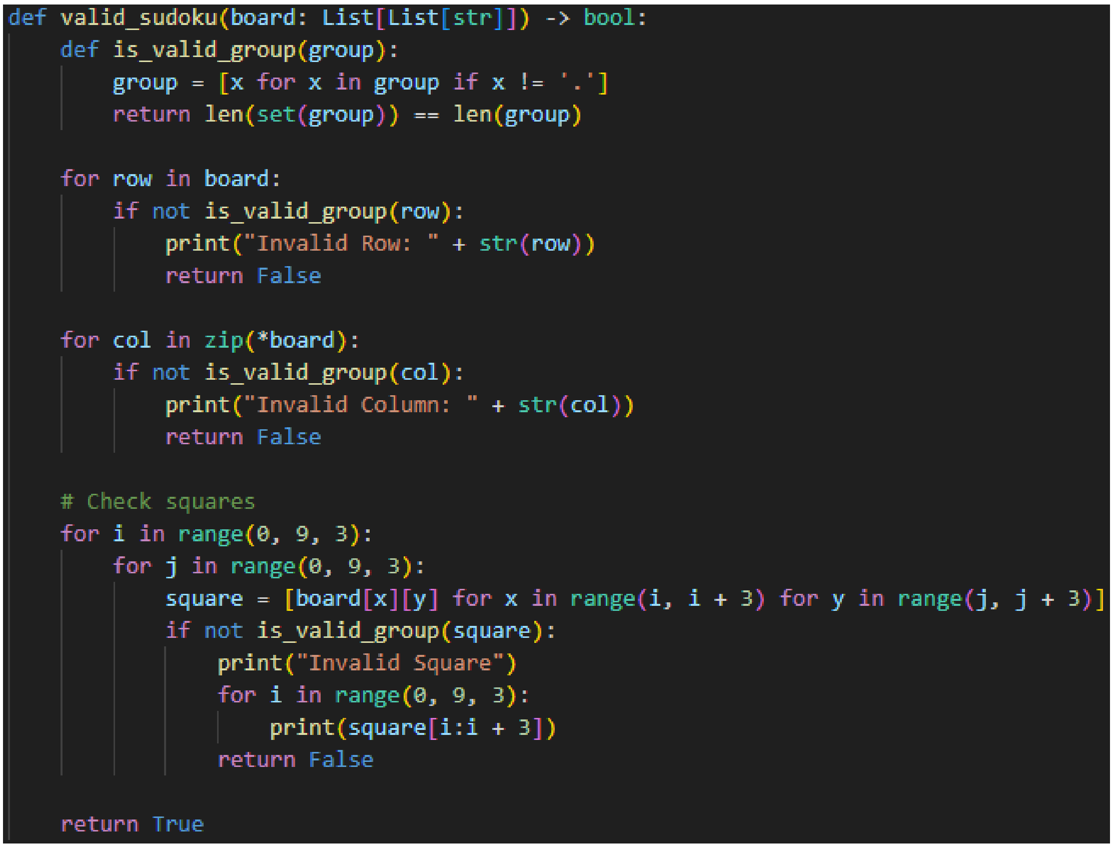
Problem Statement:

Given a 9x9 board, determine if it is a valid Sudoku configuration. Each row, column, and 3x3 sub-box must contain the digits 1-9 without repetition. Empty cells are represented by a ‘.’ Character.

Solution – Validate Rows, Columns, Sub Grids



```python
def valid_sudoku(board: List[List[str]]) -> bool:
    def is_valid_group(group):
        group = [x for x in group if x != '.']
        return len(set(group)) == len(group)

    for row in board:
        if not is_valid_group(row):
            print("Invalid Row: " + str(row))
            return False

    for col in zip(*board):
        if not is_valid_group(col):
            print("Invalid Column: " + str(col))
            return False

    # Check squares
    for i in range(0, 9, 3):
        for j in range(0, 9, 3):
            square = [board[x][y] for x in range(i, i + 3) for y in range(j, j + 3)]
            if not is_valid_group(square):
                print("Invalid Square")
                for i in range(0, 9, 3):
                    print(square[i:i + 3])
                return False

    return True
```

Step-by-Step Breakdown

1. **Input:**

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
0	5	3	.	.	7	.	.	.	.
1	6	.	.	1	9	5	.	.	.
2	.	9	8	.	.	.	.	6	.
3	8	.	.	.	6	.	.	.	3
4	4	.	.	8	.	3	.	.	1
5	7	.	.	.	2	.	.	.	6
6	.	6	.	.	.	.	2	8	.
7	.	.	.	4	1	9	.	.	5
8	.	.	.	.	8	.	.	7	9

2. **Validation:** A validator function is created to process groups. As multiple groups require processing, a helper function here avoids repetition and simplifies the process. This function creates a new group of all non '.' Characters, returning the result of equality between the length of a set of the group and the length of the group, a simple duplicate check.

3. **Row Validation:** Each row, or array, in the provided array of arrays is validated using the validation function.

Row	Group Result	Return
Row 1	["5", "3", "7"]	T
Row 2	["6", "1", "9", "5"]	T
Row 3	["9", "8", "6"]	T

4. **Column Validation:** Columns are created by providing the input to the zip function, an easy way to get the rows of a simple matrix

Column	Group Result	Return
Col 1	["5", "6", "8", "4", "7"]	T
Col 2	["3", "9", "6"]	T

5. **Sub-grid (3x3 Box) Validation:**

- Traverse sub-grids by iterating over indices (i, j) in steps of 3.
- Extract and validate each 3x3 square.

Sub-grid Index (Top-Left)	Extracted Sub-grid	Valid Group Check Result
(0, 0)	["5", "3", "6", "9", "8"]	T
(0, 3)	["7", "1", "9", "5"]	T
(3, 0)	["8", "4", "7"]	T

Detailed Explanation:

How Iteration in Steps of Three Works: The indices (i, j) are iterated in steps of 3 (i.e., range(0, 9, 3)) to isolate the starting points of each 3x3 sub-grid on the board. Each sub-grid can be visualized as:

1. Starting at (0, 0), the first 3x3 block includes rows 0 to 2 and columns 0 to 2:

	0	**1**	**2**
0	5	3	.
1	6	.	.
2	.	9	8

- Extracting non-placeholder values results in ["5", "3", "6", "9", "8"].

2. By iterating through (3, 0), (3, 3), etc., each sub-grid is systematically processed and validated.

6. **Output:** True as all sub structures comply with the check conditions.
7. **Efficiency:**
   - **Time Complexity: O(3n) = O(n)**
     - **O(n) for row checks, O(n) for column checks, and O(n) for sub-grid checks.**
   - **Space Complexity: O(n) for storing intermediate results.**