

```

# N QUEEN PROBLEM
global N
N = 4
def printSolution(board):
    for i in range(N):
        for j in range(N):
            print(board[i][j], end = " ")
        print()

# A utility function to check if a queen can
# be placed on board[row][col]. Note that this
# function is called when "col" queens are
# already placed in columns from 0 to col -1.
# So we need to check only left side for
# attacking queens

def isSafe(board, row, col):
    # Check this row on left side
    for i in range(col):
        if board[row][i] == 1:
            return False
    # Check upper diagonal on left side
    for i, j in zip(range(row, -1, -1),
                    range(col, -1, -1)):
        if board[i][j] == 1:
            return False
    # Check lower diagonal on left side
    for i, j in zip(range(row, N, 1),
                    range(col, -1, -1)):
        if board[i][j] == 1:
            return False
    return True

def solveNQUtil(board, col):
    # base case: If all queens are placed
    # then return true
    if col >= N:
        return True
    # Consider this column and try placing
    # this queen in all rows one by one
    for i in range(N):
        if isSafe(board, i, col):
            # Place this queen in board[i][col]
            board[i][col] = 1
            # recur to place rest of the queens
            if solveNQUtil(board, col + 1) == True:
                return True
            # If placing queen in board[i][col]

```

```

        # doesn't lead to a solution, then
        # queen from board[i][col]
        board[i][col] = 0
    # if the queen can not be placed in any row in
    # this column col then return false
    return False

# This function solves the N Queen problem using
# Backtracking. It mainly uses solveNQUtil() to
# solve the problem. It returns false if queens
# cannot be placed, otherwise return true and
# placement of queens in the form of 1s.
# note that there may be more than one
# solutions, this function prints one of the
# feasible solutions.
def solveNQ():
    board = [ [0, 0, 0, 0],
              [0, 0, 0, 0],
              [0, 0, 0, 0],
              [0, 0, 0, 0] ]
    if solveNQUtil(board, 0) == False:
        print ("Solution does not exist")
        return False
    printSolution(board)
    return True

# Driver Code
solveNQ()

```

OUTPUT :

The screenshot shows a Visual Studio Code window with a file named `n_queen.py` open. The editor displays the Python code for solving the N-Queen problem using backtracking. The code includes a `solveNQUtil` function and a `solveNQ` function. The `solveNQ` function initializes a 4x4 board and calls `solveNQUtil` to find a solution. The output of the program is shown in the terminal window, which displays the solution board as a 4x4 grid of 0s and 1s.

```

56
57 # This function solves the N Queen problem using
58 # Backtracking. It mainly uses solveNQUtil() to
59 # solve the problem. It returns false if queens

```

Windows PowerShell  
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell <https://aka.ms/pscore6>

```

PS C:\Users\Asus\Desktop\Harshada Docs\LP-III Practical Assignments with Write-up format\machine 1\LP-III Practical Assignments with Write-up format\daa> & 'C:\Users\Asus\AppData\Local\Programs\Python\Python39\python.exe' 'c:\Users\Asus\.vscode\extensions\ms-python.python-2022.16.1\pythonFiles\lib\python\debugpy\adapter\..\..\debugpy\launcher' '58549' '--' 'c:\Users\Asus\Desktop\Harshada Docs\LP-III Practical Assignments with Write-up format\machine 1\LP-III Practical Assignments with Write-up format\daa\n_queen.py'
0 0 1 0
1 0 0 0
0 0 0 1
0 1 0 0
PS C:\Users\Asus\Desktop\Harshada Docs\LP-III Practical Assignments with Write-up format\machine 1\LP-III Practical Assignments with Write-up format\daa>

```

Ln 69, Col 27 Spaces: 4 UTF-8 CRLF Python 3.9.1 64-bit Go Live 22:01 06-11-2022