Solving Sudoku Programmatically

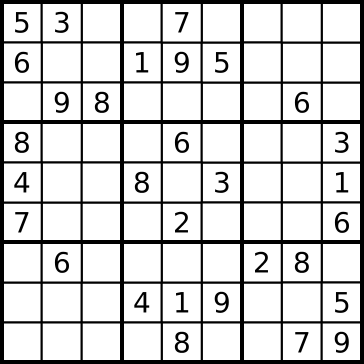
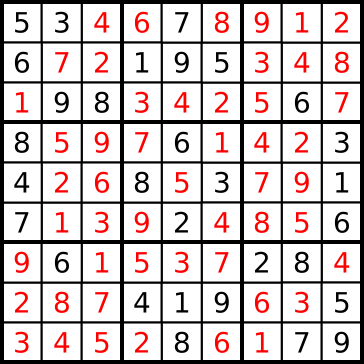
 Solving a Sudoku puzzle programmatically is a task that is both open ended and complex. Sudoku is a Japanese puzzle that usually consists of a 9 x 9 grid of squares divided into nine rows of nine squares, nine columns of nine squares, and nine 3 x 3 boxes, each with nine squares. Henceforth, these rows, columns, and boxes will all be referred to as groups, as the difference between them is not very important. Some, but not all, squares of an unsolved Sudoku puzzle are filled with the numbers 1 through 9. In order to solve a Sudoku problem, all 81 squares of the puzzle must be filled with a number 1 through 9 so that each group contains no repeated number.

Figure 2. Solved Sudoku Puzzle

Figure 1. Unsolved Sudoku Puzzle

In our Python code, a Sudoku puzzle will be represented by a string 81 characters long, where ‘.’s represent blank squares. For example, the unsolved Sudoku puzzle above would be represented by this string:’53..7....6..195....98....6.8...6...34.. 8.3..17...2...6.6....28....419..5....8..79’. Perhaps the most basic method of solving Sudoku is the recursive brute force method, which will also serve as the basis of our solver program. The brute force method, bruteForce, accepts a Sudoku puzzle string as an argument and first iterates through each position of the Sudoku puzzle string. If a position is blank (represented by a ‘.’), then bruteForce guesses a value between 1 and 9 to place in that position and returns the modified string. If the puzzle is still valid, meaning that there are no duplicate values in any group, then the recursive bruteForce method keeps inserting values into the puzzle until the puzzle becomes either invalid or solved. If the puzzle becomes invalid, then bruteForce returns a blank string and tries other values until the puzzle is solved. If the puzzle is solved, then bruteForce returns the solved puzzle string.

As one can imagine, this recursive brute force method is ponderously slow. It can be sped up and made more efficient by adding several modifications. Instead of selecting any value from 1-9 to place in a position, the brute force method can instead be made to only choose from values that would be valid if placed in that position. This is done by generating a dictionary named possible, in which the keys are the positions of blank squares in the puzzle string, represented by a number 0-80, every time bruteForce is called.