**NAME: KEERTHANA.P**

**REG.NO:192125008**

**MLA0101- ARTIFICIAL INTELLIGENCE AND EXPERT SYSTEMS FOR ENGINEERING APPLICATIONS**

**EXNO:** 6 **TITLE:** 8 QUEENS PROBLEM IN PYTHON

**INPUT:**

# Taking number of queens as input from user

print ("Enter the number of queens")

N = int(input())

# here we create a chessboard

# NxN matrix with all elements set to 0

board = [[0]\*N for \_ in range(N)]

def attack(i, j):

#checking vertically and horizontally

for k in range(0,N):

if board[i][k]==1 or board[k][j]==1:

return True

#checking diagonally

for k in range(0,N):

for l in range(0,N):

if (k+l==i+j) or (k-l==i-j):

if board[k][l]==1:

return True

return False

def N\_queens(n):

if n==0:

return True

for i in range(0,N):

for j in range(0,N):

if (not(attack(i,j))) and (board[i][j]!=1):

board[i][j] = 1

if N\_queens(n-1)==True:

return True

board[i][j] = 0

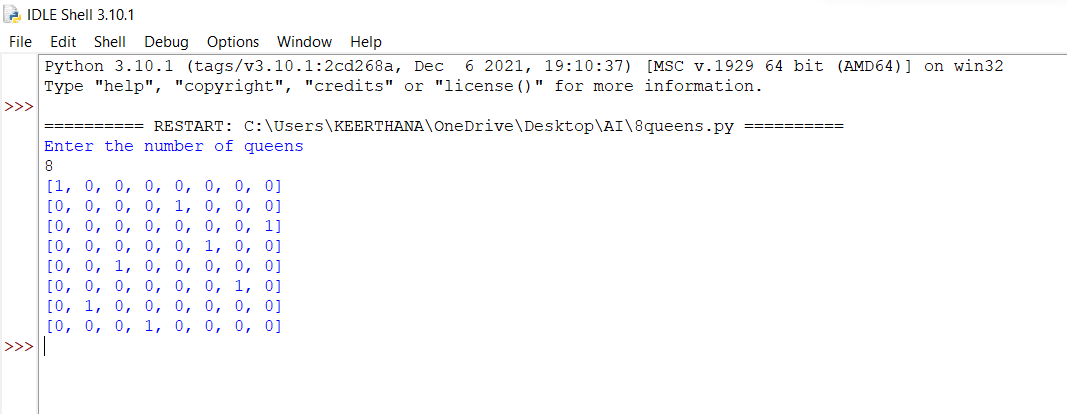
return False

N\_queens(N)

for i in board:

print (i)

**OUPUT:**



**EX NO:**7 **TITLE:** Tic-Tac-Toe GAME IN PYTHON

**INPUT:**

import random

class TicTacToe:

def \_init\_(self):

self.board = []

def create\_board(self):

for i in range(3):

row = []

for j in range(3):

row.append('-')

self.board.append(row)

def get\_random\_first\_player(self):

return random.randint(0, 1)

def fix\_spot(self, row, col, player):

self.board[row][col] = player

def is\_player\_win(self, player):

win = None

n = len(self.board)

# checking rows

for i in range(n):

win = True

for j in range(n):

if self.board[i][j] != player:

win = False

break

if win:

return win

# checking columns

for i in range(n):

win = True

for j in range(n):

if self.board[j][i] != player:

win = False

break

if win:

return win

# checking diagonals

win = True

for i in range(n):

if self.board[i][i] != player:

win = False

break

if win:

return win

win = True

for i in range(n):

if self.board[i][n - 1 - i] != player:

win = False

break

if win:

return win

return False

for row in self.board:

for item in row:

if item == '-':

return False

return True

def is\_board\_filled(self):

for row in self.board:

for item in row:

if item == '-':

return False

return True

def swap\_player\_turn(self, player):

return 'X' if player == 'O' else 'O'

def show\_board(self):

for row in self.board:

for item in row:

print(item, end=" ")

print()

def start(self):

self.create\_board()

player = 'X' if self.get\_random\_first\_player() == 1 else 'O'

while True:

print(f"Player {player} turn")

self.show\_board()

# taking user input

row, col = list(

map(int, input("Enter row and column numbers to fix spot: ").split()))

print()

# fixing the spot

self.fix\_spot(row - 1, col - 1, player)

# checking whether current player is won or not

if self.is\_player\_win(player):

print(f"Player {player} wins the game!")

break

# checking whether the game is draw or not

if self.is\_board\_filled():

print("Match Draw!")

break

# swapping the turn

player = self.swap\_player\_turn(player)

# showing the final view of board

print()

self.show\_board()

# starting the game

tic\_tac\_toe = TicTacToe()

tic\_tac\_toe.start()

**OUTPUT:**

****

**EX NO: 8 TITLE**: To write a python program to implement Travelling Salesman Problem

**INPUT:**

from sys import maxsize

from itertools import permutations

V = 4

def travellingSalesmanProblem(graph, s):

vertex = []

for i in range(V):

if i != s:

vertex.append(i)

min\_path = maxsize

next\_permutation=permutations(vertex)

for i in next\_permutation:

current\_pathweight = 0

k = s

for j in i:

current\_pathweight += graph[k][j]

k = j

current\_pathweight += graph[k][s]

min\_path = min(min\_path, current\_pathweight)

return min\_path

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

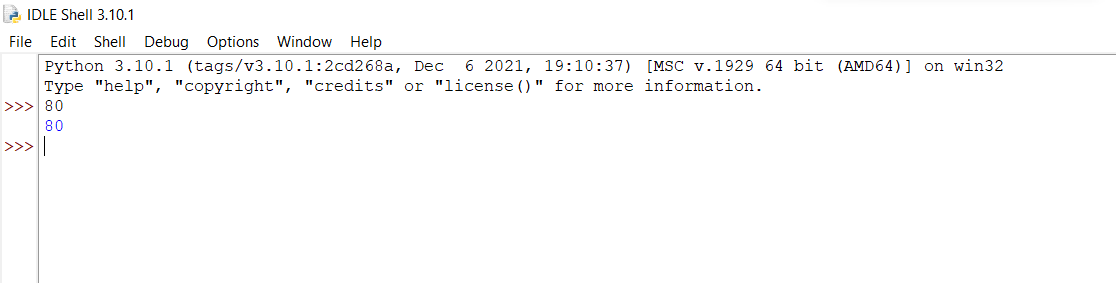
graph = [[0, 10, 15, 20], [10, 0, 35, 25],

[15, 35, 0, 30], [20, 25, 30, 0]]

s = 0

print(travellingSalesmanProblem(graph, s))

**OUTPUT:**



**EX NO:** 9 **TITLE:** write a program to solve tower of hanoi problem in python

**INPUT:**

def TowerOfHanoi(n , source, destination, auxiliary):

if n==1:

print ("Move disk 1 from source",source,"to destination",destination)

return

TowerOfHanoi(n-1, source, auxiliary, destination)

print ("Move disk",n,"from source",source,"to destination",destination)

TowerOfHanoi(n-1, auxiliary, destination, source)

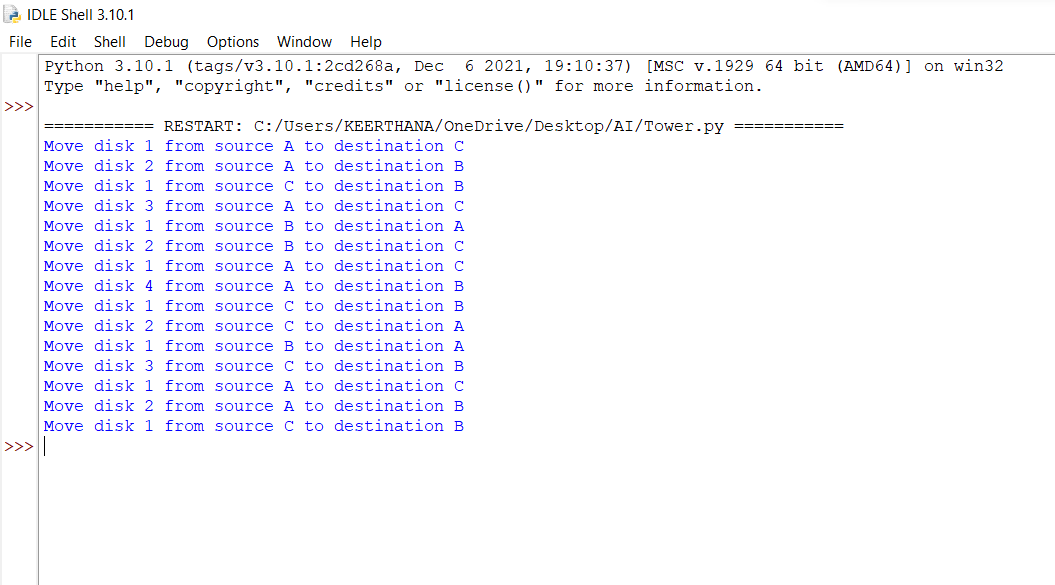
# Driver code

n = 4

TowerOfHanoi(n,'A','B','C')

# A, C, B are the name of rods

**OUTPUT:**



**EX NO**: 10 **TITLE**: Python Program for 0-1 Knapsack Problem

**INPUT:**

# a dynamic approach

# Returns the maximum value that can be stored by the bag

def knapSack(W, wt, val, n):

K = [[0 for x in range(W + 1)] for x in range(n + 1)]

#Table in bottom up manner

for i in range(n + 1):

for w in range(W + 1):

if i == 0 or w == 0:

K[i][w] = 0

elif wt[i-1] <= w:

K[i][w] = max(val[i-1] + K[i-1][w-wt[i-1]], K[i-1][w])

else:

K[i][w] = K[i-1][w]

return K[n][W]

#Main

val = [50,100,150,200]

wt = [8,16,32,40]

W = 64

n = len(val)

print(knapSack(W, wt, val, n))

**OUTPUT:**

