CMSC 201 Fall 2018
Project 3 – Sudoku

Assignment: Project 3 – Sudoku
Due Date:
Design Document: Tuesday, December 4th, 2018 by 8:59:59 PM
Project: Tuesday, December 11th, 2018 by 8:59:59 PM
Value: 80 points

Collaboration: For Project 3, **collaboration is not allowed** – you must work individually. You may still come to office hours for help, but you may not work with any other CMSC 201 students.

Make sure that you have a complete file header comment at the top of each file, and that all of the information is correctly filled out.

```
# File:    FILENAME.py
# Author:  YOUR NAME
# Date:    THE DATE
# Section: YOUR DISCUSSION SECTION NUMBER
# E-mail:  YOUR_EMAIL@umbc.edu
# Description:
#    DESCRIPTION OF WHAT THE PROGRAM DOES
```

For Project 3 you will have to turn in a "design document" in addition to the actual code. The design document is intended to help you practice deliberate construction of your program and how it will work, rather than coding as you go along, or starting without a plan.

Instructions

For this project, you will be creating a single program, but one that is bigger in size and complexity than any individual homework problem. This assignment will focus on manipulating lists, calling functions, and recursion. For this assignment, more than any other this semester, **planning ahead and designing your program will be very, *very* important**!

The design for Project 3 is entirely up to you – suggestions are provided within the project description, but you are not required to use them.

At the end, your Project 3 file must run without any errors.
It must also be called proj3.py (case sensitive).

Additional Instructions – Creating the proj3 Directory

During the semester, you'll want to keep your different Python programs organized, organizing them in appropriately named folders (also known as directories).

You should create a directory in which to store your Project 3 files. We recommend calling it **`proj3`**, and creating it inside a newly-created directory called **`Projects`** inside the **`201`** directory.

If you need help on how to do this, refer back to the detailed instructions in Homework 1.

Objective

Project 3 is designed to give you practice with two-dimensional lists, creating and calling functions, file I/O, and recursion. You'll need to use practically everything you've learned so far, and will need to do some serious thinking about how all of the pieces you need to create should fit together.

Task

You will be programming up a text-based version of the popular Sudoku puzzle game. The goal of the game is to place the digits from 1-9 in each cell of the board such that no two digits repeat in the same row, column, or 3x3 square (it's called a nonet!). The game is won if you are able to find this unique combination of digit placement, given some starting configuration of digits on the board.

You can read more about the game on the Wikipedia page, and you can play an online version of the game on this webpage. You should also take a look at the sample outputs for examples of how the game is played, won, and lost.

Your program will need to:

- Read in and store a sudoku puzzle from a file
- Solve the sudoku puzzle
- Allow the user to play numbers in the puzzle
- Allow the user to undo previous numbers placed in the puzzle
- Check if the numbers entered are correct as the user plays
- Check if the user won or lost their sudoku game after filling up the board

Your program will also need to do the following, but these functions have been provided for you, so you'll only need to call them in **`main()`**:

- Allow the user to save their sudoku puzzle progress
- Print the sudoku puzzle to the user's screen

Specification

Prior to this assignment, **you should be familiar with the entirety of the Coding Standards**, available on Blackboard under “Assignments” and linked on the course website at the top of the “Assignments” page.

You should be commenting your code, and using constants in your code (not magic numbers or strings).
Any numbers other than 0 or 1 are magic numbers!

You will **lose major points** if you do not follow the 201 coding standards.

If you have questions about commenting, whitespace, or any other coding standards, please come to office hours.

Additional Specifications

For this assignment, **you must use recursion to produce a solution to the sudoku puzzle.** No other functions are required to be recursive. You also must create and call **at least eight individual functions**, ***not*** including `main()` or the two functions provided for you. All other design decisions are up to you.

For this assignment, you do need to worry about “input validation.” You may assume that the user will enter the correct type of input (for example, an integer if one is asked for, a float if one is asked for), but the input may be negative, outside of the allowable range, or “bogus” (in the case of strings).

If the user enters a different type of data than what you asked for, your program may crash. This is acceptable.

It is also acceptable if your program crashes when a filename is entered that either does not exist, or has the wrong formatting for the sudoku board.

Details

The program starts by asking the user for the filename that contains the sudoku puzzle and reads in the file. It then asks the user if they want to play the game, or just solve the puzzle. The game can be played with or without correctness checking, which come in the form of pointing out invalid moves or filling in a cell correctly. The user wins if they fill up the board and it matches the solved puzzle, otherwise they lose. The game stops when the board fills up.

Reading in and Creating the Board

A puzzle file will have 9 lines of 9 comma separated values, where each line represents the next line in the puzzle. When the board is read in from the file, it will only contain these characters:

• A digit between 1 and 9	“**`6`**”	the number (6) belongs in this cell
• The digit 0	“**`0`**”	represents an empty cell
• A comma	“**`,`**”	separates cells in the same row

Your program must read this file and store the puzzle, so it can be manipulated and displayed to the user while they are playing Sudoku.

Displaying the Board

The board should be displayed to the user as shown below. Any empty cells should be represented with an underscore character. The row numbers should be displayed next to the corresponding row, and the column number should be displayed above the corresponding column.

puzzle.txt

```
8,9,0,0,0,5,3,7,0
3,1,0,0,7,0,5,0,0
0,0,0,0,3,9,8,0,2
0,6,0,0,0,3,7,0,8
0,5,0,0,0,0,0,2,0
9,0,7,6,0,0,0,5,0
5,0,6,1,8,0,0,0,0
0,0,9,0,5,0,0,6,4
0,2,3,9,0,0,0,8,5
```

board as displayed to user

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

We have provided a `prettyPrint()` function for you, that will print the board with the row and column numbers, and with the board spaced out to look more like a square (as seen on the right above).

You can get it from Prof. Neary's pub folder using the command:

```
cp /afs/umbc.edu/users/m/n/mneary1/pub/cs201/starterFxns.py .
```

Playing the Game

After loading the sudoku puzzle from a file, the user has the option to play the game, or to immediately solve the puzzle.

If the user chooses to solve the puzzle

- Your program must compute the solution to the puzzle
    - The solution must be displayed to the user
- Your program should exit

If the user chooses to play the game,

- They must first be asked if they want to play the game with correctness checking (this is covered in a later section)
- Your program should continuously ask the user if they want to:
    - Make a new move by placing a number, undo their previous move, save the puzzle state to a file, or quit
        - Users are not permitted to change an already entered number (they may only undo)
    - If they are playing with correctness checking, the program will behave a bit differently when they try to place a number in the incorrect spot
    - Each of these options are covered in detail on the next few pages
- Continuously prompt the user until they fill up the board or quit

The game is won when all of the cells have been correctly filled in according to the rules of Sudoku.

Making a Move

If the user chooses to place a number in the Sudoku puzzle, you must ask them for the following information:

- the row of the cell they want to fill in
- the column of the cell they want to fill in
- the number they want to place in that cell

You must validate the row and column values to be sure that they do not try to fill a cell that is out of bounds (there are only 9 rows, and 9 columns).

You must also validate the number they want to play, as the only valid numbers that can be placed in a Sudoku puzzle are 1 to 9.

The user is not allowed to place a number in a cell that already has a number in it. In other words, they cannot replace one number with another.

The user is not allowed to place a number in a cell that would violate any of the three rules of Sudoku (this is not the same as correctness checking):

- cannot play a number that is already in the same square (nonet!)
- cannot play a number that is already in the same row
- cannot play a number that is already in the same column

Undoing a Move

If the user decides to undo a move, you only need to change their last move back to an empty cell. The undo process always happens one move/cell at a time. However, the user should be able to undo every move they made since they started playing the game. If there are no moves to undo, the user should be alerted that they cannot go back further.

HINT: As the user places numbers, it would be a good idea to append each row and column value they use to a list where you are keeping track of their moves. If a user wants to undo, refer to the last element in this list to get their last move!

Saving the Game

The user should be able to save their Sudoku puzzle in between making or undoing a move. When the user decides to save, your program should write the current state of the sudoku puzzle to a file. It must ask the user for the name of a file to save the game to.

This **`savePuzzle()`** function has been provided for you, and can be copied from Prof. Neary's pub folder using the command:

```
cp /afs/umbc.edu/users/m/n/mneary1/pub/cs201/starterFxns.py .
```

You must use the provided **`savePuzzle()`** function ***exactly*** as it is provided to you. If you alter the function or the format in which the file is saved, you will lose **major** points.

Correctness Checking

The user is able to play the game with correctness checking. When this option is enabled, the user should not be allowed to make any move that is incorrect according to the solution. Each number they want to place should be compared against the number in the same position of the solved version of that puzzle. If the numbers don't match, the user should be alerted that number does not go in that position. No incorrect value should ever be placed in a puzzle while correctness checking is turned on.

NOTE: If correctness checking is not turned on, it is still the case that only moves that follow the rules of Sudoku should be allowed.

Solving a Sudoku Puzzle

There are quite a few strategies that (human) expert sudoku-solvers use to conquer super hard puzzles. These strategies require a lot of logic rules, which would be pretty tedious to put into our program. Luckily for us, we don't have to worry about coding up all the different strategies if we can understand how a Sudoku puzzle is actually recursive!

Think about it this way: each time you put a number in a cell, you are shrinking the number of cells that are left to fill in. That means that all you have to do to solve a Sudoku is put a number in a cell, and then solve the rest of the puzzle. (That's the recursion!) We can come up with the solution to any puzzle by filling in the first empty cell we find with a number, and then trying to solve that new, smaller, puzzle.

Of course, it is a little bit more complicated than that. What happens if you reach a point, after filling in some cells in the current puzzle, where no number can be put into the next empty cell in the puzzle? How can you find the next empty cell to try to fill in the puzzle? Should you try every number from 1 to 9 in that empty cell, or can you discount some numbers based on the rules of Sudoku?

This part of the project must be implemented using recursion. As always, think about what the base case(s) should be, and what the recursive case(s) should be. The questions posed in the previous paragraph should offer some insight into what the base case(s) and recursive case(s) are.

IMPORTANT: You should only have to generate the solution to the puzzle once! You should solve the puzzle after it is loaded, save the result, and use that solved puzzle everywhere in your code that requires the solution.

You will be provided with solved versions of the test puzzles you are given so that you can test components of your project that rely on a solved puzzle without waiting until you can solve the sudoku puzzle recursively.

Additional Information and Examples

For more information, look at the sample output files available. They contain an example of the required input validation, how moves work, how undoing works, how correctness checking works, and a game that is won (you can't lose).

You can download all of the sample output by using the following command:

```
     cp /afs/umbc.edu/users/m/n/mneary1/pub/cs201/sample* .
```

You can download all of the puzzles used in the sample output (including their solutions, so you can test correctness checking without needing to finishing solving) by using the following command:

```
     cp /afs/umbc.edu/users/m/n/mneary1/pub/cs201/puzzle* .
```

You are also highly encouraged to make your own test boards – you can create and share these with your classmates if you want as well.

Points

The project is worth a total of 80 points. Of those points, 10 will be based on your design document (creation of it and following it), 10 will be based on following the coding standards, and the other 60 will be based on the functionality and completeness of your project.

Design Document

The design document will ensure that you begin seriously thinking about your project way early on. This will not only give you important experience doing design work, but will help you gauge the number of hours you'll need to set aside to be able to complete the project. **Your design document must be called design3.txt.**

For Project 3, you are creating the design entirely on your own.
You **may NOT work with another student** to "brainstorm" a solution or discuss any general approaches or requirements. If you need assistance with the design document, come to office hours.

Your design document must have four separate parts:

1. A file header, similar to those for your assignments
2. Constants
    a. A list of all the constants your program will need, including a short comment describing what each "group" of constants is for
3. Function headers
    a. A complete function header comment for each function your plan to create, including the description, inputs, and outputs
4. Pseudocode for **`main()`**
    a. A brief but descriptive breakdown of the steps your **`main()`** function will take to completely solve the problem; note function calls under the relevant comment (if applicable)

Your design can follow the same general format as the design for Project 1.

Your **`design3.txt`** file will be compared to the **`proj3.py`** file that you submit. Minor changes to the design are allowed. A minor change might be the addition of another function, or a small change to **`main()`**.

Major changes between the design and your project will lose you points. This would indicate that you didn't give sufficient thought to your design.
(If your submitted design doesn't work, it is generally better to lose the points on the design, and to have a functional program, rather than turning in a broken program that follows the design. The ultimate decision is up to you.)

Submitting

Once your **`proj3.py`** or **`design3.txt`** file is complete, it is time to turn it in with the **`submit`** command. (You may also turn the design or project in multiple times, as you reach new milestones or complete each piece. To do so, run **`submit`** as normal.)

To submit your design file (which is due Tuesday, December 4th, 2018 by 8:59:59 PM), use the command:

```
linux1[4]% submit cs201 PROJ3_DESIGN design3.txt
Submitting design3.txt...OK
linux1[5]%
```

To submit your project file (which is due Tuesday, December 11th, 2018 by 8:59:59 PM), use the command:

```
linux1[4]% submit cs201 PROJ3 proj3.py
Submitting proj3.py...OK
linux1[5]%
```

If you don't get a confirmation like the one above, check that you have not made any typos or errors in the command.

You can check that your project and/or design was submitted by following the directions in Homework 0. Double-check that you submitted your files correctly, since **an empty file will result in a grade of zero for this assignment.**