class NQueens:

def \_\_init\_\_(self, n):

self.n = n

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

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

# Check this row on the left side

for i in range(col):

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

return False

# Check lower diagonal on left side

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

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

return False

return True

def solve\_nqueens\_util(self, col):

if col >= self.n:

return True

for i in range(self.n):

if self.is\_safe(i, col):

self.board[i][col] = 1

if self.solve\_nqueens\_util(col + 1):

return True

self.board[i][col] = 0 # Backtrack

return False

def solve\_nqueens(self):

if not self.solve\_nqueens\_util(0):

print("Solution does not exist")

return

self.print\_solution()

def print\_solution(self):

print("\nOne of the solutions:")

for row in self.board:

print(" ".join("Q" if cell == 1 else "." for cell in row))

print()

# === User Input Section ===

def main():

try:

n = int(input("Enter the number of queens (N): "))

if n < 1:

print("N must be a positive integer.")

return

solver = NQueens(n)

solver.solve\_nqueens()

except ValueError:

print("Invalid input. Please enter a positive integer.")

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

main()