Last updated: January 2, 2024

Sudoku Solver

Author: @jazreen

Just a mini-report documenting the progress of the project (in reverse chronological order).

Project started: August 2023 (summer break)
File described: sudoku.py
Python experience level: HarvardX CS50 Intro to Python online course

- Majority of functions and code completed
  - Code broken into multiple functions for easy debugging and maintainability
  - Backtracking algorithm used to solve the sudoku matrix
- Lots of comments used to keep record of thinking process
- Debugging started, using print statements to track output
  - Initial logic didn't go beyond first recursion as there was no saved list of visited cells—hence stuck in an infinite loop. Thus, stack created using LIFO system used to keep record of all the past visited cells (used in backtracking)
  - realised an upper bound needed to be passed into certain functions when backtracking so that we only rewrite and rerun previous cells with new numbers (between up_bound to 9) – prevents a type of infinite recursion
- Identified potential sources of error
- Lots of detailed comments included

Original test sudoku matrix (from Wikipedia):

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

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

[Work postponed during school semester, Sept - Dec]

Restarted work: January 1, 2024 (winter break)
Python experience level: previous online course + Software Design, CPEN 333 (university course)
[Solver successfully debugged on January 2, 2024!]

Log of all the changes made on this attempt:

- Issue detected with choose_val function as it always seemed to give a value of 0
  - Issue found in check_square function: incorrect if condition, instead of checking if numbers in any of the other cells in the square match with the number at the specified matrix position, we should be checking it with the proposed integer value for that position. Solution: add a new input parameter to check_square so that the proposed integer can be passed into the function.

```
 matrix[mrow + i][mcol + j] == int #new
 matrix[mrow + i][mcol + j] == matrix[nrow][ncol] #old
```

- When doing backward recursive calls, there was nothing to tackle the case if up_bound became >= 10 (indicates that none of the number 1-9 fits the criteria for this cell → indicates issue lies in one of the preciously visited cells)
    - Added an if condition that popped one more previous cell off the stack i.e. went back one step further since the issue couldn't be solved here
- Created a new function called pop_prev_record, the code under it was originally part of backtrack but it was separated to avoid repeating code sections. Instead, now we just call that function where necessary inside backtrack
- Changed the `def main: ….. main()` structure of the code to `if __name__ == "__main__":` so that the file can be imported into unit testing files AND run as it is
- Unit testing files created to test the different functions individually
- BIG issue encountered after the initial debugging: Maximum recursion depth exceeded. Initially, I assumed it meant I was going into infinite recursion but after some debugging with simpler matrices, it seemed obvious that the code was working. After some research it turned out I just had to increase Python's standard recursion limit to get working! The following line was added:

  ```
  sys.setrecursionlimit(100000) #usually its 1000
  ```

  The maze I started off with was solved with a 10,000 limit, but when I tried a more difficult matrix, I had to increase the limit to 100,000 (more recursions needed). More recursion also means more time taken to reach the solution. Note that increasing recursion limit can sometimes lead to system crashes if no proper ending condition has actually been implemented (so best to only do it if you're sure it will eventually stop).

```
super_easy = [
            [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, 0],
            [2, 8, 7, 4, 0, 9, 6, 3, 5],
            [3, 4, 5, 2, 8, 6, 1, 7, 0]
            ]
#tried a very easy one just to ensure the logic was working in initial stages of
debugging
```

- Changed how the solved matrix looks when printed i.e. from normal list to grid. Discovered and used a new Python library called Tabulate.

  Screenshots of the solver working:

```
no remaining empty cell
Sudoku solved!
+---+---+---+---+---+---+---+---+---+
| 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 |
+---+---+---+---+---+---+---+---+---+
```

The original matrix solved

The ‘hard’ matrix:

```
hard = [
        [8, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 3, 6, 0, 0, 0, 0, 0],
        [0, 7, 0, 0, 9, 0, 2, 0, 0],
        [0, 5, 0, 0, 0, 7, 0, 0, 0],
        [0, 0, 0, 0, 4, 5, 7, 0, 0],
        [0, 0, 0, 1, 0, 0, 0, 3, 0],
        [0, 0, 1, 0, 0, 0, 0, 6, 8],
        [0, 0, 8, 5, 0, 0, 0, 1, 0],
        [0, 9, 0, 0, 0, 0, 4, 0, 0]
    ]
```

Unsolved

```
no remaining empty cell
Sudoku solved!
+---+---+---+---+---+---+---+---+---+
| 8 | 1 | 2 | 7 | 5 | 3 | 6 | 4 | 9 |
+---+---+---+---+---+---+---+---+---+
| 9 | 4 | 3 | 6 | 8 | 2 | 1 | 7 | 5 |
+---+---+---+---+---+---+---+---+---+
| 6 | 7 | 5 | 4 | 9 | 1 | 2 | 8 | 3 |
+---+---+---+---+---+---+---+---+---+
| 1 | 5 | 4 | 2 | 3 | 7 | 8 | 9 | 6 |
+---+---+---+---+---+---+---+---+---+
| 3 | 6 | 9 | 8 | 4 | 5 | 7 | 2 | 1 |
+---+---+---+---+---+---+---+---+---+
| 2 | 8 | 7 | 1 | 6 | 9 | 5 | 3 | 4 |
+---+---+---+---+---+---+---+---+---+
| 5 | 2 | 1 | 9 | 7 | 4 | 3 | 6 | 8 |
+---+---+---+---+---+---+---+---+---+
| 4 | 3 | 8 | 5 | 2 | 6 | 9 | 1 | 7 |
+---+---+---+---+---+---+---+---+---+
| 7 | 9 | 6 | 3 | 1 | 8 | 4 | 5 | 2 |
+---+---+---+---+---+---+---+---+---+
```

Solved

Discussion:

- Definitely understood the importance of writing comment explanations. If I didn’t leave in those comments back in August, I doubt I’d be able to understand half of what I was thinking just by looping at the code right now
- In my Software Design course (CPEN 333) this term, I learnt about Python’s unit testing library. However, it wasn’t a topic we implemented in our labs so I’m eager to integrate it into this project and get some practical experience on it!