Game Challenge

Su Doku / Latin square

A *Latin square* is an $n \times n$ array filled with n different symbols, each occurring exactly once in each row and exactly once in each column. Su Doku is a puzzle game that has become popular in recent years. It is a variant of a 9×9 Latin square, where the player has to fill out missing numbers, without violating the rules; each row, column and each of the nine 3×3 sub-grids can only contain each symbol once.

In this exercise we implement the game Su Doku. For this we need to represent and check these puzzles in Python.

Inspired by a projecteuler.org exercise.

On the left you find an example of a typically Su Doku puzzle after solving it. The player had to fill in all numbers that are displayed in red. A good Su Doku should only have one possible solution. Note, how it is not always apparent if a number is in the right location. Many players start by excluding candidate numbers and searching for spots with only a single candidate left.

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

Try to solve each challenge without looking at the hints, but feel free to use whatever else information source. If you feel a bit stuck look at the hint. If you are still stuck, do not hesitate to reach out to one of the trainers. After finishing one of the challenges, you might find it useful to also reach out to one of the trainers to discuss it.

If you cannot finish it all, that does not matter. If you want to you can finish it at home. If you need help or have your answers checked, you can reach out to us at an time later.

Enjoy the challenges!

Challenge 1

We have to store and print a grid of 9×9 numbers. Try to figure out how to do that sensibly in Python! Write a function that prints the grid. *Bonus:* Add visual aids to distinguish the nine sub-grids.

Hint 1.1

Use nested lists (see exercise sheet).

Hint 1.2

Read the first section of lesson 9 on snakify.org.

Challenge 2

Copy the black numbers from above example into your list. Represent empty (red) numbers with a zero. Now implement two functions checking the rules for a given row and column. Let it return a Boolean. Test them exhaustively to make sure they work! *Bonus:* Do not copy code! Use a third function that is used by the other two.

Hint 2.1

Make use of the `.sort()` function of lists.

Hint 2.2

Slicing can be very helpful for this.

Challenge 3

Now implement the function that checks the rule for a sub-grid. (This one is a bit tricky!)

Hint 3.1

Try to copy all the required elements into a new list. Then use one the same method as before.

Challenge 4

Now write a function, that uses loops to check the rules for all rows, columns and sub-grids. Let it return whether or not everything checks out!

Challenge 5 (Bonus)

Now allow the user to fill out numbers. Only accept it, if it does not directly violate the rules.

Challenge 6 (Bonus)

Now allow the user to delete numbers. Only delete it, if it was a guessed number, not a number that was initially part of the puzzle.

Hint 6.1

You can use two separate lists or use the numbers 1-9 for guessed numbers and 10-18 for the predetermined numbers. The *'modulo function'* can be very helpful for the latter approach!

Challenge 7 (Bonus)

Let's try to have the computer do some of the work. After each user input, check all fields if there is only one option remaining (i.e. all other options are forbidden by the three rules). If that is the case, fill in that number.

Hint 7.1

Automatically filling in another number can reduce the options for another field to one. So make sure, that you keep checking until you finish one round of checking all fields without filling any number out.