Program 3 - Calcudoku Solver: 10 points (the other projects are 5 points.)

CPE 101, Spring 2021

Updates

- Release date: Monday, 19 April. Due date: Sunday, 25 April, 11:59:59, early turn in, total 10 points, bonus credit of 1 point for Sunday submission: 11 possible pts.

- Full credit due date: Wednesday, 28 April, 11:59:59. 70% credit due date would then be Wednesday, 5 May, 11:59:59.

- LINK for explanatory video by Julie Workman: https://screencast-o-matic.com/watch/crniVNRPQR

Purpose

To write a program requiring use of loops and lists.

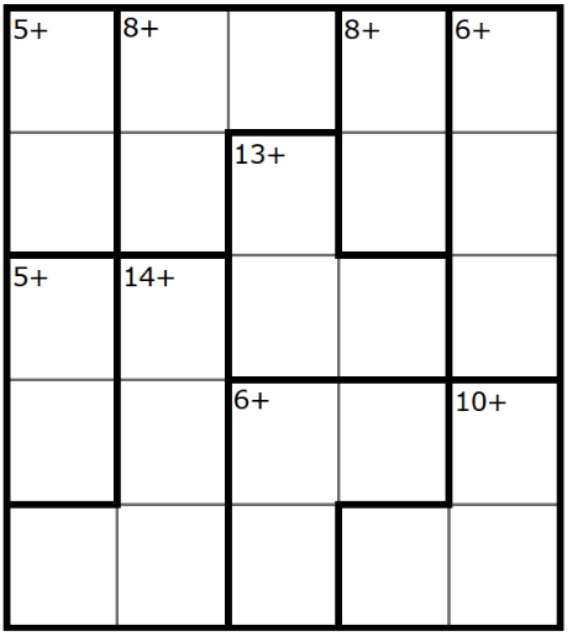
Programming environment. This is a solo programming project. You are responsible for all the work related to the program development, testing, and submission. This is a comprehensive project and requires that you begin early and in earnest.

Program Description

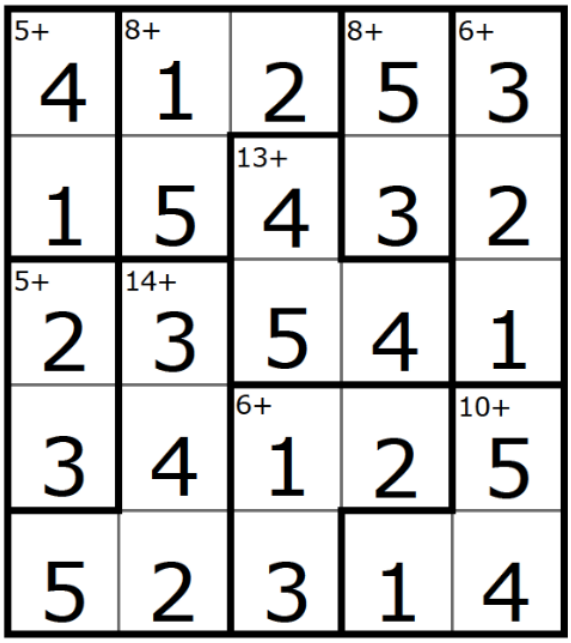
For this program, you will be writing a solver for 5x5 Calcudoku puzzles. A Calcudoku puzzle is an NxN grid where the solution satisfies the following:

- each row can only have the numbers 1 through N with no duplicates
- each column can only have the numbers 1 through N with no duplicates
- the sum of the numbers in a “cage” (areas with a bold border) should equal the number shown in the upper left portion of the cage

Here is a sample puzzle:



Here is the solution for the puzzle:



The description for each puzzle will be input in the following format:

```
number_of_cages
cage_sum number_of_cells (list of cells)
cage_sum number_of_cells (list of cells)
…
…
…
```

The first line contains of the number of cages in the puzzle.

After the first line, each subsequent line describes a cage. The first number of a line is the sum of the cage, the second is the number of cells in the cage, and the third number onward contains a list of the cells contained in the cage. In the puzzle, the cells are numbered starting with 0 for the upper left cell and increase from left-to-right, and then from top to bottom. (In the above puzzle, the squares be numbered from 0 to 24.)

For the sample puzzle shown above, the input would look like:

```
9
5 2 0 5
8 3 1 2 6
8 2 3 8
…
…
…
```

Solving a Puzzle

For this program, you will be solving the puzzle using 'brute-force' where the program will test all possible solutions until it finds the correct one. Your algorithm should perform the following:

- Place a 1 in the current cell (starting with the upper left cell)
    - 'Check' if the number is valid, continue to the next cell to the right (advancing to the next row when necessary)
    - If the number is not valid, increment the number and 'check' for validity again
        - If the number is the maximum possible and is still invalid, then 'backtrack' by setting the current cell to 0 and moving back to the previous cell

In order for this algorithm to work, you will need to write a function to test if a puzzle is in a valid state. **You should initialize the values in your puzzle to 0.** As you populate the puzzle, a puzzle is invalid if:

- Duplicates exist in any row or column
- The sum of values in a fully populated cage does not equal the required sum
- The sum of values in a partially populated cage exceeds or equals required sum

In your program, count the number of times you check that a number is valid and count the number of backtracks.

Input

A sample run is shown below (user input is in bold):

```
Number of cages:9
Cage number 0: 9 3 0 5 6
Cage number 1: 7 2 1 2
Cage number 2: 10 3 3 8 13
Cage number 3: 14 4 4 9 14 19
Cage number 4: 3 1 7
Cage number 5: 8 3 10 11 16
Cage number 6: 13 4 12 17 21 22
Cage number 7: 5 2 15 20
Cage number 8: 6 3 18 23 24

--Solution--

3 5 2 1 4
5 1 3 4 2
2 4 1 5 3
1 2 4 3 5
4 3 5 2 1

checks: 2745 backtracks: 534
```

Testing Data

I will provide a few files with sample runs for sample problems with the extra feature that it prints out each iteration of the grid so that you can actually "see" the brute force algorithm and how it does its job.

General Notes

- To read in multiple values in one line, you can use:

  ```
  input("Enter line:").split()
  ```

- You must implement your functions in a file: `solverFuncs.py (NAMED EXACTLY THAT.)`

Additional Requirements

You must store your puzzle and cages as a list of lists.

Puzzle is a list of lists.

```
[[0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0],
 …
```

Cages is a list of lists.

```
[[5, 2, 0, 5],
 [8, 3, 1, 2, 6],
 …
```

Required Functions (**ALL MUST BE NAMED EXACTLY AS SHOWN BELOW**, else you will receive no credit for your submission. All return `True/False` except `get_cages`):

```
get_cages() # returns list of cages (each cage is a list)

check_rows_valid(puzzle)

check_columns_valid(puzzle)

check_cages_valid(puzzle, cages)

check_valid(puzzle, cages)
```

Handin on Canvas under "Assignments"

submit the 2 files (not screenshots, the files, and do not zip them): solver.py and solverFuncs.py
AND submit screenshots to show operation of your script and the tests you've done. Give one screenshot to show that your script performs the given example in this instruction sheet and give one other screenshot with some interesting example. Give one screenshot to show you did your unit testing on the functions. That would make 3 screenshots.

Total: submit 2 files and 3 screenshots.