**Practical 1**

**Aim:-** a) implement BFS

b) implement DFS

**Code :-**

graph = {

'5' : ['3','7'],

'3' : ['2','4'],

'7' : ['8'],

'2' : [],

'4' : ['8'],

'8' : []

}

visited = []

queue = []

def bfs(visited, graph, node):

visited.append(node)

queue.append(node)

while queue:

m = queue.pop(0)

print(m,end=" ")

for neighbour in graph[m]:

if neighbour not in visited:

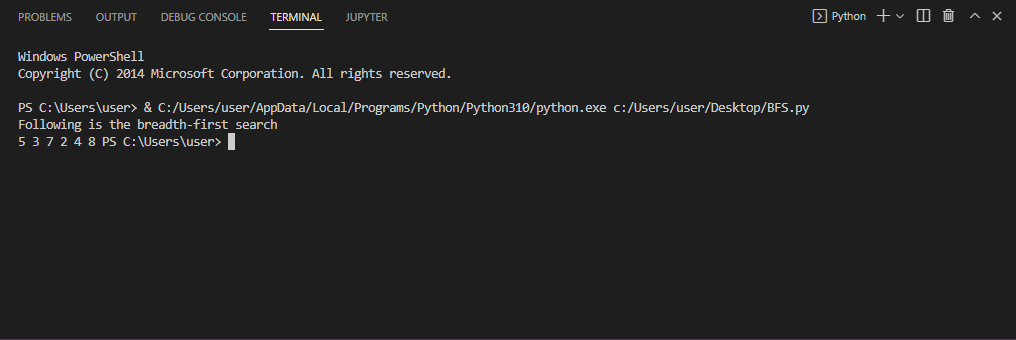
visited.append(neighbour)

queue.append(neighbour)

print("Following is the breadth-first search")

bfs(visited,graph,'5')

**Output:**



b) **DFS**

**Code**:

graph = {

'5' : ['3','7'],

'3' : ['2','4'],

'7' : ['8'],

'2' : [],

'4' : ['8'],

'8' : []

}

visited = set()

def dfs(visited,graph,node):

if node not in visited:

print(node)

visited.add(node)

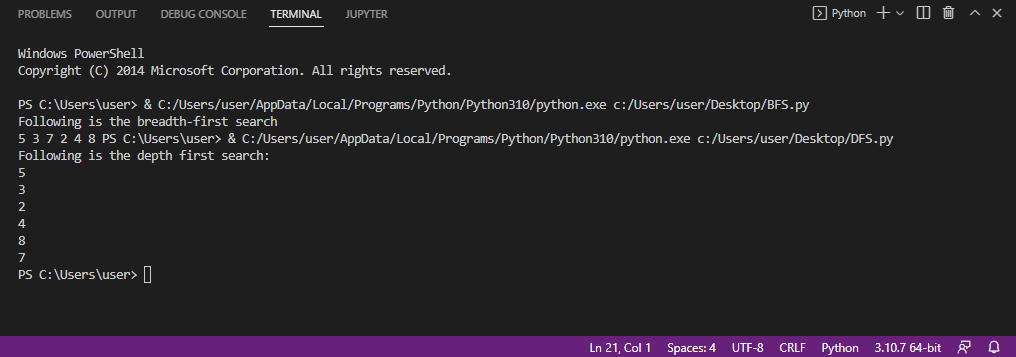
for neighbour in graph[node]:

dfs(visited,graph,neighbour)

print("Following is the depth first search:")

dfs(visited,graph,'5')

**Output:**



**RMP MODULE:**

dict\_hn={'Arad':336,'Bucharest':0,'Craiova':160,'Drobeta':242,'Eforie':161,

'Fagaras':176,'Giurgiu':77,'Hirsova':151,'Iasi':226,'Lugoj':244,

'Mehadia':241,'Neamt':234,'Oradea':380,'Pitesti':100,'Rimnicu':193,

'Sibiu':253,'Timisoara':329,'Urziceni':80,'Vaslui':199,'Zerind':374}

dict\_gn=dict(

Arad=dict(Zerind=75,Timisoara=118,Sibiu=140),

Bucharest=dict(Urziceni=85,Giurgiu=90,Pitesti=101,Fagaras=211),

Craiova=dict(Drobeta=120,Pitesti=138,Rimnicu=146),

Drobeta=dict(Mehadia=75,Craiova=120),

Eforie=dict(Hirsova=86),

Fagaras=dict(Sibiu=99,Bucharest=211),

Giurgiu=dict(Bucharest=90),

Hirsova=dict(Eforie=86,Urziceni=98),

Iasi=dict(Neamt=87,Vaslui=92),

Lugoj=dict(Mehadia=70,Timisoara=111),

Mehadia=dict(Lugoj=70,Drobeta=75),

Neamt=dict(Iasi=87),

Oradea=dict(Zerind=71,Sibiu=151),

Pitesti=dict(Rimnicu=97,Bucharest=101,Craiova=138),

Rimnicu=dict(Sibiu=80,Pitesti=97,Craiova=146),

Sibiu=dict(Rimnicu=80,Fagaras=99,Arad=140,Oradea=151),

Timisoara=dict(Lugoj=111,Arad=118),

Urziceni=dict(Bucharest=85,Hirsova=98,Vaslui=142),

Vaslui=dict(Iasi=92,Urziceni=142),

Zerind=dict(Oradea=71,Arad=75))

**Practical 2**

**Aim:**  Implement Breadth first search algorithm with RMP

**Program :-**

import queue as Q

from RMP import dict\_hn

from RMP import dict\_gn

start='Arad'

goal='Bucharest'

result=''

def BFS(city, cityq, visitedq):

global result

if city==start:

result=result+' '+city

for eachcity in dict\_gn[city].keys():

if eachcity==goal:

result=result+' '+eachcity

return

if eachcity not in cityq.queue and eachcity not in visitedq.queue:

cityq.put(eachcity)

result=result+' '+eachcity

visitedq.put(city)

BFS(cityq.get(),cityq,visitedq)

def main():

cityq=Q.Queue()

visitedq=Q.Queue()

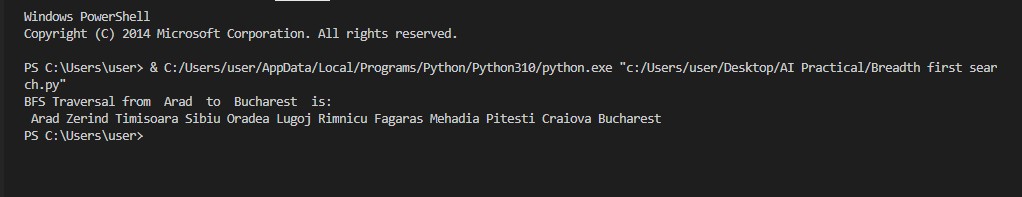
BFS(start, cityq, visitedq)

print("BFS Traversal from ",start," to ",goal," is: ")

print(result)

main()

**Output :-**

****

**Practical 3**

**Aim:-**  Implement Iterative depth first search for Romanian map problem

**Code:-**

import queue as Q

from RMP import dict\_hn

from RMP import dict\_gn

start = 'Arad'

goal = 'Bucharest'

result = ''

def DLS(city, visitedstack, startlimit, endlimit):

global result

found = 0

result = result+city+' '

visitedstack.append(city)

if city == goal:

return 1

if startlimit == endlimit:

return 0

for eachcity in dict\_gn[city].keys():

if eachcity not in visitedstack:

found = DLS(eachcity, visitedstack, startlimit+1, endlimit)

if found:

return found

def IDDFS(city, visitedstack, endlimit):

global result

for i in range(0, endlimit):

print("Searching at Limit: ", i)

found = DLS(city, visitedstack, 0, i)

if found:

print("Found")

break

else:

print("Not Found! ")

print(result)

print("-----")

result = ' '

visitedstack = []

def main():

visitedstack = []

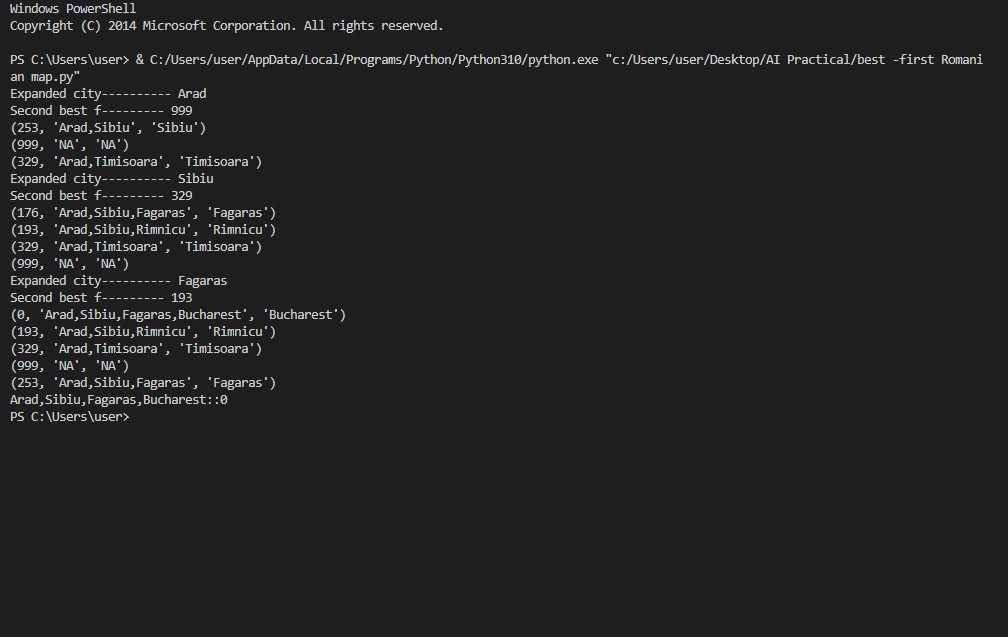
IDDFS(start, visitedstack, 9)

print("IDDFS Traversal from ", start, " to ", goal, " is: ")

print(result)

main()

**Output:**

****

**Practical 4**

**Aim:-** Implement A \* algorithm for Romanian map problem

**Code :-**

import queue as Q

from RMP import dict\_gn

from RMP import dict\_hn

start='Arad'

goal='Bucharest'

result=''

def get\_fn(citystr):

cities=citystr.split(" , ")

hn=gn=0

for ctr in range(0, len(cities)-1):

gn=gn+dict\_gn[cities[ctr]][cities[ctr+1]]

hn=dict\_hn[cities[len(cities)-1]]

return(hn+gn)

def expand(cityq):

global result

tot, citystr, thiscity=cityq.get()

if thiscity==goal:

result=citystr+" : : "+str(tot)

return

for cty in dict\_gn[thiscity]:

cityq.put((get\_fn(citystr+" , "+cty), citystr+" , "+cty, cty))

expand(cityq)

def main():

cityq=Q.PriorityQueue()

thiscity=start

cityq.put((get\_fn(start),start,thiscity))

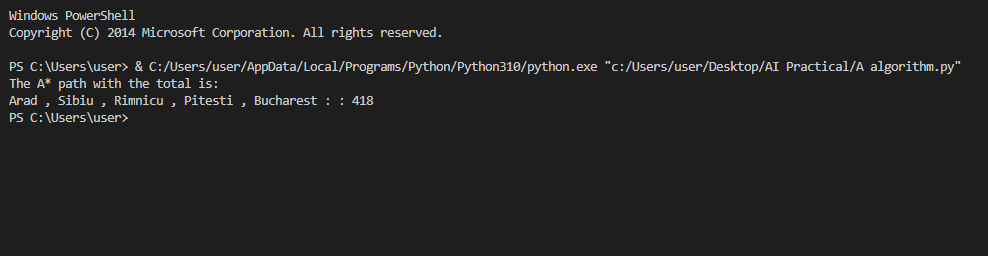
expand(cityq)

print("The A\* path with the total is: ")

print(result)

main()

**Output:-**



**Practical 5**

**Aim :-** Implement recursive best -first search algorithm for Romanian map problem

**Code** :-

import queue as Q

from RMP import dict\_gn

from RMP import dict\_hn

start='Arad'

goal='Bucharest'

result=''

def get\_fn(citystr):

cities=citystr.split(',')

hn=gn=0

hn=dict\_hn[cities[len(cities)-1]]

return(hn+gn)

def printout(cityq):

for i in range(0,cityq.qsize()):

print(cityq.queue[i])

def expand(cityq):

global result

tot,citystr,thiscity=cityq.get()

nexttot=999

if not cityq.empty():

nexttot,nextcitystr,nextthiscity=cityq.queue[0]

if thiscity==goal and tot<nexttot:

result=citystr+'::'+str(tot)

return

print("Expanded city----------",thiscity)

print("Second best f---------",nexttot)

tempq=Q.PriorityQueue()

for cty in dict\_gn[thiscity]:

tempq.put((get\_fn(citystr+','+cty),citystr+','+cty,cty))

for ctr in range(1,3):

ctrtot,ctrcitystr,ctrthiscity=tempq.get()

if ctrtot<nexttot:

cityq.put((ctrtot,ctrcitystr,ctrthiscity))

else:

cityq.put((ctrtot,citystr,thiscity))

break

printout(cityq)

expand(cityq)

def main():

cityq=Q.PriorityQueue()

thiscity=start

cityq.put((999,"NA","NA"))

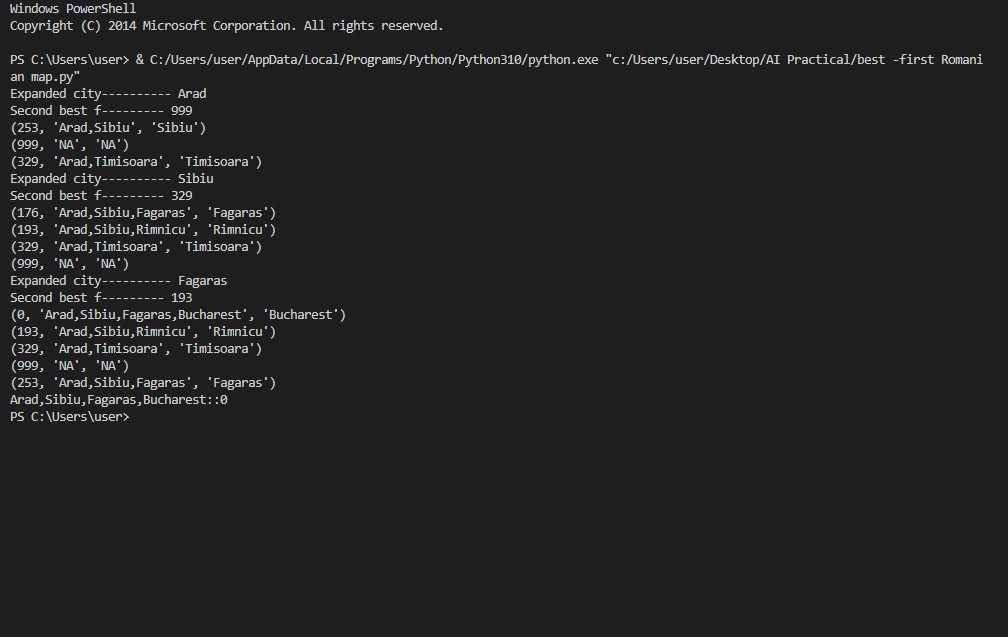
cityq.put((get\_fn(start),start,thiscity))

expand(cityq)

print(result)

main()

**Output:-**



**RWP Module**

rwp\_examples = dict(

x1=dict(Alt='Y', Bar='N', Fri='N',Hun='Y',Pat='S',Price='$$$',Rain='N',Res='Y',Type='F',Est='0-10',ans='Y'),

x2=dict(Alt='Y', Bar='N', Fri='N',Hun='Y',Pat='F',Price='$',Rain='N',Res='N',Type='T',Est='30-60',ans='N'),

x3=dict(Alt='N', Bar='Y', Fri='N',Hun='N',Pat='S',Price='$',Rain='N',Res='N',Type='B',Est='0-10',ans='Y'),

x4=dict(Alt='Y', Bar='N', Fri='Y',Hun='Y',Pat='F',Price='$',Rain='Y',Res='N',Type='T',Est='10-30',ans='Y'),

x5=dict(Alt='Y', Bar='N', Fri='Y',Hun='N',Pat='F',Price='$$$',Rain='N',Res='Y',Type='F',Est='>60',ans='N'),

x6=dict(Alt='N', Bar='Y', Fri='N',Hun='Y',Pat='S',Price='$$',Rain='Y',Res='Y',Type='I',Est='0-10',ans='Y'),

x7=dict(Alt='N', Bar='Y', Fri='N',Hun='N',Pat='N',Price='$',Rain='Y',Res='N',Type='B',Est='0-10',ans='N'),

x8=dict(Alt='N', Bar='N', Fri='N',Hun='Y',Pat='S',Price='$$',Rain='Y',Res='Y',Type='T',Est='0-10',ans='Y'),

x9=dict(Alt='N', Bar='Y', Fri='Y',Hun='N',Pat='F',Price='$',Rain='Y',Res='N',Type='B',Est='>60',ans='N'),

x10=dict(Alt='Y', Bar='Y', Fri='Y',Hun='Y',Pat='F',Price='$$$',Rain='N',Res='Y',Type='I',Est='10-30',ans='N'),

x11=dict(Alt='N', Bar='N', Fri='N',Hun='N',Pat='N',Price='$',Rain='N',Res='N',Type='T',Est='0-10',ans='N'),

x12=dict(Alt='Y', Bar='Y', Fri='Y',Hun='Y',Pat='F',Price='$',Rain='N',Res='N',Type='B',Est='0-10',ans='Y')

)

**Practical 6**

**Aim:-** Implement Naïve-Bayes Learning algorithm for RWP

**Code:-**

rwp\_examples = dict(

x1=dict(Alt='Y', Bar='N', Fri='N',Hun='Y',Pat='S',Price='$$$',Rain='N',Res='Y',Type='F',Est='0-10',ans='Y'),

x2=dict(Alt='Y', Bar='N', Fri='N',Hun='Y',Pat='F',Price='$',Rain='N',Res='N',Type='T',Est='30-60',ans='N'),

x3=dict(Alt='N', Bar='Y', Fri='N',Hun='N',Pat='S',Price='$',Rain='N',Res='N',Type='B',Est='0-10',ans='Y'),

x4=dict(Alt='Y', Bar='N', Fri='Y',Hun='Y',Pat='F',Price='$',Rain='Y',Res='N',Type='T',Est='10-30',ans='Y'),

x5=dict(Alt='Y', Bar='N', Fri='Y',Hun='N',Pat='F',Price='$$$',Rain='N',Res='Y',Type='F',Est='>60',ans='N'),

x6=dict(Alt='N', Bar='Y', Fri='N',Hun='Y',Pat='S',Price='$$',Rain='Y',Res='Y',Type='I',Est='0-10',ans='Y'),

x7=dict(Alt='N', Bar='Y', Fri='N',Hun='N',Pat='N',Price='$',Rain='Y',Res='N',Type='B',Est='0-10',ans='N'),

x8=dict(Alt='N', Bar='N', Fri='N',Hun='Y',Pat='S',Price='$$',Rain='Y',Res='Y',Type='T',Est='0-10',ans='Y'),

x9=dict(Alt='N', Bar='Y', Fri='Y',Hun='N',Pat='F',Price='$',Rain='Y',Res='N',Type='B',Est='>60',ans='N'),

x10=dict(Alt='Y', Bar='Y', Fri='Y',Hun='Y',Pat='F',Price='$$$',Rain='N',Res='Y',Type='I',Est='10-30',ans='N'),

x11=dict(Alt='N', Bar='N', Fri='N',Hun='N',Pat='N',Price='$',Rain='N',Res='N',Type='T',Est='0-10',ans='N'),

x12=dict(Alt='Y', Bar='Y', Fri='Y',Hun='Y',Pat='F',Price='$',Rain='N',Res='N',Type='B',Est='0-10',ans='Y')

)

from rwp import rwp\_examples

total\_exp = 12

def tot(attribute, value):

count = 0

for key, val in rwp\_examples.items():

for key1, val1 in val.items():

if key1 == attribute:

if val1 == value:

count += 1

return count

def getProbab(attribute, attribval, value):

count = 0

for key, val in rwp\_examples.items():

val1 = rwp\_examples[key][attribute]

val2 = rwp\_examples[key]['ans']

if val1 == attribval and val2 == value:

count += 1

probab=count / tot('ans', value)

return probab

def main():

PAItYes=tot('Alt', 'Y') / total\_exp

PAItNo=tot('Alt', 'N') / total\_exp

PBarYes=tot('Bar', 'Y') / total\_exp

PBarNo=tot('Bar', 'N') / total\_exp

PFriYes=tot('Fri', 'Y') / total\_exp

PFriNo=tot('Fri', 'N') / total\_exp

PHunYes=tot('Hun', 'Y') / total\_exp

PHunNo=tot('Hun', 'N') / total\_exp

PPatSome=tot('Pat', 'S') / total\_exp

PPatFull=tot('Pat', 'F') / total\_exp

PPatNone=tot('Pat', 'N') / total\_exp

PPriceCheap=tot('Price', '$') / total\_exp

PPriceAvg=tot('Price', '$$') / total\_exp

PPriceExp=tot('Price', '$$$') / total\_exp

PRainYes=tot('Rain', 'Y') / total\_exp

PRainNo=tot('Rain', 'N') / total\_exp

PResYes=tot('Res', 'Y') / total\_exp

PResNo=tot('Res', 'N') / total\_exp

PTypeFrench=tot('Type', 'F') / total\_exp

PTypeltalian=tot('Type', 'I') / total\_exp

PTypeBurger=tot('Type', 'B') / total\_exp

PTypeThai=tot('Type', 'T') / total\_exp

PEstFew=tot('Est', '0-10') / total\_exp

PEstMore=tot('Est', '10-30') /total\_exp

PEstStillMore=tot('Est', '30-60') /total\_exp

PEstTooMuch=tot('Est', '>60') / total\_exp

PAnsYes=tot('ans', 'Y') / total\_exp

PAnsNo=tot('ans', 'N') / total\_exp

print('Probability for will wait if there is an Alternate Restaurant Nearby:')

print('Yes: Will Wait', (getProbab('Alt', 'Y', 'Y') \* PAnsYes/PAItYes) \* 100, '%')

print('No: Will Wait', (getProbab('Alt', 'Y', 'N') \* PAnsNo/PAItYes) \* 100, '%')

print('Probability for will wait if there No is an Alternate Restaurant Nearby:')

print('Yes: Will Wait', (getProbab('Alt', 'N', 'Y') \* PAnsYes/PAItNo) \* 100, '%')

print('No: Will Wait', (getProbab('Alt', 'N', 'N') \* PAnsNo/PAItNo) \* 100, '%')

print('Probability for will wait if Estimated Wait time is 0-10 minutes:')

print('Yes: Will Wait', (getProbab('Est', '0-10', 'Y') \* PAnsYes/PEstFew) \* 100, '%')

print('No: Will Wait', (getProbab('Est', '0-10','N')\* PAnsNo/PEstFew) \* 100, '%')

print('Probability for will wait if Estimated Wait time is 10-30 minutes')

print('Yes: Will Wait', (getProbab('Est', '10 30', 'Y')\* PAnsYes/PEstMore) \* 100, '%')

print('No: Will Wait', (getProbab('Est', '10-30','N') \* PAnsNo/PEstMore) \* 100, '%')

print("Probability for Will Wait if the Estimated Wait Time is 30-60 mins: ")

print("Yes: Will Wait: ",(getProbab('Est','30','Y')\*PAnsYes/PEstStillMore)\*100,"%")

print("No: Will Wait: ",(getProbab('Est','30-60','N')\*PAnsNo/PEstStillMore)\*100,"%")

print("Probability for Will Wait if the Estimated Wait Time is >60 mins: ")

print("Yes: Will Wait: ",(getProbab('Est','>60','Y')\*PAnsYes/PEstTooMuch)\*100,"%")

print("No: Will Wait: ",(getProbab('Est','>60','N')\*PAnsNo/PEstTooMuch)\*100,"%")

print('Probability for will wait if there are Some Patrons')

print('Yes: Will Wait', (getProbab('Pat', 'S', 'Y') \* PAnsYes/PPatSome) \* 100, '%')

print('No: Will Wait ', (getProbab('Pat', 'S', 'N')\*PAnsNo/PPatSome) \* 100, '%')

print("Probability for Will Wait if there are None Patrons: ")

print("Yes: Will Wait: ", (getProbab('Pat', 'N', 'Y')\*PAnsYes/PPatNone)\*100, "%")

print("No: Will Wait: ", (getProbab('Pat', 'N', 'N') \* PAnsNo/PPatNone)\*100, "%")

print("Probability for Will Wait if there areFull Patrons: ")

print("Yes: Will Wait: ", (getProbab('Pat', 'F', 'Y') \* PAnsYes/PPatFull)\*100, "%")

print("No: Will Wait: ", (getProbab('Pat', 'F', 'N') \* PAnsNo/PPatFull)\*100, "%")

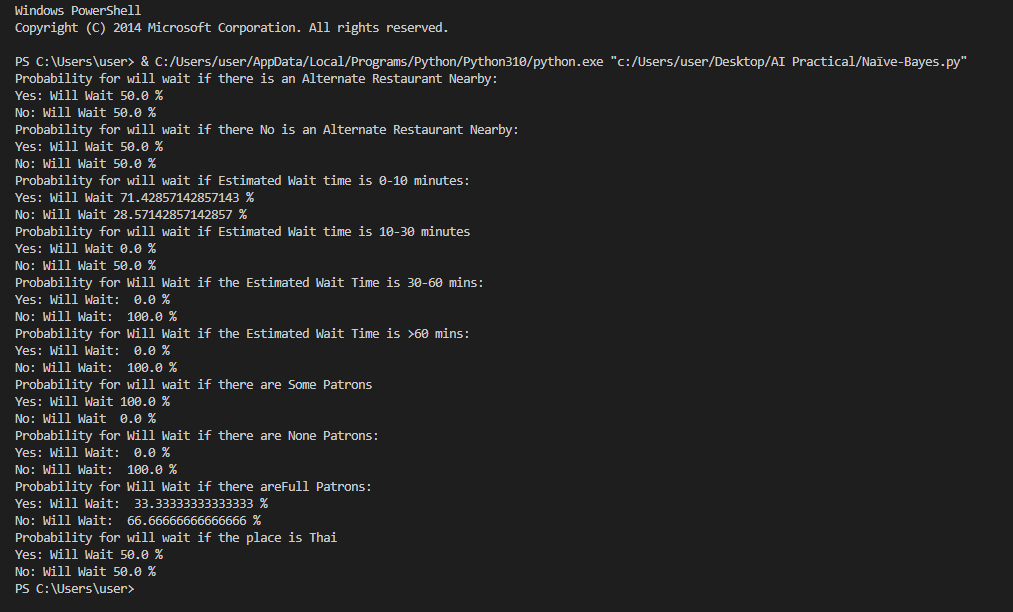
print('Probability for will wait if the place is Thai')

print('Yes: Will Wait', (getProbab('Type', 'T', 'Y') \* PAnsYes/PTypeThai) \* 100, '%')

print('No: Will Wait', (getProbab('Type', 'T', 'N') \* PAnsNo/PTypeThai) \* 100, '%')

main()

**Output:-**

****

**Practical 7**

**Aim:-** Implement linear regression for RWP

**Code:-**

import numpy as np

import matplotlib.pyplot as plt

def estimate\_coef(x, y):

# number of observations/points

n = np.size(x)

# mean of x and y vector

m\_x = np.mean(x)

m\_y = np.mean(y)

# calculating cross-deviation and deviation about x

SS\_xy = np.sum(y \* x) - n \* m\_y \* m\_x

SS\_xx = np.sum(x \* x) - n \* m\_x \* m\_x

# calculating regression coefficients

b\_1 = SS\_xy / SS\_xx

b\_0 = m\_y - b\_1 \* m\_x

return b\_0, b\_1

def plot\_regression\_line(x, y, b):

# plotting the actual points as scatter plot 25

plt.scatter(x, y, color="m", marker="o", s=30)

# predicted response vector

y\_pred = b[0] + b[1] \* x

# plotting the regression line 32

plt.plot(x, y\_pred, color="g")

# putting labels

plt.xlabel('x')

plt.ylabel('y')

# function to show plot 39

plt.show()

def main():

# observations / data

x = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

y = np.array([1, 3, 2, 5, 7, 8, 8, 9, 10, 12])

# estimating coefficients

b = estimate\_coef(x, y)

print("Estimated coefficients: \nb\_0 = {} \\nb\_1 = {}".format(b[0], b[1]))

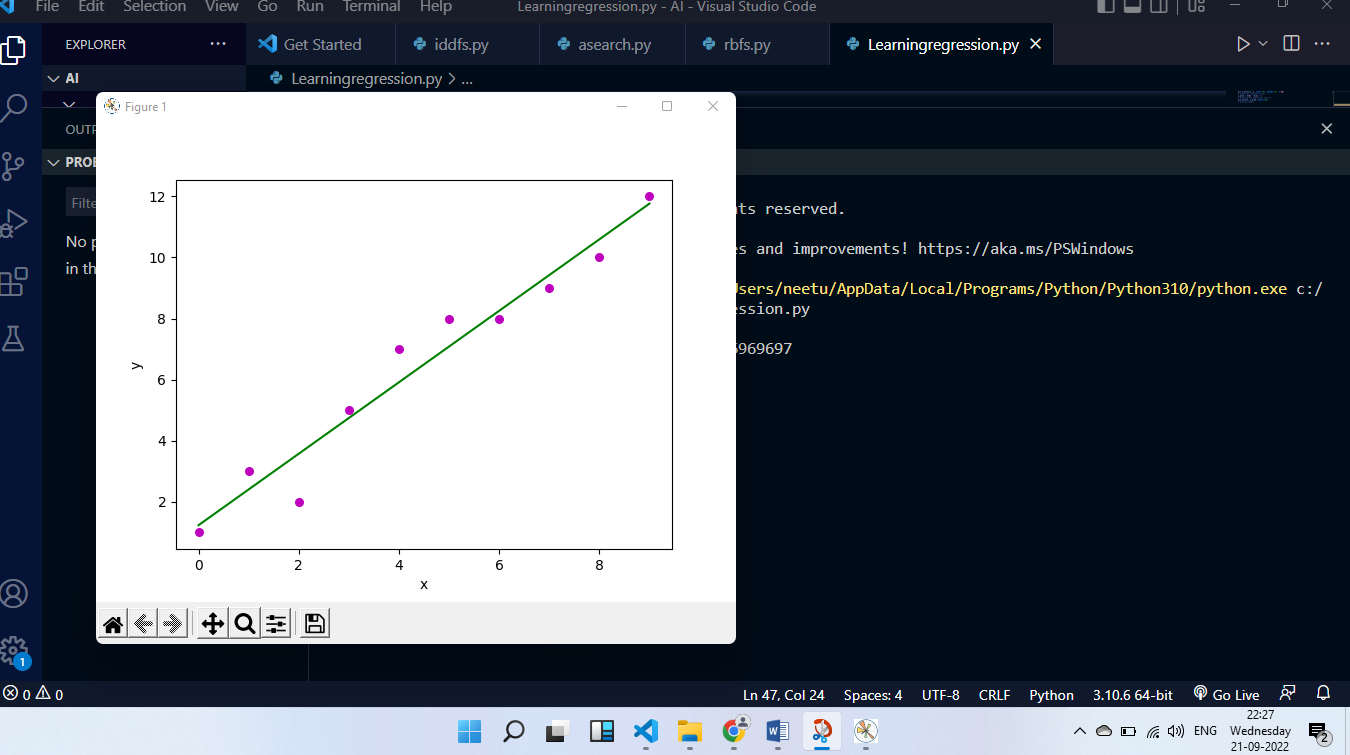
# plotting regression line

plot\_regression\_line(x, y, b)

if name == "main":

main()

**Output:-**

****

**Practical 8**

**Aim:-**  Implement feed forward back propagation neural network learning algorithm for the RWP

**Code**

print("FEED FORWARD BACK PROPAGATIOn NEURAL NETWORK LEARNING ALGORITHM")

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\_iterations):

for iteration in range(training\_iterations):

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: "))

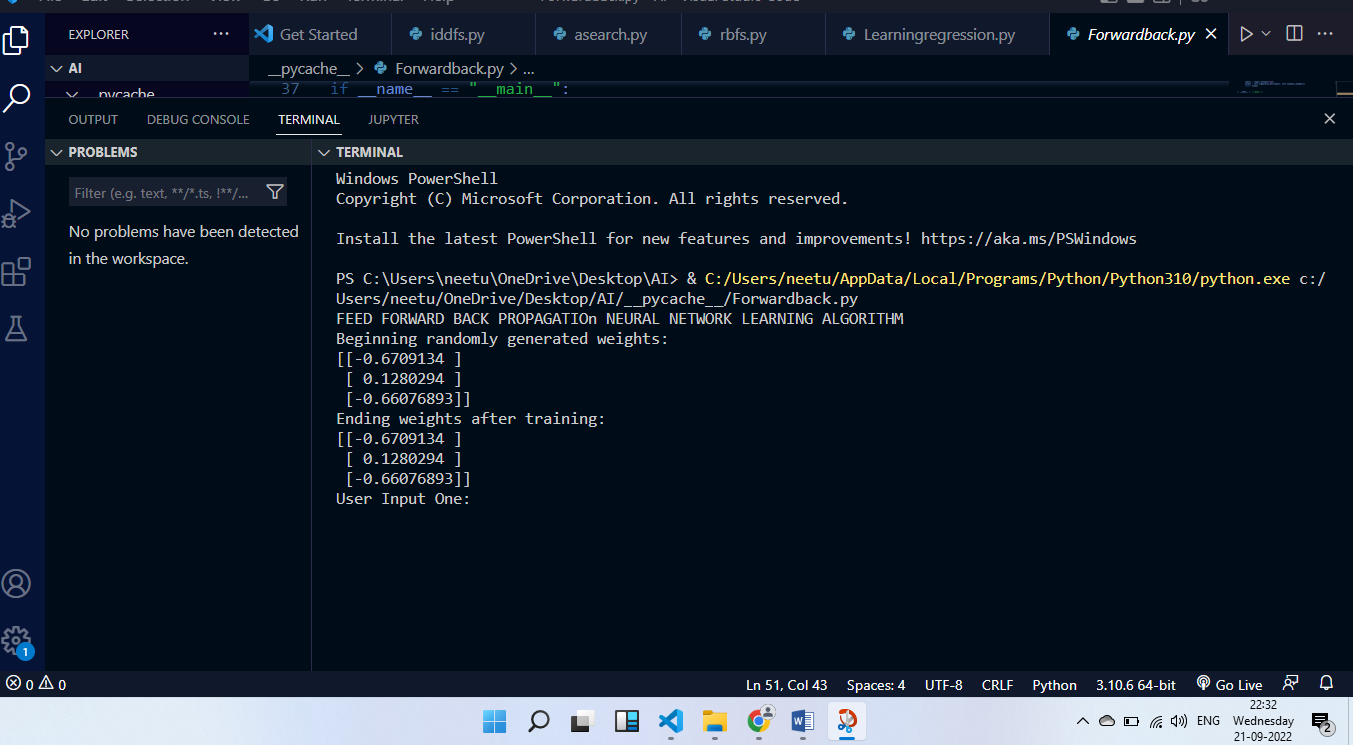
user\_input\_three = str(input("User Input Three:"))

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:-**

****