8 Queens

Darin Critchlow
CSIS 2430-001

Objective:

The eight queens puzzle is the problem of placing eight chess queens on an 8×8 chessboard so that no two queens attack each other. Thus, a solution requires that no two queens share the same row, column, or diagonal. The eight queens puzzle is an example of the more general n-queens problem of placing n queens on an $n \times n$ chessboard, where solutions exist for all natural numbers n with the exception of $n = 2$ and $n = 3$.

What Worked:

Everything worked properly

What Didn't Work:

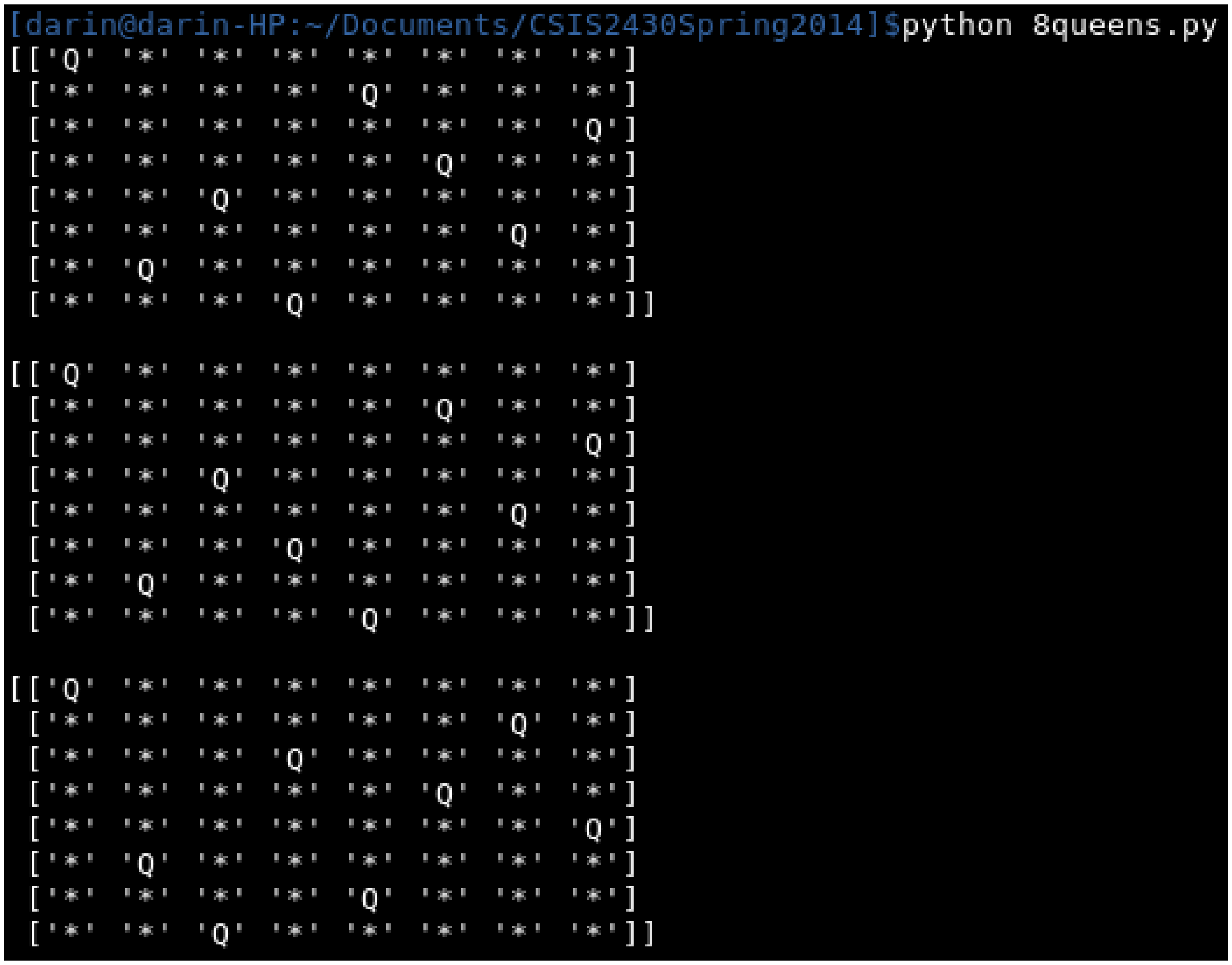
Everything worked properly

Comments:

It was more difficult than I first thought it would be to decide on what data structure to use to store the solutions. I wanted a way to keep the row and column so that I could output a pretty version to the console. I decided to use a set structure because it provided very convenient methods to iterate through row and column values.

Code:

```
#!/usr/bin/python
import numpy as np

# Store the solution set
x = {}

def nQueen(k, n):
    """
    Builds solutions for 'n' size of chess board
    """
    for i in range(1, n+1):
        if place(k, i):
            x[k] = i
            if k == n:
                # print x.values()
                printBoard(x)
            else:
                nQueen(k+1, n)

def place(row, col):
    """
    Backtracking algorithm to check queen placement
    """
    for j in range(1, row):
        # Check for 'rook' and 'bishop' conflicts
        if x[j] == col or abs(j - row) == abs(x[j] - col):
            return False
    return True

def printBoard(solution):
    """
    Prints a pretty board using numpy
    """
    board = np.array([['*'] * n] * n)
    for row, col in solution.items():
        board[row-1, col-1] = 'Q'
    print board,'\n'

if __name__ == '__main__':
    # Solve for board size 8x8
    n = 8
    nQueen(1, n)
```



```
[darin@darin-HP:~/Documents/CSIS2430Spring2014]$python 8queens.py
[['Q' '*' '*' '*' '*' '*' '*' '*']
 ['*' '*' '*' '*' 'Q' '*' '*' '*']
 ['*' '*' '*' '*' '*' '*' '*' 'Q']
 ['*' '*' '*' '*' '*' 'Q' '*' '*']
 ['*' '*' 'Q' '*' '*' '*' '*' '*']
 ['*' '*' '*' '*' '*' '*' 'Q' '*']
 ['*' 'Q' '*' '*' '*' '*' '*' '*']
 ['*' '*' '*' 'Q' '*' '*' '*' '*']]

[['Q' '*' '*' '*' '*' '*' '*' '*']
 ['*' '*' '*' '*' '*' 'Q' '*' '*']
 ['*' '*' '*' '*' '*' '*' '*' 'Q']
 ['*' '*' 'Q' '*' '*' '*' '*' '*']
 ['*' '*' '*' '*' '*' '*' 'Q' '*']
 ['*' '*' '*' 'Q' '*' '*' '*' '*']
 ['*' 'Q' '*' '*' '*' '*' '*' '*']
 ['*' '*' '*' '*' 'Q' '*' '*' '*']]

[['Q' '*' '*' '*' '*' '*' '*' '*']
 ['*' '*' '*' '*' '*' '*' 'Q' '*']
 ['*' '*' '*' 'Q' '*' '*' '*' '*']
 ['*' '*' '*' '*' '*' 'Q' '*' '*']
 ['*' '*' '*' '*' '*' '*' '*' 'Q']
 ['*' 'Q' '*' '*' '*' '*' '*' '*']
 ['*' '*' '*' '*' 'Q' '*' '*' '*']
 ['*' '*' 'Q' '*' '*' '*' '*' '*']]
```

Screenshot of the first 3 solutions