**Aim:**

To write and execute the python program for solving 8 queens problem

**Procedure:**

1. **Initialize the Chess Board:**
   * The program creates an 8x8 chess board represented as a 2D list (**board**) initially filled with zeros.
2. **Check Safety of a Queen Placement:**
   * The **is\_safe** function checks whether placing a queen at a specified position (**row**, **col**) on the board is safe.
   * It checks for conflicts with existing queens in the same column and diagonals.
3. **Recursively Solve the N-Queens Problem:**
   * The **solve\_queens** function uses backtracking to find a solution to the N-Queens problem.
   * It starts from the first row (**row=0**) and iterates through each column, attempting to place a queen.
   * If a safe position is found, it recursively calls itself for the next row (**row + 1**).
   * If a solution is found, the function returns **True**; otherwise, it backtracks by setting the current position to 0.
4. **Print the Solution:**
   * The **print\_solution** function prints the final arrangement of queens on the board in a readable format.
5. **Main Program:**
   * The **main** function initializes the chess board and calls **solve\_queens** with the initial row (**row=0**).
   * If a solution is found, it prints the solution using **print\_solution**.
   * If no solution exists, it prints a message indicating that no solution was found.
6. **Run the Program:**
   * The **main** function is called when the script is executed (**\_\_name\_\_ == "\_\_main\_\_"**).
7. **Output:**
   * The program either prints the solution with queen placements or indicates that no solution exists.

**Code:**

def is\_safe(board, row, col):

for i in range(row):

if board[i][col] == 1:

return False

for i, j in zip(range(row, -1, -1), range(col, -1, -1)):

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

return False

for i, j in zip(range(row, -1, -1), range(col, len(board))):

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

return False

return True

def solve\_queens(board, row):

if row == len(board):

return True

for col in range(len(board)):

if is\_safe(board, row, col):

board[row][col] = 1

if solve\_queens(board, row + 1):

return True

board[row][col] = 0

return False

def print\_solution(board):

for row in board:

print(" ".join(map(str, row)))

def main():

board = [[0] \* 8 for \_ in range(8)]

if solve\_queens(board, 0):

print("Solution found:")

print\_solution(board)

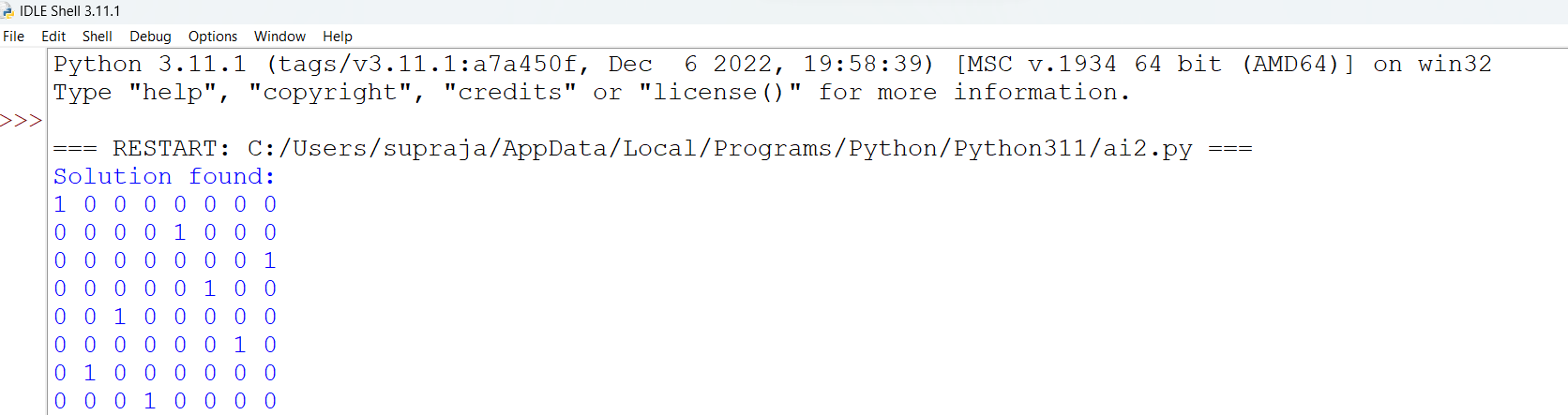
else:

print("No solution exists.")

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

main()

**output:**



**Result:**

Hence the program has been successfully executed and verified.