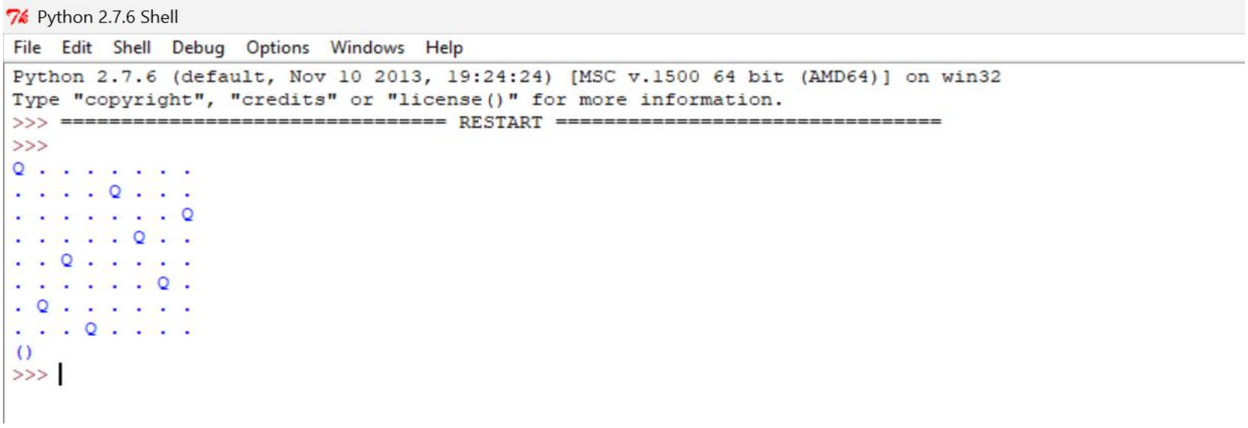
PROGRAM 2:

```
# Function to print the board
def print_board(board):
    for row in board:
        print(" ".join("Q" if x == 1 else "." for x in row))
    print()

# Function to check if it's safe to place a queen at board[row][col]
def is_safe(board, row, col):
    # Check the column
    for i in range(row):
        if board[i][col] == 1:
            return False

    # Check the upper-left diagonal
    for i, j in zip(range(row-1, -1, -1), range(col-1, -1, -1)):
        if board[i][j] == 1:
            return False

    # Check the upper-right diagonal
    for i, j in zip(range(row-1, -1, -1), range(col+1, len(board))):
        if board[i][j] == 1:
            return False

    return True

# Function to solve the N-Queen problem using backtracking
def solve_n_queen(board, row):
    # If all queens are placed
    if row >= len(board):
        return True

    for col in range(len(board)):
        if is_safe(board, row, col):
            board[row][col] = 1  # Place the queen
            if solve_n_queen(board, row + 1):
                return True
            board[row][col] = 0  # Backtrack

    return False

# Function to solve the 8-Queen problem
def solve_8_queen():
    N = 8
    board = [[0] * N for _ in range(N)]  # Create an 8x8 board initialized to 0 (no queens)

    if solve_n_queen(board, 0):
        print_board(board)
    else:
        print("Solution does not exist")

# Run the program
solve_8_queen()
```

OUTPUT:



```
Python 2.7.6 (default, Nov 10 2013, 19:24:24) [MSC v.1500 64 bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ================================ RESTART ================================
>>>
Q . . . . . . .
. . . . Q . . .
. . . . . . . Q
. . . . . Q . .
. . Q . . . . .
. . . . . . Q .
. Q . . . . . .
. . . Q . . . .
()
>>>
```