DM550 Introduction to Programming
Fall 2017 Re-re-exam Project (Python)

Department of Mathematics and Computer Science
University of Southern Denmark

July 2, 2018

Introduction
The purpose of this project is to try in practice the use of programming techniques and knowledge about the programming language Python. Please make sure to read this entire note before starting your work on this part of the project. Pay close attention to the three sections below.

Exam Rules
This project is part of the re-re-exam for DM550. To pass the re-re-exam this Python project (or the one from the ordinary exam or the re-exam) and a Java project (from the re-re-exam or from the re-exam or form the ordinary exam) have to be passed in order to pass the overal exam. This project hast to be done individually.

Deliverables
A short project report (at least 6 pages without front page and appendix) contain the following 6 sections has to be delivered:

- **front page** (course number, name, section, date of birth)
- **specification** (what the program is supposed to do)
- **design** (how the program was planned)
- **implementation** (how the program was written)
- **testing** (what tests you performed)
- **conclusion** (how satisfying the result is)
- **appendix** (complete source code)

The report has to be delivered as a single PDF file electronically using Blackboard's SDU Assignment functionality. No printed copies are required. **Do not forget to include the complete source code!**

Deadline

August 26, 2018, 23:59

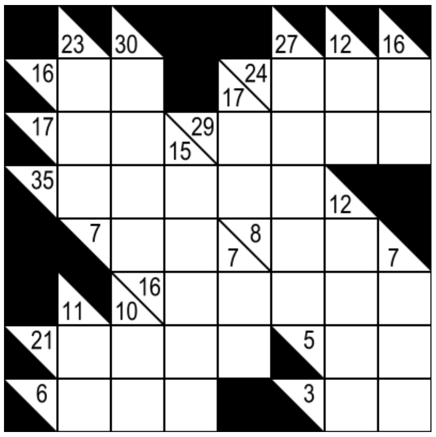
Late deliveries cannot be accepted, so please plan your time accordingly.

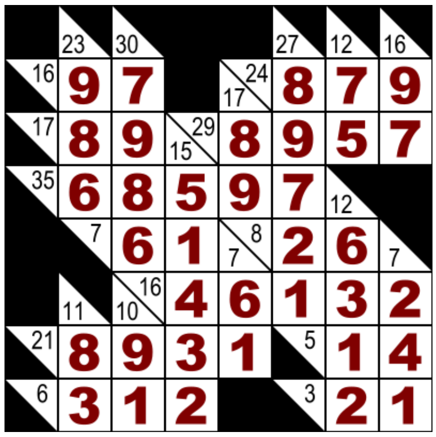
The Problem

Your task in this part of the project is to write a solver for Kakuro puzzles. Kakuro puzzles are a kind of crossword puzzles with numbers where the following two conditions have to be met:

- In each consecutive row or column of empty fields, all numbers have to be from the set $\{1, 2, 3, 4, 5, 6, 7, 8, 9\}$ and they must all be different.
- For each consecutive row or column of empty fields, a number at the to the left or above of the fields specifies the sum of all these numbers.

The following two figures show a Kakuro puzzle and its solution (taken from Wikipedia, both graphics are under the GNU Free Documentation License).





The Input

For input to your program, the Kakuro puzzles are represented as matrices (lists of rows which are lists of fields) where the fields can be one of the following five types, which can be distinguished by the `types` function:

- A frame block with no sum information (black in the graphics above) is represented by the value `(0,0)`.
- A frame block with sum information for a row (black lower left, number N in upper right) is represented by the expression `(0,N)`, e.g., `(0,16)` or `(0,24)`.
- A frame block with sum information for a column (number N in lower left, black upper right) is represented by the value `(N,0)`, e.g., `(23,0)`.

- A frame block with sum information for a column and a row (number *N* in lower left, number *M* in upper right) is represented by the value `(N,M`, e.g., `(15,29)`.
- An empty field is represented by just the value `0`.

Thus, for our example above we obtain the following expression:

```
[[(0, 0), (23,0), (30, 0), ( 0, 0), ( 0, 0), (27,0), (12,0), (16,0)],
 [(0,16),      0,       0, ( 0, 0), (17,24),      0,      0,      0],
 [(0,17),      0,       0, (15,29),       0,      0,      0,      0],
 [(0,35),      0,       0,       0,       0,      0, (12,0), ( 0,0)],
 [(0, 0), ( 0,7),       0,       0, ( 7, 8),      0,      0, ( 7,0)],
 [(0, 0), (11,0), (10,16),       0,       0,      0,      0,      0],
 [(0,21),      0,       0,       0,       0, ( 0,5),      0,      0],
 [(0, 6),      0,       0,       0, ( 0, 0), ( 0,3),      0,      0]]
```

The Output

The output of your solver should also be a matrix. The representation is similar to the one for the Input except for all single `0`s being replaced by the appropriate number. For our example we obtain the following expression:

```
[[(0, 0), (23,0), (30, 0), ( 0, 0), ( 0, 0), (27,0), (12,0), (16,0)],
 [(0,16),      9,       7, ( 0, 0), (17,24),      8,      7,      9],
 [(0,17),      8,       9, (15,29),       8,      9,      5,      7],
 [(0,35),      6,       8,       5,       9,      7, (12,0), ( 0,0)],
 [(0, 0), ( 0,7),       6,       1, ( 7, 8),      2,      6, ( 7,0)],
 [(0, 0), (11,0), (10,16),       4,       6,      1,      3,      2],
 [(0,21),      8,       9,       3,       1, ( 0,5),      1,      4],
 [(0, 6),      3,       1,       2, ( 0, 0), ( 0,3),      2,      1]]
```

The Task

Implement a function `solve` that takes an unsolved Kakuro puzzle as its argument and returns its solved form. Also implement functions `load` and `store` for loading and storing Kakuros from and to text files, respectively. Finally, implement a main function that loads, solves and stores a Kakuro.

Keep in mind, that there are many different ways how to implement the `solve` function. Explain your approach, implement it, and produce the report. You could for example choose one of the following approaches:

- Use a generate-and-test approach, i.e., go through all solution candidates and test them until you find a solution. This can be done either recursively or iteratively.
- Implement solving rules as used by human players and use brute-force only as a last resort.