PRACTICAL 4

MAX, MIN = 1000, -1000

def minimax(depth, nodeIndex, maximizingPlayer,

            values, alpha, beta):

    if depth == 3:

        return values[nodeIndex]

    if maximizingPlayer:

        best = MIN

        for i in range(0, 2):

            val = minimax(depth + 1, nodeIndex \* 2 + i,

                        False, values, alpha, beta)

            best = max(best, val)

            alpha = max(alpha, best)

            if beta <= alpha:

                break

        return best

    else:

        best = MAX

        for i in range(0, 2):

            val = minimax(depth + 1, nodeIndex \* 2 + i,

                            True, values, alpha, beta)

            best = min(best, val)

            beta = min(beta, best)

            if beta <= alpha:

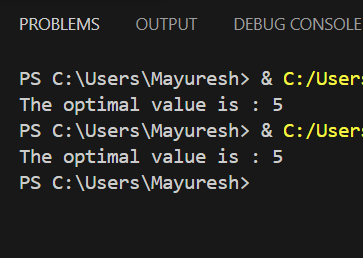
                break

        return best

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

    values = [3, 5, 6, 9, 1, 2, 0, -1]

    print("The optimal value is :", minimax(0, 0, True, values, MIN, MAX))



PRACTICAL 5  
import nltk

from nltk.chat import Chat

# Reflection dictionary can be left empty as it's not used here

reflection = {}

# Define the pattern-response pairs

pairs = [

    [r'hello', ['Hello! How can I assist you today?']],

    [r'how are you', ["I'm just a computer program, but I'm doing well. How can I help you?"]],

    [r'name', ["I don't have a name, you can call me Chatbot."]],

    [r'bye', ['Goodbye! If you have more questions, feel free to ask.']]

]

# Function to initiate the chatbot

def chat():

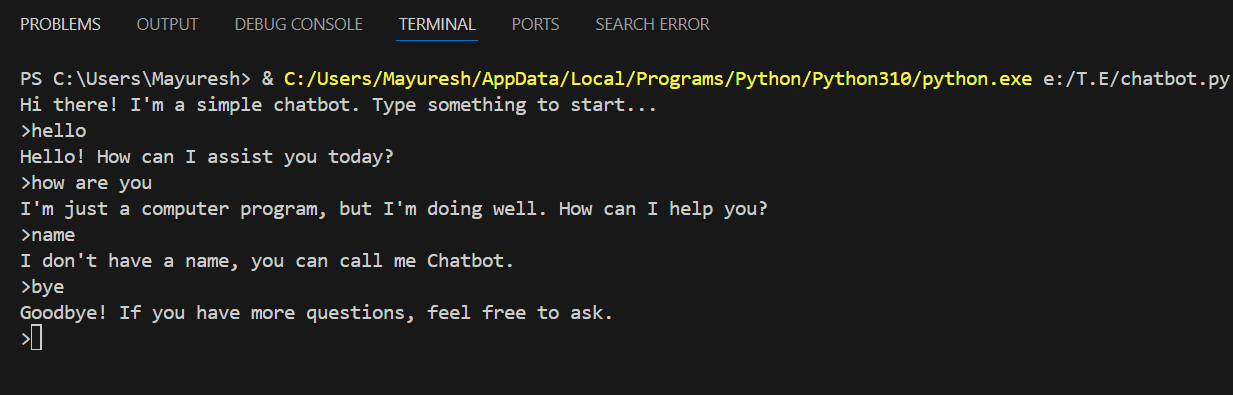
    print("Hi there! I'm a simple chatbot. Type something to start...")

    chat = Chat(pairs, reflection)

    chat.converse()

# Start the chat

chat()



PRACTICAL 6

#Practical No. 4

INF = 9999999

# number of vertices in graph

N = 5

#creating graph by adjacency matrix method

G = [[0, 19, 5, 0, 0],

     [19, 0, 5, 9, 2],

     [5, 5, 0, 1, 6],

     [0, 9, 1, 0, 1],

     [0, 2, 6, 1, 0]]

selected\_node = [0, 0, 0, 0, 0]

no\_edge = 0

selected\_node[0] = True

# printing for edge and weight

print("Edge : Weight\n")

while (no\_edge < N - 1):

    minimum = INF

    a = 0

    b = 0

    for m in range(N):

        if selected\_node[m]:

            for n in range(N):

                if ((not selected\_node[n]) and G[m][n]):

                    # not in selected and there is an edge

                    if minimum > G[m][n]:

                        minimum = G[m][n]

                        a = m

                        b = n

    print(str(a) + "-" + str(b) + ":" + str(G[a][b]))

    selected\_node[b] = True

    no\_edge += 1

