def solve\_n\_queens(n):

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

if solve\_n\_queens\_util(board, 0, n):

print\_board(board)

else:

print("No solution exists")

def solve\_n\_queens\_util(board, row, n):

if row >= n:

return True

for col in range(n):

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

board[row][col] = 1

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

return True

board[row][col] = 0

return False

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

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, n)):

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

return False

return True

def print\_board(board):

for row in board:

print(" ".join("Q" if cell else "." for cell in row))

n = int(input("Enter the size of the board (n for n-Queens): "))

solve\_n\_queens(n)

def solve\_n\_queens(n):

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

solutions = []

count = solve\_n\_queens\_util(board, 0, n, solutions)

print(f"Total solutions: {count}")

for idx, solution in enumerate(solutions, 1):

print(f"\nSolution {idx}:")

print\_board(solution)

def solve\_n\_queens\_util(board, row, n, solutions):

if row >= n:

solutions.append([row[:] for row in board])

return 1

count = 0

for col in range(n):

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

board[row][col] = 1

count += solve\_n\_queens\_util(board, row + 1, n, solutions)

board[row][col] = 0

return count

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

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, n)):

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

return False

return True

def print\_board(board):

for row in board:

print(" ".join("Q" if cell else "." for cell in row))

n = int(input("Enter the size of the board (n for n-Queens): "))

solve\_n\_queens(n)