Brittany Lee
EECS 348 HW 2

1. Code
    a. Code in file 'sudoku.py'
    b. Wrote entire code myself.

2. Writeup
    a. Briefly explain your code's representation of each of the following, in 1-2 sentences each: the board, variables, constraints, and the states.
    b. The board is represented by the Board object, specifically a board array of rows (CurrentGameboard). The variables are the individual indicies of the arrays, a given row and column pair in the board array. The constraints are represented by three functions check_row, check_col, check_box. Each function checks that the row, column, or sub-box contains numbers zero through the size of the game board only once. The states are represented by each recursion through each solve function.

    c. How do you implement forward checking?
    Forward checking determines the first blank space and finds the domain of values that are available for that blank space. Then the function iterates through each value in the domain and recurses on the next blank space. If the function finds a conflict with a particular variable/value pair, then the function resets to the previous variable/value pair and continues looping through possible values.
    d. Time Table

Problem	Backtracking	Forward Checking	MRV + MCV	MRV + MCV + LCV
4 x 4	1.001e-05	1.693e-05	1.597e-05	1.597e-05
9 x 9	3.505e-05	2.098e-05	2.098e-05	5.221e-05
16 x 16	--	--	--	--
25 x 25	--	--	--	--

    e. Comment on your results in about 100 words: what trends do you see, in terms of scalability across the various heuristics? Are these as expected?
    Generally it seems that MRV + MCV and Forward Checking is faster than brute force backtracking and MRV + MCV + LCV. Brute force takes a long time because it checks every possible value for each row, col pair, and MRV + MCV + LCV takes a long time because it requires a lot of processing power. It seems that Backtracking, Forward Checking, and MRV + MCV scale steadily as the puzzle size gets larger, but MRV + MCV + LCV slows down dramatically as the puzzle increases. My implementations did not finish solving the 16x16 and the 25x25 puzzle, but these are the trends I would expect if they did finish. I expected the Forward Checking heuristic to slow down dramatically

because of the processing power needed, which I may have observed if 16x16 or 25x25 finished.

f. Implement the following heuristics. For each one, describe how you implemented it and evaluate it within the table in problem 3 above.
    i. MRV + most constraining variable:
       MRV is implemented by looking for the variable with shortest array of possible solutions. To find the MRV, create an array of all possible solutions and iterate through all the solutions to find the solution array of the shortest length. Then that row, col pair is returned to the backtracking function and is used to set the next value. In small puzzles, the MRV + MCV backtracking algorithm is slower because there is added processing power that takes place, but once the puzzles get larger it speeds up the process by finding a solution quicker.

    ii. MRV + most constraining variable + least constraining value:
       MRV is implemented in the same way as above, but after finding the row, col pair the LCV is used to choose which value to set the variable to. The LCV function looks at the possible domain values by checking the possible solutions for that row, col pair. The function then iterates through the possible solution values and sets the row, col pair with that value. The function finds all the possible solutions using this value and counts the number of solutions in that solution set. The function keeps track of the number of solutions in each value set and returns the value with the largest solution set. This is the value that imposes the least number of constraints on the remaining variables. In small puzzles, the MRV+ MCV + LCV algorithm only takes slightly longer than regular backtracking because of the added processing time. But doing LCV processing is time intensive and makes the backtracking algorithm much slower.