COSC 130 – Project 02 Instructions

Introduction

The goal of this project is to develop an automated sudoku solver. If you are unfamiliar with sudoku puzzles, you can find a description of them by clicking on the following link: Rules for Sudoku

General Instructions

Create a Python script file named **`Sudoku.py`** and a Jupyter notebook file named **`Project_02_YourLastName.ipynb`**. You will use the script to define your functions and the notebook will be used to load the script and to test your functions.

Please download the files **`puzzle01.txt`**, **`puzzle02.txt`**, **`puzzle03.txt`**, and **`puzzle04.txt`**, storing these in the same directory as your script file and notebook file. It is important that these files are all in the same directory. Otherwise, your code will not run correctly when I run it.

You will represent a given state for a sudoku puzzle as a list of nine lists. Each of the 9 sublists will contain 9 elements, and will represent a single row in the puzzle. Empty cells will be indicated by storing a value of 0 in the location representing that cell. As an example, the list displayed on the left below would represent the puzzle displayed on the right.

```
puzzle = [
    [8, 0, 0, 0, 0, 0, 0, 0, 0],
    [0, 0, 3, 6, 0, 0, 0, 0, 0],
    [0, 7, 0, 0, 9, 0, 2, 0, 0],
    [0, 5, 0, 0, 0, 7, 0, 0, 0],
    [0, 0, 0, 0, 4, 5, 7, 0, 0],
    [0, 0, 0, 1, 0, 0, 0, 3, 0],
    [0, 0, 1, 0, 0, 0, 0, 6, 8],
    [0, 0, 8, 5, 0, 0, 0, 1, 0],
    [0, 9, 0, 0, 0, 0, 4, 0, 0]
]
```

```
+-------+-------+-------+
| 8 . . | . . . | . . . |
| . . 3 | 6 . . | . . . |
| . 7 . | . 9 . | 2 . . |
+-------+-------+-------+
| . 5 . | . . 7 | . . . |
| . . . | . 4 5 | 7 . . |
| . . . | 1 . . | . 3 . |
+-------+-------+-------+
| . . 1 | . . . | . 6 8 |
| . . 8 | 5 . . | . 1 . |
| . 9 . | . . . | 4 . . |
+-------+-------+-------+
```

Instructions for the Script File

Define functions with the following names: **`load_puzzle()`**, **`display_puzzle()`**, **`get_next()`**, **`get_options()`**, **`copy_puzzle()`**,and **`solve()`**. Descriptions of each of these functions are provided below.

`load_puzzle()`

This function should accept a single parameter named **`path`**. This parameter is expected to be a string indicating the path to a text file containing the initial state of a sudoku puzzle. The file will be expected to contain 9 lines of text, each of which contains 9 numbers separated by spaces. The function should open the file and process the lines within. The function should return a list of lists, with each sublist containing the 9 values on any one line of the file, stored as ints.

`display_puzzle()`

This function should accept a single parameter named **`puzzle`**. This parameter is expected to be a list of lists representing a puzzle state. The function should print the puzzle state using the format displayed above on the right.

`get_next()`

This function should accept two parameters named **`row`** and **`col`**. These parameters are expected to be integers 0 – 8 representing the location of a specific cell. The function should return two integers, which should be the row and column indices for the next cell in the puzzle, assuming we are moving through the rows left-to-right and top-to-bottom.

`get_options()`

This function should accept three parameters named **`puzzle`**, **`row`**, and **`col`**. The parameters **`row`** and **`col`** should be integers indicating the position of a cell in the puzzle represented by **`puzzle`**. If the cell contains a value, then the function should return **None**. If the cell is empty, then the function should return a list of valid numbers that could be placed in that cell. It should do this by scanning the row, column, and 3x3 block that contain the cell, eliminating from the possible options any values that appear in those collections of cells.

`copy_puzzle()`

This function should accept a single parameter named **`puzzle`**, which is intended to represent a puzzle state. The function should create a deep copy of the list **`puzzle`**. To accomplish this, the function should create an empty list named **`new_puzzles`**, and then append to this list copies of each of the sublists contained in **`puzzle`**. The function should return this new list.

`solve()`

This function should accept three parameters named **`puzzle`**, **`row`**, and **`col`**. The parameter **`puzzle`** is expected to be a list representing a puzzle state, while **`row`** and **`col`** are expected to be integers indicating the position of a specific cell in the puzzle. The default values of **`row`** and **`col`** should be set to zero. The function should return a list of lists representing the solved state of the puzzle, if one exists. If no solution exists, then the function should return **None**.

The function should find the solution to the puzzle recursively by following the steps below:

1. If the current cell being considered is not blank, then get the location of the row and column using **`get_next`**.
   - If we are at the last cell of the puzzle, then the solution should have been found. Return **`puzzle`**.
   - If we are not at the last cell of the puzzle, then call **`solve()`** again, passing it **`puzzle`** and the location of the next cell.
2. If the current cell is not blank, then get a list of possible values for that cell using **`get_options()`.**
   - If no options are available, then return **None**.
   - If there are options available, then for each possible option, perform the following steps:
     a. Use **`copy_puzzle()`** to create a copy of the puzzle named **`new_puzzle`**.
     b. Set the current cell in **new_puzzle** to be equal to the current option.
     c. Call **solve()** again, passing it **new_puzzle** and the location of the current cell. Store the value returned.
        - If the value returned by **`solve()`** is **None**, then do nothing. Using this value in the cell eventually led to an unsolvable state.
        - If the value returned by **`solve()`** was something other than **None**, then the function was able to find a solution along this path, and it will be stored in the return value. Return this value, cancelling any further iterations of the loop.

Instructions for the Notebook

Any set of instructions you see in this document with an orange bar to the left will indicate a place where you should create a markdown cell. If no instructions are provided regarding formatting, then the text should be unformatted.

Any set of instructions you see with a blue bar to the left will provide instructions for creating a single code cell.

Assignment Header

Create a markdown cell with a level 1 header that reads: "COSC – Project 03". Add your name as a level 3 header.

Introduction

Create a markdown cell with a level 2 header that reads "Introduction". Add unformatted text explaining the purpose of this project. You may paraphrase the statement in the first paragraph of this document. Explain that you will be by running the script in which you have created the functions that you will be using.

We will start by running the script.

Use the Magic command **`%run -i Sudoku.py`** to run the contents of your script.

Puzzle 1

Create a markdown cell with a level 2 header that reads "**Puzzle 1**". Add unformatted text explaining that you will test your function on the puzzle whose initial state is stored in the file **`puzzle01.txt`**, and that you will begin by loading the puzzle from the file.

Use **`load_puzzle()`** to load the contents of **`puzzle01.txt`** into a list named **`puzzle01`**. Use **`display_puzzle()`** to print the puzzle.

Add a markdown cell explaining that you will now attempt to find a solution to the puzzle.

Use **`solve()`** to attempt to find the solution of the puzzle stored in **`puzzle01`**. If a solution exists, then use **`display_puzzle()`** to print the solution. Otherwise, print the text “`No solution exists for this puzzle.`”

Puzzle 2, Puzzle 3, and Puzzle 4

Repeat the steps detailed for Puzzle 1, but using the information stored in the files **`puzzle02.txt`**, **`puzzle03.txt`**, and **`puzzle04.txt`**.

Submission Instructions

When you are done, click **Kernel > Restart and Run All**. If any cell produces an error, then manually run every cell after that one, in order. Save your notebook, and then export the notebook as an HTML file. Upload the HTML file to Canvas and upload the IPYNB file to CoCalc, placing the file into the folder **Projects/Project 02**.