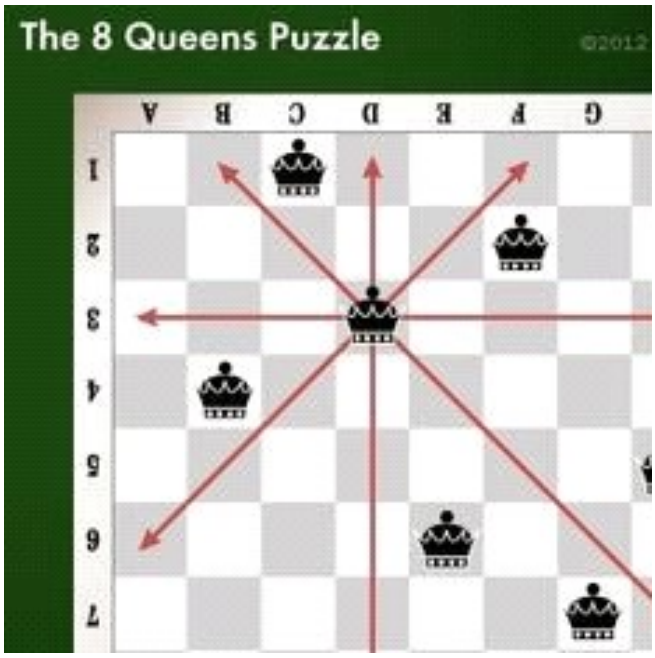
EX.NO: 1

DATE:

8- QUEENS PROBLEM

You are given an 8x8 board; find a way to place 8 queens such that no queen can attack any other queen on the chessboard. A queen can only be attacked if it lies on the same row, or same column, or the same diagonal of any other queen. Print all the possible configurations.

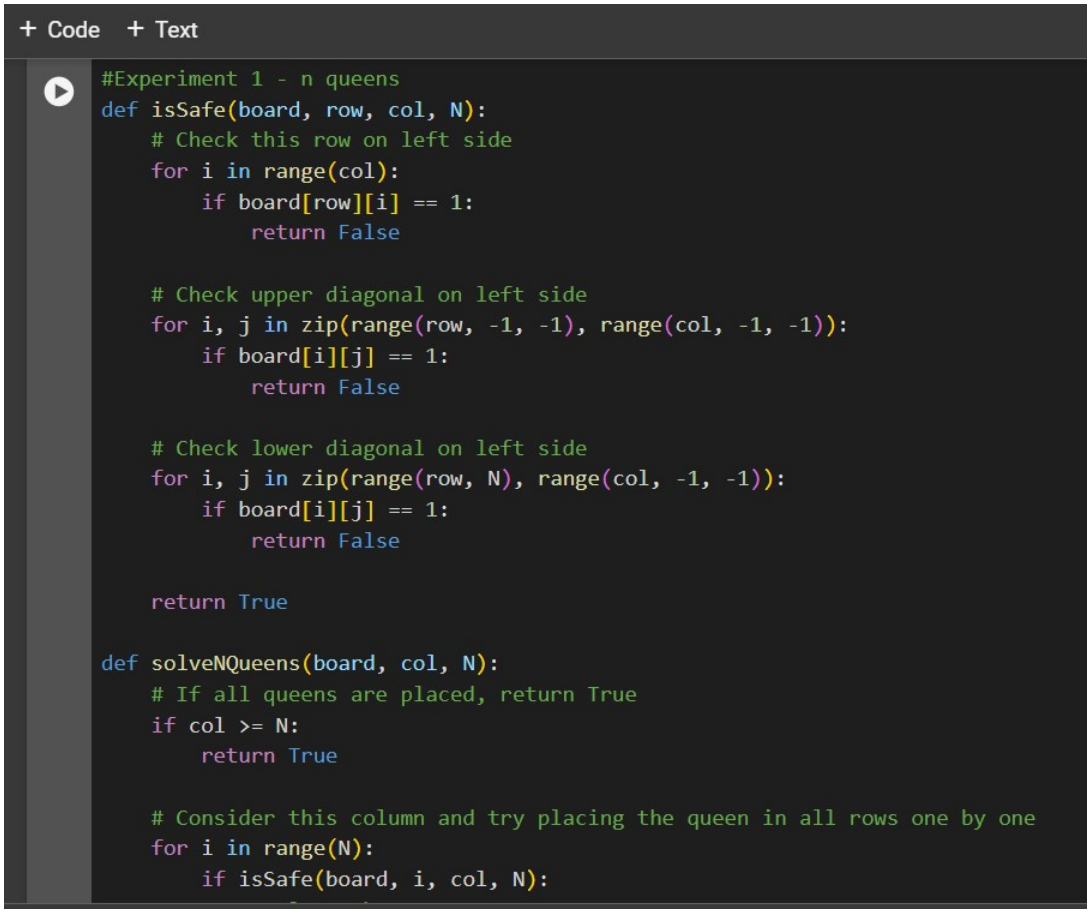
To solve this problem, we will make use of the Backtracking algorithm. The backtracking algorithm, in general checks all possible configurations and test whether the required result is obtained or not. For the given problem, we will explore all possible positions the queens can be relatively placed at. The solution will be correct when the number of placed queens = 8.



AIM :

To implement an 8-Queesns problem using python.

CODE :



```
+ Code  + Text

#Experiment 1 - n queens
def isSafe(board, row, col, N):
    # 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), range(col, -1, -1)):
        if board[i][j] == 1:
            return False

    return True

def solveNQueens(board, col, N):
    # If all queens are placed, return True
    if col >= N:
        return True

    # Consider this column and try placing the queen in all rows one by one
    for i in range(N):
        if isSafe(board, i, col, N):
```

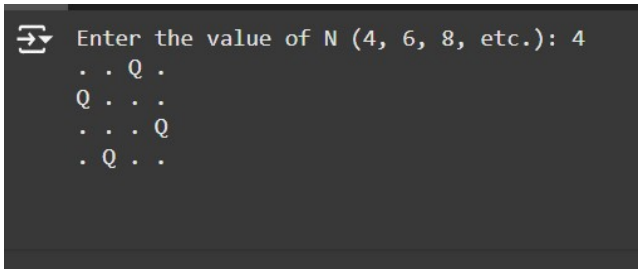
+ Code + Text

```
    # Consider this column and try placing the queen in all rows one by one
    for i in range(N):
        if isSafe(board, i, col, N):
            # Place the queen
            board[i][col] = 1

            # Recur to place the rest of the queens
            if solveNQueens(board, col + 1, N):
                return True

            # If placing queen in board[i][col] doesn't lead to a solution, backtrack
            board[i][col] = 0

    return False

def printSolution(board, N):
    for i in range(N):
        for j in range(N):
            if board[i][j] == 1:
                print("Q", end=" ")
            else:
                print(".", end=" ")
        print()
    print("\n")

def solveNQueensProblem(N):
    board = [[0 for _ in range(N)] for _ in range(N)]
```

+ Code + Text

```
def printSolution(board, N):
    for i in range(N):
        for j in range(N):
            if board[i][j] == 1:
                print("Q", end=" ")
            else:
                print(".", end=" ")
        print()
    print("\n")

def solveNQueensProblem(N):
    board = [[0 for _ in range(N)] for _ in range(N)]

    if not solveNQueens(board, 0, N):
        print("Solution does not exist")
        return False

    printSolution(board, N)
    return True

# Main code
if __name__ == "__main__":
    N = int(input("Enter the value of N (4, 6, 8, etc.): "))
    solveNQueensProblem(N)
```

OUTPUT:



RESULT:

Thus the program is successfully executed and output is verified