EAST POINT COLLEGE OF ENGINEERING & TECHNOLOGY

Department of Computer Science and Engineering

Approved by AICTE New Delhi | Affiliated to VTU, Belagavi,
Virgo Nagar, Bengaluru-560049

Department of Computer Science and Engineering

BE - VII SEMESTER

ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING LABORATORY

18CSL76

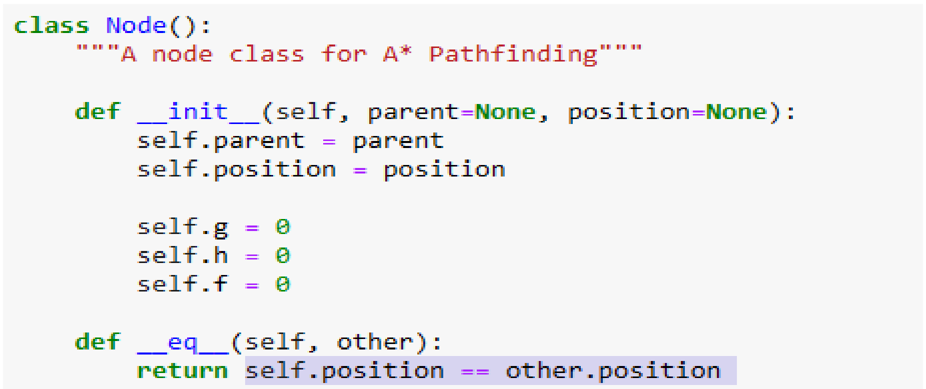
Course Instructors:
Dr.I.Manimozhi/ Prof. Manimegalai/ Dr. Shanker Ganesh/ Prof. Divya

Name of the student:	
USN:	
Section & Batch:	

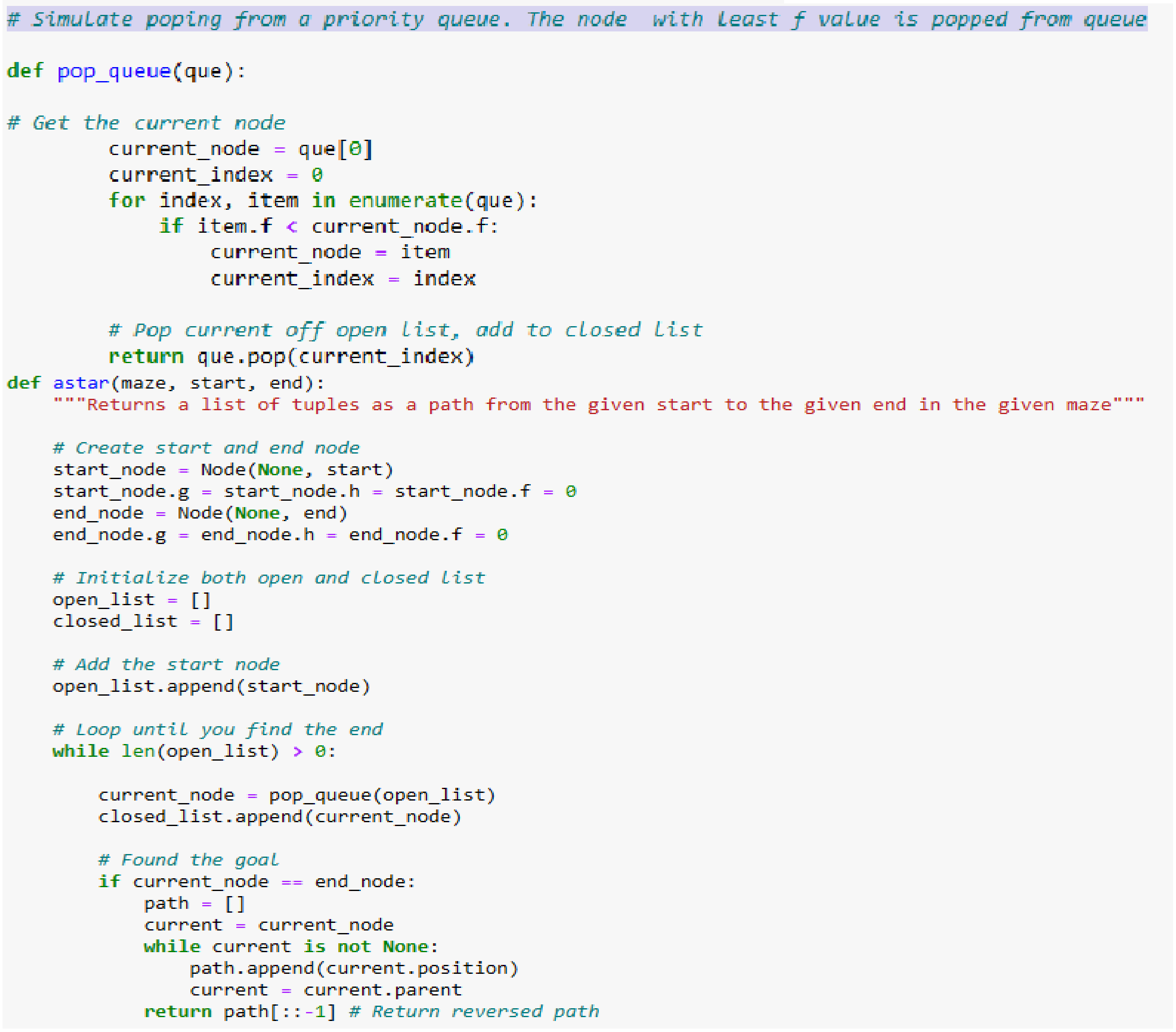
INDEX

1. Implement A* Search Algorithm.

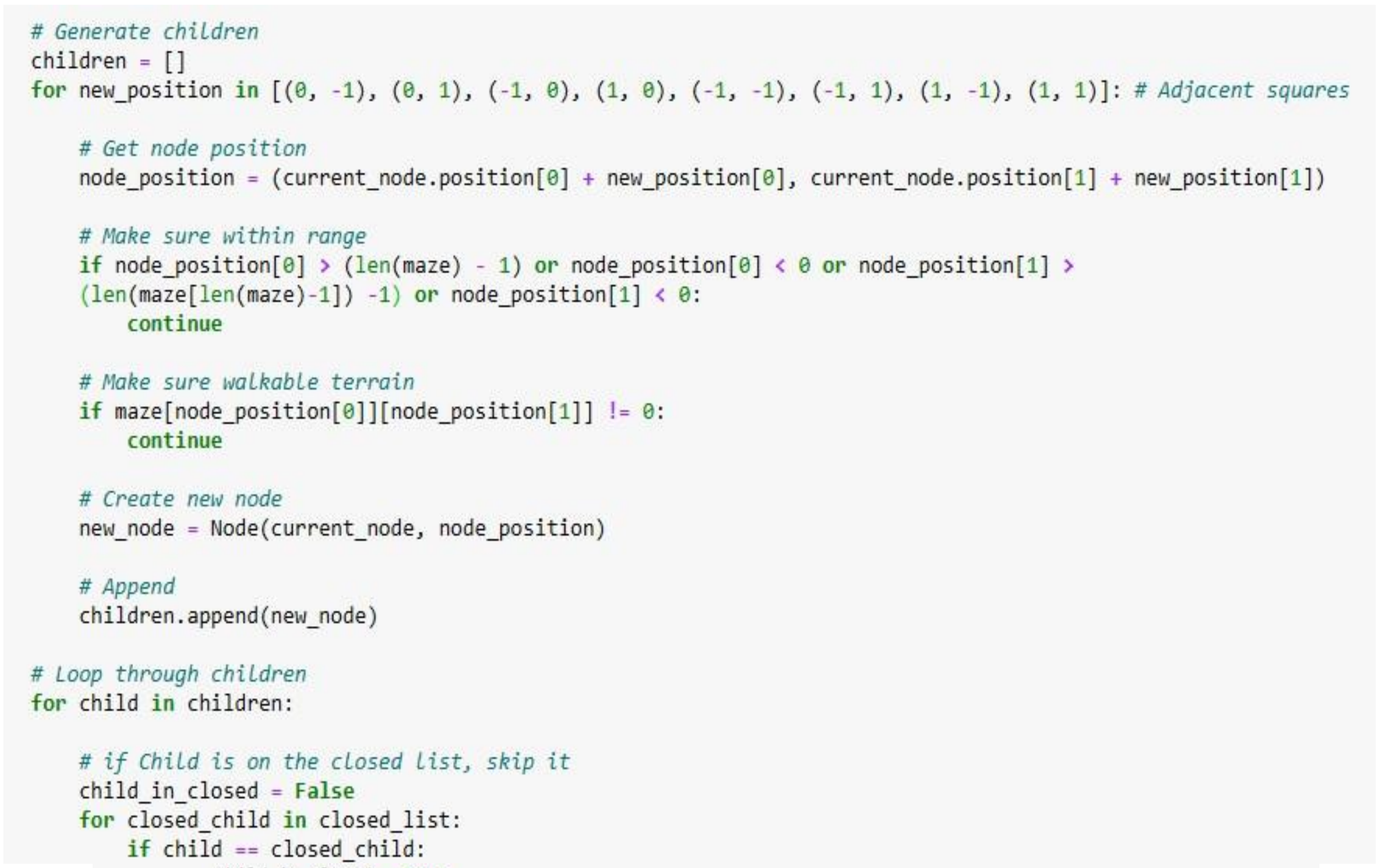
Program :



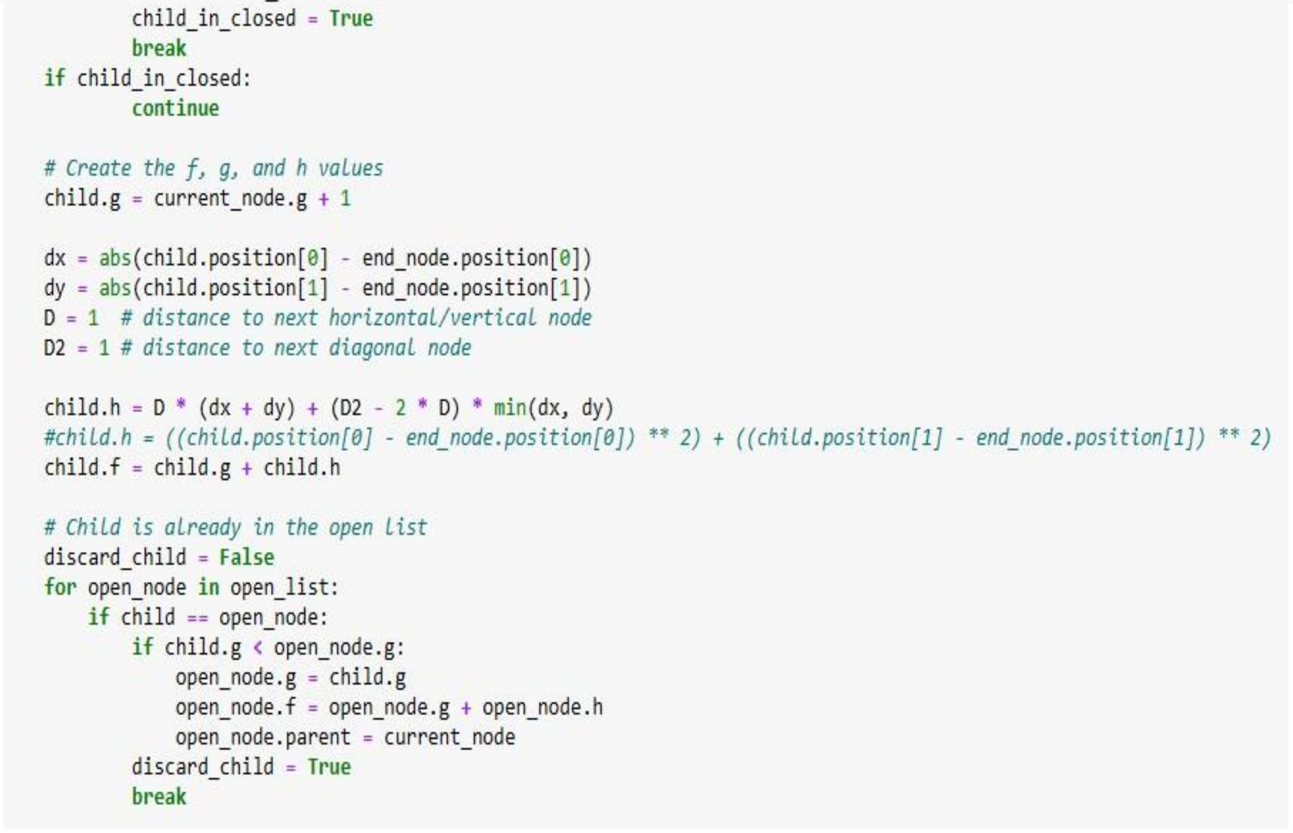
```
class Node():
    """A node class for A* Pathfinding"""

    def __init__(self, parent=None, position=None):
        self.parent = parent
        self.position = position

        self.g = 0
        self.h = 0
        self.f = 0

    def __eq__(self, other):
        return self.position == other.position
```



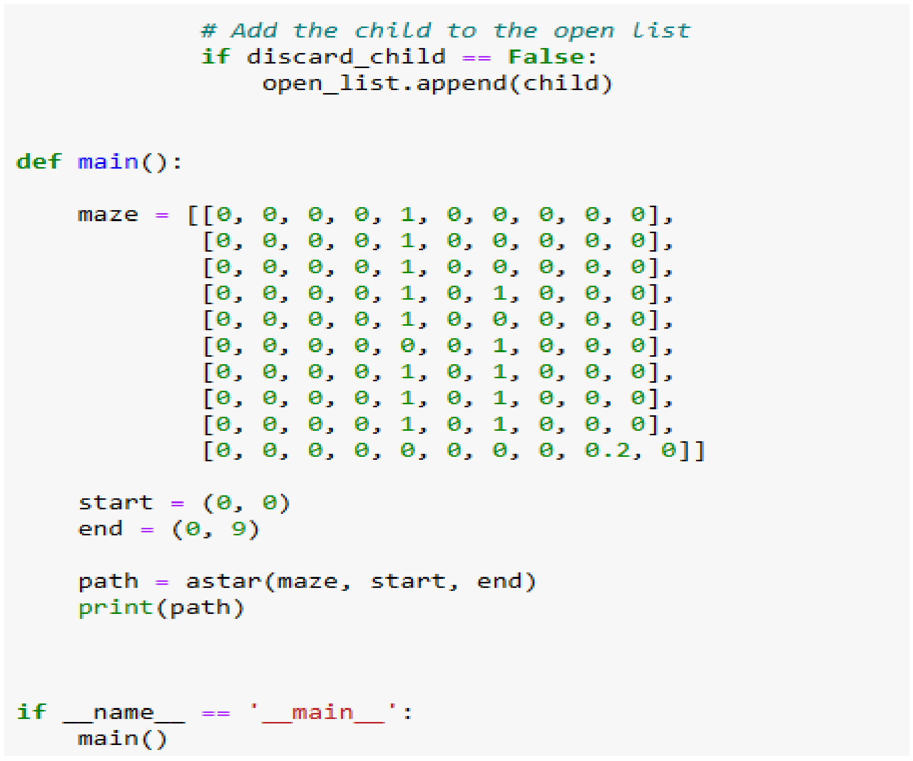
```
# Simulate poping from a priority queue. The node  with least f value is popped from queue

def pop_queue(que):

# Get the current node
    current_node = que[0]
    current_index = 0
    for index, item in enumerate(que):
        if item.f < current_node.f:
            current_node = item
            current_index = index

    # Pop current off open list, add to closed list
    return que.pop(current_index)
def astar(maze, start, end):
    """Returns a list of tuples as a path from the given start to the given end in the given maze"""

    # Create start and end node
    start_node = Node(None, start)
    start_node.g = start_node.h = start_node.f = 0
    end_node = Node(None, end)
    end_node.g = end_node.h = end_node.f = 0

    # Initialize both open and closed list
    open_list = []
    closed_list = []

    # Add the start node
    open_list.append(start_node)

    # Loop until you find the end
    while len(open_list) > 0:

        current_node = pop_queue(open_list)
        closed_list.append(current_node)

        # Found the goal
        if current_node == end_node:
            path = []
            current = current_node
            while current is not None:
                path.append(current.position)
                current = current.parent
            return path[::-1] # Return reversed path
```



```python
# Generate children
children = []
for new_position in [(0, -1), (0, 1), (-1, 0), (1, 0), (-1, -1), (-1, 1), (1, -1), (1, 1)]: # Adjacent squares

    # Get node position
    node_position = (current_node.position[0] + new_position[0], current_node.position[1] + new_position[1])

    # Make sure within range
    if node_position[0] > (len(maze) - 1) or node_position[0] < 0 or node_position[1] >
    (len(maze[len(maze)-1]) -1) or node_position[1] < 0:
        continue

    # Make sure walkable terrain
    if maze[node_position[0]][node_position[1]] != 0:
        continue

    # Create new node
    new_node = Node(current_node, node_position)

    # Append
    children.append(new_node)

# Loop through children
for child in children:

    # if Child is on the closed list, skip it
    child_in_closed = False
    for closed_child in closed_list:
        if child == closed_child:
```



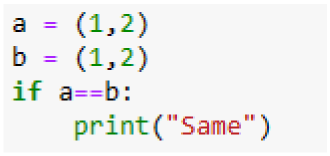
```python
            child_in_closed = True
            break
    if child_in_closed:
            continue

    # Create the f, g, and h values
    child.g = current_node.g + 1

    dx = abs(child.position[0] - end_node.position[0])
    dy = abs(child.position[1] - end_node.position[1])
    D = 1  # distance to next horizontal/vertical node
    D2 = 1 # distance to next diagonal node

    child.h = D * (dx + dy) + (D2 - 2 * D) * min(dx, dy)
    #child.h = ((child.position[0] - end_node.position[0]) ** 2) + ((child.position[1] - end_node.position[1]) ** 2)
    child.f = child.g + child.h

    # Child is already in the open list
    discard_child = False
    for open_node in open_list:
        if child == open_node:
            if child.g < open_node.g:
                open_node.g = child.g
                open_node.f = open_node.g + open_node.h
                open_node.parent = current_node
            discard_child = True
            break
```



```python
            # Add the child to the open list
            if discard_child == False:
                open_list.append(child)


def main():

    maze = [[0, 0, 0, 0, 1, 0, 0, 0, 0, 0],
            [0, 0, 0, 0, 1, 0, 0, 0, 0, 0],
            [0, 0, 0, 0, 1, 0, 0, 0, 0, 0],
            [0, 0, 0, 0, 1, 0, 1, 0, 0, 0],
            [0, 0, 0, 0, 1, 0, 0, 0, 0, 0],
            [0, 0, 0, 0, 0, 0, 1, 0, 0, 0],
            [0, 0, 0, 0, 1, 0, 1, 0, 0, 0],
            [0, 0, 0, 0, 1, 0, 1, 0, 0, 0],
            [0, 0, 0, 0, 1, 0, 1, 0, 0, 0],
            [0, 0, 0, 0, 0, 0, 0, 0, 0.2, 0]]

    start = (0, 0)
    end = (0, 9)

    path = astar(maze, start, end)
    print(path)


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

Output :

```
[(0, 0), (1, 1), (2, 2), (3, 3), (4, 3), (5, 4), (4, 5), (4, 6), (3, 7), (2, 7), (1, 8), (0, 9)]
```



```python
a = (1,2)
b = (1,2)
if a==b:
    print("Same")
```

Output :

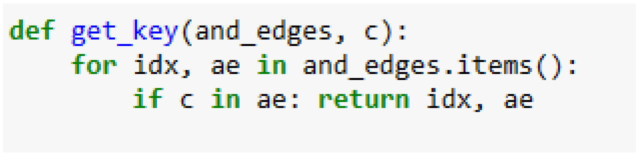
```
Same
```

2. Implement AO* Search Algorithm.

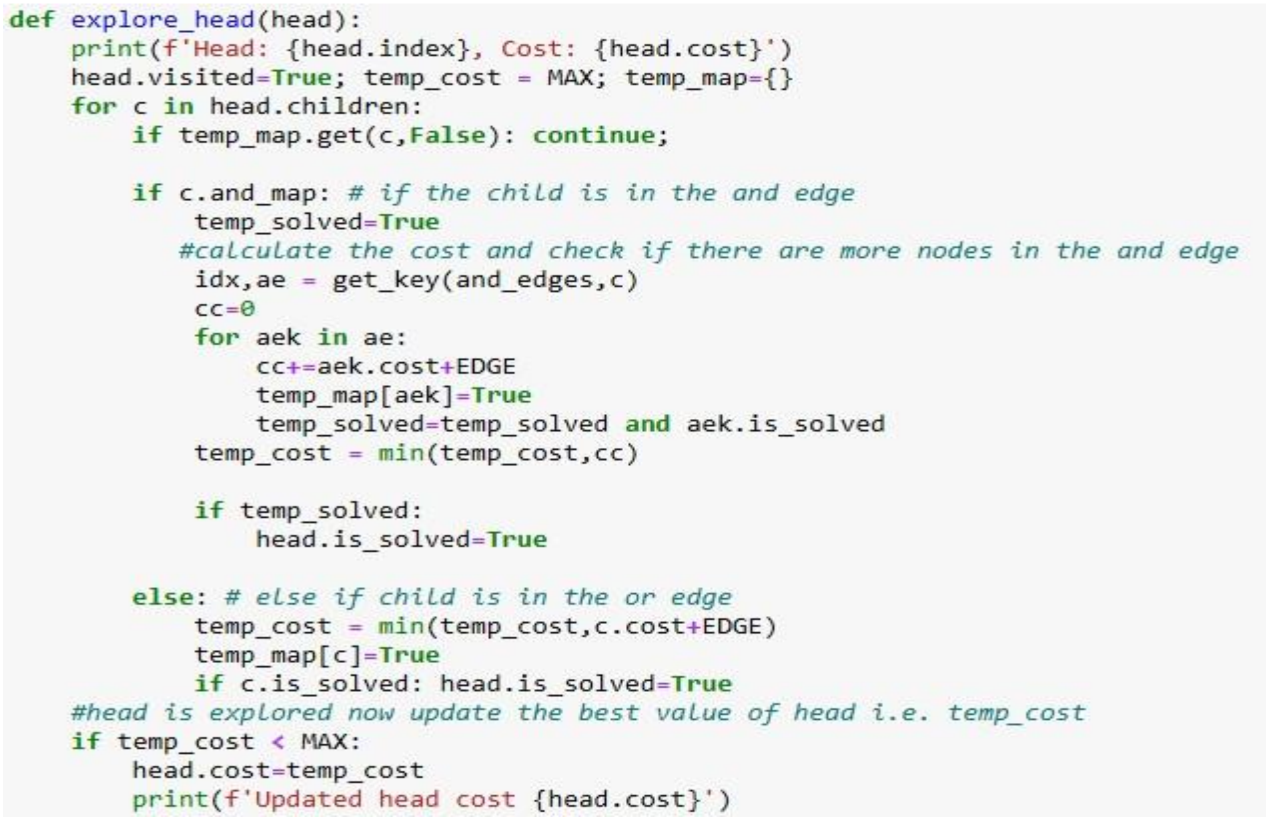
Program :

```
class Node:
    def __init__(self, index,cost,visited=False,is_solved=False,and_map=False, or_map=False):
        self.index=index
        self.cost=cost
        self.visited=visited
        self.is_solved=is_solved
        self.and_map= and_map
        self.or_map = or_map
        self.children=()

    def __str__(self):
        return f'{self.index}: {self.cost}'

    def set_children(self,ch):
        self.children=ch
```

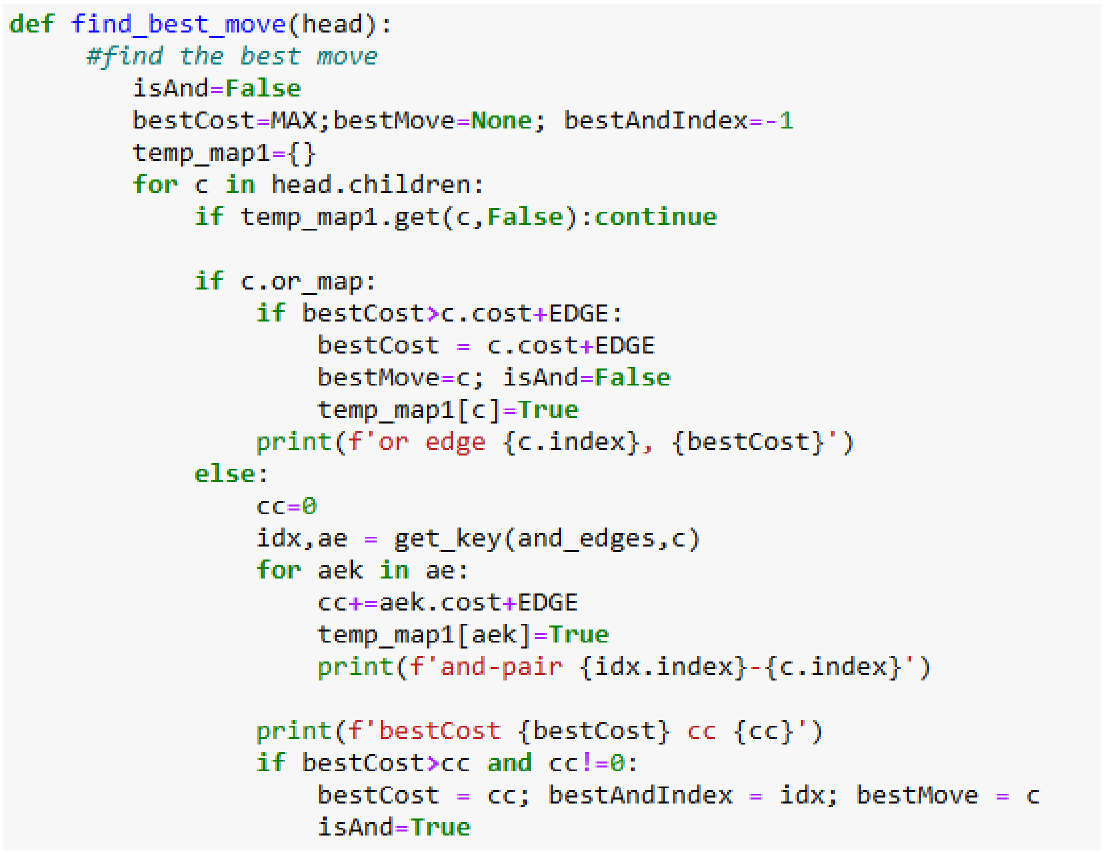
```
adj=[]
n_nodes = 21
#heuristic costs
cost=[None,0,40,2,4,1,2,3,50,60,70,80,4,5,8,9,6,7,90,90,90,90]
and_edges={}
for i in range(n_nodes+1):
    n=Node(i, cost[i])
    adj.append(n)

adj[1].set_children((adj[2],adj[3],adj[4]))
adj[2].set_children((adj[5],adj[6],adj[7]))
adj[3].set_children((adj[8],adj[9]))
adj[4].set_children((adj[10],adj[11]))
adj[5].set_children((adj[12],adj[13])); adj[6].set_children((adj[14],adj[15]))
adj[7].set_children((adj[16],adj[17])); adj[8].set_children((adj[18],))
adj[9].set_children((adj[19],)); adj[10].set_children((adj[20],)); adj[11].set_children((adj[21],))

and_edges[adj[1]] = (adj[3],adj[4])
adj[3].and_map = adj[4].and_map = True

and_edges[adj[2]] = (adj[5],adj[6], adj[7])
adj[5].and_map = adj[6].and_map = adj[7].and_map=True

for a in adj:
    if len(a.children)==0: a.is_solved=True
    if a.and_map==False: a.or_map=True
    #print(f'{a.index} and {a.and_map} or {a.or_map}')
```



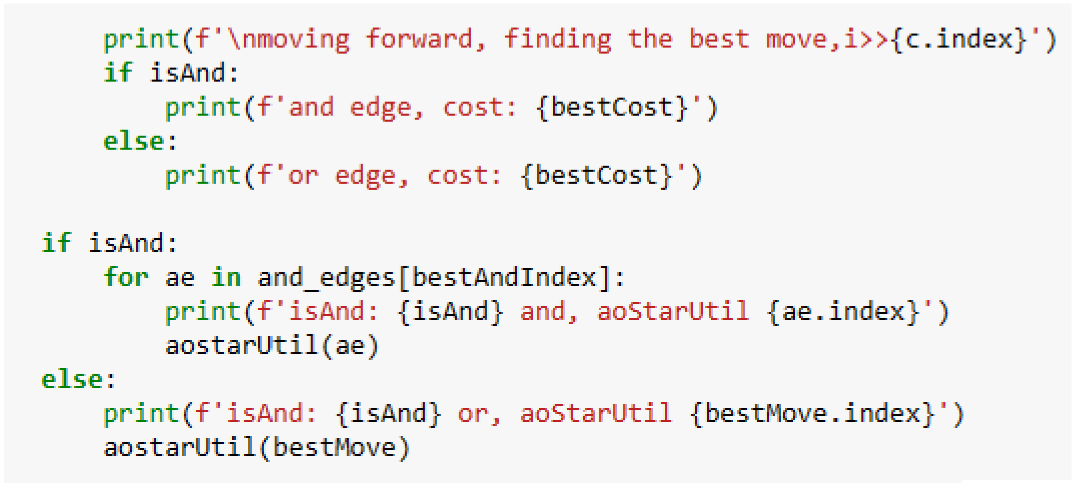
```
def get_key(and_edges, c):
    for idx, ae in and_edges.items():
        if c in ae: return idx, ae
```



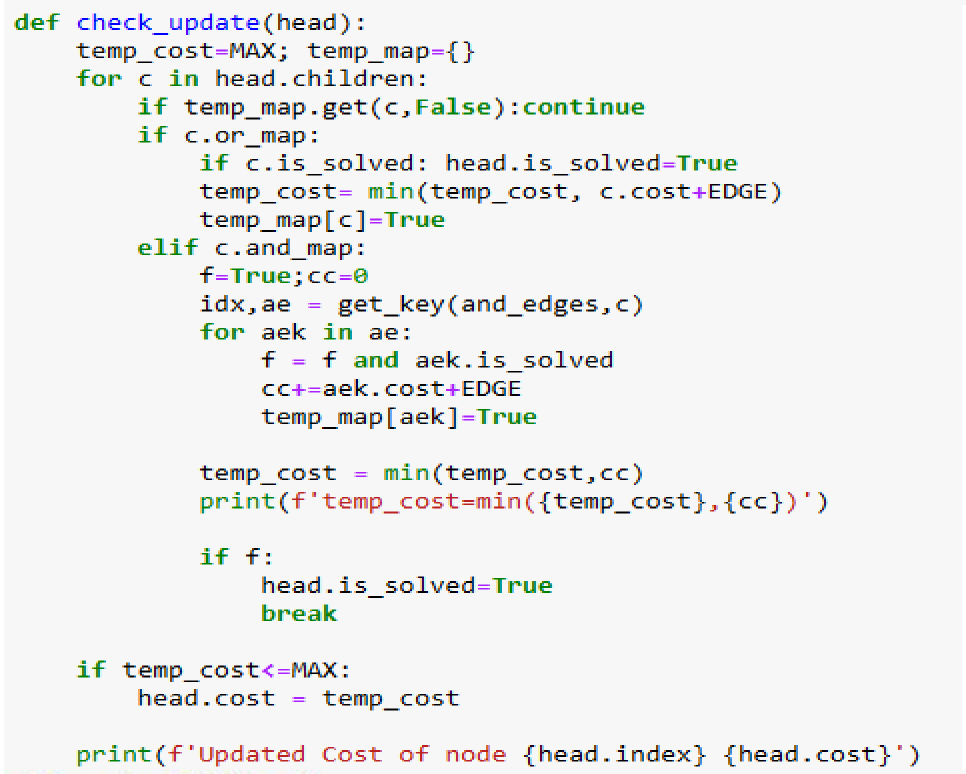
```
def explore_head(head):
    print(f'Head: {head.index}, Cost: {head.cost}')
    head.visited=True; temp_cost = MAX; temp_map={}
    for c in head.children:
        if temp_map.get(c,False): continue;

        if c.and_map: # if the child is in the and edge
            temp_solved=True
           #calculate the cost and check if there are more nodes in the and edge
            idx,ae = get_key(and_edges,c)
            cc=0
            for aek in ae:
                cc+=aek.cost+EDGE
                temp_map[aek]=True
                temp_solved=temp_solved and aek.is_solved
            temp_cost = min(temp_cost,cc)

            if temp_solved:
                head.is_solved=True

        else: # else if child is in the or edge
            temp_cost = min(temp_cost,c.cost+EDGE)
            temp_map[c]=True
            if c.is_solved: head.is_solved=True
    #head is explored now update the best value of head i.e. temp_cost
    if temp_cost < MAX:
        head.cost=temp_cost
        print(f'Updated head cost {head.cost}')
```



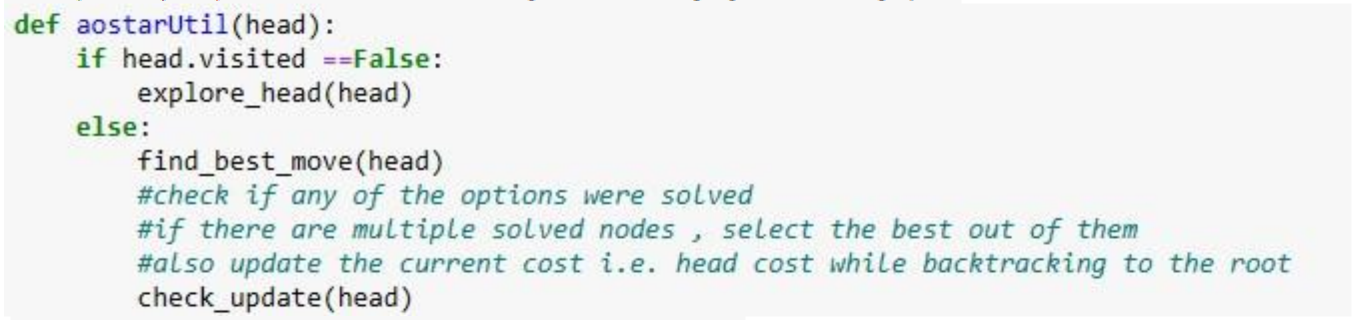
```
def find_best_move(head):
     #find the best move
        isAnd=False
        bestCost=MAX;bestMove=None; bestAndIndex=-1
        temp_map1={}
        for c in head.children:
            if temp_map1.get(c,False):continue

            if c.or_map:
                if bestCost>c.cost+EDGE:
                    bestCost = c.cost+EDGE
                    bestMove=c; isAnd=False
                    temp_map1[c]=True
                print(f'or edge {c.index}, {bestCost}')
            else:
                cc=0
                idx,ae = get_key(and_edges,c)
                for aek in ae:
                    cc+=aek.cost+EDGE
                    temp_map1[aek]=True
                    print(f'and-pair {idx.index}-{c.index}')

                print(f'bestCost {bestCost} cc {cc}')
                if bestCost>cc and cc!=0:
                    bestCost = cc; bestAndIndex = idx; bestMove = c
                    isAnd=True
```



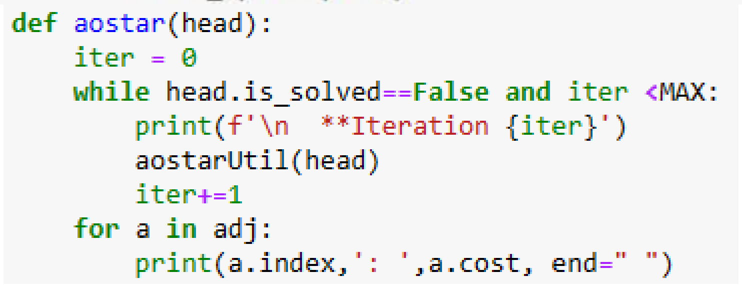
```
        print(f'\nmoving forward, finding the best move,i>>{c.index}')
        if isAnd:
            print(f'and edge, cost: {bestCost}')
        else:
            print(f'or edge, cost: {bestCost}')

    if isAnd:
        for ae in and_edges[bestAndIndex]:
            print(f'isAnd: {isAnd} and, aoStarUtil {ae.index}')
            aostarUtil(ae)
    else:
        print(f'isAnd: {isAnd} or, aoStarUtil {bestMove.index}')
        aostarUtil(bestMove)
```



```
def check_update(head):
    temp_cost=MAX; temp_map={}
    for c in head.children:
        if temp_map.get(c,False):continue
        if c.or_map:
            if c.is_solved: head.is_solved=True
            temp_cost= min(temp_cost, c.cost+EDGE)
            temp_map[c]=True
        elif c.and_map:
            f=True;cc=0
            idx,ae = get_key(and_edges,c)
            for aek in ae:
                f = f and aek.is_solved
                cc+=aek.cost+EDGE
                temp_map[aek]=True

            temp_cost = min(temp_cost,cc)
            print(f'temp_cost=min({temp_cost},{cc})')

            if f:
                head.is_solved=True
                break

    if temp_cost<=MAX:
        head.cost = temp_cost

    print(f'Updated Cost of node {head.index} {head.cost}')
```



```
def aostarUtil(head):
    if head.visited ==False:
        explore_head(head)
    else:
        find_best_move(head)
        #check if any of the options were solved
        #if there are multiple solved nodes , select the best out of them
        #also update the current cost i.e. head cost while backtracking to the root
        check_update(head)
```

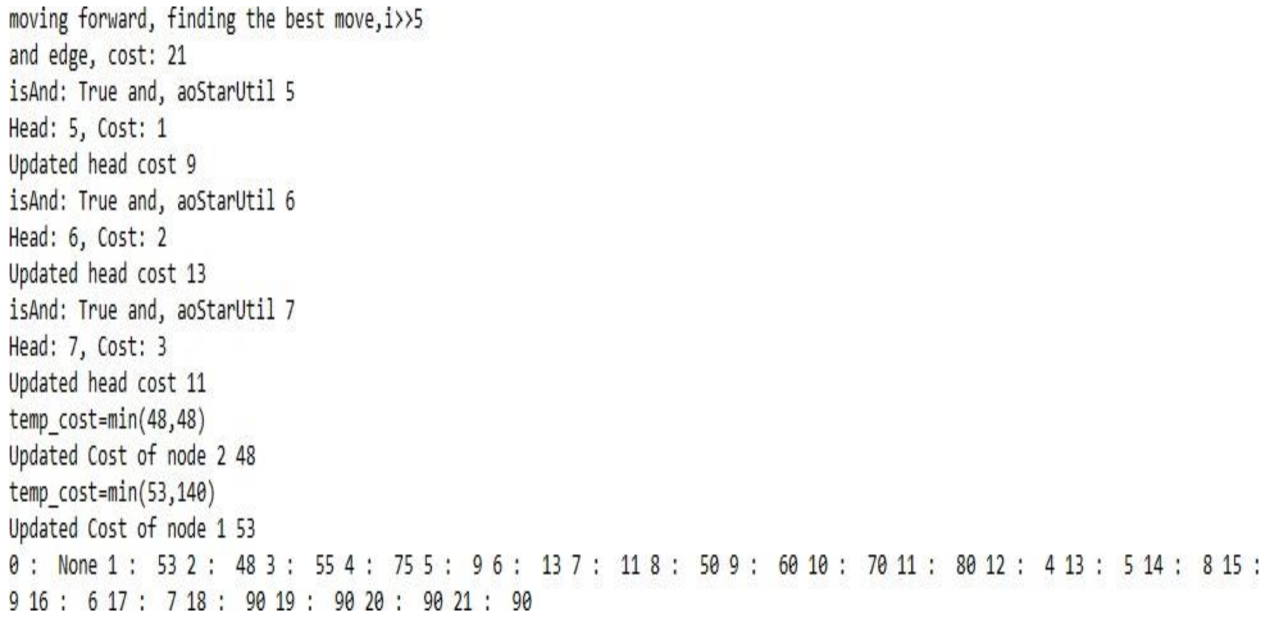


```
def aostar(head):
    iter = 0
    while head.is_solved==False and iter <MAX:
        print(f'\n  **Iteration {iter}')
        aostarUtil(head)
        iter+=1
    for a in adj:
        print(a.index,': ',a.cost, end=" ")
```

```
MAX=1000
EDGE=5 #g cost of edge
aostar(adj[1])
```

Output :

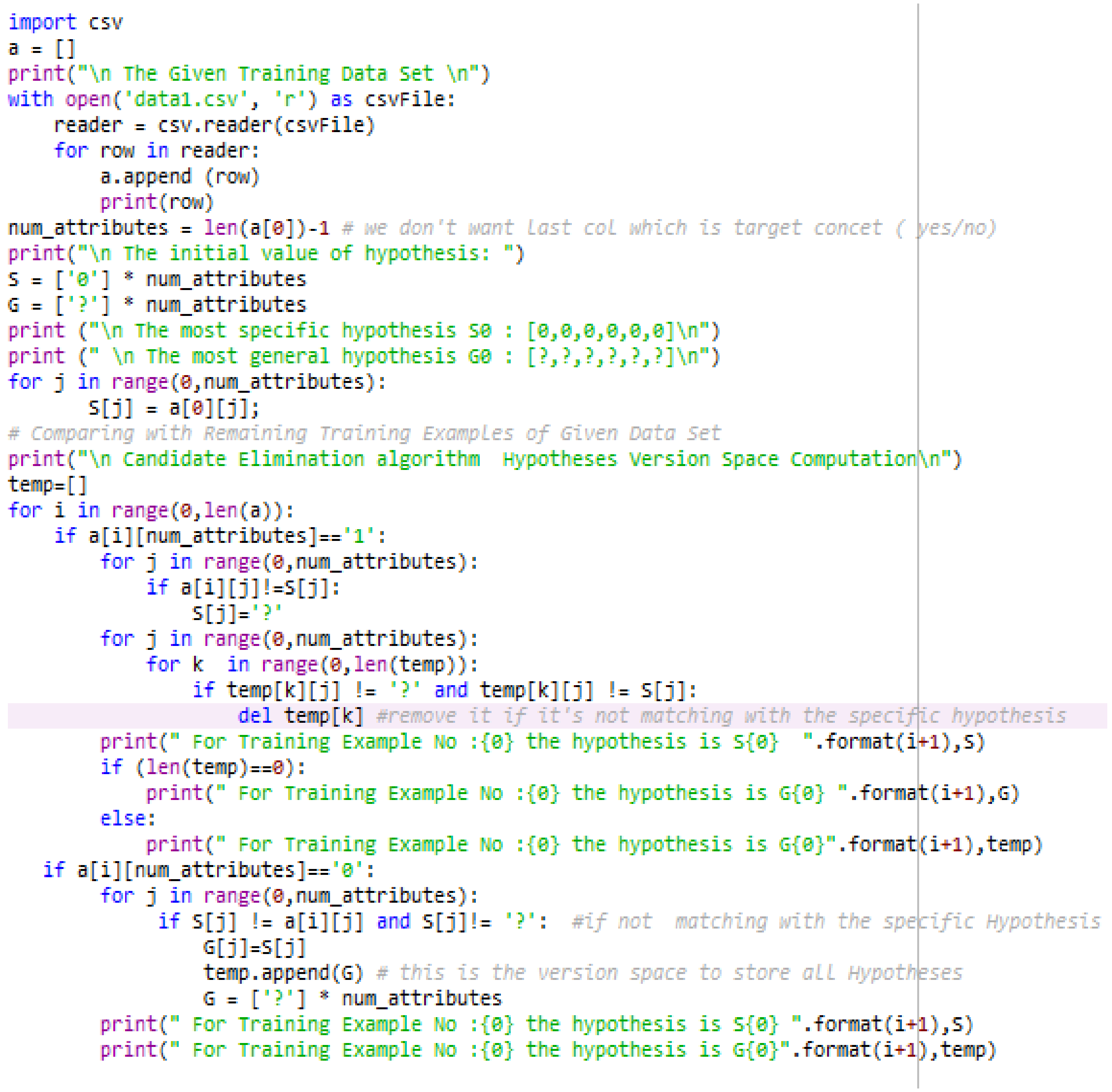
```
  **Iteration 0
Head: 1, Cost: 0
Updated head cost 16

  **Iteration 1
or edge 2, 45

moving forward, finding the best move,i>>2
or edge, cost: 45
and-pair 1-3
and-pair 1-3
bestCost 45 cc 16

moving forward, finding the best move,i>>3
and edge, cost: 16
isAnd: True and, aoStarUtil 3
Head: 3, Cost: 2
Updated head cost 55
isAnd: True and, aoStarUtil 4
Head: 4, Cost: 4
Updated head cost 75
temp_cost=min(45,140)
Updated Cost of node 1 45

  **Iteration 2
or edge 2, 45

moving forward, finding the best move,i>>2
or edge, cost: 45
and-pair 1-3
and-pair 1-3
bestCost 45 cc 140

moving forward, finding the best move,i>>3
or edge, cost: 45
isAnd: False or, aoStarUtil 2
Head: 2, Cost: 40
Updated head cost 21
temp_cost=min(26,140)
Updated Cost of node 1 26

  **Iteration 3
or edge 2, 26

moving forward, finding the best move,i>>2
or edge, cost: 26
and-pair 1-3
and-pair 1-3
bestCost 26 cc 140

moving forward, finding the best move,i>>3
or edge, cost: 26
isAnd: False or, aoStarUtil 2
and-pair 2-5
and-pair 2-5
and-pair 2-5
bestCost 1000 cc 21
```



```
moving forward, finding the best move,i>>5
and edge, cost: 21
isAnd: True and, aoStarUtil 5
Head: 5, Cost: 1
Updated head cost 9
isAnd: True and, aoStarUtil 6
Head: 6, Cost: 2
Updated head cost 13
isAnd: True and, aoStarUtil 7
Head: 7, Cost: 3
Updated head cost 11
temp_cost=min(48,48)
Updated Cost of node 2 48
temp_cost=min(53,140)
Updated Cost of node 1 53
0 :  None 1 :  53 2 :  48 3 :  55 4 :  75 5 :  9 6 :  13 7 :  11 8 :  50 9 :  60 10 :  70 11 :  80 12 :  4 13 :  5 14 :  8 15 :
9 16 :  6 17 :  7 18 :  90 19 :  90 20 :  90 21 :  90
```

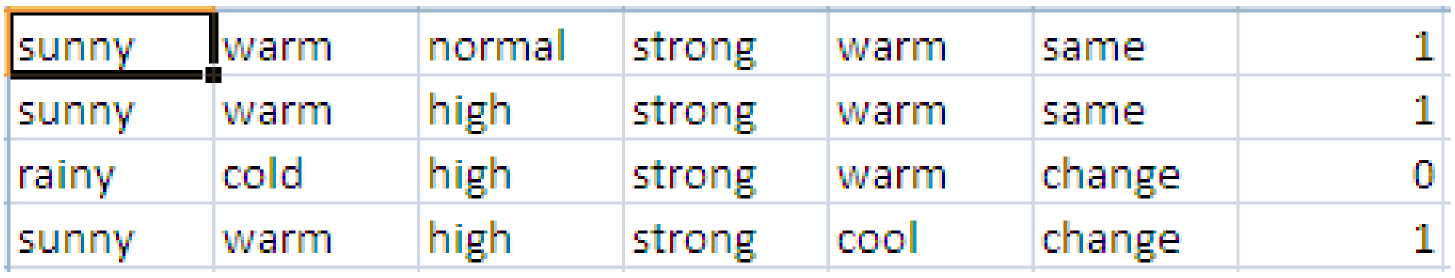
3. For a given set of training data examples stored in a .CSV file, implement and demonstrate the Candidate-Elimination algorithm to output a description of the set of all Hypotheses consistent with the training examples.

Program:



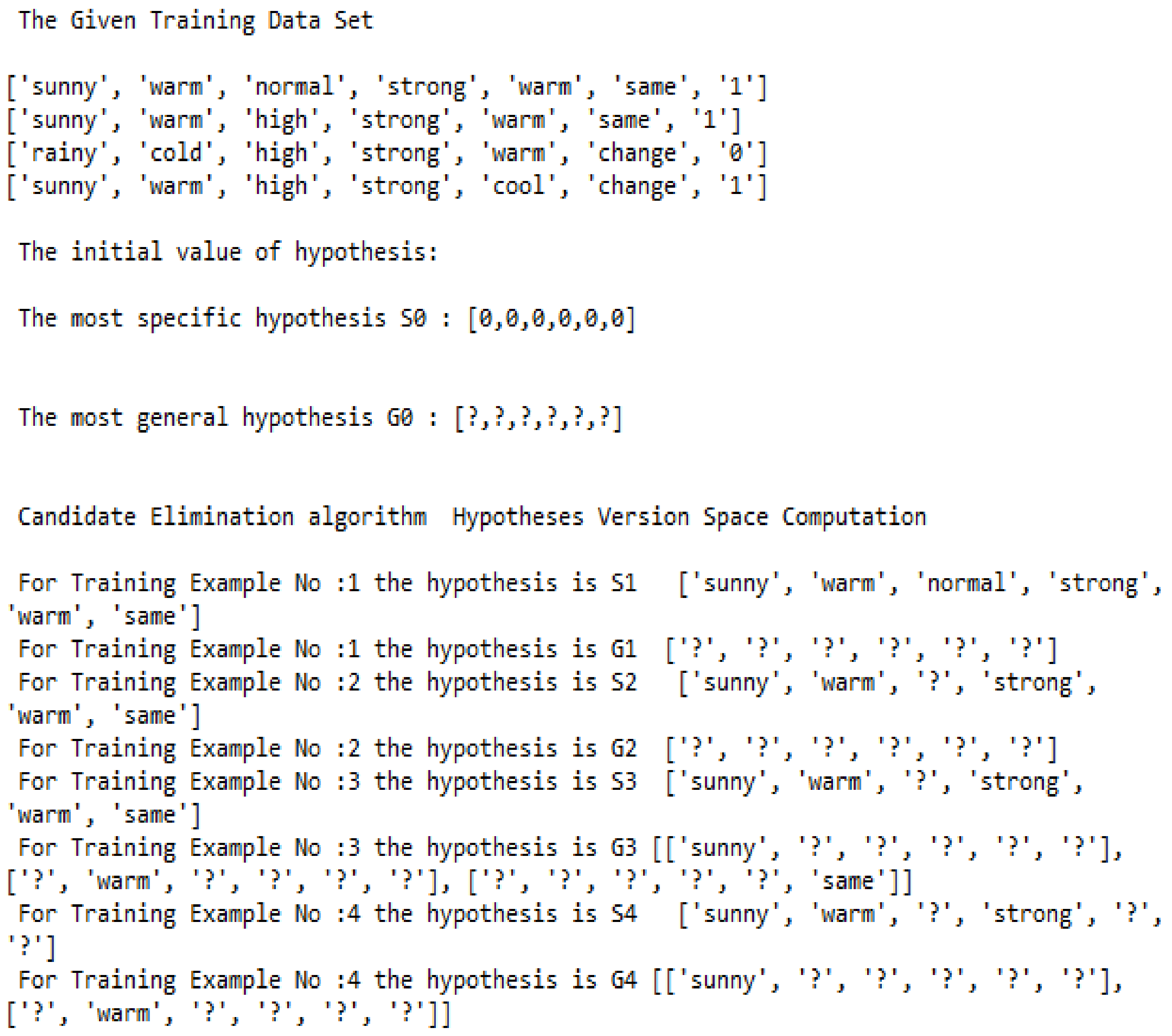
```
import csv
a = []
print("\n The Given Training Data Set \n")
with open('data1.csv', 'r') as csvFile:
    reader = csv.reader(csvFile)
    for row in reader:
        a.append (row)
        print(row)
num_attributes = len(a[0])-1 # we don't want last col which is target concet ( yes/no)
print("\n The initial value of hypothesis: ")
S = ['0'] * num_attributes
G = ['?'] * num_attributes
print ("\n The most specific hypothesis S0 : [0,0,0,0,0,0]\n")
print (" \n The most general hypothesis G0 : [?,?,?,?,?,?]\n")
for j in range(0,num_attributes):
       S[j] = a[0][j];
# Comparing with Remaining Training Examples of Given Data Set
print("\n Candidate Elimination algorithm  Hypotheses Version Space Computation\n")
temp=[]
for i in range(0,len(a)):
    if a[i][num_attributes]=='1':
        for j in range(0,num_attributes):
            if a[i][j]!=S[j]:
                S[j]='?'
        for j in range(0,num_attributes):
            for k  in range(0,len(temp)):
                if temp[k][j] != '?' and temp[k][j] != S[j]:
                    del temp[k] #remove it if it's not matching with the specific hypothesis
        print(" For Training Example No :{0} the hypothesis is S{0}  ".format(i+1),S)
        if (len(temp)==0):
            print(" For Training Example No :{0} the hypothesis is G{0} ".format(i+1),G)
        else:
            print(" For Training Example No :{0} the hypothesis is G{0}".format(i+1),temp)
   if a[i][num_attributes]=='0':
        for j in range(0,num_attributes):
             if S[j] != a[i][j] and S[j]!= '?':  #if not  matching with the specific Hypothesis
                 G[j]=S[j]
                 temp.append(G) # this is the version space to store all Hypotheses
                 G = ['?'] * num_attributes
        print(" For Training Example No :{0} the hypothesis is S{0} ".format(i+1),S)
        print(" For Training Example No :{0} the hypothesis is G{0}".format(i+1),temp)
```

Dataset to be considered is as follows:-



sunny	warm	normal	strong	warm	same	1
sunny	warm	high	strong	warm	same	1
rainy	cold	high	strong	warm	change	0
sunny	warm	high	strong	cool	change	1

The output is as follows:-



```
 The Given Training Data Set

['sunny', 'warm', 'normal', 'strong', 'warm', 'same', '1']
['sunny', 'warm', 'high', 'strong', 'warm', 'same', '1']
['rainy', 'cold', 'high', 'strong', 'warm', 'change', '0']
['sunny', 'warm', 'high', 'strong', 'cool', 'change', '1']

 The initial value of hypothesis:

 The most specific hypothesis S0 : [0,0,0,0,0,0]


 The most general hypothesis G0 : [?,?,?,?,?,?]


 Candidate Elimination algorithm  Hypotheses Version Space Computation

 For Training Example No :1 the hypothesis is S1   ['sunny', 'warm', 'normal', 'strong',
'warm', 'same']
 For Training Example No :1 the hypothesis is G1  ['?', '?', '?', '?', '?', '?']
 For Training Example No :2 the hypothesis is S2   ['sunny', 'warm', '?', 'strong',
'warm', 'same']
 For Training Example No :2 the hypothesis is G2  ['?', '?', '?', '?', '?', '?']
 For Training Example No :3 the hypothesis is S3  ['sunny', 'warm', '?', 'strong',
'warm', 'same']
 For Training Example No :3 the hypothesis is G3 [['sunny', '?', '?', '?', '?', '?'],
['?', 'warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', 'same']]
 For Training Example No :4 the hypothesis is S4   ['sunny', 'warm', '?', 'strong', '?',
'?']
 For Training Example No :4 the hypothesis is G4 [['sunny', '?', '?', '?', '?', '?'],
['?', 'warm', '?', '?', '?', '?']]
```

4. **Write a program to demonstrate the working of the decision tree based ID3 algorithm. Use an appropriate data set for building the decision tree and apply this knowledge to classify a new sample.**

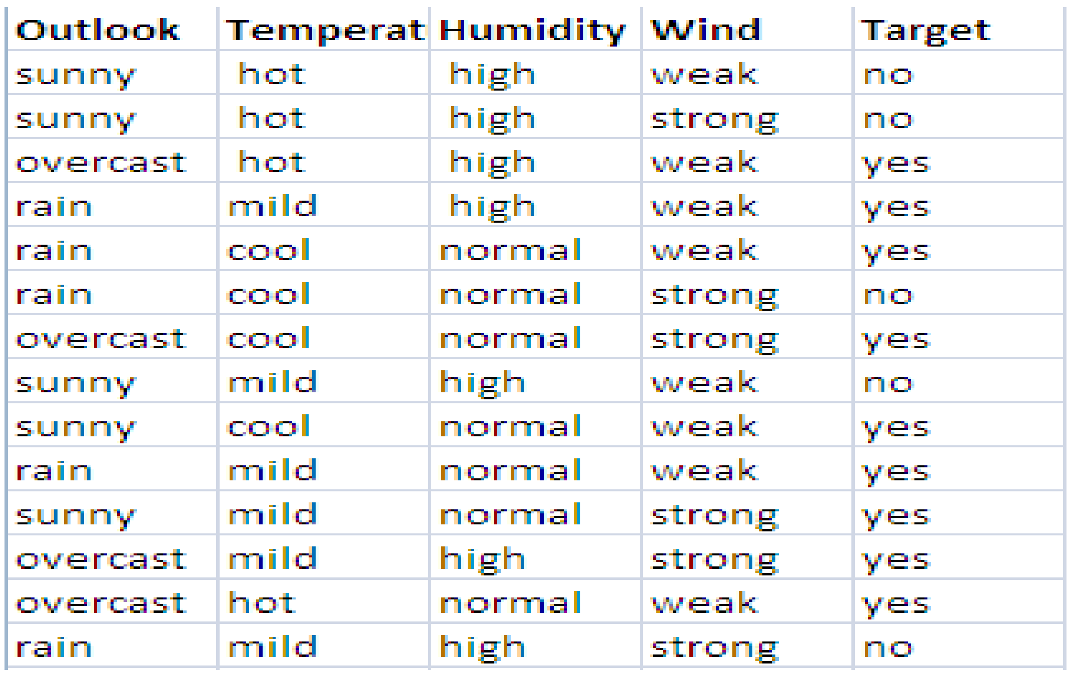
Program:

```
import pandas as pd
import math
import numpy as np

data = pd.read_csv("3_data.csv")
features = [feat for feat in data]
features.remove("Target")

class Node:
    def __init__(self):
        self.children = []
        self.value = ""
        self.isLeaf = False
        self.pred = ""

def entropy(examples):
    pos = 0.0
    neg = 0.0
    for _, row in examples.iterrows():
        if row["Target"] == "yes":
            pos += 1
        else:
            neg += 1
    if pos == 0.0 or neg == 0.0:
        return 0.0
    else:
        p = pos / (pos + neg)
        n = neg / (pos + neg)
        return -(p * math.log(p, 2) + n * math.log(n, 2))
```

```
def info_gain(examples, attr):
    uniq = np.unique(examples[attr])
    #print ("\n",uniq)
    gain = entropy(examples)
    #print ("\n",gain)
    for u in uniq:
        subdata = examples[examples[attr] == u]
        #print ("\n",subdata)
        sub_e = entropy(subdata)
        gain -= (float(len(subdata)) / float(len(examples))) * sub_e
        #print ("\n",gain)
    return gain

def ID3(examples, attrs):
    root = Node()

    max_gain = 0
    max_feat = ""
    for feature in attrs:
        #print ("\n",examples)
        gain = info_gain(examples, feature)
        if gain > max_gain:
            max_gain = gain
            max_feat = feature
    root.value = max_feat
    #print ("\nMax feature attr",max_feat)
    uniq = np.unique(examples[max_feat])
    #print ("\n",uniq)
    for u in uniq:
        #print ("\n",u)
        subdata = examples[examples[max_feat] == u]
        #print ("\n",subdata)
        if entropy(subdata) == 0.0:
            newNode = Node()
```

```
                newNode.isLeaf = True
                newNode.value = u
                newNode.pred = np.unique(subdata["Target"])
                root.children.append(newNode)
            else:
                dummyNode = Node()
                dummyNode.value = u
                new_attrs = attrs.copy()
                new_attrs.remove(max_feat)
                child = ID3(subdata, new_attrs)
                dummyNode.children.append(child)
                root.children.append(dummyNode)
    return root

def printTree(root: Node, depth=0):
    for i in range(depth):
        print("\t", end="")
    print(root.value, end="")
    if root.isLeaf:
        print(" -> ", root.pred)
    print()
    for child in root.children:
        printTree(child, depth + 1)

root = ID3(data, features)
printTree(root)
```

The dataset to be considered is as follows :-



Outlook	Temperat	Humidity	Wind	Target
sunny	hot	high	weak	no
sunny	hot	high	strong	no
overcast	hot	high	weak	yes
rain	mild	high	weak	yes
rain	cool	normal	weak	yes
rain	cool	normal	strong	no
overcast	cool	normal	strong	yes
sunny	mild	high	weak	no
sunny	cool	normal	weak	yes
rain	mild	normal	weak	yes
sunny	mild	normal	strong	yes
overcast	mild	high	strong	yes
overcast	hot	normal	weak	yes
rain	mild	high	strong	no

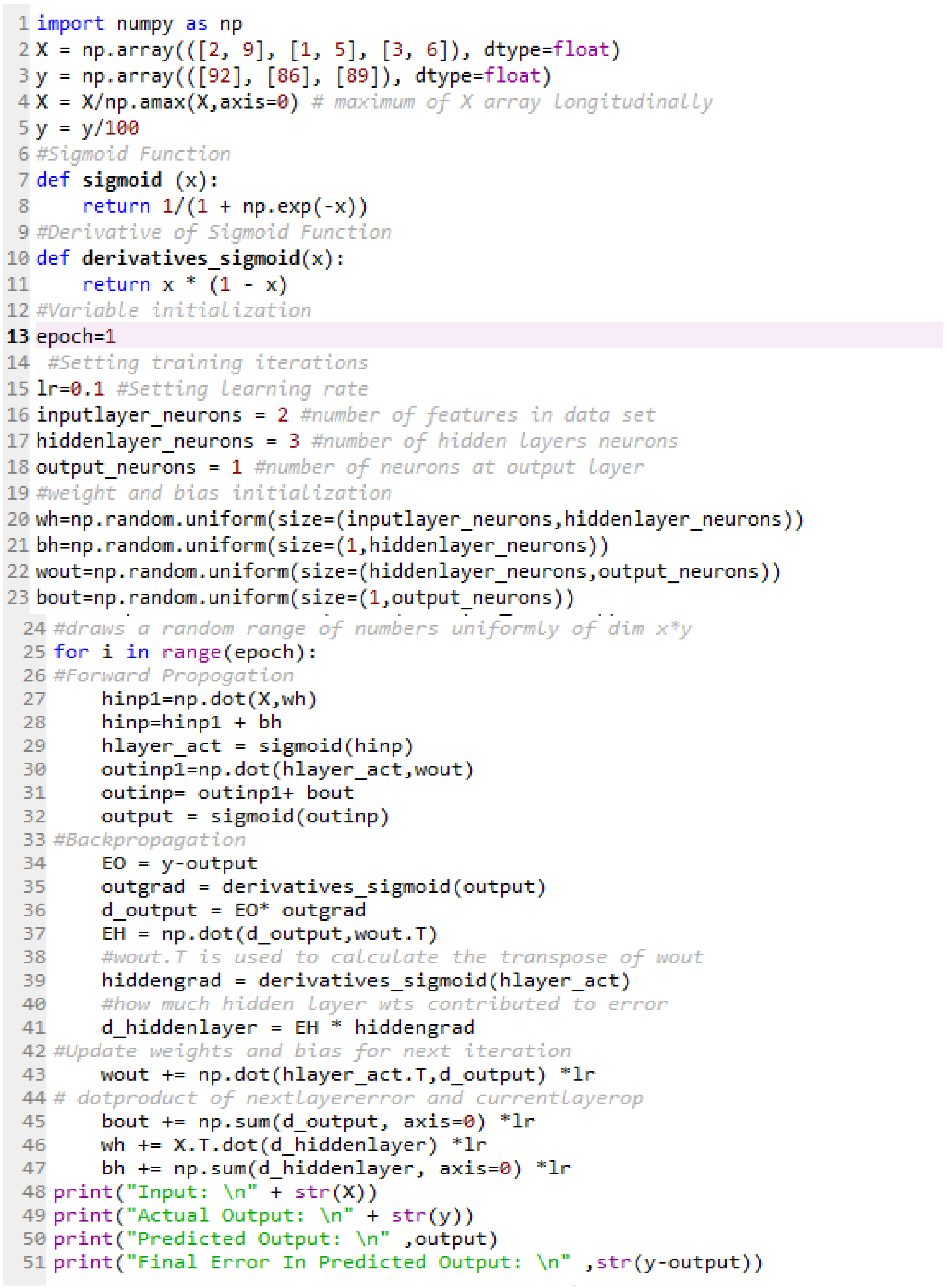
The output is:

Output:

```
The decision tree for the dataset using ID3 algorithm is

Outlook
  rain
        Wind
                strong
                        no
                weak
                        yes
  overcast
        yes

  sunny
        Humidity
                normal
                        yes
                high
                        no

The test instance: ['rain', 'cool', 'normal', 'strong']
The label for test instance:   no
```

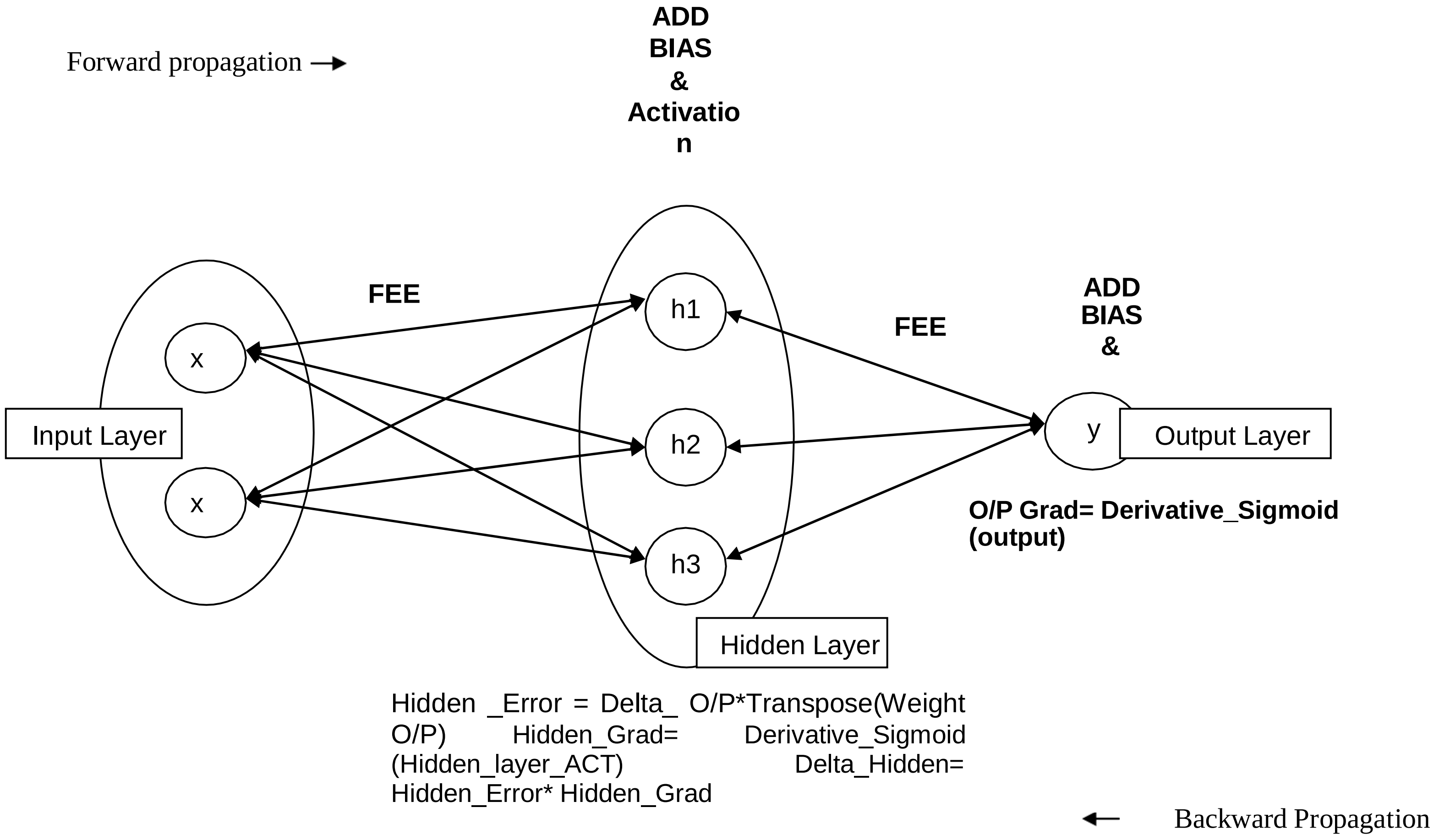
5. Build an Artificial Neural Network by implementing the Back propagation algorithm and test the same using appropriate data sets.

Program:

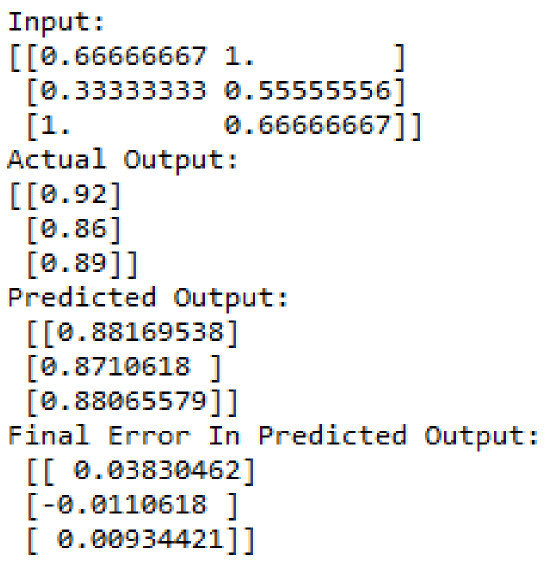


```python
import numpy as np
X = np.array(([2, 9], [1, 5], [3, 6]), dtype=float)
y = np.array(([92], [86], [89]), dtype=float)
X = X/np.amax(X,axis=0) # maximum of X array longitudinally
y = y/100
#Sigmoid Function
def sigmoid (x):
    return 1/(1 + np.exp(-x))
#Derivative of Sigmoid Function
def derivatives_sigmoid(x):
    return x * (1 - x)
#Variable initialization
epoch=1
 #Setting training iterations
lr=0.1 #Setting learning rate
inputlayer_neurons = 2 #number of features in data set
hiddenlayer_neurons = 3 #number of hidden layers neurons
output_neurons = 1 #number of neurons at output layer
#weight and bias initialization
wh=np.random.uniform(size=(inputlayer_neurons,hiddenlayer_neurons))
bh=np.random.uniform(size=(1,hiddenlayer_neurons))
wout=np.random.uniform(size=(hiddenlayer_neurons,output_neurons))
bout=np.random.uniform(size=(1,output_neurons))
#draws a random range of numbers uniformly of dim x*y
for i in range(epoch):
#Forward Propogation
    hinp1=np.dot(X,wh)
    hinp=hinp1 + bh
    hlayer_act = sigmoid(hinp)
    outinp1=np.dot(hlayer_act,wout)
    outinp= outinp1+ bout
    output = sigmoid(outinp)
#Backpropagation
    EO = y-output
    outgrad = derivatives_sigmoid(output)
    d_output = EO* outgrad
    EH = np.dot(d_output,wout.T)
    #wout.T is used to calculate the transpose of wout
    hiddengrad = derivatives_sigmoid(hlayer_act)
    #how much hidden layer wts contributed to error
    d_hiddenlayer = EH * hiddengrad
#Update weights and bias for next iteration
    wout += np.dot(hlayer_act.T,d_output) *lr
# dotproduct of nextlayererror and currentlayerop
    bout += np.sum(d_output, axis=0) *lr
    wh += X.T.dot(d_hiddenlayer) *lr
    bh += np.sum(d_hiddenlayer, axis=0) *lr
print("Input: \n" + str(X))
print("Actual Output: \n" + str(y))
print("Predicted Output: \n" ,output)
print("Final Error In Predicted Output: \n" ,str(y-output))
```

For this we consider:


Forward propagation
ADD BIAS & Activation
FEE
FEE
ADD BIAS &
x
x
Input Layer
h1
h2
h3
Hidden Layer
y
Output Layer
O/P Grad= Derivative_Sigmoid (output)
Hidden _Error = Delta_ O/P*Transpose(Weight O/P) Hidden_Grad= Derivative_Sigmoid (Hidden_layer_ACT) Delta_Hidden= Hidden_Error* Hidden_Grad
Backward Propagation

The output is as follows:



```
Input:
[[0.66666667 1.        ]
 [0.33333333 0.55555556]
 [1.         0.66666667]]
Actual Output:
[[0.92]
 [0.86]
 [0.89]]
Predicted Output:
 [[0.88169538]
 [0.8710618 ]
 [0.88065579]]
Final Error In Predicted Output:
 [[ 0.03830462]
 [-0.0110618 ]
 [ 0.00934421]]
```

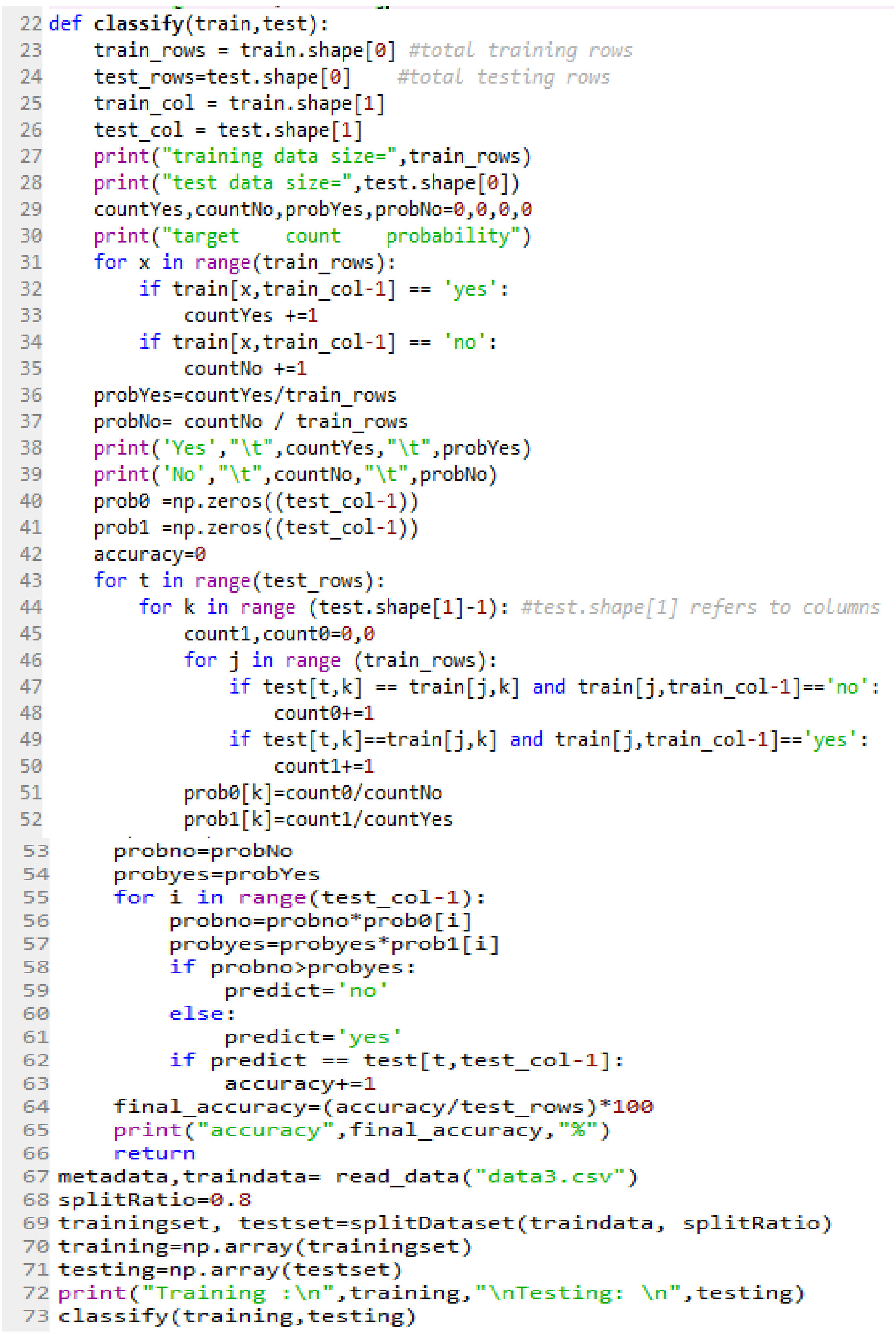
6. **Write a program to implement the naïve Bayesian classifier for a sample training data set stored as a .CSV file. Compute the accuracy of the classifier, considering few test data sets.**

<u>The dataset to be considered is as follows :-</u>

Outlook	Temperat	Humidity	Wind	Target
sunny	hot	high	weak	no
sunny	hot	high	strong	no
overcast	hot	high	weak	yes
rain	mild	high	weak	yes
rain	cool	normal	weak	yes
rain	cool	normal	strong	no
overcast	cool	normal	strong	yes
sunny	mild	high	weak	no
sunny	cool	normal	weak	yes
rain	mild	normal	weak	yes
sunny	mild	normal	strong	yes
overcast	mild	high	strong	yes
overcast	hot	normal	weak	yes
rain	mild	high	strong	no

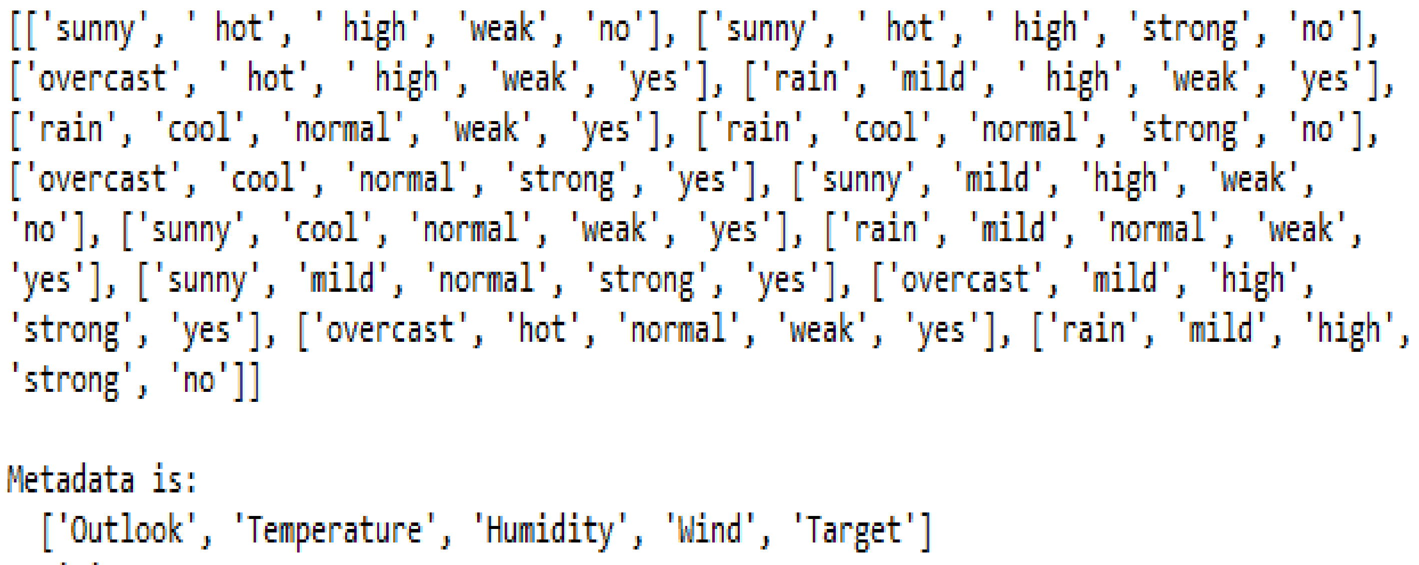
Program:

```
import numpy as np
import csv
def read_data(filename):
    with open(filename,'r') as csvfile:
        datareader = csv.reader(csvfile)
        metadata = next(datareader)
        traindata=[]
        for row in datareader:
            traindata.append(row)
        print(traindata,"\n")
        print("Metadata is:\n ",metadata)
    return (metadata, traindata)

def splitDataset(dataset, splitRatio):
    trainSize = int(len(dataset) * splitRatio)
    trainSet = []
    testset = list(dataset)
    i=0
    while len(trainSet) <trainSize:
        trainSet.append(testset.pop(i))
    return [trainSet, testset]
```

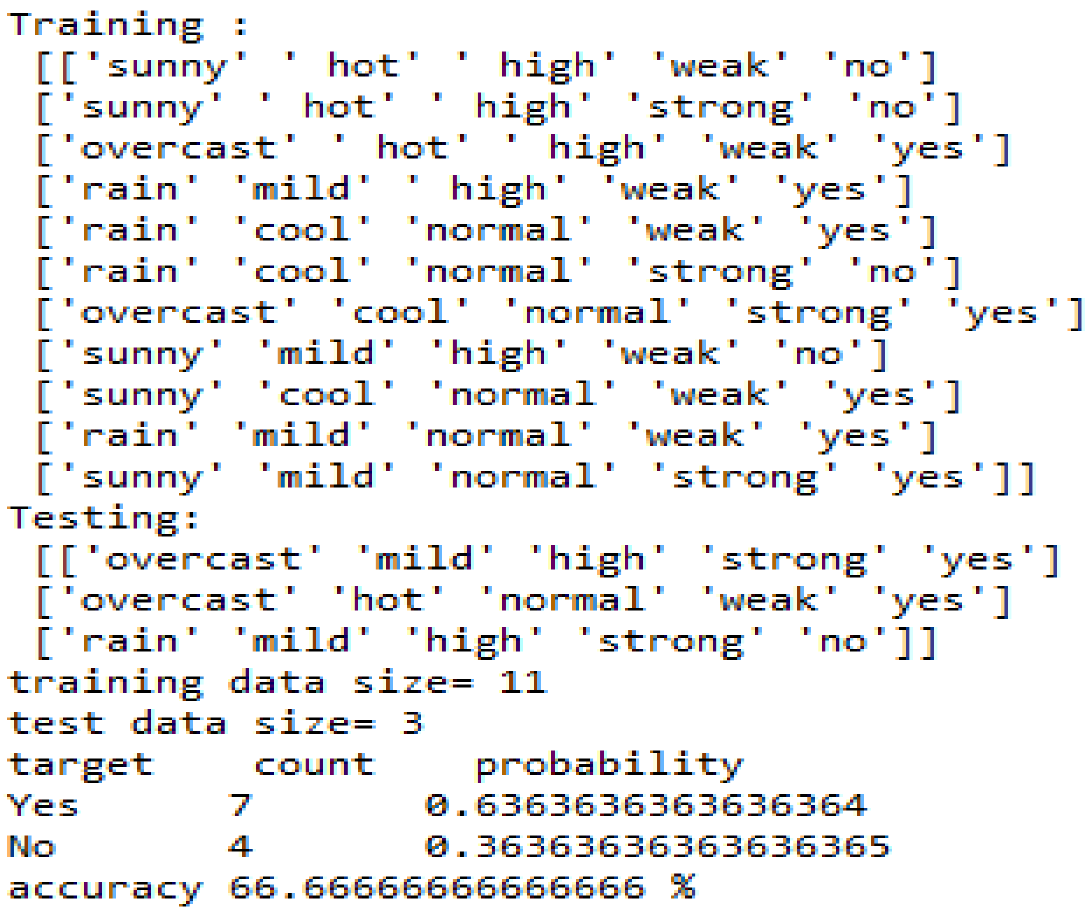


```python
def classify(train,test):
    train_rows = train.shape[0] #total training rows
    test_rows=test.shape[0]    #total testing rows
    train_col = train.shape[1]
    test_col = test.shape[1]
    print("training data size=",train_rows)
    print("test data size=",test.shape[0])
    countYes,countNo,probYes,probNo=0,0,0,0
    print("target    count    probability")
    for x in range(train_rows):
        if train[x,train_col-1] == 'yes':
            countYes +=1
        if train[x,train_col-1] == 'no':
            countNo +=1
    probYes=countYes/train_rows
    probNo= countNo / train_rows
    print('Yes',"\t",countYes,"\t",probYes)
    print('No',"\t",countNo,"\t",probNo)
    prob0 =np.zeros((test_col-1))
    prob1 =np.zeros((test_col-1))
    accuracy=0
    for t in range(test_rows):
        for k in range (test.shape[1]-1): #test.shape[1] refers to columns
            count1,count0=0,0
            for j in range (train_rows):
                if test[t,k] == train[j,k] and train[j,train_col-1]=='no':
                    count0+=1
                if test[t,k]==train[j,k] and train[j,train_col-1]=='yes':
                    count1+=1
            prob0[k]=count0/countNo
            prob1[k]=count1/countYes
    probno=probNo
    probyes=probYes
    for i in range(test_col-1):
        probno=probno*prob0[i]
        probyes=probyes*prob1[i]
        if probno>probyes:
            predict='no'
        else:
            predict='yes'
        if predict == test[t,test_col-1]:
            accuracy+=1
    final_accuracy=(accuracy/test_rows)*100
    print("accuracy",final_accuracy,"%")
    return
metadata,traindata= read_data("data3.csv")
splitRatio=0.8
trainingset, testset=splitDataset(traindata, splitRatio)
training=np.array(trainingset)
testing=np.array(testset)
print("Training :\n",training,"\nTesting: \n",testing)
classify(training,testing)
```

The output is as follows:



```
[['sunny', ' hot', ' high', 'weak', 'no'], ['sunny', ' hot', ' high', 'strong', 'no'],
['overcast', ' hot', ' high', 'weak', 'yes'], ['rain', 'mild', ' high', 'weak', 'yes'],
['rain', 'cool', 'normal', 'weak', 'yes'], ['rain', 'cool', 'normal', 'strong', 'no'],
['overcast', 'cool', 'normal', 'strong', 'yes'], ['sunny', 'mild', 'high', 'weak',
'no'], ['sunny', 'cool', 'normal', 'weak', 'yes'], ['rain', 'mild', 'normal', 'weak',
'yes'], ['sunny', 'mild', 'normal', 'strong', 'yes'], ['overcast', 'mild', 'high',
'strong', 'yes'], ['overcast', 'hot', 'normal', 'weak', 'yes'], ['rain', 'mild', 'high',
'strong', 'no']]

Metadata is:
  ['Outlook', 'Temperature', 'Humidity', 'Wind', 'Target']
```

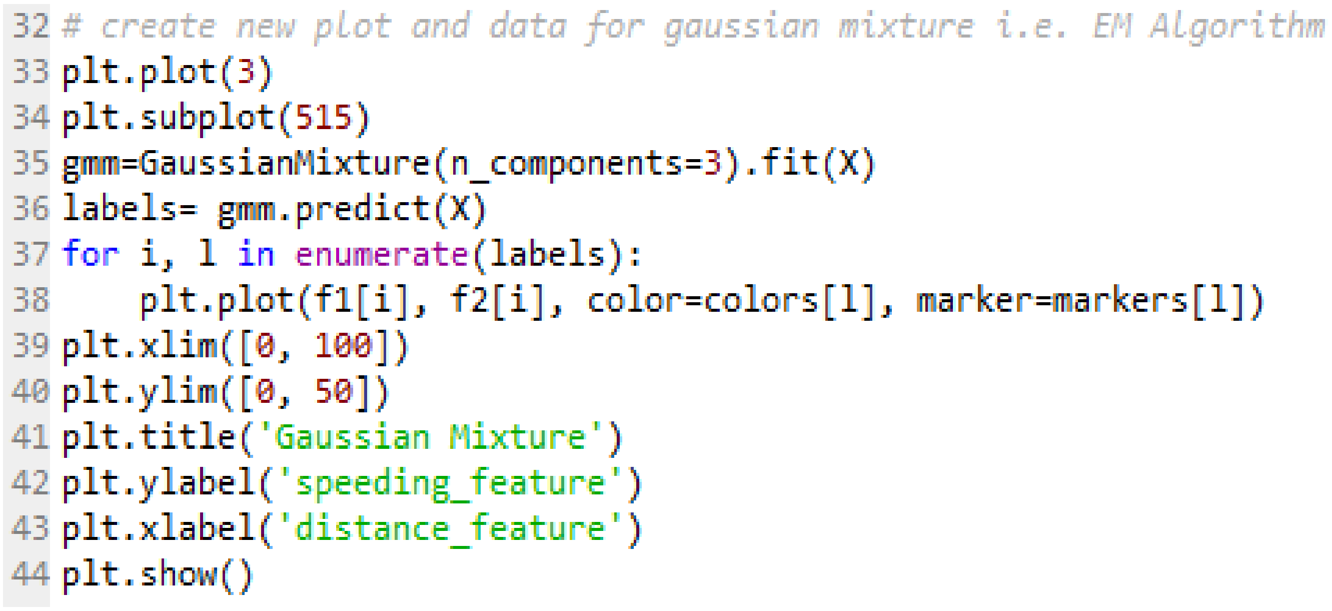


```
Training :
 [['sunny' ' hot' ' high' 'weak' 'no']
 ['sunny' ' hot' ' high' 'strong' 'no']
 ['overcast' ' hot' ' high' 'weak' 'yes']
 ['rain' 'mild' ' high' 'weak' 'yes']
 ['rain' 'cool' 'normal' 'weak' 'yes']
 ['rain' 'cool' 'normal' 'strong' 'no']
 ['overcast' 'cool' 'normal' 'strong' 'yes']
 ['sunny' 'mild' 'high' 'weak' 'no']
 ['sunny' 'cool' 'normal' 'weak' 'yes']
 ['rain' 'mild' 'normal' 'weak' 'yes']
 ['sunny' 'mild' 'normal' 'strong' 'yes']]
Testing:
 [['overcast' 'mild' 'high' 'strong' 'yes']
 ['overcast' 'hot' 'normal' 'weak' 'yes']
 ['rain' 'mild' 'high' 'strong' 'no']]
training data size= 11
test data size= 3
target    count    probability
Yes       7        0.6363636363636364
No        4        0.36363636363636365
accuracy 66.66666666666666 %
```

7. **Apply EM algorithm to cluster a set of data stored in a .CSV file. Use the same data set for clustering using *k*-Means algorithm. Compare the results of these two algorithms and comment on the quality of clustering. You can add Java/Python ML library classes/API in the program.**

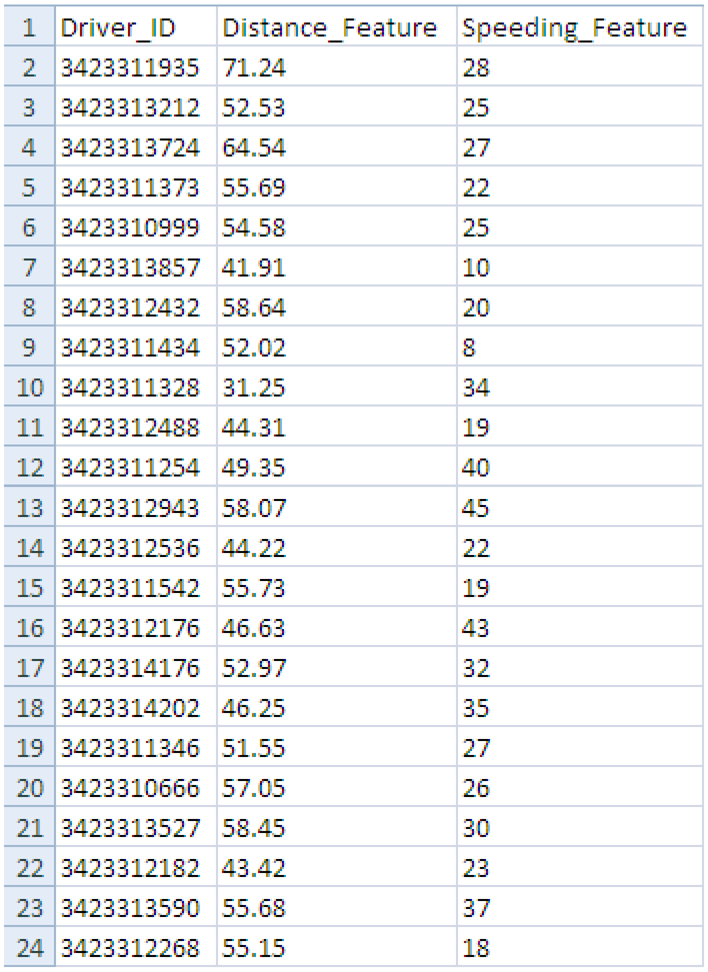
Program:

```python
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.cluster import KMeans
from sklearn.mixture import GaussianMixture
df1 = pd.read_csv("data8.csv")
print(df1)
f1 = df1['Distance_Feature'].values
f2 = df1['Speeding_Feature'].values
X = np.matrix(list(zip(f1,f2)))
plt.plot(1)
plt.subplot(511)
plt.xlim([0, 100])
plt.ylim([0, 50])
plt.title('Dataset')
plt.ylabel('speeding_feature')
plt.xlabel('distance_feature')
plt.scatter(f1,f2)
colors = ['b', 'g', 'r']
markers = ['o', 'v', 's']
# create new plot and data for K- means algorithm
plt.plot(2)
ax=plt.subplot(513)
kmeans_model = KMeans(n_clusters=3).fit(X)
for i, l in enumerate(kmeans_model.labels_):
    plt.plot(f1[i], f2[i], color=colors[l],marker=markers[l])
plt.xlim([0, 100])
plt.ylim([0, 50])
plt.title('K- Means')
plt.ylabel('speeding_feature')
plt.xlabel('distance_feature')
```



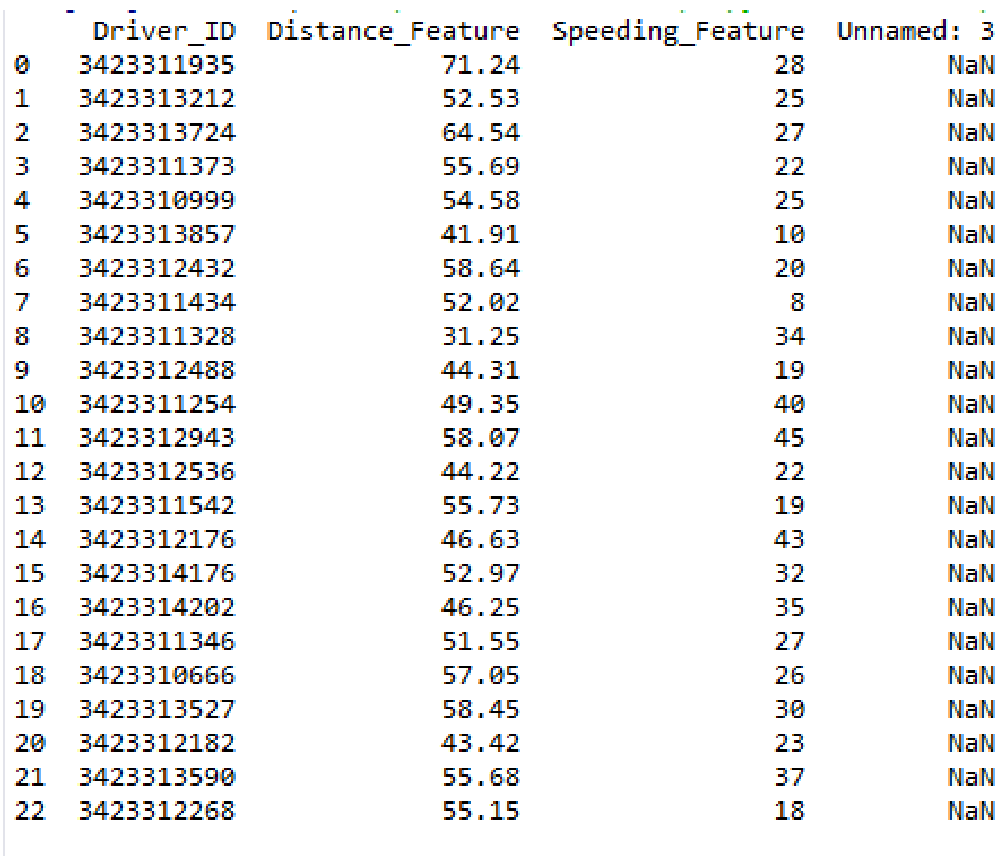
```python
# create new plot and data for gaussian mixture i.e. EM Algorithm
plt.plot(3)
plt.subplot(515)
gmm=GaussianMixture(n_components=3).fit(X)
labels= gmm.predict(X)
for i, l in enumerate(labels):
    plt.plot(f1[i], f2[i], color=colors[l], marker=markers[l])
plt.xlim([0, 100])
plt.ylim([0, 50])
plt.title('Gaussian Mixture')
plt.ylabel('speeding_feature')
plt.xlabel('distance_feature')
plt.show()
```

<u>**The dataset to be considered is:**</u>

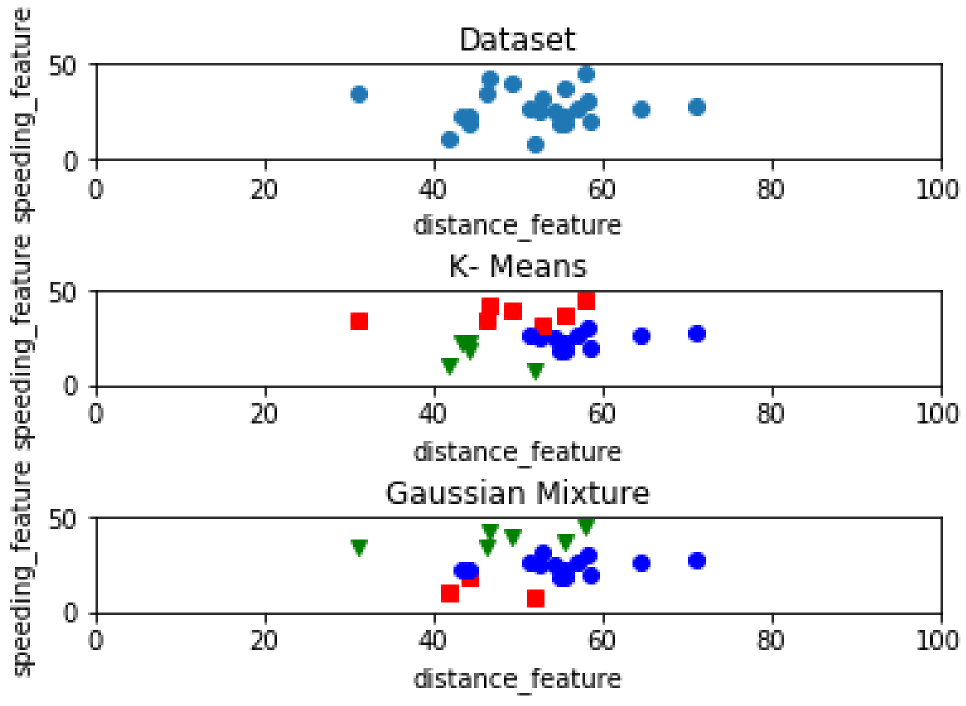


1	Driver_ID	Distance_Feature	Speeding_Feature
2	3423311935	71.24	28
3	3423313212	52.53	25
4	3423313724	64.54	27
5	3423311373	55.69	22
6	3423310999	54.58	25
7	3423313857	41.91	10
8	3423312432	58.64	20
9	3423311434	52.02	8
10	3423311328	31.25	34
11	3423312488	44.31	19
12	3423311254	49.35	40
13	3423312943	58.07	45
14	3423312536	44.22	22
15	3423311542	55.73	19
16	3423312176	46.63	43
17	3423314176	52.97	32
18	3423314202	46.25	35
19	3423311346	51.55	27
20	3423310666	57.05	26
21	3423313527	58.45	30
22	3423312182	43.42	23
23	3423313590	55.68	37
24	3423312268	55.15	18

The output is as follows:

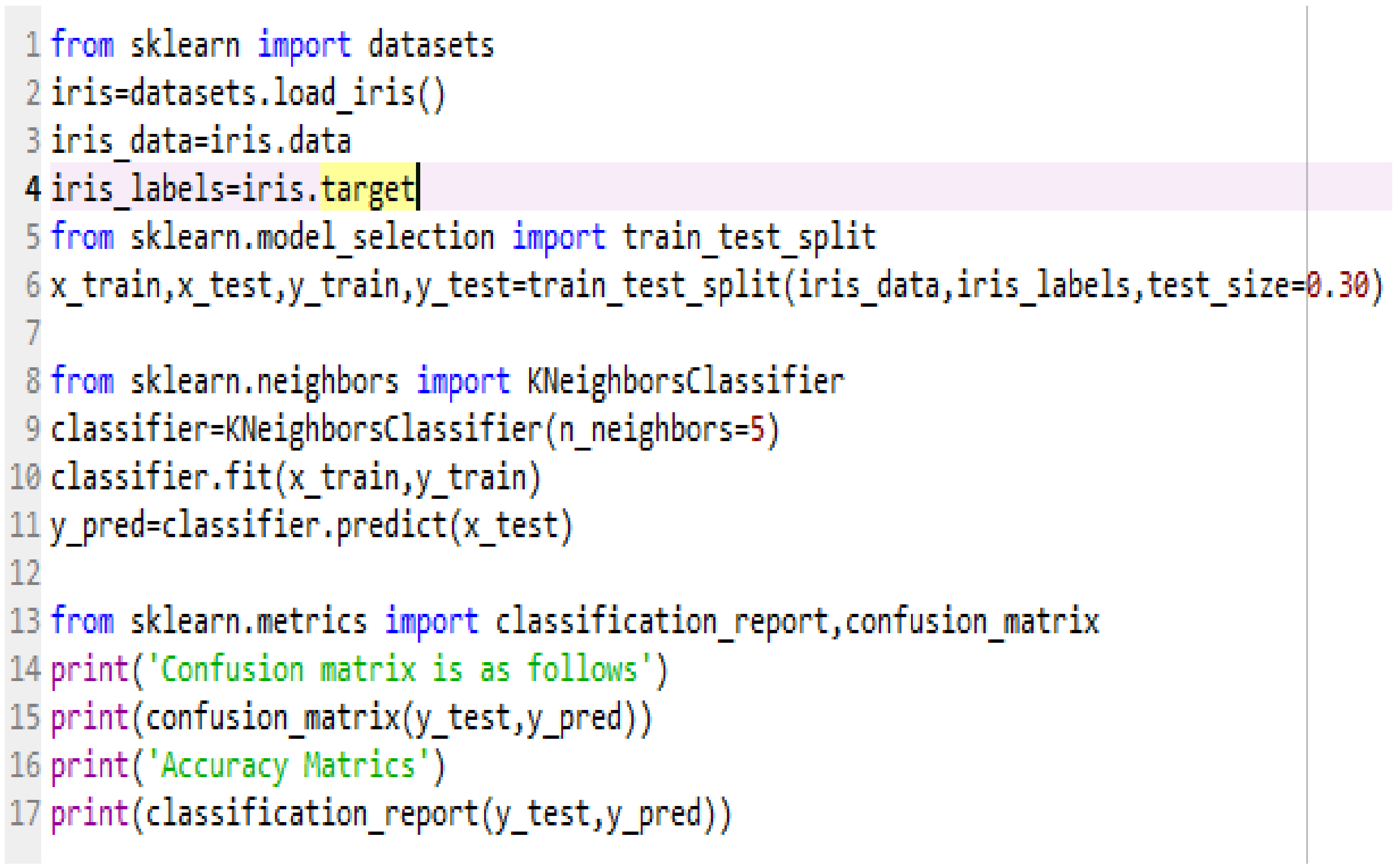


```
    Driver_ID  Distance_Feature  Speeding_Feature  Unnamed: 3
0   3423311935             71.24                28         NaN
1   3423313212             52.53                25         NaN
2   3423313724             64.54                27         NaN
3   3423311373             55.69                22         NaN
4   3423310999             54.58                25         NaN
5   3423313857             41.91                10         NaN
6   3423312432             58.64                20         NaN
7   3423311434             52.02                 8         NaN
8   3423311328             31.25                34         NaN
9   3423312488             44.31                19         NaN
10  3423311254             49.35                40         NaN
11  3423312943             58.07                45         NaN
12  3423312536             44.22                22         NaN
13  3423311542             55.73                19         NaN
14  3423312176             46.63                43         NaN
15  3423314176             52.97                32         NaN
16  3423314202             46.25                35         NaN
17  3423311346             51.55                27         NaN
18  3423310666             57.05                26         NaN
19  3423313527             58.45                30         NaN
20  3423312182             43.42                23         NaN
21  3423313590             55.68                37         NaN
22  3423312268             55.15                18         NaN
```



8. Write a program to implement *k*-Nearest Neighbor algorithm to classify the iris data set. Print both correct and wrong predictions. Java/Python ML library classes can be used for this problem.

Program:

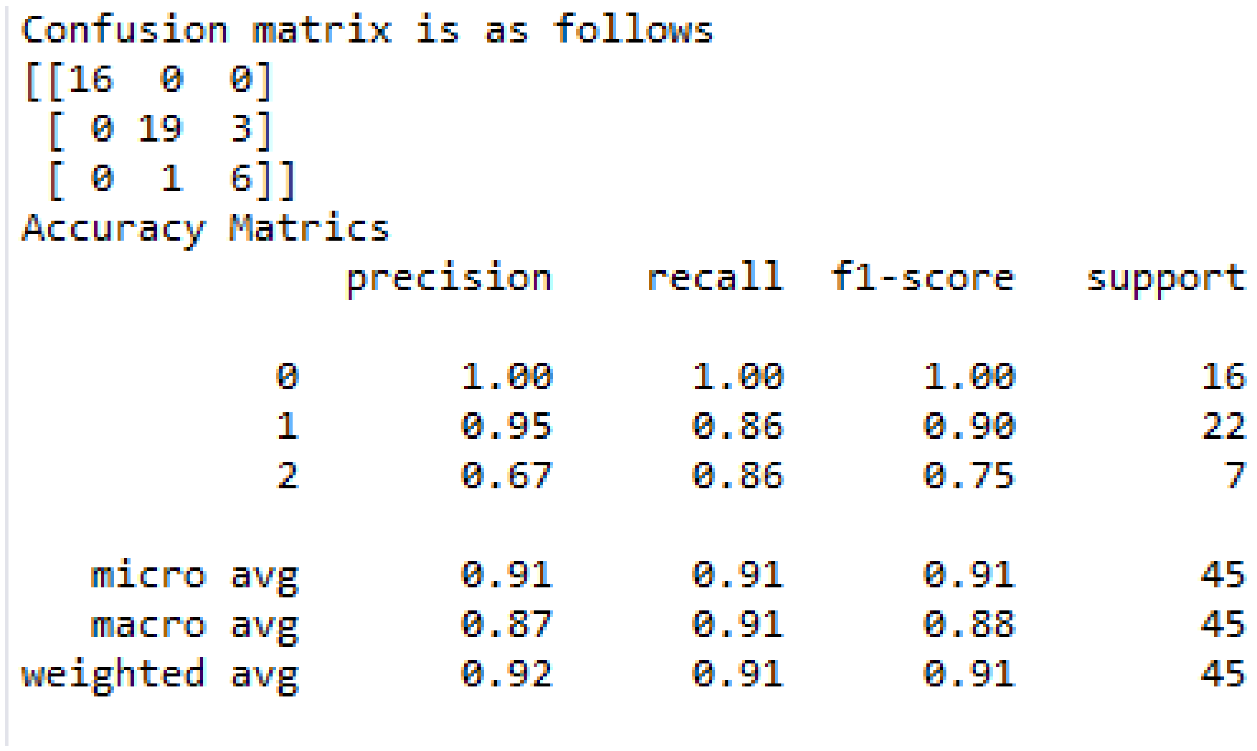


```python
from sklearn import datasets
iris=datasets.load_iris()
iris_data=iris.data
iris_labels=iris.target
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(iris_data,iris_labels,test_size=0.30)

from sklearn.neighbors import KNeighborsClassifier
classifier=KNeighborsClassifier(n_neighbors=5)
classifier.fit(x_train,y_train)
y_pred=classifier.predict(x_test)

from sklearn.metrics import classification_report,confusion_matrix
print('Confusion matrix is as follows')
print(confusion_matrix(y_test,y_pred))
print('Accuracy Matrics')
print(classification_report(y_test,y_pred))
```

<u>The data set to be considered is:</u>

Iris Flower dataset from SKLearn is imported .

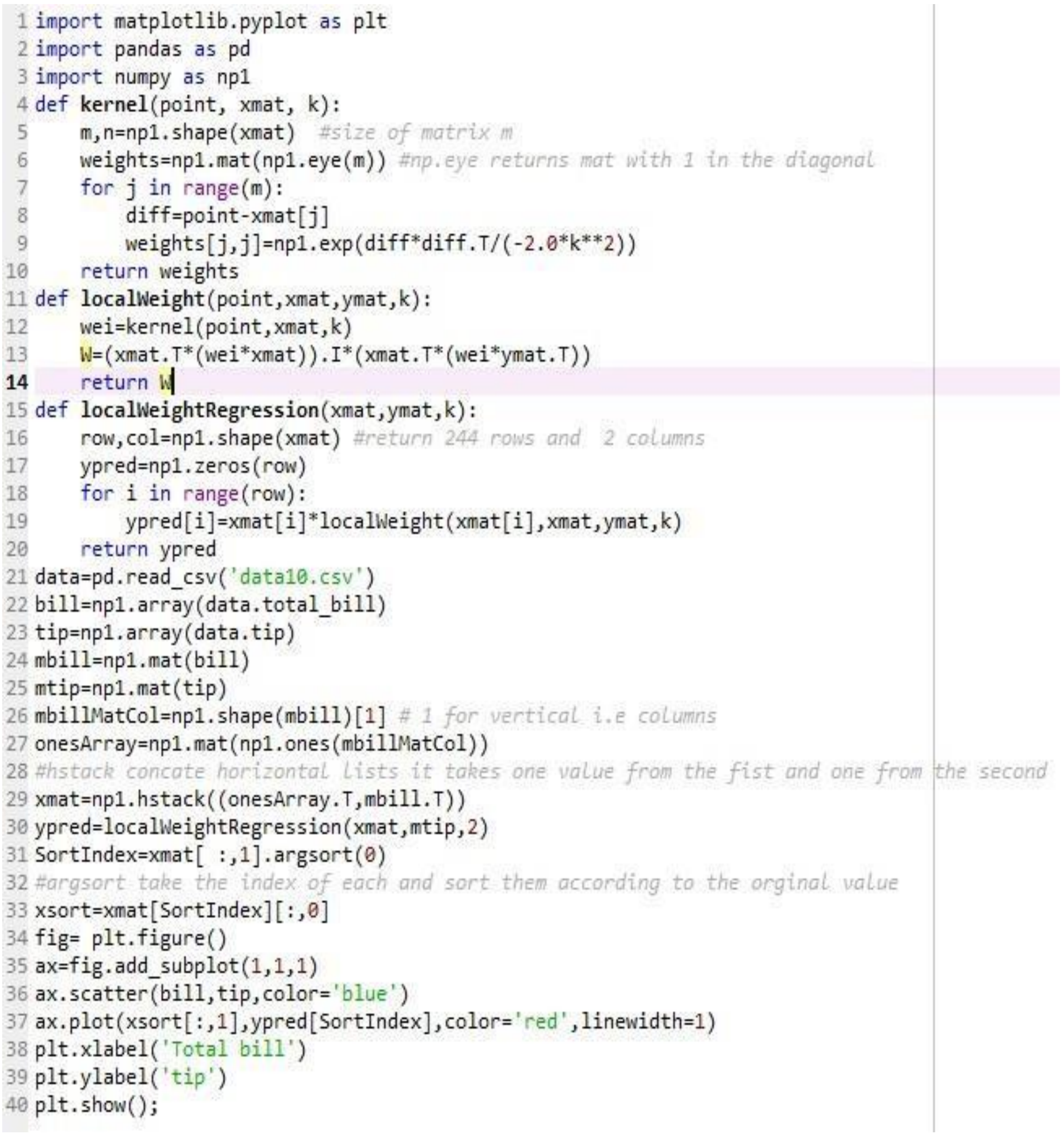
The output is as follows:



```
Confusion matrix is as follows
[[16  0  0]
 [ 0 19  3]
 [ 0  1  6]]
Accuracy Matrics
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        16
           1       0.95      0.86      0.90        22
           2       0.67      0.86      0.75         7

   micro avg       0.91      0.91      0.91        45
   macro avg       0.87      0.91      0.88        45
weighted avg       0.92      0.91      0.91        45
```

9. **Implement the non-parametric Locally Weighted Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs.**

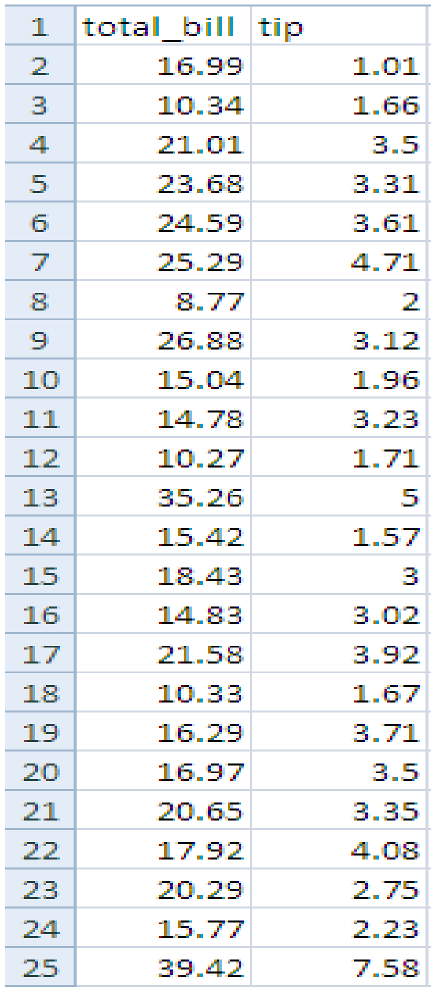
Program:



```python
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np1
def kernel(point, xmat, k):
    m,n=np1.shape(xmat)  #size of matrix m
    weights=np1.mat(np1.eye(m)) #np.eye returns mat with 1 in the diagonal
    for j in range(m):
        diff=point-xmat[j]
        weights[j,j]=np1.exp(diff*diff.T/(-2.0*k**2))
    return weights
def localWeight(point,xmat,ymat,k):
    wei=kernel(point,xmat,k)
    W=(xmat.T*(wei*xmat)).I*(xmat.T*(wei*ymat.T))
    return W
def localWeightRegression(xmat,ymat,k):
    row,col=np1.shape(xmat) #return 244 rows and  2 columns
    ypred=np1.zeros(row)
    for i in range(row):
        ypred[i]=xmat[i]*localWeight(xmat[i],xmat,ymat,k)
    return ypred
data=pd.read_csv('data10.csv')
bill=np1.array(data.total_bill)
tip=np1.array(data.tip)
mbill=np1.mat(bill)
mtip=np1.mat(tip)
mbillMatCol=np1.shape(mbill)[1] # 1 for vertical i.e columns
onesArray=np1.mat(np1.ones(mbillMatCol))
#hstack concate horizontal lists it takes one value from the fist and one from the second
xmat=np1.hstack((onesArray.T,mbill.T))
ypred=localWeightRegression(xmat,mtip,2)
SortIndex=xmat[ :,1].argsort(0)
#argsort take the index of each and sort them according to the orginal value
xsort=xmat[SortIndex][:,0]
fig= plt.figure()
ax=fig.add_subplot(1,1,1)
ax.scatter(bill,tip,color='blue')
ax.plot(xsort[:,1],ypred[SortIndex],color='red',linewidth=1)
plt.xlabel('Total bill')
plt.ylabel('tip')
plt.show();
```

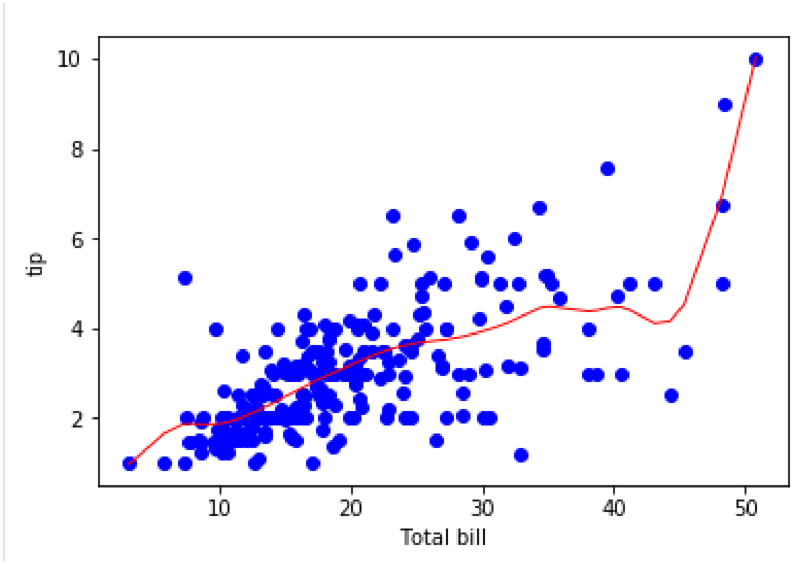
The dataset considered is:

The overall dataset consists of over 200 hypothesis values. Out of this First 24 are given below:



1	total_bill	tip
2	16.99	1.01
3	10.34	1.66
4	21.01	3.5
5	23.68	3.31
6	24.59	3.61
7	25.29	4.71
8	8.77	2
9	26.88	3.12
10	15.04	1.96
11	14.78	3.23
12	10.27	1.71
13	35.26	5
14	15.42	1.57
15	18.43	3
16	14.83	3.02
17	21.58	3.92
18	10.33	1.67
19	16.29	3.71
20	16.97	3.5
21	20.65	3.35
22	17.92	4.08
23	20.29	2.75
24	15.77	2.23
25	39.42	7.58

The output is:


tip
Total bill