Sudoku Solver Documentation

Introduction

This documentation provides an overview of a Python program to solve Sudoku puzzles by filling in the empty cells. Sudoku is a logic-based combinatorial number-placement puzzle. The objective is to fill a 9×9 grid with digits so that each column, each row, and each of the nine 3×3 subgrids contain all of the digits from 1 to 9.

Functionality

The program provides a class `Solution` with a method `solveSudoku` to solve the Sudoku puzzle. It modifies the input Sudoku board in-place to fill the empty cells with valid digits.

Input

The input Sudoku puzzle is provided as a 9x9 grid represented as a list of lists, where each inner list represents a row. The digits are represented as strings, and empty cells are represented by the '.' character.

Output

The solved Sudoku puzzle is represented as the modified input Sudoku board, where all empty cells are filled with valid digits, satisfying the Sudoku rules.

Constraints

- The input Sudoku board must be a 9x9 grid.
- Each inner list representing a row must contain exactly 9 elements.
- Each element in the board must be a digit (1-9) represented as a string or the '.' character.
- It is guaranteed that the input board has only one solution.

Implementation

The program implements a backtracking algorithm to solve the Sudoku puzzle efficiently. It iterates over each cell in the Sudoku grid, and if the cell is empty, it tries filling it with digits from 1 to 9, checking for the validity of the digit based on Sudoku rules. If a valid digit is found, it recursively tries to solve the puzzle. If the puzzle cannot be solved with the current digit, it backtracks and tries the next digit. This process continues until a solution is found or all possibilities are exhausted.

Methods

1. **`solveSudoku(board: List[List[str]]) -> None`:** This method takes the Sudoku board as input and modifies it in-place to solve the Sudoku puzzle.

2. **`solve(board:** List[List[str]]) -> bool`: This is a helper method used internally by `solveSudoku`. It implements the backtracking algorithm to recursively solve the Sudoku puzzle.

3. **`is_valid(board: List[List[str]], row: int, col: int, num: str) -> bool`:** This method checks the validity of placing a digit `num` at the given row and column position in the Sudoku board.

Example

```
board = [
   ["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"]
]


solution = Solution()
solution.solveSudoku(board)
print(board)
```

Output:

```
[
  ["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"]
]
```

Explanation: The input board is shown above, and the solved Sudoku puzzle is provided as the output.

Conclusion

This documentation provides a comprehensive overview of the Sudoku Solver program, including its functionality, input/output format, constraints, implementation details, methods, and an example demonstrating how to use the program to solve a Sudoku puzzle.