Project

Sudoku Solver

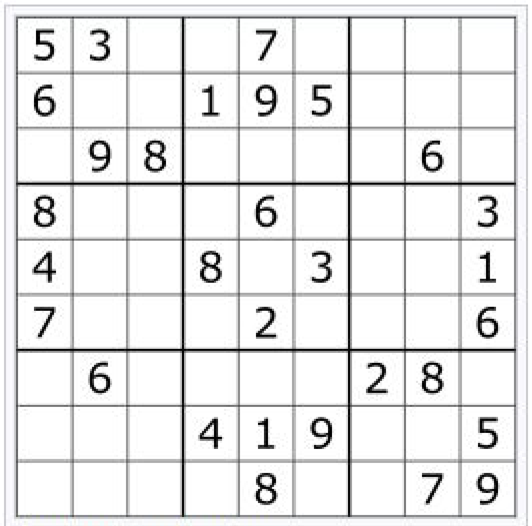
Summary

For this project, you will be writing a function to solve a Sudoku puzzle. After going through all the lessons of this course, you are now well equipped to tackle this challenge. We will break down the Sudoku Solver into pieces so that it is more manageable. Each problem will be working towards creating the final Sudoku Solver function.

Problem Description

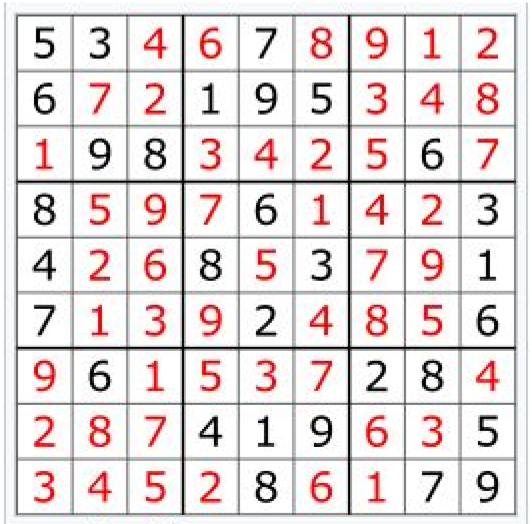
What is Sudoku?

In classic sudoku, the objective is to fill a 9×9 grid with digits so that each column, each row, and each of the nine 3×3 subgrids that compose the grid (also called "boxes", "blocks", or "regions") contain all of the digits from 1 to 9. The puzzle setter provides a partially completed grid, which for a well-posed puzzle has a single solution. Here is an example provided from Wikipedia:



5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

A typical Sudoku puzzle...



5	3	4	6	7	8	9	1	2
6	7	2	1	9	5	3	4	8
1	9	8	3	4	2	5	6	7
8	5	9	7	6	1	4	2	3
4	2	6	8	5	3	7	9	1
7	1	3	9	2	4	8	5	6
9	6	1	5	3	7	2	8	4
2	8	7	4	1	9	6	3	5
3	4	5	2	8	6	1	7	9

...and its solution

Format of the coding problem

You will be given a list of lists of numbers, which will represent the board. For example:

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

This board represents the example shown above, where `0`s represent empty cells that you have to fill.

You will be writing the final function of this form:

```
def solve(board):
    # your code here
    return
```

The output should be a list of lists of the board but without and `0`s and completely filled out with numbers. Note that you will also implement `print_board` function which will print it as a matrix which will make it easier to read and debug your board.

```
> solution = solve(board)
> solution
[[5, 3, 4, 6, 7, 8, 9, 1, 2], [6, 7, 2, 1, 9, 5, 3, 4, 8], [1, 9, 8, 3, 4,
2, 5, 6, 7], [8, 5
, 9, 7, 6, 1, 4, 2, 3], [4, 2, 6, 8, 5, 3, 7, 9, 1], [7, 1, 3, 9, 2, 4, 8,
5, 6], [9, 6, 1, 5
, 3, 7, 2, 8, 4], [2, 8, 7, 4, 1, 9, 6, 3, 5], [3, 4, 5, 2, 8, 6, 1, 7, 9]]
> print_board(solution)
5 3 4 6 7 8 9 1 2
6 7 2 1 9 5 3 4 8
1 9 8 3 4 2 5 6 7
8 5 9 7 6 1 4 2 3
4 2 6 8 5 3 7 9 1
7 1 3 9 2 4 8 5 6
9 6 1 5 3 7 2 8 4
2 8 7 4 1 9 6 3 5
3 4 5 2 8 6 1 7 9
```

Problem 1: Print the Sudoku Board

Write a function that prints the board as a matrix:

```
def print_board(board):
    # your code here
    return
```

Here is what the output should look like:

```
> board = [[5, 3, 0, 0, 7, 0, 0, 0, 0],
           [6, 0, 0, 1, 9, 5, 0, 0, 0],
           [0, 9, 8, 0, 0, 0, 0, 6, 0],
           [8, 0, 0, 0, 6, 0, 0, 0, 3],
           [4, 0, 0, 8, 0, 3, 0, 0, 1],
           [7, 0, 0, 0, 2, 0, 0, 0, 6],
           [0, 6, 0, 0, 0, 0, 2, 8, 0],
           [0, 0, 0, 4, 1, 9, 0, 0, 5],
           [0, 0, 0, 0, 8, 0, 0, 7, 9]]
> print_board(board)
5 3 0 0 7 0 0 0 0
6 0 0 1 9 5 0 0 0
0 9 8 0 0 0 0 6 0
8 0 0 0 6 0 0 0 3
4 0 0 8 0 3 0 0 1
7 0 0 0 2 0 0 0 6
0 6 0 0 0 0 2 8 0
0 0 0 4 1 9 0 0 5
0 0 0 0 8 0 0 7 9
```

Problem 2: Find empty cell

Write a function that finds the first empty cell. This will be useful for your function to know which cell to try to fill. In our case, let's read the board from left to right, and top down. So in our example, the first cell is the cell in the first row and on the 3rd column (shown in **red**):

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

Here is the signature of the function you should write

```
def find_zero(board):
    # your code here
    return
```

The output of that function should be a list of size 2. The first element of the list is the row number and the second element of the list is the col number.

```
> board = [[5, 3, 0, 0, 7, 0, 0, 0, 0],
           [6, 0, 0, 1, 9, 5, 0, 0, 0],
           [0, 9, 8, 0, 0, 0, 0, 6, 0],
           [8, 0, 0, 0, 6, 0, 0, 0, 3],
           [4, 0, 0, 8, 0, 3, 0, 0, 1],
           [7, 0, 0, 0, 2, 0, 0, 0, 6],
           [0, 6, 0, 0, 0, 0, 2, 8, 0],
           [0, 0, 0, 4, 1, 9, 0, 0, 5],
           [0, 0, 0, 0, 8, 0, 0, 7, 9]]
> find_zero(board)
[0, 2]
```

Problem 3: Is Valid Placement?

Write a function that takes in the board, the position (row and column) and a value (1 to 9) and determines if it's legal to place that number on the cell. To do that, you will check the entire row to see if there is a number that matches your value. If there is, then you cannot place that value on that cell. You will do the same with the entire column, and then you will do the same with the 3x3 grid. The output should simply be a Boolean value.

Here is the signature of the function you should write

```
def is_valid(board, row, col, value):
    # your code here
    return
```

The output of the function should be True or False. True meaning you can place that value on that cell. Here I have highlighted the cell in **RED** for the example:

```
> board = [[5, 3, 0, 0, 7, 0, 0, 0, 0],
           [6, 0, 0, 1, 9, 5, 0, 0, 0],
           [0, 9, 8, 0, 0, 0, 0, 6, 0],
           [8, 0, 0, 0, 6, 0, 0, 0, 3],
           [4, 0, 0, 8, 0, 3, 0, 0, 1],
           [7, 0, 0, 0, 2, 0, 0, 0, 6],
           [0, 6, 0, 0, 0, 0, 2, 8, 0],
           [0, 0, 0, 4, 1, 9, 0, 0, 5],
           [0, 0, 0, 0, 8, 0, 0, 7, 9]]
> is_valid(board, 0, 2, 1)
True
> is_valid(board, 0, 2, 7)
False # There is a 7 on the same row
> is_valid(board, 0, 2, 6)
False # There is a 6 on the same 3x3 grid
> is_valid(board, 3, 5, 9)
False # Look at the green example, there is a 9 on the same column
```

Problem 4: Solve the Sudoku Problem

Finally with those functions you defined in the previous problem, you will use them together to solve your final problem. This is a backtracking problem, so you will be using recursion to solve this problem. The key is to fill out the `board` and use it as your partial solution. Here is the general solution which you will code yourself:

Base Cases

- Use your function `find_zero` to find the first empty cell. If there are no more 0s on the board. You are done! You solve it, return your board

Recursive Case

- If there is an empty cell (`0`) then check if you can put a value (`1` to `9`, but start with `1`) on there. You can check if it's a valid placement by using your function `is_valid`.
    - If it's not a valid placement, try then try the next number and do this all over again
    - If it is a valid placement, assign it to that value
        - Recurse by solving the updated board (since you assigned a cell a new value)
            - If there is a solution, return that solution
            - If there isn't a solution, assign that cell back to 0 to backtrack and do this all over again with the next number.

Here is the signature of the function you should write

```
def solve(board):
    # your code here
    return
```

The output should be a list of lists of the board but without and `0`s and completely filled out with numbers. Note that you will also implement `print_board` function which will print it as a matrix which will make it easier to read and debug your board. **NOTE: If there are no solution to the Sudoku return None**

```
> solution = solve(board)
> solution
[[5, 3, 4, 6, 7, 8, 9, 1, 2], [6, 7, 2, 1, 9, 5, 3, 4, 8], [1, 9, 8, 3, 4,
2, 5, 6, 7], [8, 5
, 9, 7, 6, 1, 4, 2, 3], [4, 2, 6, 8, 5, 3, 7, 9, 1], [7, 1, 3, 9, 2, 4, 8,
5, 6], [9, 6, 1, 5
, 3, 7, 2, 8, 4], [2, 8, 7, 4, 1, 9, 6, 3, 5], [3, 4, 5, 2, 8, 6, 1, 7, 9]]
> print_board(solution)
5 3 4 6 7 8 9 1 2
6 7 2 1 9 5 3 4 8
1 9 8 3 4 2 5 6 7
8 5 9 7 6 1 4 2 3
4 2 6 8 5 3 7 9 1
7 1 3 9 2 4 8 5 6
9 6 1 5 3 7 2 8 4
2 8 7 4 1 9 6 3 5
3 4 5 2 8 6 1 7 9
```

Problem 5 (optional): Print Pretty

Just for fun, rewrite the print_board function to print it in a nicer format

```python
def print_board(board):
    # your code here
    return
```

Here is what the output should look like:

```
> board = [[5, 3, 0, 0, 7, 0, 0, 0, 0],
           [6, 0, 0, 1, 9, 5, 0, 0, 0],
           [0, 9, 8, 0, 0, 0, 0, 6, 0],
           [8, 0, 0, 0, 6, 0, 0, 0, 3],
           [4, 0, 0, 8, 0, 3, 0, 0, 1],
           [7, 0, 0, 0, 2, 0, 0, 0, 6],
           [0, 6, 0, 0, 0, 0, 2, 8, 0],
           [0, 0, 0, 4, 1, 9, 0, 0, 5],
           [0, 0, 0, 0, 8, 0, 0, 7, 9]]
> print_board(board)
-------------------------
| 5 3   |   7   |       |
| 6     | 1 9 5 |       |
|   9 8 |       |   6   |
-------------------------
| 8     |   6   |     3 |
| 4     | 8   3 |     1 |
| 7     |   2   |     6 |
-------------------------
|   6   |       | 2 8   |
|       | 4 1 9 |     5 |
|       |   8   |   7 9 |
-------------------------
```