Daily Coding Problem #54

Problem

This problem was asked by Dropbox.

Sudoku is a puzzle where you're given a partially-filled 9 by 9 grid with digits. The objective is to fill the grid with the constraint that every row, column, and box (3 by 3 subgrid) must contain all of the digits from 1 to 9.

Implement an efficient sudoku solver.

Solution

Trying brute force on a sudoku board will take a really long time: we will need to try every permutation of the numbers 1-9 for all the non-empty squares.

Let's try using backtracking to solve this problem instead. What we can do is try filling each empty cell one by one, and backtrack once we hit an invalid state.

To do this, we'll need an `valid_so_far` function that tests the board for its validity by checking all the rows, columns, and squares. Then we'll backtrack as usual:

```
X = None # Placeholder empty value

def sudoku(board):
    if is_complete(board):
        return board

    r, c = find_first_empty(board)
    # set r, c to a val from 1 to 9
    for i in range(1, 10):
        board[r][c] = i
        if valid_so_far(board):
            result = sudoku(board)
            if is_complete(result):
                return result
        board[r][c] = X
    return board

def is_complete(board):
    return all(all(val is not X for val in row) for row in board)

def find_first_empty(board):
    for i, row in enumerate(board):
        for j, val in enumerate(row):
            if val == X:
                return i, j
    return False
```

```
def valid_so_far(board):
    if not rows_valid(board):
        return False
    if not cols_valid(board):
        return False
    if not blocks_valid(board):
        return False
    return True

def rows_valid(board):
    for row in board:
        if duplicates(row):
            return False
    return True

def cols_valid(board):
    for j in range(len(board[0])):
        if duplicates([board[i][j] for i in range(len(board))]):
            return False
    return True

def blocks_valid(board):
    for i in range(0, 9, 3):
        for j in range(0, 9, 3):
            block = []
            for k in range(3):
                for l in range(3):
                    block.append(board[i + k][j + l])
            if duplicates(block):
                return False
    return True

def duplicates(arr):
    c = {}
    for val in arr:
        if val in c and val is not X:
            return True
        c[val] = True
    return False
```