WITHYOU
WITHME

Python Programming Fundamentals

Assignment

Scenario

Your puzzle-loving (and insufferable) Uncle has long claimed that Sudoku puzzles are the peak of the puzzle solving art, and no more true a test of intelligence exists. Incensed, you have decided to prove him wrong and create a simple program that can solve Sudoku puzzles with ease, to demonstrate that he doesn’t need to waste any more of his time.

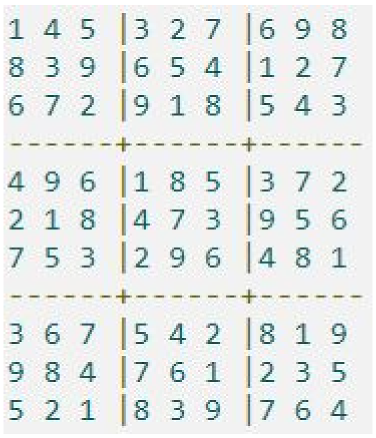
So what is a Sudoku, other than a “denial of service attack on human intellect”?. The Sudoku is a Latin Square puzzle that first appeared in French newspapers in the late 19th century. Appearing as a 9x9 square containing 9 smaller 3x3 squares, the goal of a Sudoku is to ensure every row, column, and smaller 3x3 square contains each of the numbers 1 - 9 exactly once. Typically, Sudoku puzzles start with a number of squares already containing numbers, clues from which the solver must use to divine the correct placement of every other number until all 81 squares are filled. Below is an example of an unsolved puzzle:

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

Task

In the included attachment is a number of Sudoku puzzles, each represented as nine nine-character strings, each on a separate line. Blank spaces are represented with the number 0, spaces with a number are represented by that number.

Your task is to develop a short Python program that is capable of solving them correctly, outputting the completed puzzle in a separate text document, formatted like so:



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

You are not required to extend the program with a GUI or an interface for a human to solve the puzzle, although if this is a problem that appeals to you the library PyGame is a great place to start.

Your program should be able to read a text document of an arbitrary length, formatted as the one attached, and for each puzzle, output a correct solution.

There are several ways to solve a Sudoku puzzle, chief among them backtracking (a brute force method of solving them). You are not required to use this method and can do whatever you prefer - treat it as an exact cover problem, or constraint propagation for example. There are a lot of potential avenues to solutions presented online. The technique you use to solve the puzzles doesn't matter, only the results will be tested.

A solution that has been copied and pasted off of the internet will not be a pass.

Your program will be required to pass the test puzzles provided to you, as well as a number of puzzles hidden from you that will only be used to test your submission. This is in order to prevent students hard-coding a solution.

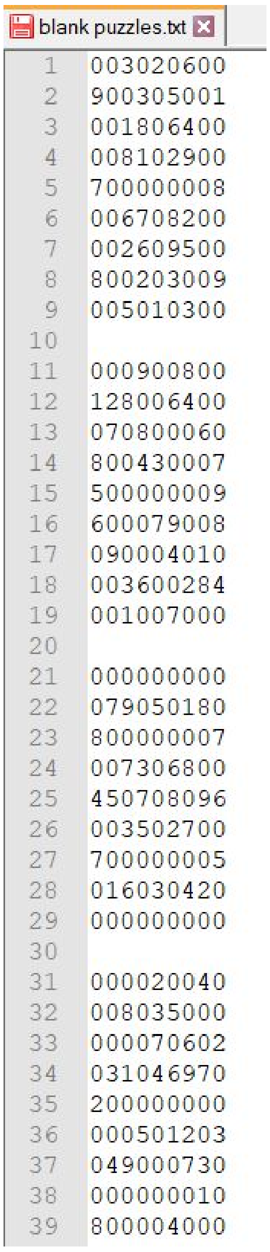
You may wish to test your program with more puzzles than those we've provided to you. You can find several, as well as answers to test your solutions online. Websudoku and Sudoku.com are both good sites to start with.

Submission

Submit your finished Python code as a single zipped package. All the modules you use need to be contained within this zip. You don't need to submit any of your test cases.

Blank Puzzles

The raw text file can be found in the assignment zip, but is reproduced below to give you an idea of how the input is formatted.



```
003020600
900305001
001806400
008102900
700000008
006708200
002609500
800203009
005010300

000900800
128006400
070800060
800430007
500000009
600079008
090004010
003600284
001007000

000000000
079050180
800000007
007306800
450708096
003502700
700000005
016030420
000000000

000020040
008035000
000070602
031046970
200000000
000501203
049000730
000000010
800004000
```

The minimum possible score for the assignment is 0. The maximum possible score is 10, and the pass mark is a 7 out of 10.

Criterion	**Below Standard**	**At Standard**	**Above Standard**
White space and following PEP 8 Style Guide	Code isn't spaced out, crammed in, is hard to read. No consistent styling throughout the code (0 points)	Some effort has been made to make code readable. Styled consistently throughout. Follows some PEP 8 guidelines. (1 point)	Code follows PEP8 guidelines (2 points)
Comments	No comments in the code, or the comments don't adequately explain what the code is doing, or there are too many comments and they obfuscate the code (0 points)	Comments explain function behaviour (1 point)	Comments are sufficient, without being obfuscating. In line with PEP8 guidelines. (2 points)
File Handling	Submission doesn't read from a file. Submission doesn't write to a file (0 points)	Submission reads from a file, or writes to a file, but not both. Submission file handling is unsafe. Submission writes to the file in an incorrect format (1 point)	Code correctly reads and writes to files, and handles files in a safe manner (2 points)
Problem Solving	Code fails to solve even the most basic sudoku puzzles within a reasonable time frame (0 points)	Code correctly solves half of the basic puzzles Code recognises when it can't solve a puzzle, and provides feedback (1 point)	Code can solve all of the basic puzzles, quickly. (2 puzzles)

Problem Solving (Advanced)	Code fails to solve harder problems within a reasonable time frame (0 points)	Code can correctly solve the intermediate difficulty puzzles (1 point)	Code can correctly solve the harder puzzles, or recognise when no solution is possible (2 points)