1 Environments and instructions

1. Python version **3.8** required
2. Run with `python sodoku_solver.py`
3. The terminal will show the results with both two implementations and the time cost comparison between them.

2 Define Sodoku puzzle as a CSP

The components of CSP in this problem:

1. **Variables**: Every cell in the Sodoku is a variable. We use a matrix X to represent all variables where X_{ij} is the cell in the $i+1$th row, $j+1$th colomn of the origin sodoku.
2. **Domain**: For the pre-filled cells, the domain is a single value. For the empty cells, the domain is the set contains all integers from 1 to 9. i.e. $D = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$;
3. **Constraints**:
    1. C_1 = No identical elements per row
    2. C_2 = No identical elements per colomn
    3. C_3 = No identical elements per 3×3 square

3 Implementation Details

We formulate this problem as a CSP and using uninformed search method to solve it. Sodoku puzzle is represented as a two-dimensional array, with `-1` for positions that have no pre-filled elements. In detail, we use ***backtracking+forward checking*** as the basic implementation. We first update the domain of each variable by forward checking, and at the completion of each round of assignment, check the consistency and update the domain of each variable. Eventually we get the answer to Sudoku or no solution exists.

For the later requirement, we implement a faster sodoku solver with conflict-directed backjumping. As we expected, this method is far more quick than the former one, especially on the second evil sodoku test case.

	BASIC IMPLEMENTATION	CONFLICT-DIRECTED BACKJUMPING
EASY	0.02s	0.00s
EVIL	0.24s	0.00s