CS 120 (Spring 21): Introduction to Computer Programming II

Short Project #4

due at 5pm, Thu 11 Feb 2021

1 Overview

Our Long project is going to be a tool which will help people solve Sudoku puzzles (`https://en.wikipedia.org/wiki/Sudoku`). In this Short project, you will write some utility functions which you may find useful in the Long project. (Some of them you may be able to use as-is, some of them you may have to adapt.)

You will implement each of these as functions in the file `grid_funcs.py` .

2 Terminology and Conventions

In this Short and Long project, we'll often be talking about "rows," "columns," and "squares."

- A row is a horizontal line; when receiving input from the user (or printing output to the user), we will number from 1-9, with 1 on the top.

  However, in the funtions below, if I ask for a certain row, note that I will number them **0 through 8, inclusive!**

- A column is a vertical line; when interacting with the user, 1 is the leftmost line, and 9 is the rightmost. (As with rows, internally we will call these columns 0-8.)

- A "square" is a 3x3 subregion of the board; we identify them with (sx, sy) coordinates. $(0, 0)$ is the upper-left square; $(2, 0)$ is the upper-right.

(spec continues on next page)

- When printing out the board (or reading it in from an input file), we will use blank lines and spaces to make it easy to see the sub-regions of the board, and we will print periods for not-yet-set spaces. Thus, the example board from Wikipedia would look like this:

```
53. .7. ...
6.. 195 ...
.98 ... .6.

8.. .6. ..3
4.. 8.3 ..1
7.. .2. ..6

.6. ... 28.
... 419 ..5
... .8. .79
```

  (Use blank lines - not lots of spaces - for the vertical spacing.)

3 Required Functions

Implement each of these functions in the file `grid_funcs.py` :

3.1 `arr_of_strs_to_2d_array(strs)`

Implement a function that takes one parameter, which will be an array of strings. It converts the data into a 2d array of single characters. But it's not as simple as just calling `list()` to convert each string in turn - we want to be able to access the elements of the array like this:

```
grid = ...
some_value = grid[x][y]
```

In order for a 2d array to work this way, we have to use the first index to represent the x value, and the second to represent the y. This means that we must have organize our data into an **array of columns.**

To understand this, let's use a small example of a 4x4 grid. (This problem will actually use 9x9 grids, but those are really annoying to draw!) Here's the grid, as you might print it out to the user:

```
abcd
....
x..y
1234
```

Here's what the input would look like, as an array of strings:

```
strs = [ "abcd",
         "....",
         "x..y",
         "1234" ]
```

While this is very handy for printing, it's not very handy for code - because when you access `strs[0]` you are accessing the first row, not the first column.

For the 2d array that you return, You will need to organize it as follows:

```
grid = [ ['a', '.', 'x', '1'],
         ['b', '.', '.', '2']
         ['c', '.', '.', '3']
         ['d', '.', 'y', '4'] ]
```

The cool thing about this form is that `grid[0]` gives you the first column, and `grid[0][3]` is the bottom-left corner.

3.1.1 Details

- You may assume that the input is an array of exactly 9 strings, and that each string is exactly 9 characters long.
- Do not make any assumptions about what characters are in the strings; while we're practicing for Sudoku, I can send any string I want.

3.1.2 Hints

- One of the easiest way to build an array is to start with `[]`, and then use `append()` repeatedly in a loop.
- Yes, it's possible to append an array into another array - that's a great way to build nested arrays.
- Nested loops will be your friend. I make a lot of use of variants on the following:

```
for x in range(9):
    for y in range(9):
```

4 print_grid(grid)

This function has one parmater, which is a 2d array of integers. It will always be exactly 9x9 in size, it will be organized as an "array of columns," as we discussed in the previous function - meaning that you can index into the array with the syntax `grid[x][y]`.

Each of the values stored in the array will be integers, in the range 0 through 9 (inclusive). When you print out the values, you should out a period every place the array has a zero; otherwise, print out the integer you've been given.

You will print out the 9x9 values in the standard Sudoku shape (see above). So, you will need to add spaces in two places on each line, to give empty columns, and blank lines to give empty rows.

5 dup_grid(grid)

This function should duplicate a 2d array of integers. You may assume that it is 9x9 in size.

It's important to make sure that the internal arrays you return are **not** aliases of the original data - they need to be new arrays! To check that, try out the following code snippet:

```
orig = ... some grid ...
copy = dup_grid(grid)
copy[0][0] = 9
print(orig)
print(copy)     # should be different!
```

6 get_col(grid, x)

This function takes a 9x9 grid. Don't make any assumptions about what sort of data it contains, but you can assume that it's arranged in the "array of columns" format.

The `x` parameter is the column number; it will be in the range 0 through 8 (inclusive).

Return all of the values in that column. Note that I don't care what order you return them, since I will be sorting your return values before I check them.

7 get_row(grid, y)

This function is just like `get_col()`, except that this returns all of the values in a row.

8 get_square(grid, sx,sy)

This function is just like `get_col()`, except that it returns the 9 values inside one of the "squares" of the grid. Remember, $(0,0)$ is the upper-left square, and $(2,0)$ is the upper-right.

9 Turning in Your Solution

You must turn in your code using GradeScope.