The N-Queens Problem

```
#!/usr/bin/env python3

#-------------------------------------------------------------------------------
# queens n : display a solution to the n-queens problem, if one exists
#-------------------------------------------------------------------------------
# A placement of queens on the chessboard is represented by a list 'b' in which
#     b[ r ] = c
# denotes that there is a queen on row 'r' and column 'c' of the chessboard
#-------------------------------------------------------------------------------
# 'n', the dimension of the problem board, is a global variable
#-------------------------------------------------------------------------------

def Solve( b = [ ] ) :

    # With queens already placed safely on each row of the partial board 'b',
    # attempt to now place queens safely on all remaining rows up to row 'n';
    # return a tuple  ( success, solution )  where, if a solution was completed,
    # 'success' is True and 'solution' is this solution, or if no such solution
    # was found, 'success' is False and 'solution' is the unchanged board 'b'

    r = len( b )

    if r == n :
        return ( True, b )

    else :
        for c in range( n ) :
            if IsSafe( b, r, c ) :
                ( success, solution ) = Solve( b + [ c ] )
                if success :
                    return ( True, solution )

        return ( False, b )

#-------------------------------------------------------------------------------

def IsSafe( b, r, c ) :

    # Would a queen on row 'r' and column 'c' of board 'b'
    # be safe from each of the queens on rows 0 to r-1 ?

    for row in range( r ) :
        col = b[ row ]
        if col == c or row - col == r - c or row + col == r + c :
            return False

    return True

#-------------------------------------------------------------------------------

def WriteBoard( b ) :

    # Output a drawing of board 'b'

    print( )
    for row in range( n ) :
        print( b[ row ] * " +" + " Q" + ( n - 1 - b[ row ] ) * " +" )
    print( )

#-------------------------------------------------------------------------------
```

```
#-------------------------------------------------------------------------------

from sys import argv

n = int( argv[ 1 ] )  # no error checking

( success, solution ) = Solve( )

if success :
    print( "\nSolution for the %i-queens problem:" % ( n ) )
    WriteBoard( solution )
else :
    print( "\nNo solution found for the %i-queens problem\n" % ( n ) )

#-------------------------------------------------------------------------------

$ queens 3

No solution found for the 3-queens problem

#-------------------------------------------------------------------------------

$ queens 4

Solution for the 4-queens problem:

 + Q + +
 + + + Q
 Q + + +
 + + Q +

#-------------------------------------------------------------------------------

$ queens 5

Solution for the 5-queens problem:

 Q + + + +
 + + Q + +
 + + + + Q
 + Q + + +
 + + + Q +

#-------------------------------------------------------------------------------

$ queens 8

Solution for the 8-queens problem:

 Q + + + + + + +
 + + + + Q + + +
 + + + + + + + Q
 + + + + + Q + +
 + + Q + + + + +
 + + + + + + Q +
 + Q + + + + + +
 + + + Q + + + +

#-------------------------------------------------------------------------------
```

The N-Queens Problem

```
#-------------------------------------------------------------------------------
# 'queens-trace' inserts statement 'print( b )' at the start of function 'Solve'
#-------------------------------------------------------------------------------

$ queens-trace 5

[]
[0]
[0, 2]
[0, 2, 4]
[0, 2, 4, 1]
[0, 2, 4, 1, 3]

Solution for the 5-queens problem:

 Q + + + +
 + + Q + +
 + + + + Q
 + Q + + +
 + + + Q +
```

```
$ queens-trace 6

[]
[0]
[0, 2]
[0, 2, 4]
[0, 2, 4, 1]
[0, 2, 4, 1, 3]
[0, 2, 5]
[0, 2, 5, 1]
[0, 3]
[0, 3, 1]
[0, 3, 1, 4]
[0, 3, 1, 4, 2]
[0, 3, 5]
[0, 3, 5, 2]
[0, 4]
[0, 4, 1]
[0, 4, 1, 5]
[0, 4, 1, 5, 2]
[0, 5]
[0, 5, 1]
[0, 5, 1, 4]
[0, 5, 3]
[0, 5, 3, 1]
[1]
[1, 3]
[1, 3, 0]
[1, 3, 0, 2]
[1, 3, 0, 2, 4]
[1, 3, 5]
[1, 3, 5, 0]
[1, 3, 5, 0, 2]
[1, 3, 5, 0, 2, 4]

Solution for the 6-queens problem:

 + Q + + + +
 + + + Q + +
 + + + + + Q
 Q + + + + +
 + + Q + + +
 + + + + Q +
```