Assignment 2

COMP9021, Term 2, 2021

1. General matter

1.1. **Aims.** The purpose of the assignment is to:

- design and implement an interface based on the desired behaviour of an application program;
- practice the use of Python syntax;
- develop problem solving skills.

1.2. **Submission.** Your program will be stored in a file named `sudoku.py`. After you have developed and tested your program, upload it using Ed (unless you worked directly in Ed). Assignments can be submitted more than once; the last version is marked. Your assignment is due by August 9, 10:00am.

1.3. **Assessment.** The assignment is worth 13 marks. It is going to be tested against a number of input files. For each test, the automarking script will let your program run for 30 seconds.

Late assignments will be penalised: the mark for a late submission will be the minimum of the awarded mark and 13 minus the number of full and partial days that have elapsed from the due date.

The outputs of your programs should be **exactly** as indicated.

1.4. **Reminder on plagiarism policy.** You are permitted, indeed encouraged, to discuss ways to solve the assignment with other people. Such discussions must be in terms of algorithms, not code. But you must implement the solution on your own. Submissions are routinely scanned for similarities that occur when students copy and modify other people's work, or work very closely together on a single implementation. Severe penalties apply.

2. Background

A sudoku grid consists of 9 lines and 9 columns, making up 81 cells, that are grouped in nine 3x3 boxes. In a sudoku puzzle, some but not all of the cells already contain digits between 1 and 9. Here is an example of a sudoku puzzle.

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

Solving a sudoku puzzle means completing the grid so that each digit from 1 to 9 occurs once and only once in every row, once and only one in every column, and once and only once in every box. For instance, the previous puzzle has the following solution.

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

Solving a sudoku puzzle is a very common assignment; it is not difficult and moderately interesting as a "solution" (the completed grid) tells nothing about *how* the solution was reached. More interesting solvers are *logical* in the sense that they (possibly partially only) solve the puzzle in steps and at every step, explain how they made some progress; they do so by using some of the well known techniques that most people who solve sudoku puzzles apply. Two remarks are in order.

- Methods that only discover digits in empty cells are fairly limited; most methods need to keep track of the list of possible digits that can go into a given cell, and making progress might mean reducing that list. To apply techniques of the second kind, it is necessary to first *mark* the grid.
- Often, it is not possible to completely solve a puzzle using exclusively the chosen methods; at some point no progress can be made and then a *random guess* has to be made to either put a digit into a given empty cell, or to remove a digit from the list of possible digits that can go into a given cell. It might subsequently be necessary to *backtrack* and make alternative guesses if the earlier guesses turn out to be inconsistent with a solution.

For this assignment, you will have to implement two such techniques, based on the notions of *forced digits* and *preemptive sets* described in the paper *A Pencil-and-Paper Algorithm for Solving Sudoku Puzzles* by J. F. Crook, Notices of the AMS, 56(4), pp. 460–468. Before anything else, you should study this paper. The forced digits technique is applied first, followed by the preemptive set technique. When no progress can be made, the forced digits techniques could be applied again, but that might not yield anything; an alternative would be to try and fill some empty cell with one of the possible digits for that cell and apply the preemptive set technique applied again, knowing that that guess might prove wrong and that other possible digits might have to be used instead. In this assignment, we will stop at the point where the preemptive set technique can no longer be applied; hence we can expect that our implementation will only partially solve most puzzles. But the technique is very powerful and as explained in the article, subsumes many of the well known techniques.

You will design and implement a program that will read a sudoku grid whose representation is stored in a file *filename*.txt and create a Sudoku object, with a number of methods:

- a method `preassess()` that prints out to standard output whether the representation is correct and has no digit that occurs twice on the same row, on the same column or in the same box;
- a method `bare_tex_output()` that outputs some Latex code to a file, *`filename`*`_bare.tex`, that can be compiled by pdflatex to produce a pictorial representation of the grid;
- a method `forced_tex_output()` that outputs some Latex code to a file, *`filename`*`_forced.tex`, that can be compiled by pdflatex to produce a pictorial representation of the grid to which the forced digits technique has been applied;
- a method `marked_tex_output()` that outputs some Latex code to a file, *`filename`*`_marked.tex`, that can be compiled by pdflatex to produce a pictorial representation of the grid to which the forced digits technique has been applied and that has been marked;
- a method `worked_tex_output()` that outputs some Latex code to a file, *`filename`*`_worked.tex`, that can be compiled by pdflatex to produce a pictorial representation of the grid to which the forced digits technique has been applied, that has been marked, and to which the preemptive set technique has been applied.

The input is expected to consist of 9 lines of digits, with possibly lines consisting of spaces only that will be ignored and with possibly spaces anywhere on the lines with digits. If the input is incorrect, that is, does not satisfy the conditions just spelled out, then the program should generate a `SudokuError` with `Incorrect input` as message.

Here is a possible interaction:

```
$ python3
...
>>> from sudoku import *
>>> Sudoku('sudoku_wrong_1.txt')
...
sudoku.SudokuError: Incorrect input
>>> Sudoku('sudoku_wrong_2.txt')
...
sudoku.SudokuError: Incorrect input
>>> Sudoku('sudoku_wrong_3.txt')
...
sudoku.SudokuError: Incorrect input
```

3. Examples

3.1. **First example.** The file sudoku_1.txt has the following contents.

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

Here is a possible interaction:

```
$ python3
...
>>> from sudoku import *
>>> sudoku = Sudoku('sudoku_1.txt')
>>> sudoku.preassess()
There is clearly no solution.
```

3.2. **Second example.** The file sudoku_2.txt has the following contents.

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

Here is a possible interaction:

```
$ python3
...
>>> from sudoku import *
>>> sudoku = Sudoku('sudoku_2.txt')
>>> sudoku.preassess()
There is clearly no solution.
```

3.3. **Third example.** The file sudoku_3.txt has the following contents.

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

Here is a possible interaction:

```
$ python3
...
>>> from sudoku import *
>>> sudoku = Sudoku('sudoku_3.txt')
>>> sudoku.preassess()
There might be a solution.
>>> sudoku.bare_tex_output()
>>> sudoku.forced_tex_output()
>>> sudoku.marked_tex_output()
>>> sudoku.worked_tex_output()
```

The effect of executing sudoku.bare_tex_output() is to produce a file named sudoku_3_bare.tex that can be given as argument to pdflatex to produce a file named sudoku_3_bare.pdf that views as the grid on page 2.

The effect of executing `sudoku.forced_tex_output()` is to produce a file named `sudoku_3_forced.tex` that can be given as argument to `pdflatex` to produce a file named `sudoku_3_forced.pdf` that views as follows.

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

The effect of executing `sudoku.marked_tex_output()` is to produce a file named `sudoku_3_marked.tex` that can be given as argument to `pdflatex` to produce a file named `sudoku_3_marked.pdf` that views as follows.

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

The effect of executing `sudoku.worked_tex_output()` is to produce a file named `sudoku_3_worked.tex` that can be given as argument to `pdflatex` to produce a file named `sudoku_3_worked.pdf` that views identically to `sudoku_3_marked.pdf`.

3.4. **Fourth example.** The file sudoku_4.txt has the following contents.

```
039500000
000800070
000010904
100400003
000000000
007000860
006708200
010090005
000001008
```

Here is a possible interaction:

```
$ python3
...
>>> from sudoku import *
>>> sudoku = Sudoku('sudoku_4.txt')
>>> sudoku.preassess()
There might be a solution.
>>> sudoku.bare_tex_output()
>>> sudoku.forced_tex_output()
>>> sudoku.marked_tex_output()
>>> sudoku.worked_tex_output()
```

The effect of executing `sudoku.bare_tex_output()` is to produce a file named `sudoku_4_bare.tex` that can be given as argument to `pdflatex` to produce a file named `sudoku_4_bare.pdf` that views as follows.

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

The effect of executing `sudoku.forced_tex_output()` is to produce a file named `sudoku_4_forced.tex` that can be given as argument to `pdflatex` to produce a file named `sudoku_4_forced.pdf` that views as follows.

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

The effect of executing `sudoku.marked_tex_output()` is to produce a file named `sudoku_4_marked.tex` that can be given as argument to `pdflatex` to produce a file named `sudoku_4_marked.pdf` that views as follows.

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

The effect of executing `sudoku.worked_tex_output()` is to produce a file named `sudoku_4_worked.tex` that can be given as argument to `pdflatex` to produce a file named `sudoku_4_worked.pdf` that views as follows.

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

3.5. **Fifth example.** The file sudoku_5.txt has the following contents.

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

Here is a possible interaction:

```
$ python3
...
>>> from sudoku import *
>>> sudoku = Sudoku('sudoku_5.txt')
>>> sudoku.preassess()
There might be a solution.
>>> sudoku.bare_tex_output()
>>> sudoku.forced_tex_output()
>>> sudoku.marked_tex_output()
>>> sudoku.worked_tex_output()
```

The effect of executing `sudoku.bare_tex_output()` is to produce a file named `sudoku_5_bare.tex` that can be given as argument to `pdflatex` to produce a file named `sudoku_5_bare.pdf` that views as follows.

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

The effect of executing `sudoku.forced_tex_output()` is to produce a file named `sudoku_5_forced.tex` that can be given as argument to `pdflatex` to produce a file named `sudoku_5_forced.pdf` that views as follows.

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

The effect of executing `sudoku.marked_tex_output()` is to produce a file named `sudoku_5_marked.tex` that can be given as argument to `pdflatex` to produce a file named `sudoku_5_marked.pdf` that views as follows.

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

The effect of executing `sudoku.worked_tex_output()` is to produce a file named `sudoku_5_worked.tex` that can be given as argument to `pdflatex` to produce a file named `sudoku_5_worked.pdf` that views as follows.

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

4. Precisions

The `preassess()` method prints out `There is clearly no solution.` in case some row, column or box contains twice the same digit`, and There might be a solution.` otherwise, that is, in case no row, column or box contains twice the same digit.

For the `.tex` files output by the program, pay attention to the expected format, including spaces and blank lines. Lines that start with `%` are comments; there are 9 such lines. The output of your program redirected to a file will be compared with the expected output saved in a file (of a different name of course) using the `diff` command. For your program to pass the associated test, `diff` should silently exit, which requires that the contents of both files be absolutely identical, character for character, including spaces and blank lines.

The `forced_tex_output()` method produces a file designed to depicts the grid where all forced digits have been added. A forced digit is a digit that must fill an empty cell in a box because that box does not contain that digit yet and all other empty cells in that box are on a row or on a column that contains that digit. As forced digits are being discovered and fill empty cells, more forced digits might be discovered that could not be discovered in the first round. So the program must make sure that no forced digit can be added to the grid that will be produced when executing that method. The provided examples illustrate.

The `marked_tex_output()` method produces a file designed to depicts the grid where all forced digits have been added and all possible digits have been added to the corners of the empty cells. The possible digits for an empty cell are the the digits that do not occur on the same row, on the same column or in the same box. The provided examples illustrate.

The `worked_tex_output()` method produces a file designed to depicts the grid where all forced digits have been added, all possible digits have been added to the corners of the empty cells, and the preemptive set technique has been applied until it cannot allow one to eliminate any possible digit from any cell (which might be because the puzzle has been solved). The provided examples illustrate.

The 13 marks will be distributed as follows:

- 1 mark for `preassess()`;
- 3 marks for `bare_tex_output()`;
- 3 marks for `forced_tex_output()`;
- 3 marks for `marked_tex_output()`;
- 3 marks for `worked_tex_output()`.