Documentation: Valid Sudoku Checker

Overview

This documentation outlines the usage, functionality, and implementation details of a solution to the problem of determining whether a given 9x9 Sudoku board is valid. The solution adheres to the rules of Sudoku, validating the filled cells based on three criteria: rows, columns, and 3x3 sub-boxes.

Problem Statement

A valid Sudoku board must satisfy the following conditions:

1. Each row must contain the digits 1-9 without repetition.
2. Each column must contain the digits 1-9 without repetition.
3. Each of the nine 3x3 sub-boxes of the grid must contain the digits 1-9 without repetition.

Solution Approach

The provided solution utilizes a class named `Solution` containing methods to validate rows, columns, and 3x3 sub-boxes. It employs a straightforward approach to iteratively check each row, column, and sub-box against the rules of Sudoku.

Solution Implementation

1. **isValidSudoku Method:**
    a. This method serves as the entry point for the solution. It iterates through each row, column, and 3x3 sub-box of the Sudoku board, checking their validity using the `isValidUnit` method.
    b. It returns `True` if the entire board satisfies all Sudoku rules, otherwise `False`.

2. **isValidUnit Method:**
    a. This method takes a list of numbers (representing a row, column, or sub-box) as input and checks whether it contains duplicates.
    b. It utilizes a set to keep track of seen numbers and returns `True` if no duplicates are found, otherwise `False`.

3. **Example Usage:**
    a. The example usage demonstrates how to instantiate the `Solution` class and use it to validate Sudoku boards.
    b. Two example Sudoku boards are provided with expected outputs.

Example Usage

```
solution = Solution()
```

Example 1

```
board1 = [
    ["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"]
]

print(solution.isValidSudoku(board1))  # Output: True
```

Example 2

```
board2 = [
    ["8","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"]
]

print(solution.isValidSudoku(board2))  # Output: False
```

Constraints

- `board` is a 9x9 grid.
- Each element of `board` is either a digit from 1 to 9 or a period `'.'`.