_file=None)**

**graph=graphviz.Source(dot\_data)**

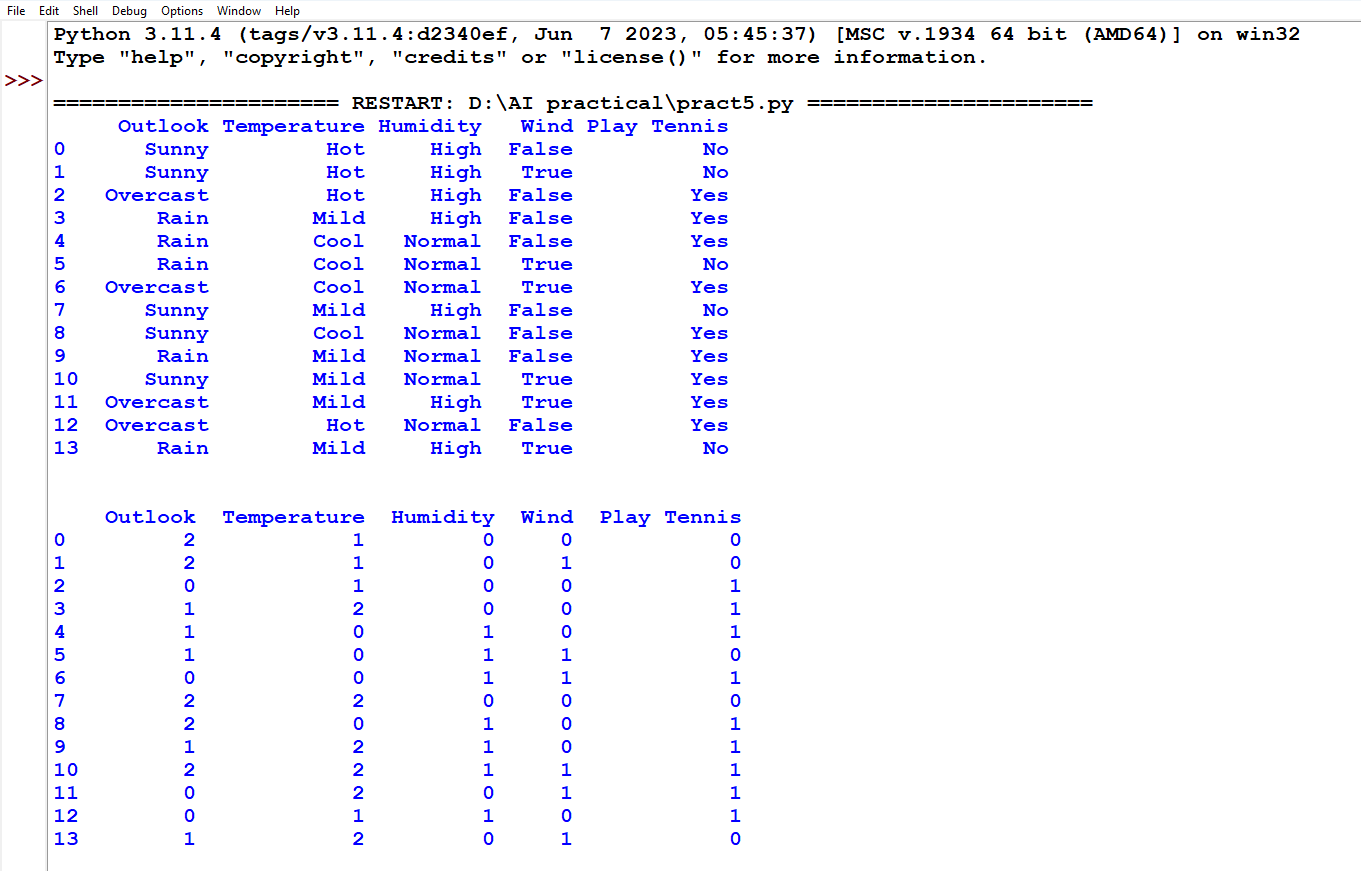
**print(graph)**

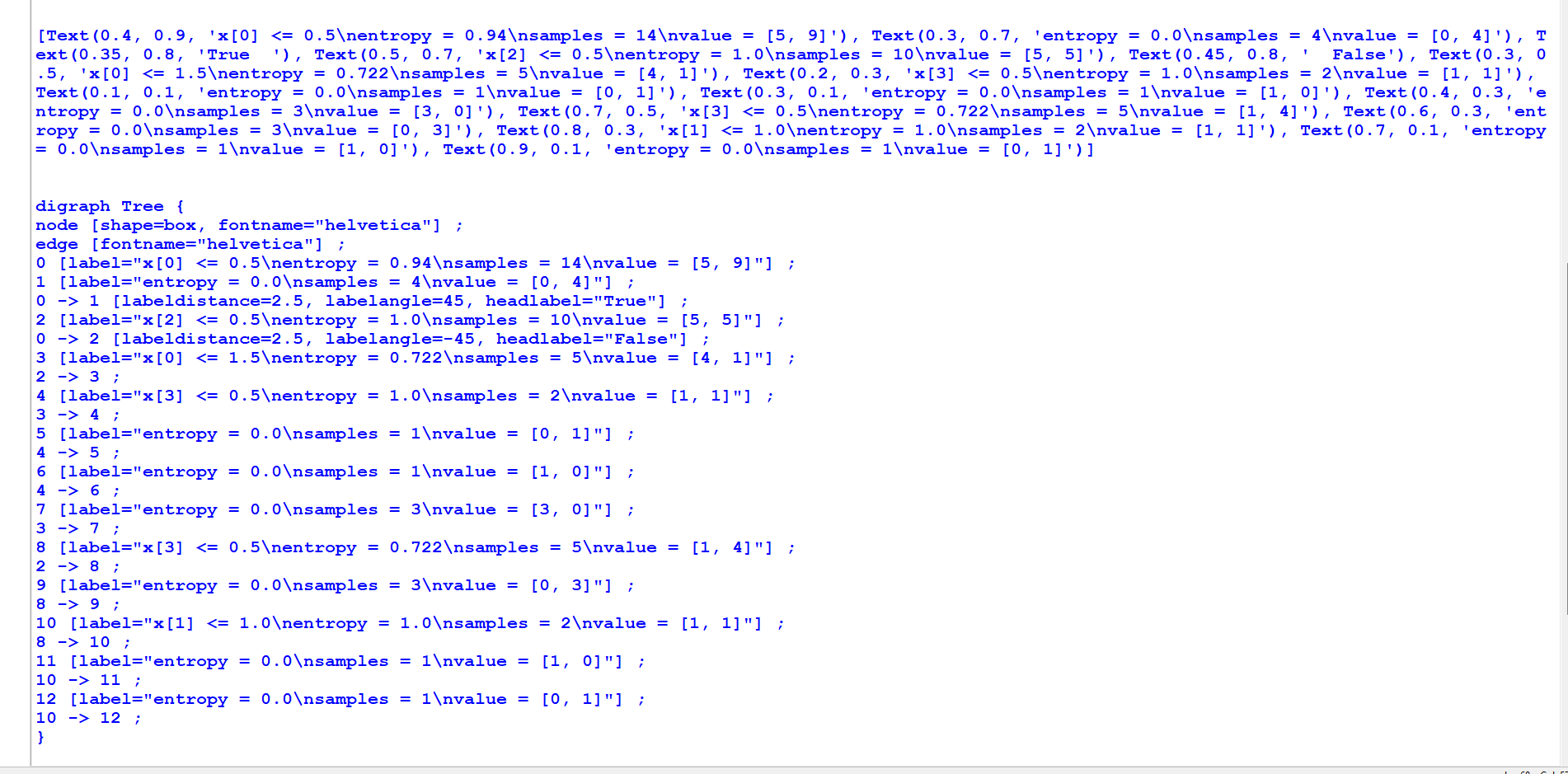
**# python shell can't print graph image so, it will print the code for diagraph implementation**

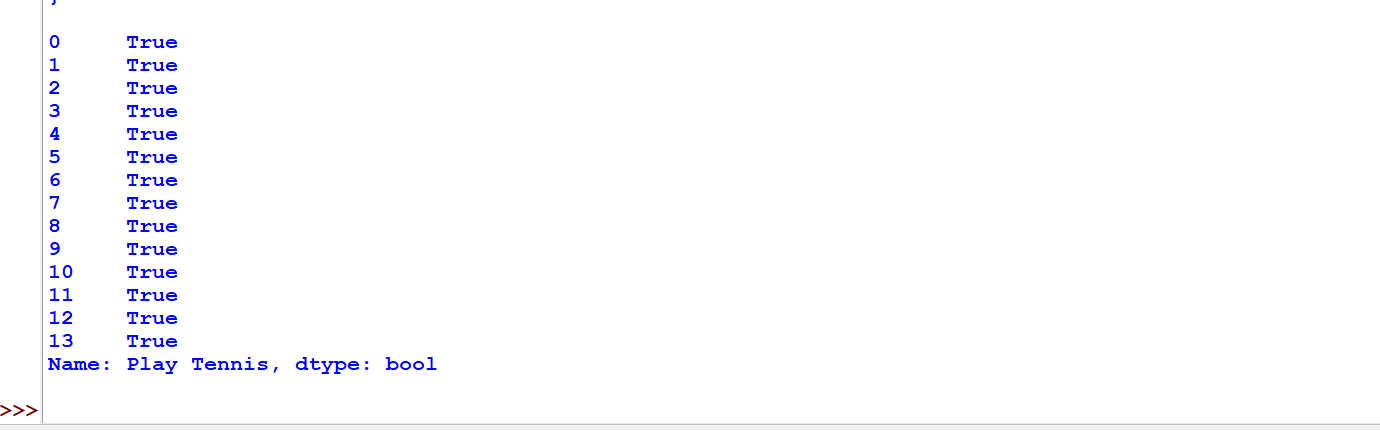
**X\_pred=clf.predict(X)**

**print(X\_pred==y,"\n")**

**Output:-**







**Practical 4**

**Aim:-Feed Forward Backpropagation neural network**

**from doctest import OutputChecker**

**import numpy as np**

**class NeuralNetwork():**

**def \_\_init\_\_(self):**

**np.random.seed()**

**self.synaptic\_weights=2\*np.random.random((3,1))-1**

**def sigmoid(self,x):**

**return 1/(1+np.exp(-x))**

**def sigmoid\_derivative(self,x):**

**return x\*(1-x)**

**def train(self,training\_inputs,training\_outputs,training\_iteration):**

**for iteration in range(training\_iteration):**

**output=self.think(training\_inputs)**

**error=training\_outputs-output**

**adjustments=np.dot(training\_inputs.T,error\*self.sigmoid\_derivative(output))**

**self.synaptic\_weights+=adjustments**

**def think(self,inputs):**

**inputs=inputs.astype(float)**

**output=self.sigmoid(np.dot(inputs,self.synaptic\_weights))**

**return output**

**if \_\_name\_\_ == "\_\_main\_\_":**

**neural\_network = NeuralNetwork()**

**print("Beginning Randomly Generated Weights: ")**

**print(neural\_network.synaptic\_weights)**

**training\_inputs = np.array([[0,0,1],[1,1,1],[1,0,1],[0,1,1]])**

**training\_outputs = np.array([[0,1,1,0]]).T**

**neural\_network.train(training\_inputs,training\_outputs,15000)**

**print("Ending Weights After Training : ")**

**print(neural\_network.synaptic\_weights)**

**user\_input\_one=str(input("User Input One: "))**

**user\_input\_two=str(input("User Input Two: "))**

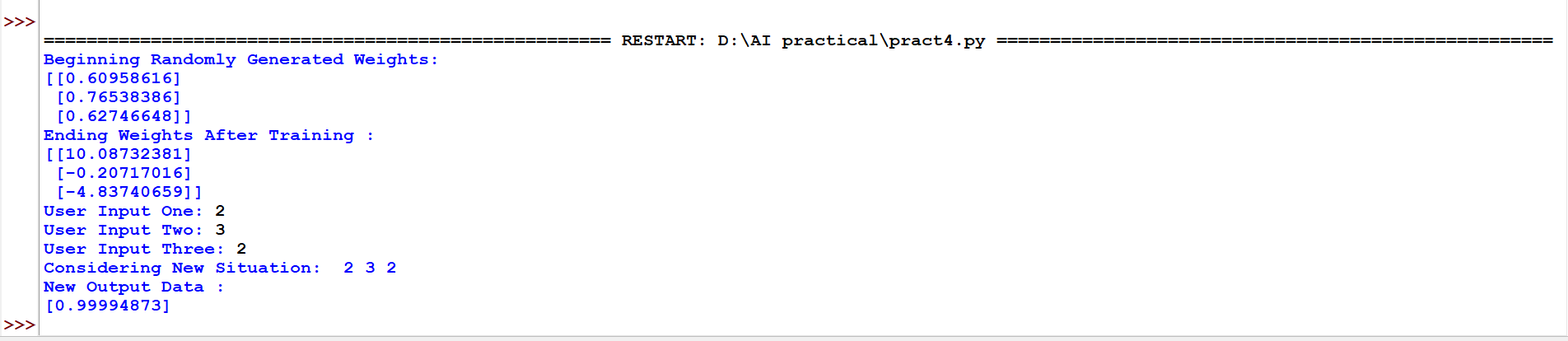
**user\_input\_three=str(input("User Input Three: "))**

**print("Considering New Situation: ", user\_input\_one, user\_input\_two, user\_input\_three)**

**print("New Output Data : ")**

**print(neural\_network.think(np.array([user\_input\_one,user\_input\_two,user\_input\_three])))**

**OUTPUT:-**

****