Python assignment 4

Sudoku

Sudoku is a mathematical puzzle game from Japan. A Sudoku puzzle contains a grid of 3 by 3 boxes, each box of which contains 3 by 3 squares. The puzzle thus comprises 9 by 9 squares in total. Initially some squares are filled with a number, and the challenge of the puzzle is to fill all squares with the numbers 1 to 9 such that: each row, column and box contain each number 1 to 9 exactly once.

In this assignment we are going to write a Sudoku puzzle solver.

Approach

In this assignment, we regard a cell with number 0 as an empty cell.

Start with the provided sudoku_template.py. We will build our solution step by step. It is not required to stick to these steps, but they may help.

1. Take a look at the `full_grid` variable. It contains a complete Sudoku puzzle. You can access each cell by using 2D-array indexing. E.g. to access cell on row 2 column 5 you can write `full_grid[1][4]`. You can manually set one or multiple cells to 0 when writing the assignment, it is easier to debug compared to the actual quiz.

2. In the process of solving we will look for an empty cell, try to fill a number and see if it fits, in other words, to see if it has no conflict. A conflict means: the same number occurs in the same row, column or box. The skeleton for a function named `is_conflict(grid, row_idx, column_idx, value)` has been made for you. This function should take the current `grid`, the coordinates (`row_idx, column_idx`) for the cell you want to fill, and the `value` for the cell. The function needs to calculate if the given value for the given position will result in a conflict or not, and return the result. It is suggested to breakdown this function into `row_conflict()`, `col_conflict()` and `box_conflict()`. Test these functions.

3. Function `get_empty_cell_index(grid)` is provided for you. It returns the coordinates of the first found empty cell (number in cell is 0), or `-1, -1` if there is no empty cell.

4. Complete the recursive function `solve(grid, answers)`. The `grid` parameter is the current Sudoku grid (2D-array), and `answers` is the list of all answers (list of 2D-array).

Our strategy for solving the puzzle is "brute force": we look for an empty cell, fill it with a number and see if it fits (`is_conflict()` returns False). If it doesn't fit, we try the next number. When we have found a number that fits, we will fill that number and call `solve(grid, answers)` to find the solutions for the new grid (because we filled in a number).

Note that when calling the `solve(grid, answers)`, it is essential to make a copy (use `nested_list_copy() or build your own`) the current grid and pass that copy to the function call. If you send your local grid rather than the copy to that recursive call, it will solve the current grid and stores it, which means you lost all other possible solutions.

5. Finally, print out the number of solutions.

Answer: `quiz` has 11 solutions.