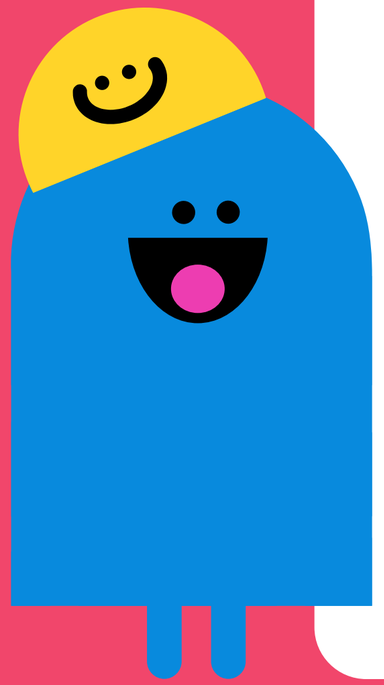
Let's Validate SUDOKU

Presented by:
Sutisan N. 6213221
Wanitcha S. 6213365

Start

Presentation Template by:
Marketplace Designers, CANVA

OUTLINE



1 INTRODUCTION: SUDOKU

2 PROBLEM OVERVIEW

3 SOLUTION

4 CONCLUSION

5 REFERENCE

INTRO

Let's go

What is SUDOKU?

SUDOKU

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

SUDOKU

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

- Number Placement Game
- 9x9 Grid (9 Rows, 9 Columns)
- 3x3 Sub-boxes
- Each row, column, sub-box contains number 1-9
- Given at least 17 numbers

SUDOKU

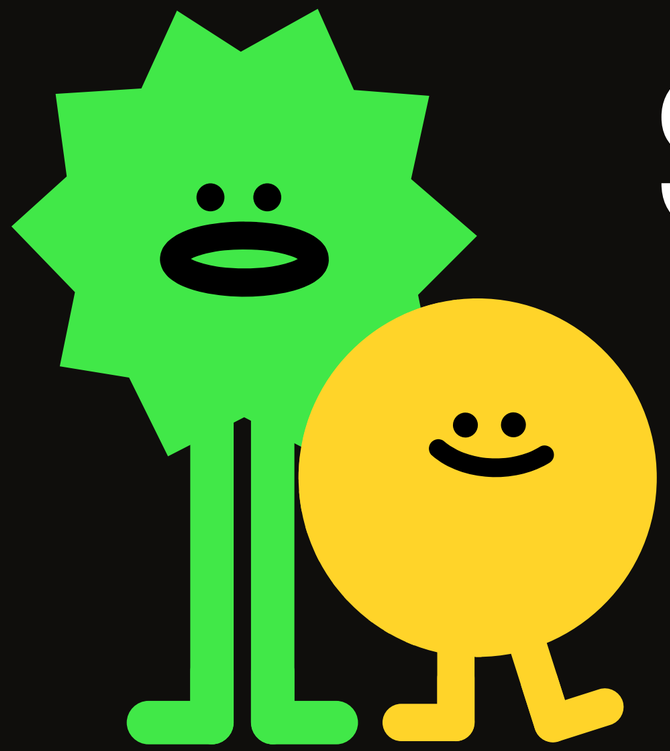
5	3	4	6	7	8	9	1	2
6	7	2	1	9	5	3	4	8
1	9	8	3	4	2	5	6	7
8	5	9	7	6	1	4	2	3
4	2	6	8	5	3	7	9	1
7	1	3	9	2	4	8	5	6
9	6	1	5	3	7	2	8	4
2	8	7	4	1	9	6	3	5
3	4	5	2	8	6	1	7	9

Problem..

VALID SUDOKU

leetcode.com

Difficulty: Medium



PROBLEM OVERVIEW

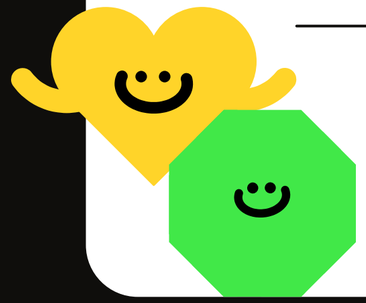
Let's go

VALID SUDOKU

Determine if a 9 x 9 Sudoku board is valid.
Only the filled cells need to be validated according to the following:

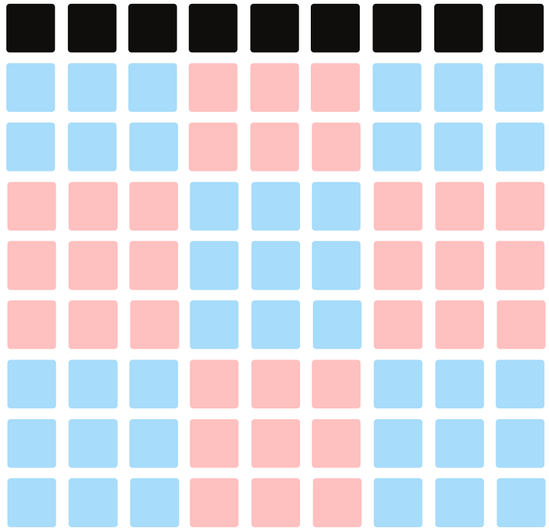
RULES:

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 3 X 3 SUB-BOXES OF THE GRID MUST CONTAIN THE DIGITS 1-9 WITHOUT REPETITION.

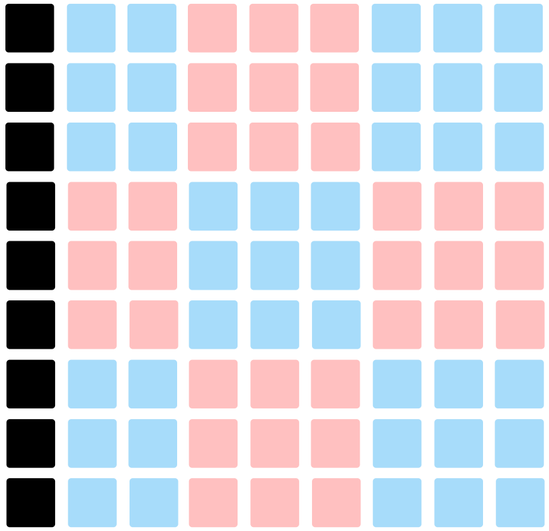


RULES:

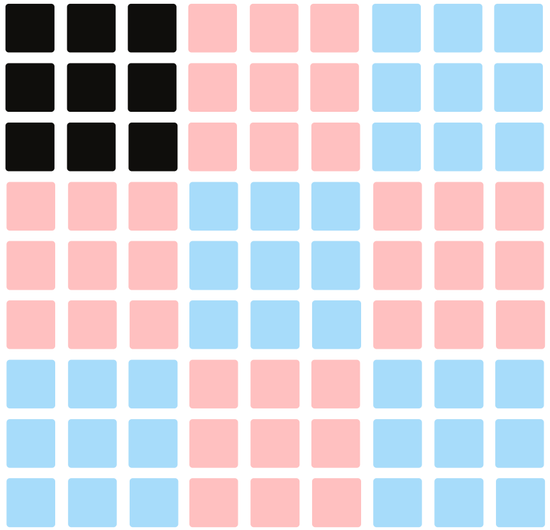
1) Each row must contain the digits 1-9 without repetition.
2) Each column must contain the digits 1-9 without repetition.
3) Each of the nine 3 x 3 sub-boxes of the grid must contain the digits 1-9 without repetition.



Each Row



Each Column



Each Sub-box

SUDOKU

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

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

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

VALID SUDOKU

Determine if a 9 x 9 Sudoku board is valid.
Only the filled cells need to be validated according to the following:

RULES:

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 3 X 3 SUB-BOXES OF THE GRID MUST CONTAIN THE DIGITS 1-9 WITHOUT REPETITION.

Note:

- A Sudoku board (partially filled) could be valid but is **not necessarily solvable**.
- **Only the filled cells** need to be validated according to the mentioned rules.

INPUT/OUTPUT

Input

2-dimensional array indicating the sudoku board

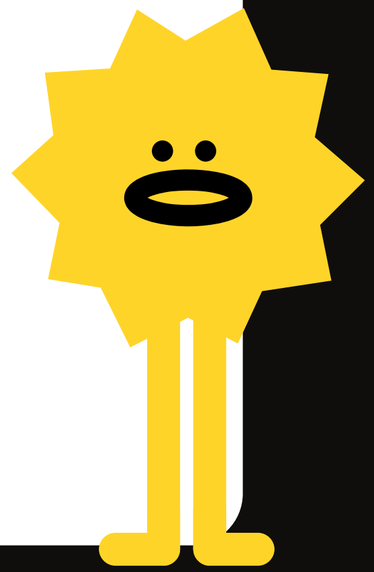
Constraints:

- board.length == 9
- board[i].length == 9
- board[i][j] is a digit 1-9 or '.'
  - '.' means empty

Output

- true
- false

(whether the input sudoku is valid or not)



SAMPLE 1

Input

board =

```
[["5","3",".",".","7",".",".",".",".",
,["6",".",".","1","9","5",".",".",".",
,[".","9","8",".",".",".",".",".","6",
,["8",".",".",".","6",".",".",".",".",
,["4",".",".","8",".","3",".",".","1"]
,["7",".",".",".","2",".",".",".",".",
,[".","6",".",".",".",".",".","2","8",
,[".",".",".","4","1","9",".",".","5"]
,[".",".",".",".","8",".",".","7","9"]
]
```

Output

true

(valid)

Constraints:

- board.length == 9
- board[i].length == 9
- board[i][j] is a digit 1-9 or '.'
  - '.' means empty

SAMPLE 1

Input

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"]
]
```

Output

true

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

SAMPLE 2

Input

board =

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

Output

false

(invalid)

Constraints:

- board.length == 9
- board[i].length == 9
- board[i][j] is a digit 1-9 or '.'
  - '.' means empty

SAMPLE 2

Input

board =

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

Output

false

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

SOLUTION

Let's go

PARAMETERS

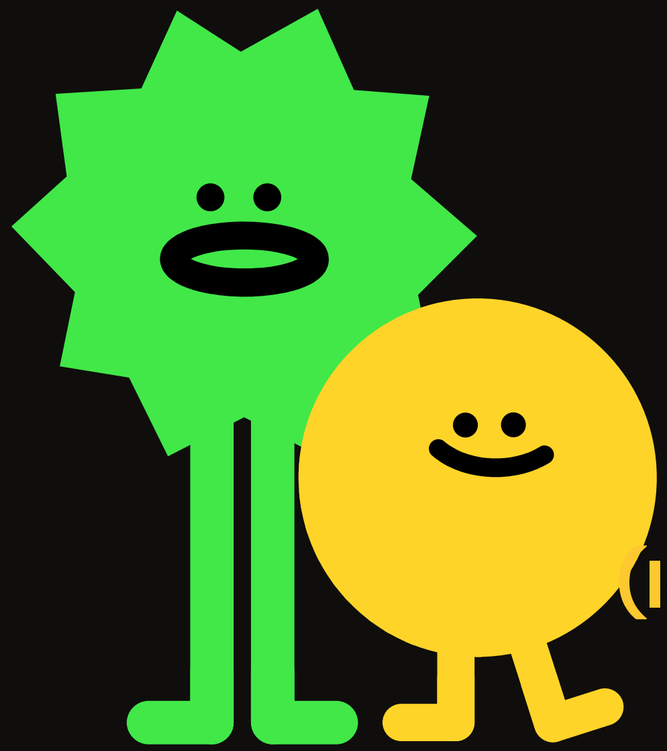
```
1  class Solution {
2      public boolean isValidSudoku(char[][] board) {
3
4      }
5  }
```

---

char[][] board

Approach

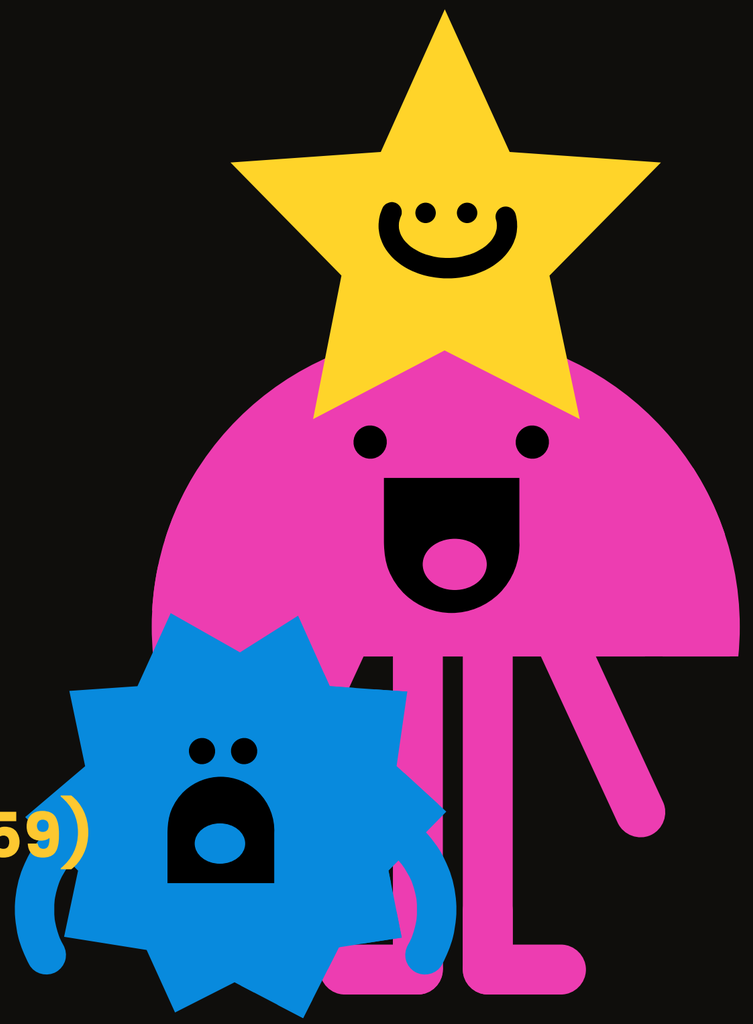
BRUTE FORCE



IDEA IS TAKEN FROM:

Shivam Slingn

(leetcode.com: 2000shivam659)

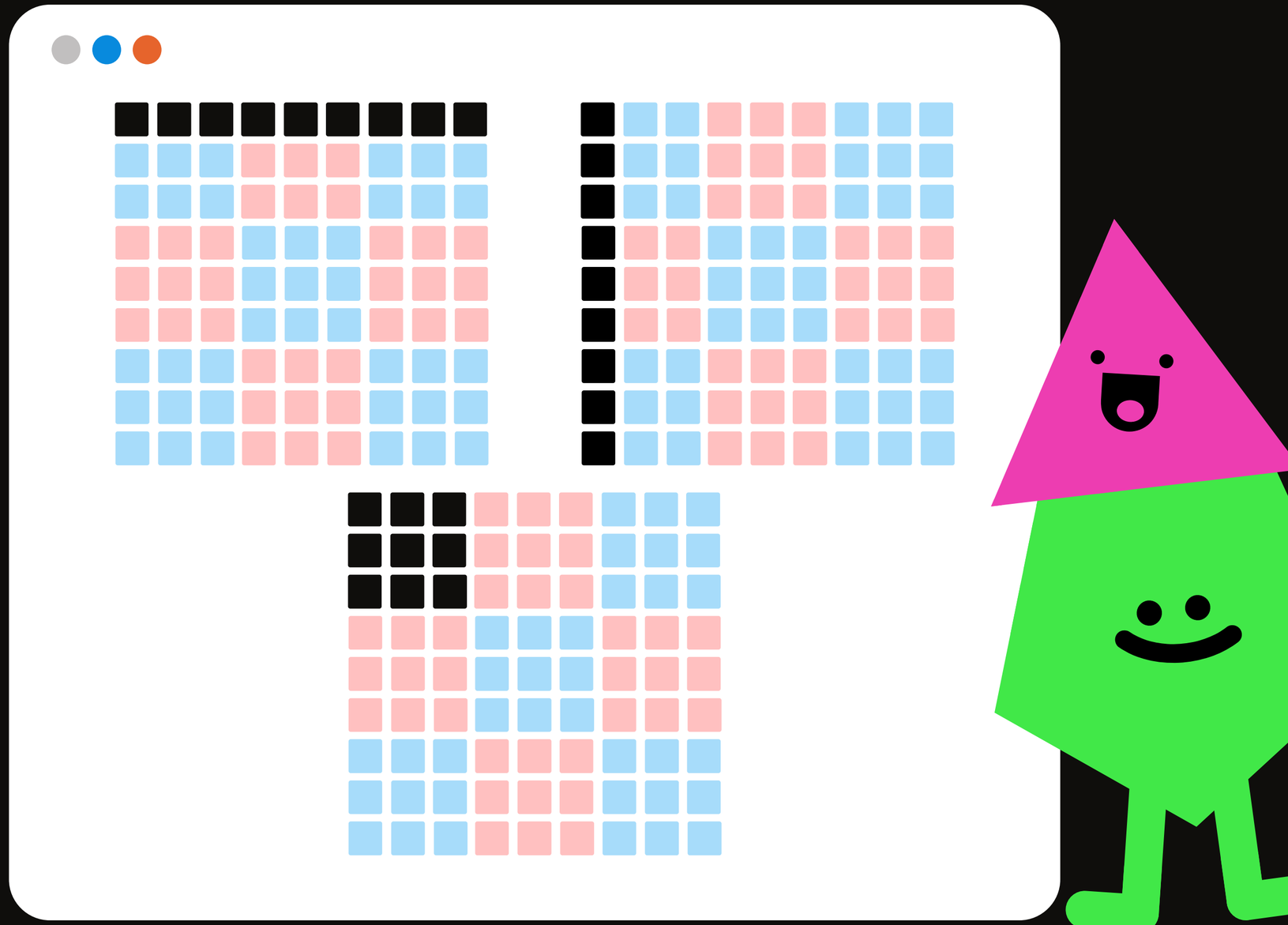


PROBLEM ANALYSIS

1 CHECK each ROW

2 CHECK each COLUMN

3 CHECK each SUB-BOX



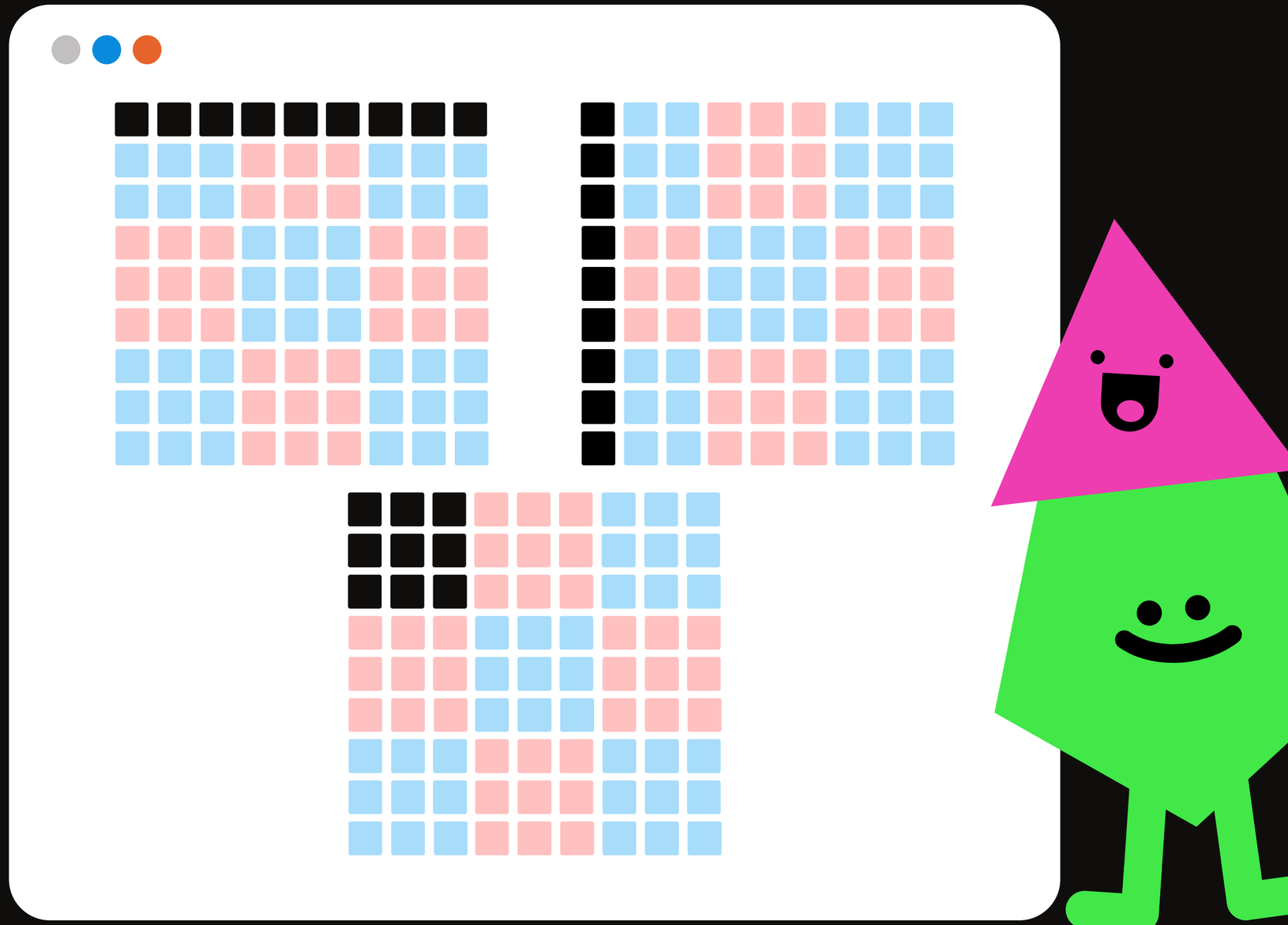
PROBLEM ANALYSIS

1 CHECK each ROW

2 CHECK each COLUMN

3 CHECK each SUB-BOX

NO REPETITION!



Approach

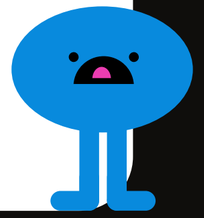
BRUTE FORCE

JAVA

REFERENCE: **Shivam Slingn**

2000shivam659, Leetcode.com

```java
class Solution {
    public boolean isValidSudoku(char[][] board) {

        boolean[][] checkRow = new boolean[9][9];
        boolean[][] checkCol = new boolean[9][9];
        boolean[][] checkBox = new boolean[9][9];

        for (int row = 0; row < board.length; row++) {
            for (int col = 0; col < board.length; col++) {

                if (board[row][col] != '.') {

                    int num = board[row][col] - '1';

                    if (checkRow[num][row]) {
                        return false;
                    } checkRow[num][row] = true;

                    if (checkCol[num][col]) {
                        return false;
                    } checkCol[num][col] = true;

                    int box = ((row/3)*3) + (col/3);

                    if (checkBox[num][box]) {
                        return false;
                    } checkBox[num][box] = true;
                }
            }
        }
        return true;
    }
}
```



Approach

BRUTE FORCE

REFERENCE: **Shivam Slingn**

2000shivam659, Leetcode.com

FIELDS

- checkRow: boolean[][]
- checkCol: boolean[][]
- checkBox: boolean[][]
- row:int
- col: int
- num: int
- box: int

```java
class Solution {
    public boolean isValidSudoku(char[][] board) {

        boolean[][] checkRow = new boolean[9][9];
        boolean[][] checkCol = new boolean[9][9];
        boolean[][] checkBox = new boolean[9][9];

        for (int row = 0; row < board.length; row++) {
            for (int col = 0; col < board.length; col++) {

                if (board[row][col] != '.') {

                    int num = board[row][col] - '1';

                    if (checkRow[num][row]) {
                        return false;
                    } checkRow[num][row] = true;

                    if (checkCol[num][col]) {
                        return false;
                    } checkCol[num][col] = true;

                    int box = ((row/3)*3) + (col/3);

                    if (checkBox[num][box]) {
                        return false;
                    } checkBox[num][box] = true;
                }
            }
        }
        return true;
    }
}
```

```java
class Solution {
    public boolean isValidSudoku(char[][] board) {

        boolean[][] checkRow = new boolean[9][9];
        boolean[][] checkCol = new boolean[9][9];
        boolean[][] checkBox = new boolean[9][9];

        for (int row = 0; row < board.length; row++) {
            for (int col = 0; col < board.length; col++) {

                if (board[row][col] != '.') {

                    int num = board[row][col] - '1';

                    if (checkRow[num][row]) {
                        return false;
                    } checkRow[num][row] = true;

                    if (checkCol[num][col]) {
                        return false;
                    } checkCol[num][col] = true;

                    int box = ((row/3)*3) + (col/3);

                    if (checkBox[num][box]) {
                        return false;
                    } checkBox[num][box] = true;
                }
            }
        }
        return true;
    }
}
```

IDEA REFERENCE:

2000shivam659, Leetcode.com

2D ARRAYS

```
boolean[][] checkRow = new boolean[9][9];
boolean[][] checkCol = new boolean[9][9];
boolean[][] checkBox = new boolean[9][9];
```

RUN >> System.out.print(Arrays.deepToString(checkRow))
OUTPUT >> [[false, false, false, false, false, false, false, false, false], [false,...]]

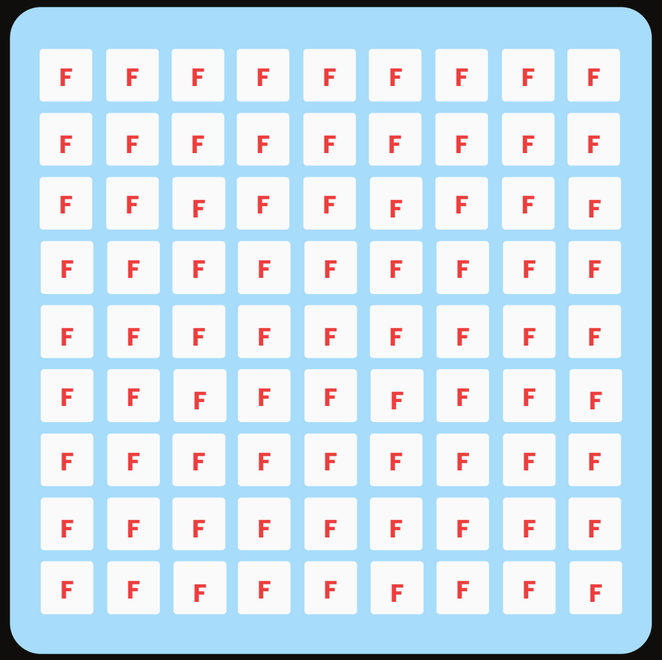
DUTY

- Keeps the status of each row/column/sub-box whether it has already been checked or not (duplicated)
- true = already checked
- false = not yet checked

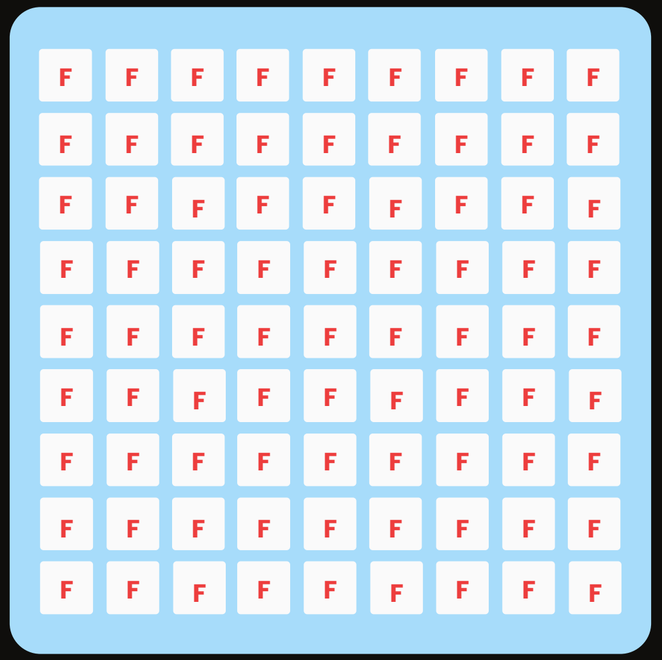
2D ARRAYS

```
boolean[][] checkRow = new boolean[9][9];
boolean[][] checkCol = new boolean[9][9];
boolean[][] checkBox = new boolean[9][9];
```

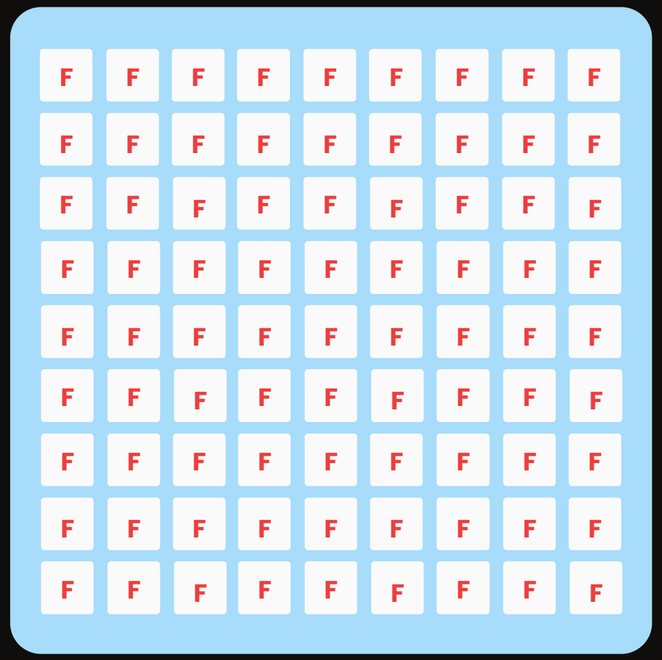
checkRow



checkCol

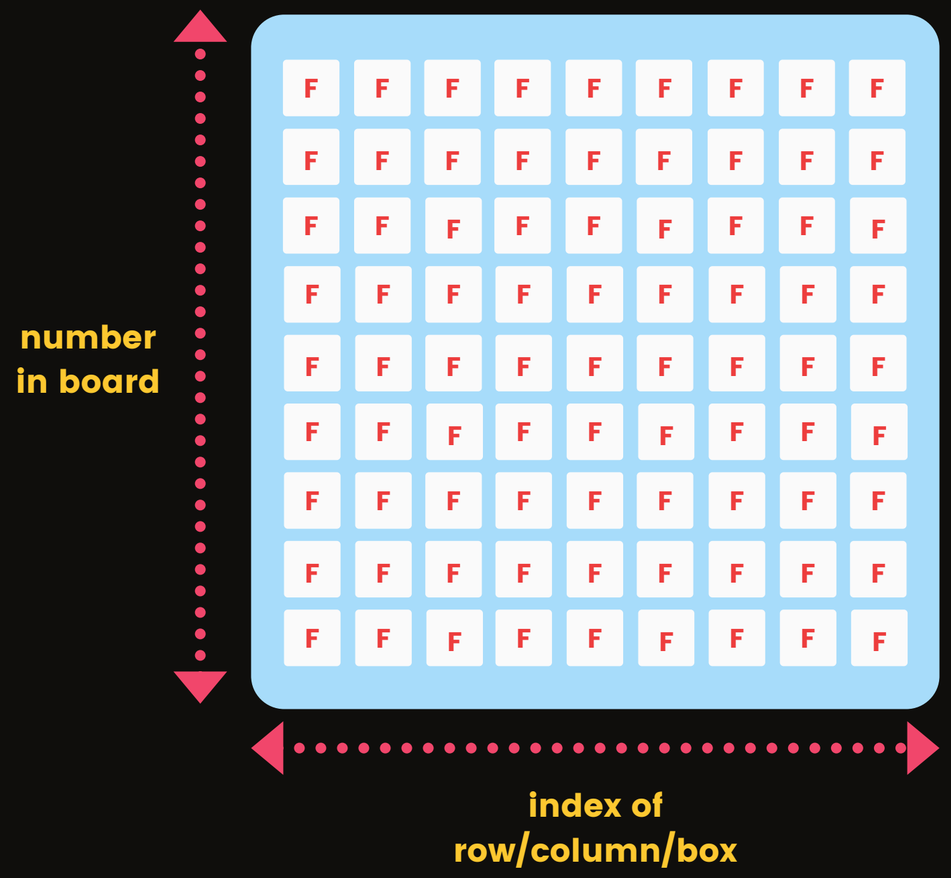


checkBox



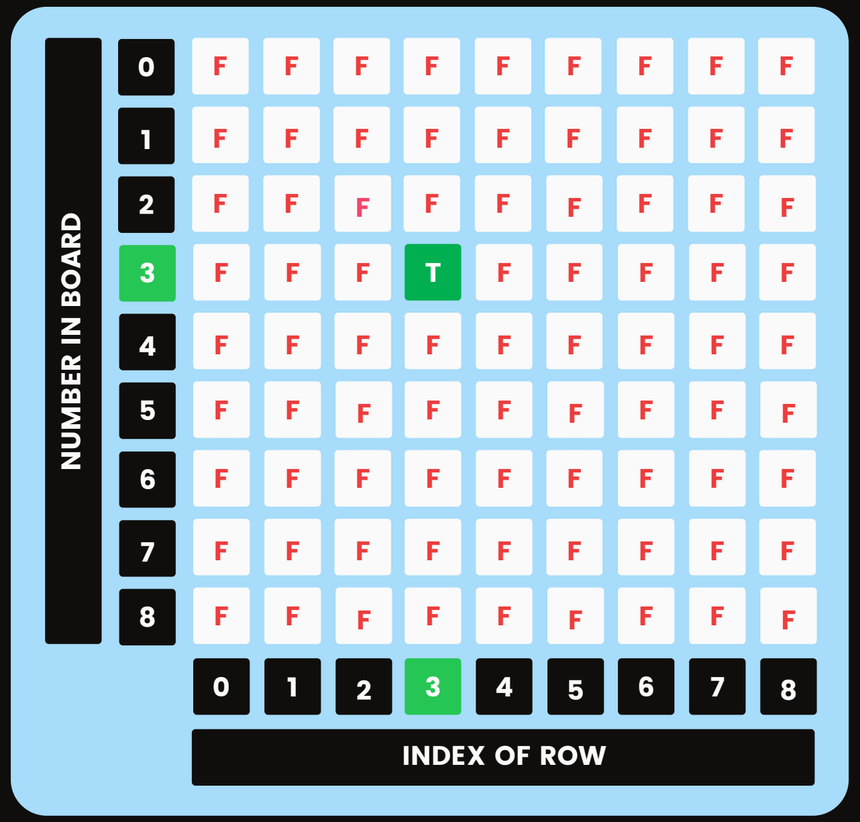
2D ARRAYS

```
boolean[][] checkRow = new boolean[9][9];
boolean[][] checkCol = new boolean[9][9];
boolean[][] checkBox = new boolean[9][9];
```



2D ARRAYS

```
boolean[][] checkRow = new boolean[9][9];
boolean[][] checkCol = new boolean[9][9];
boolean[][] checkBox = new boolean[9][9];
```



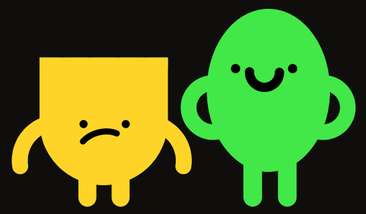
RUN >> System.out.print(checkRow[3][3])
OUTPUT >> true

DUTY

- Loop through the board
  - Access the numbers
- Loop through access status's 2D arrays
  - Access & Update the boolean values

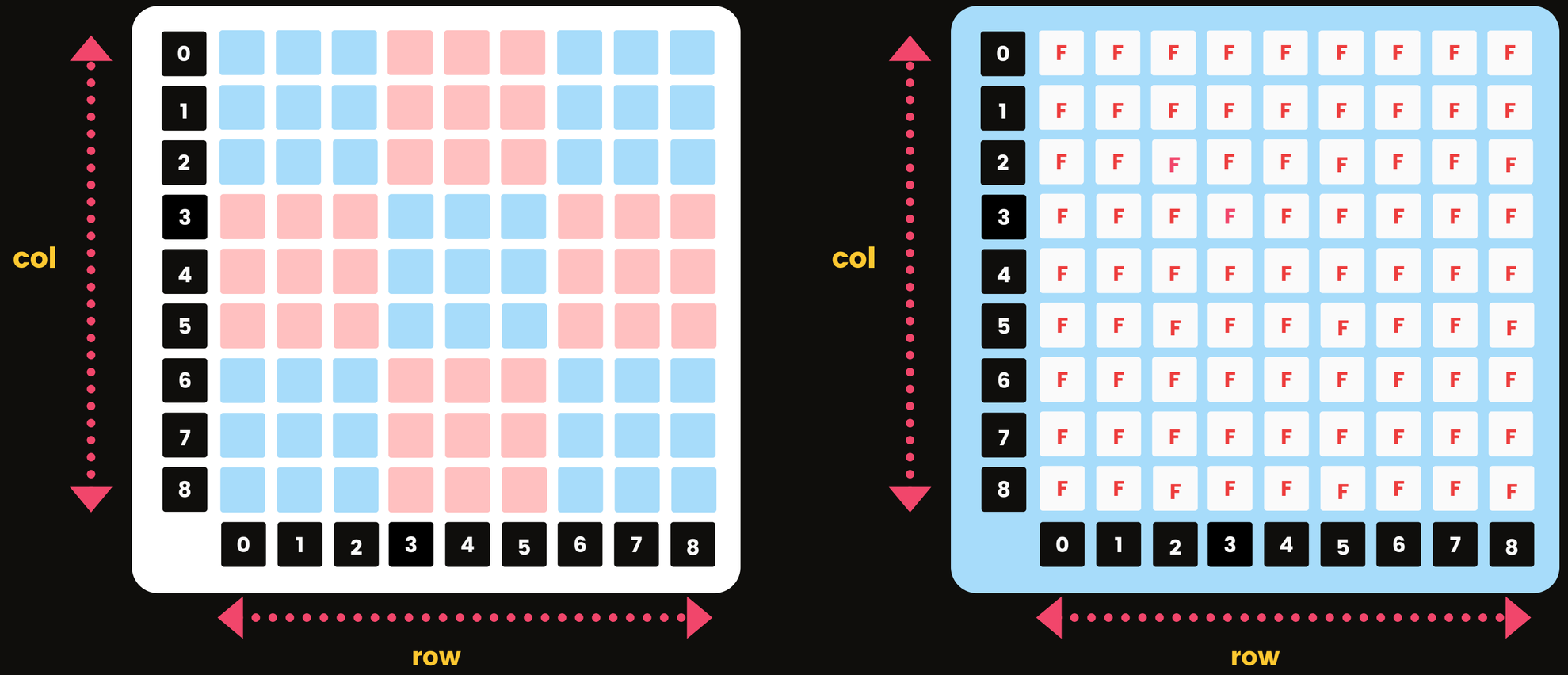
REFERENCE: **Shivam Slingn**
2000shivam659, Leetcode.com

```java
class Solution {
    public boolean isValidSudoku(char[][] board) {

        boolean[][] checkRow = new boolean[9][9];
        boolean[][] checkCol = new boolean[9][9];
        boolean[][] checkBox = new boolean[9][9];

        for (int row = 0; row < board.length; row++) {
            for (int col = 0; col < board.length; col++) {

                if (board[row][col] != '.') {

                    int num = board[row][col] - '1';

                    if (checkRow[num][row]) {
                        return false;
                    } checkRow[num][row] = true;

                    if (checkCol[num][col]) {
                        return false;
                    } checkCol[num][col] = true;

                    int box = ((row/3)*3) + (col/3);

                    if (checkBox[num][box]) {
                        return false;
                    } checkBox[num][box] = true;
                }
            }
        }
        return true;
    }
}
```



FOR-LOOPS

```
for (int row = 0; row < board.length; row++) {
    for (int col = 0; col < board.length; col++) {
```



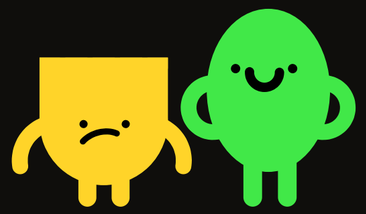
DUTY

- Skip the empty cell

REFERENCE: **Shivam Slingn**

2000shivam659, Leetcode.com

```java
class Solution {
    public boolean isValidSudoku(char[][] board) {

        boolean[][] checkRow = new boolean[9][9];
        boolean[][] checkCol = new boolean[9][9];
        boolean[][] checkBox = new boolean[9][9];

        for (int row = 0; row < board.length; row++) {
            for (int col = 0; col < board.length; col++) {

                if (board[row][col] != '.') {

                    int num = board[row][col] - '1';

                    if (checkRow[num][row]) {
                        return false;
                    } checkRow[num][row] = true;

                    if (checkCol[num][col]) {
                        return false;
                    } checkCol[num][col] = true;

                    int box = ((row/3)*3) + (col/3);

                    if (checkBox[num][box]) {
                        return false;
                    } checkBox[num][box] = true;
                }
            }
        }
        return true;
    }
}
```

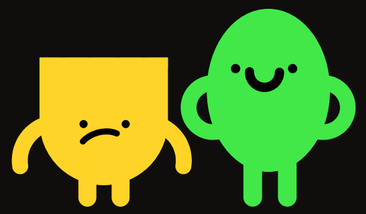


DUTY

- Be the number value (1-9) in board
- Convert char to int
- Used to update:
  - checkRow
  - checkCol
  - checkBox
- as a 'col' index of boolean[][]
- (that they are already checked)

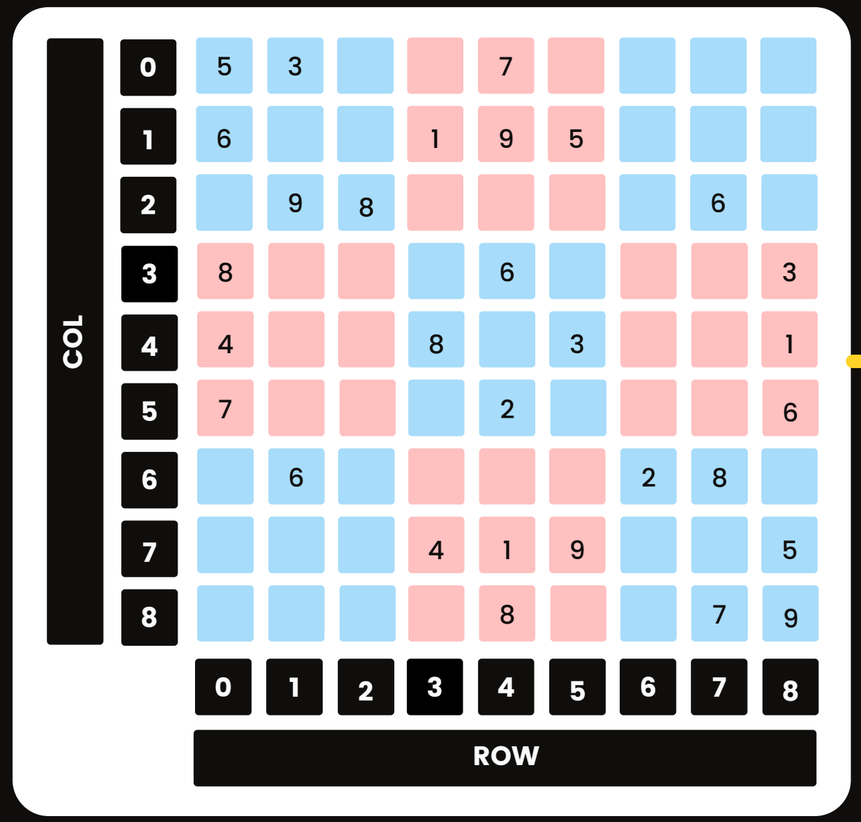
REFERENCE: **Shivam Slingn**
2000shivam659, Leetcode.com

```java
class Solution {
    public boolean isValidSudoku(char[][] board) {

        boolean[][] checkRow = new boolean[9][9];
        boolean[][] checkCol = new boolean[9][9];
        boolean[][] checkBox = new boolean[9][9];

        for (int row = 0; row < board.length; row++) {
            for (int col = 0; col < board.length; col++) {

                if (board[row][col] != '.') {

                    int num = board[row][col] - '1';

                    if (checkRow[num][row]) {
                        return false;
                    } checkRow[num][row] = true;

                    if (checkCol[num][col]) {
                        return false;
                    } checkCol[num][col] = true;

                    int box = ((row/3)*3) + (col/3);

                    if (checkBox[num][box]) {
                        return false;
                    } checkBox[num][box] = true;
                }
            }
        }
        return true;
    }
}
```

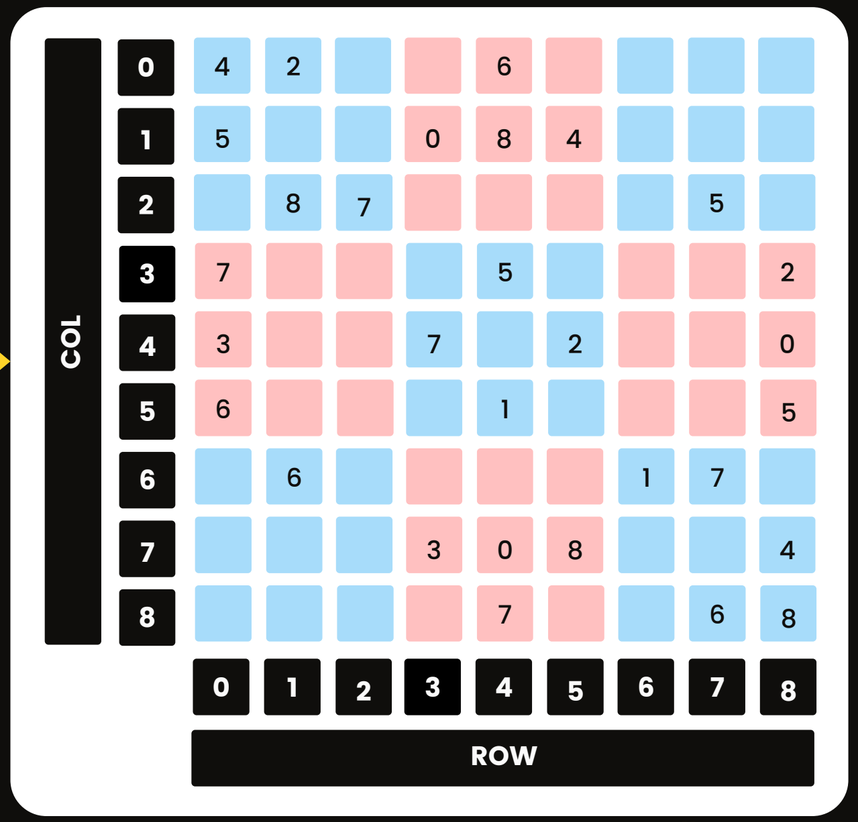


num

```
int num = board[row][col] - '1';
```



num's view



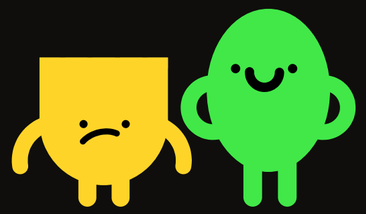
*not actually updated

DUTY

- Find out if a particular number is duplicated in a certain row or not
  - If so, return False (invalid)
  - true = already checked
  - false = not yet checked
- Update checkRow

REFERENCE: **Shivam Slingn**
2000shivam659, Leetcode.com

```java
class Solution {
    public boolean isValidSudoku(char[][] board) {

        boolean[][] checkRow = new boolean[9][9];
        boolean[][] checkCol = new boolean[9][9];
        boolean[][] checkBox = new boolean[9][9];

        for (int row = 0; row < board.length; row++) {
            for (int col = 0; col < board.length; col++) {

                if (board[row][col] != '.') {

                    int num = board[row][col] - '1';

                    if (checkRow[num][row]) {
                        return false;
                    } checkRow[num][row] = true;

                    if (checkCol[num][col]) {
                        return false;
                    } checkCol[num][col] = true;

                    int box = ((row/3)*3) + (col/3);

                    if (checkBox[num][box]) {
                        return false;
                    } checkBox[num][box] = true;
                }
            }
        }
        return true;
    }
}
```

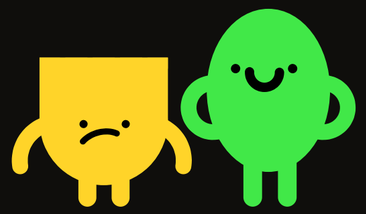


DUTY

- Find out if a particular number is duplicated in a certain column or not
  - If so, return False (invalid)
  - true = already checked
  - false = not yet checked
- Update checkCol

REFERENCE: **Shivam Slingn**
2000shivam659, Leetcode.com

```java
class Solution {
    public boolean isValidSudoku(char[][] board) {

        boolean[][] checkRow = new boolean[9][9];
        boolean[][] checkCol = new boolean[9][9];
        boolean[][] checkBox = new boolean[9][9];

        for (int row = 0; row < board.length; row++) {
            for (int col = 0; col < board.length; col++) {

                if (board[row][col] != '.') {

                    int num = board[row][col] - '1';

                    if (checkRow[num][row]) {
                        return false;
                    } checkRow[num][row] = true;

                    if (checkCol[num][col]) {
                        return false;
                    } checkCol[num][col] = true;

                    int box = ((row/3)*3) + (col/3);

                    if (checkBox[num][box]) {
                        return false;
                    } checkBox[num][box] = true;
                }
            }
        }
        return true;
    }
}
```

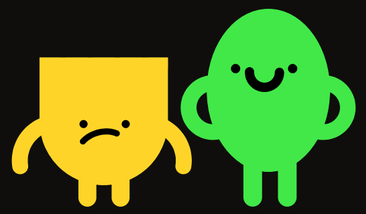


DUTY

- Be the index of numbers in each sub-box

REFERENCE: **Shivam Slingn**
2000shivam659, Leetcode.com

```java
class Solution {
    public boolean isValidSudoku(char[][] board) {

        boolean[][] checkRow = new boolean[9][9];
        boolean[][] checkCol = new boolean[9][9];
        boolean[][] checkBox = new boolean[9][9];

        for (int row = 0; row < board.length; row++) {
            for (int col = 0; col < board.length; col++) {

                if (board[row][col] != '.') {

                    int num = board[row][col] - '1';

                    if (checkRow[num][row]) {
                        return false;
                    } checkRow[num][row] = true;

                    if (checkCol[num][col]) {
                        return false;
                    } checkCol[num][col] = true;

                    int box = ((row/3)*3) + (col/3);

                    if (checkBox[num][box]) {
                        return false;
                    } checkBox[num][box] = true;
                }
            }
        }
        return true;
    }
}
```



box

```
int box = ((row/3)*3) + (col/3);
```

box's view

COL \ ROW	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

COL \ ROW	0	1	2	3	4	5	6	7	8
0	0	0			1				
1	0			1	1	1			
2		0	0					2	
3	3				4				5
4	3			4		4			5
5	3				4				5
6		6					8	8	
7				7	7	7			8
8					7			8	8

*not actually updated

box

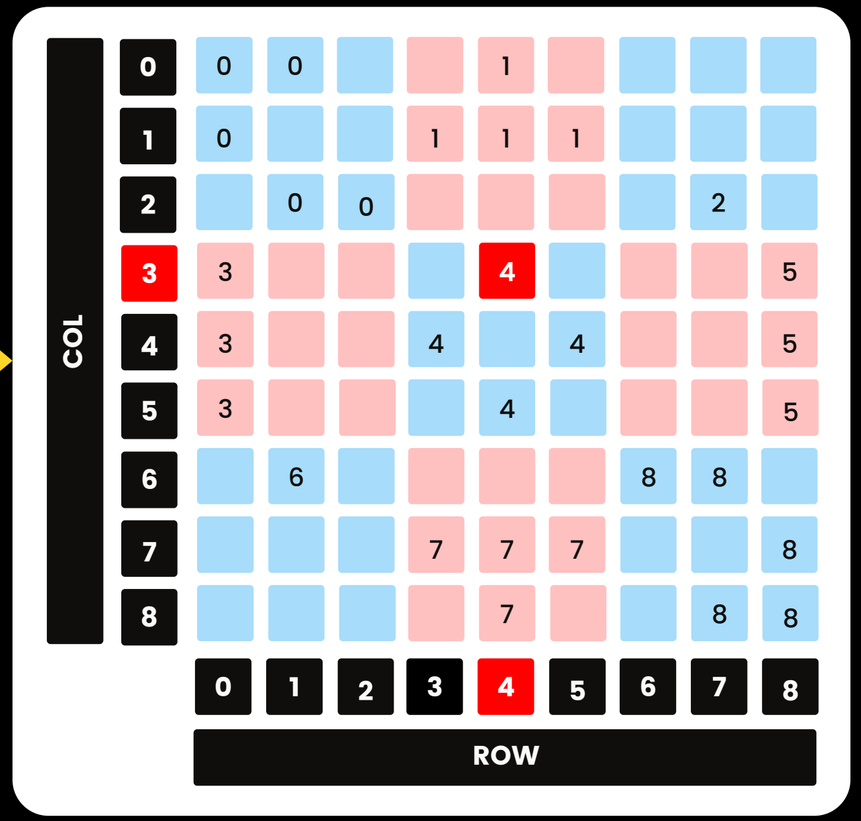
```
int box = ((row/3)*3) + (col/3);
```

EXAMPLE

- row = 4
- col = 3

- box
- = ((row/3)*3) + (col/3)
- =((4/3)*3) + (3/3)
- = (1*3) + 1
- = 4

box's view



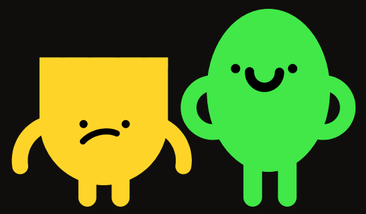
*not actually updated

DUTY

- Find out if a particular number is duplicated in a certain box or not
  - If so, return False (invalid)
  - true = already checked
  - false = not yet checked
- Update checkCol

REFERENCE: **Shivam Slingn**
2000shivam659, Leetcode.com

```java
class Solution {
    public boolean isValidSudoku(char[][] board) {

        boolean[][] checkRow = new boolean[9][9];
        boolean[][] checkCol = new boolean[9][9];
        boolean[][] checkBox = new boolean[9][9];

        for (int row = 0; row < board.length; row++) {
            for (int col = 0; col < board.length; col++) {

                if (board[row][col] != '.') {

                    int num = board[row][col] - '1';

                    if (checkRow[num][row]) {
                        return false;
                    } checkRow[num][row] = true;

                    if (checkCol[num][col]) {
                        return false;
                    } checkCol[num][col] = true;

                    int box = ((row/3)*3) + (col/3);

                    if (checkBox[num][box]) {
                        return false;
                    } checkBox[num][box] = true;
                }
            }
        }
        return true;
    }
}
```



Approach

BRUTE FORCE

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

```java
class Solution {
    public boolean isValidSudoku(char[][] board) {

        boolean[][] checkRow = new boolean[9][9];
        boolean[][] checkCol = new boolean[9][9];
        boolean[][] checkBox = new boolean[9][9];

        for (int row = 0; row < board.length; row++) {
            for (int col = 0; col < board.length; col++) {

                if (board[row][col] != '.') {

                    int num = board[row][col] - '1';

                    if (checkRow[num][row]) {
                        return false;
                    } checkRow[num][row] = true;

                    if (checkCol[num][col]) {
                        return false;
                    } checkCol[num][col] = true;

                    int box = ((row/3)*3) + (col/3);

                    if (checkBox[num][box]) {
                        return false;
                    } checkBox[num][box] = true;
                }
            }
        }
        return true;
    }
}
```

EXAMPLE

ITERATION 1

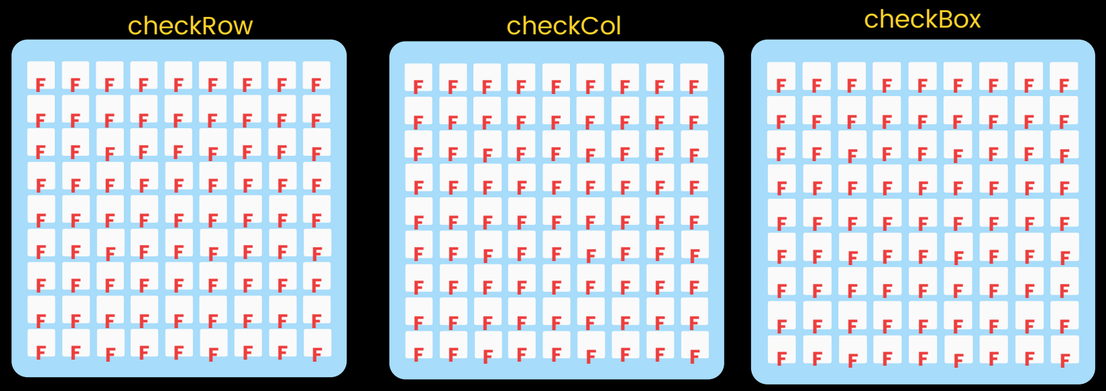
- row = 0
- col = 0

REFERENCE: **Shivam Slingn**

2000shivam659, Leetcode.com

Approach

BRUTE FORCE




```java
class Solution {
    public boolean isValidSudoku(char[][] board) {

        boolean[][] checkRow = new boolean[9][9];
        boolean[][] checkCol = new boolean[9][9];
        boolean[][] checkBox = new boolean[9][9];

        for (int row = 0; row < board.length; row++) {
            for (int col = 0; col < board.length; col++) {

                if (board[row][col] != '.') {

                    int num = board[row][col] - '1';

                    if (checkRow[num][row]) {
                        return false;
                    } checkRow[num][row] = true;

                    if (checkCol[num][col]) {
                        return false;
                    } checkCol[num][col] = true;

                    int box = ((row/3)*3) + (col/3);

                    if (checkBox[num][box]) {
                        return false;
                    } checkBox[num][box] = true;
                }
            }
        }
        return true;
    }
}
```

EXAMPLE

ITERATION 1

- row = 0
- col = 0

Approach

BRUTE FORCE

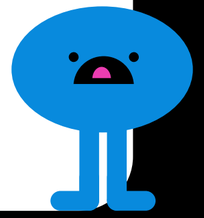
COL \ ROW	0	1	2	3	4	5	6	7	8
0	5	3			7				
1	6			1	9	5			
2		9	8					6	
3	8				6				3
4	4			8		3			1
5	7				2				6
6		6					2	8	
7				4	1	9			5
8					8			7	9

```java
class Solution {
    public boolean isValidSudoku(char[][] board) {

        boolean[][] checkRow = new boolean[9][9];
        boolean[][] checkCol = new boolean[9][9];
        boolean[][] checkBox = new boolean[9][9];

        for (int row = 0; row < board.length; row++) {
            for (int col = 0; col < board.length; col++) {

                if (board[row][col] != '.') {

                    int num = board[row][col] - '1';

                    if (checkRow[num][row]) {
                        return false;
                    } checkRow[num][row] = true;

                    if (checkCol[num][col]) {
                        return false;
                    } checkCol[num][col] = true;

                    int box = ((row/3)*3) + (col/3);

                    if (checkBox[num][box]) {
                        return false;
                    } checkBox[num][box] = true;
                }
            }
        }
        return true;
    }
}
```

EXAMPLE

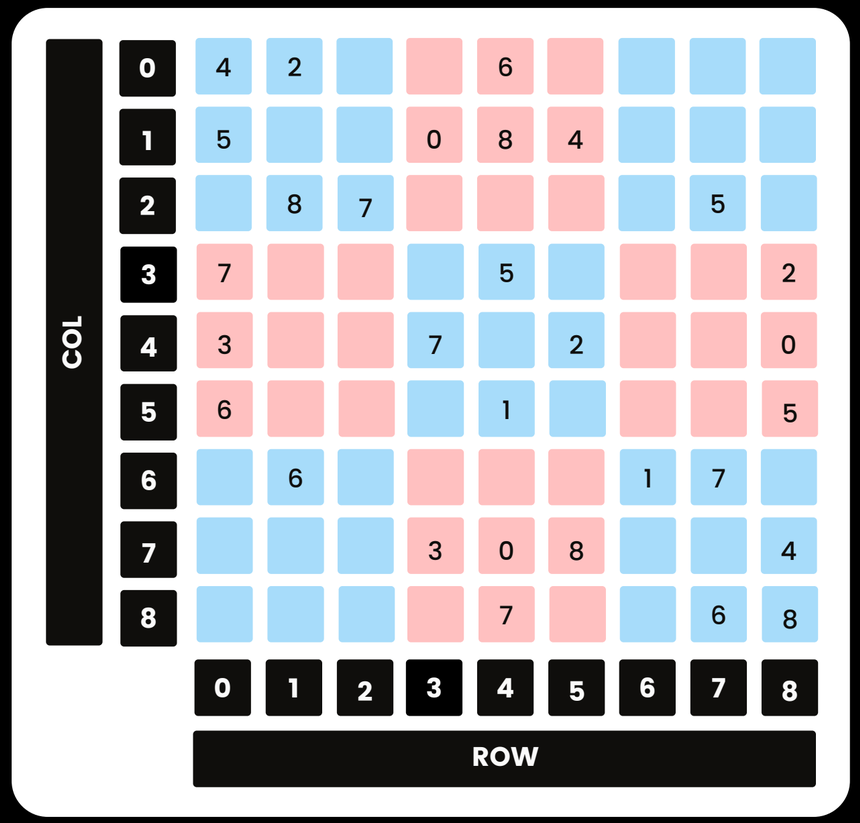
ITERATION 1

- row = 0
- col = 0



Approach

BRUTE FORCE



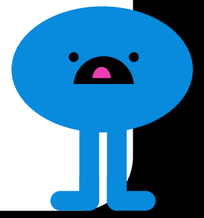
num's view

```java
class Solution {
    public boolean isValidSudoku(char[][] board) {

        boolean[][] checkRow = new boolean[9][9];
        boolean[][] checkCol = new boolean[9][9];
        boolean[][] checkBox = new boolean[9][9];

        for (int row = 0; row < board.length; row++) {
            for (int col = 0; col < board.length; col++) {

                if (board[row][col] != '.') {

                    int num = board[row][col] - '1';

                    if (checkRow[num][row]) {
                        return false;
                    } checkRow[num][row] = true;

                    if (checkCol[num][col]) {
                        return false;
                    } checkCol[num][col] = true;

                    int box = ((row/3)*3) + (col/3);

                    if (checkBox[num][box]) {
                        return false;
                    } checkBox[num][box] = true;
                }
            }
        }
        return true;
    }
}
```

EXAMPLE

ITERATION 1

- row = 0
- col = 0



if (checkRow[num][row])

```
if (checkRow[num][row]) {
    return false;
} checkRow[num][row] = true;
```

EXAMPLE
ITERATION 1
- row = 0
- col = 0

num = 4

board (num's view)

COL \ ROW	0	1	2	3	4	5	6	7	8
0	4	2			6				
1	5			0	8	4			
2		8	7					5	
3	7				5				2
4	3			7		2			0
5	6				1				5
6		6					1	7	
7				3	0	8			4
8					7			6	8

checkRow

NUMBER IN BOARD \ INDEX OF ROW	0	1	2	3	4	5	6	7	8
0	F	F	F	F	F	F	F	F	F
1	F	F	F	F	F	F	F	F	F
2	F	F	F	F	F	F	F	F	F
3	F	F	F	F	F	F	F	F	F
4	F	F	F	F	F	F	F	F	F
5	F	F	F	F	F	F	F	F	F
6	F	F	F	F	F	F	F	F	F
7	F	F	F	F	F	F	F	F	F
8	F	F	F	F	F	F	F	F	F

if (checkRow[num][row])

```
if (checkRow[num][row]) {
    return false;
} checkRow[num][row] = true;
```

EXAMPLE
ITERATION 1
- row = 0
- col = 0

num = 4

board (num's view)

COL \ ROW	0	1	2	3	4	5	6	7	8
0	4	2			6				
1	5			0	8	4			
2		8	7					5	
3	7				5				2
4	3			7		2			0
5	6				1				5
6		6					1	7	
7				3	0	8			4
8					7			6	8

checkRow

NUMBER IN BOARD \ INDEX OF ROW	0	1	2	3	4	5	6	7	8
0	F	F	F	F	F	F	F	F	F
1	F	F	F	F	F	F	F	F	F
2	F	F	F	F	F	F	F	F	F
3	F	F	F	F	F	F	F	F	F
4	T	F	F	F	F	F	F	F	F
5	F	F	F	F	F	F	F	F	F
6	F	F	F	F	F	F	F	F	F
7	F	F	F	F	F	F	F	F	F
8	F	F	F	F	F	F	F	F	F

if (checkRow[num][row])

```
if (checkRow[num][row]) {
    return false;
} checkRow[num][row] = true;
```

INVALID EXAMPLE

- row = 0
- col = 4

num = 4

board (num's view)

COL \ ROW	0	1	2	3	4	5	6	7	8
0	4	2			6	4			
1	5			0	8	4			
2		8	7					5	
3	7				5				2
4	3			7		2			0
5	6				1				5
6		6					1	7	
7				3	0	8			4
8					7			6	8

checkRow

NUMBER IN BOARD \ INDEX OF ROW	0	1	2	3	4	5	6	7	8
0	F	F	F	F	F	F	F	F	F
1	F	F	F	F	F	F	F	F	F
2	F	F	F	F	F	F	F	F	F
3	F	F	F	F	F	F	F	F	F
4	T	F	F	F	F	F	F	F	F
5	F	F	F	F	F	F	F	F	F
6	F	F	F	F	F	F	F	F	F
7	F	F	F	F	F	F	F	F	F
8	F	F	F	F	F	F	F	F	F

if (checkRow[num][row])

```
if (checkRow[num][row]) {
    return false;
} checkRow[num][row] = true;
```

INVALID EXAMPLE

- row = 0
- col = 4

num = 4

board (num's view)

COL	0	1	2	3	4	5	6	7	8
0	4	2			6	4			
1	5			0	8	4			
2		8	7					5	
3	7				5				2
4	3			7		2			0
5	6				1				
6		6							
7				3	0	8			4
8					7			6	8

ROW

INVALID

checkRow

BER IN BOARD	0	1	2	3	4	5	6	7	8
0	F	F	F	F	F	F	F	F	F
1	F	F	F	F	F	F	F	F	F
2	F	F	F	F	F	F	F	F	F
3	F	F	F	F	F	F	F	F	F
4	T	F	F	F	F	F	F	F	F
5	F	F	F	F	F	F	F	F	F
6	F	F	F	F	F	F	F	F	F
7	F	F	F	F	F	F	F	F	F
8	F	F	F	F	F	F	F	F	F

INDEX OF ROW

if (checkCol[num][col])

```
if (checkCol[num][col]) {
    return false;
} checkCol[num][col] = true;
```

EXAMPLE
ITERATION 1
- row = 0
- col = 0

num = 4

board (num's view)

COL \ ROW	0	1	2	3	4	5	6	7	8
0	4	2			6				
1	5			0	8	4			
2		8	7					5	
3	7				5				2
4	3			7		2			0
5	6				1				5
6		6					1	7	
7				3	0	8			4
8					7			6	8

checkCol

NUMBER IN BOARD \ INDEX OF COLUMN	0	1	2	3	4	5	6	7	8
0	F	F	F	F	F	F	F	F	F
1	F	F	F	F	F	F	F	F	F
2	F	F	F	F	F	F	F	F	F
3	F	F	F	F	F	F	F	F	F
4	F	F	F	F	F	F	F	F	F
5	F	F	F	F	F	F	F	F	F
6	F	F	F	F	F	F	F	F	F
7	F	F	F	F	F	F	F	F	F
8	F	F	F	F	F	F	F	F	F

if (checkCol[num][col])

```
if (checkCol[num][col]) {
    return false;
} checkCol[num][col] = true;
```

EXAMPLE
ITERATION 1
- row = 0
- col = 0

num = 4

board (num's view)

COL \ ROW	0	1	2	3	4	5	6	7	8
0	4	2			6				
1	5			0	8	4			
2		8	7					5	
3	7				5				2
4	3			7		2			0
5	6				1				5
6		6					1	7	
7				3	0	8			4
8					7			6	8

checkCol

NUMBER IN BOARD \ INDEX OF COLUMN	0	1	2	3	4	5	6	7	8
0	F	F	F	F	F	F	F	F	F
1	F	F	F	F	F	F	F	F	F
2	F	F	F	F	F	F	F	F	F
3	F	F	F	F	F	F	F	F	F
4	F	F	F	F	F	F	F	F	F
5	F	F	F	F	F	F	F	F	F
6	F	F	F	F	F	F	F	F	F
7	F	F	F	F	F	F	F	F	F
8	F	F	F	F	F	F	F	F	F

if (checkCol[num][col])

```
if (checkCol[num][col]) {
    return false;
} checkCol[num][col] = true;
```

EXAMPLE
ITERATION 1
- row = 0
- col = 0

num = 4

board (num's view)

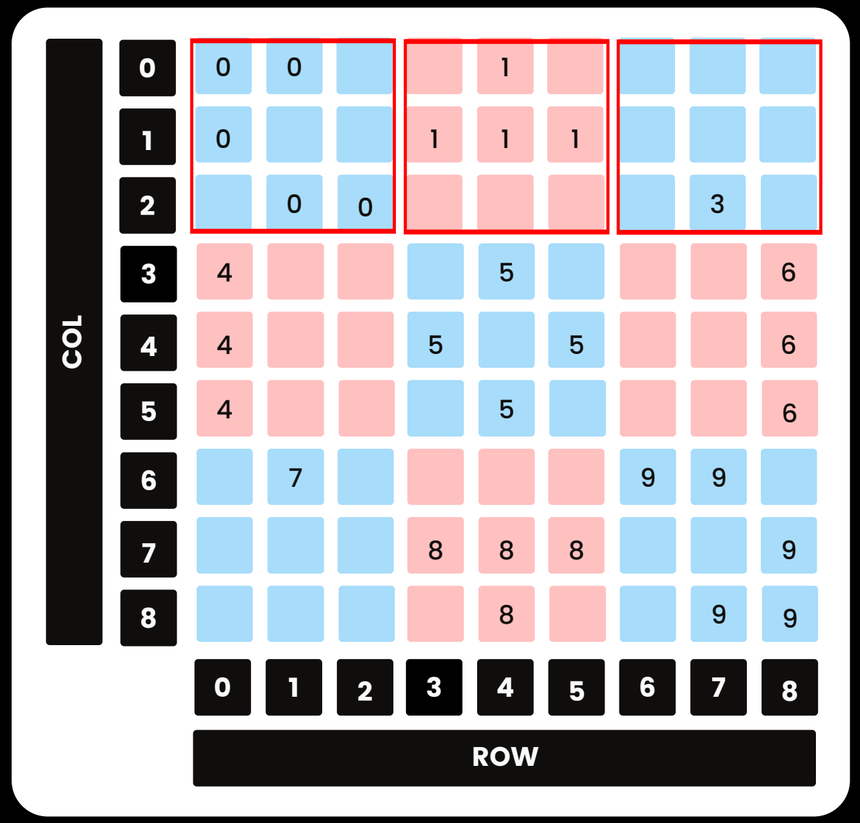
COL \ ROW	0	1	2	3	4	5	6	7	8
0	4	2			6				
1	5			0	8	4			
2		8	7					5	
3	7				5				2
4	3			7		2			0
5	6				1				5
6		6					1	7	
7				3	0	8			4
8					7			6	8

checkCol

NUMBER IN BOARD \ INDEX OF COL	0	1	2	3	4	5	6	7	8
0	F	F	F	F	F	F	F	F	F
1	F	F	F	F	F	F	F	F	F
2	F	F	F	F	F	F	F	F	F
3	F	F	F	F	F	F	F	F	F
4	T	F	F	F	F	F	F	F	F
5	F	F	F	F	F	F	F	F	F
6	F	F	F	F	F	F	F	F	F
7	F	F	F	F	F	F	F	F	F
8	F	F	F	F	F	F	F	F	F

Approach

BRUTE FORCE



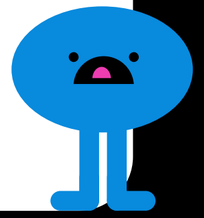
box's view

```java
class Solution {
    public boolean isValidSudoku(char[][] board) {

        boolean[][] checkRow = new boolean[9][9];
        boolean[][] checkCol = new boolean[9][9];
        boolean[][] checkBox = new boolean[9][9];

        for (int row = 0; row < board.length; row++) {
            for (int col = 0; col < board.length; col++) {

                if (board[row][col] != '.') {

                    int num = board[row][col] - '1';

                    if (checkRow[num][row]) {
                        return false;
                    } checkRow[num][row] = true;

                    if (checkCol[num][col]) {
                        return false;
                    } checkCol[num][col] = true;

                    int box = ((row/3)*3) + (col/3);

                    if (checkBox[num][box]) {
                        return false;
                    } checkBox[num][box] = true;
                }
            }
        }
        return true;
    }
}
```

EXAMPLE

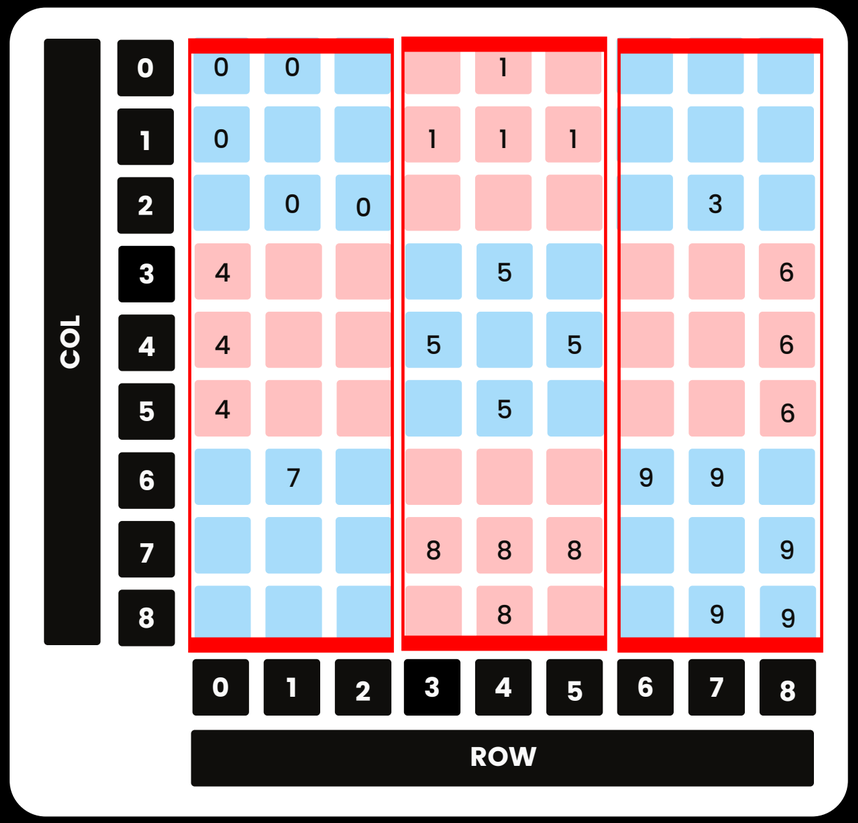
ITERATION 1

- row = 0
- col = 0



Approach

BRUTE FORCE



box's view

```java
class Solution {
    public boolean isValidSudoku(char[][] board) {

        boolean[][] checkRow = new boolean[9][9];
        boolean[][] checkCol = new boolean[9][9];
        boolean[][] checkBox = new boolean[9][9];

        for (int row = 0; row < board.length; row++) {
            for (int col = 0; col < board.length; col++) {

                if (board[row][col] != '.') {

                    int num = board[row][col] - '1';

                    if (checkRow[num][row]) {
                        return false;
                    } checkRow[num][row] = true;

                    if (checkCol[num][col]) {
                        return false;
                    } checkCol[num][col] = true;

                    int box = ((row/3)*3) + (col/3);

                    if (checkBox[num][box]) {
                        return false;
                    } checkBox[num][box] = true;
                }
            }
        }
        return true;
    }
}
```

EXAMPLE

ITERATION 1

- row = 0
- col = 0

if (checkBox[num][box])

```
if (checkBox[num][box]) {
    return false;
} checkBox[num][box] = true;
```

EXAMPLE
ITERATION 1
- row = 0
- col = 0

num = 4
box = 0

board (num's view)

box = 0

COL \ ROW	0	1	2	3	4	5	6	7	8
0	4	2			6				
1	5			0	8	4			
2		8	7					5	
3	7				5				2
4	3			7		2			0
5	6				1				5
6		6					1	7	
7				3	0	8			4
8					7			6	8

checkBox

NUMBER IN BOARD \ INDEX OF BOX	0	1	2	3	4	5	6	7	8
0	F	F	F	F	F	F	F	F	F
1	F	F	F	F	F	F	F	F	F
2	F	F	F	F	F	F	F	F	F
3	F	F	F	F	F	F	F	F	F
4	F	F	F	F	F	F	F	F	F
5	F	F	F	F	F	F	F	F	F
6	F	F	F	F	F	F	F	F	F
7	F	F	F	F	F	F	F	F	F
8	F	F	F	F	F	F	F	F	F

if (checkBox[num][box])

```
if (checkBox[num][box]) {
    return false;
} checkBox[num][box] = true;
```

EXAMPLE
ITERATION 1
- row = 0
- col = 0

num = 4
box = 0

board (num's view)

box = 0

COL \ ROW	0	1	2	3	4	5	6	7	8
0	4	2			6				
1	5			0	8	4			
2		8	7					5	
3	7				5				2
4	3			7		2			0
5	6				1				5
6		6					1	7	
7				3	0	8			4
8					7			6	8

checkBox

NUMBER IN BOARD \ INDEX OF BOX	0	1	2	3	4	5	6	7	8
0	F	F	F	F	F	F	F	F	F
1	F	F	F	F	F	F	F	F	F
2	F	F	F	F	F	F	F	F	F
3	F	F	F	F	F	F	F	F	F
4	T	F	F	F	F	F	F	F	F
5	F	F	F	F	F	F	F	F	F
6	F	F	F	F	F	F	F	F	F
7	F	F	F	F	F	F	F	F	F
8	F	F	F	F	F	F	F	F	F

SUBMISSION

Status	Runtime	Memory	Language
Accepted	1 ms	38.8 MB	java

SUBMISSION

Status	Runtime	Memory	Language
Accepted	1 ms	38.8 MB	java

Valid Sudoku

Submission Detail

507 / 507 test cases passed.

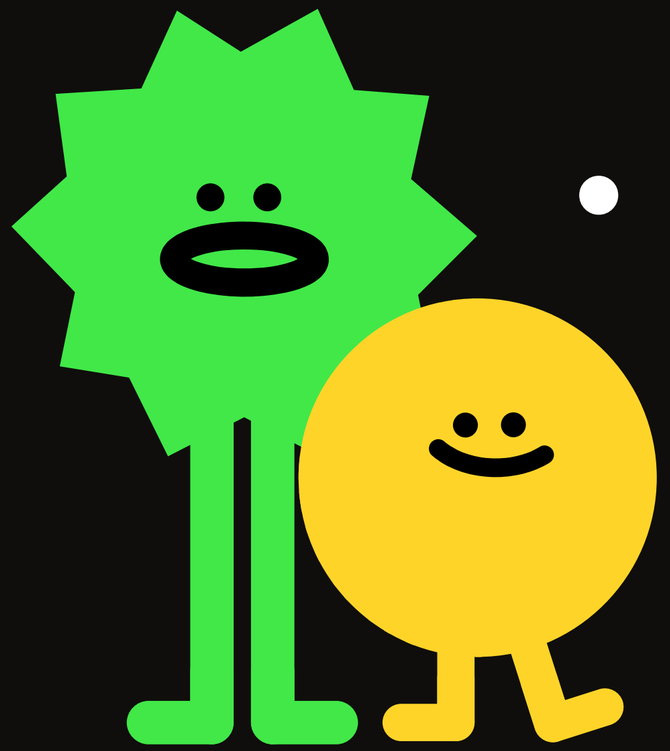
Runtime: **1 ms**
Memory Usage: **38.8 MB**

Status: Accepted

Submitted: **13 hours, 40 minutes ago**

Other Alternatives:

- HASH TABLE
- DICTIONARY
- etc.

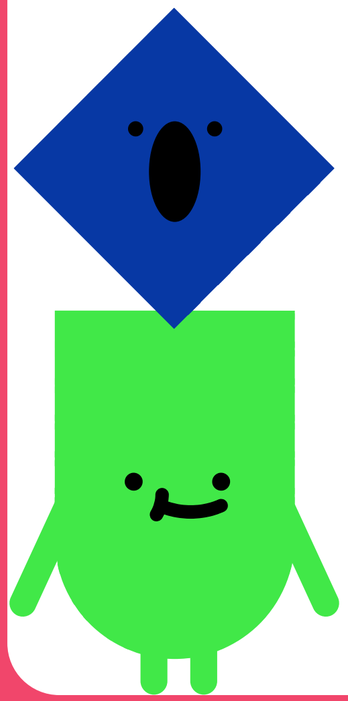


CONCLUSION

Let's go

IDEA SUMMARY

1. Create Memory Space(s)
   - to store Access Status
   - Row, Column, Sub-box
2. Loop through the Board
3. Update the Status
4. return FALSE once found a duplication
5. If not, return TRUE



Another Example

HASH TABLE

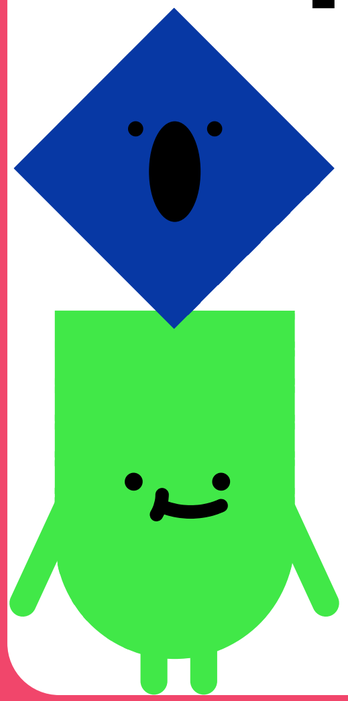
REFERENCE: **Shivam Slingn**

2000shivam659, Leetcode.com

Status	Runtime	Memory	Language
Accepted	15 ms	39.9 MB	java

```java
class Solution {
    public boolean isValidSudoku(char[][] board) {
        Map<String, Integer> map = new HashMap<>();

        for(int i=0; i<9; i++)
            for(int j=0; j<9; j++)
                if(board[i][j] != '.') {

                    String row = board[i][j] + " row " + i;
                    String col = board[i][j] + " col " + j;
                    String box = board[i][j] + " box " + i/3 + "and" + j/3;

                    map.put(row, map.getOrDefault(row, 0) + 1);
                    map.put(col, map.getOrDefault(col, 0) + 1);
                    map.put(box, map.getOrDefault(box, 0) + 1);

                    if(map.get(row) > 1 || map.get(col) > 1 || map.get(box) > 1)
                        return false;
                }

        return true;
    }
}
```

THE SAME LOGIC CAN BE APPLIED IN DIFFERENT ALTERNATIVES



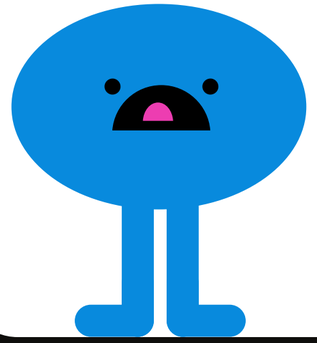
REFERENCES

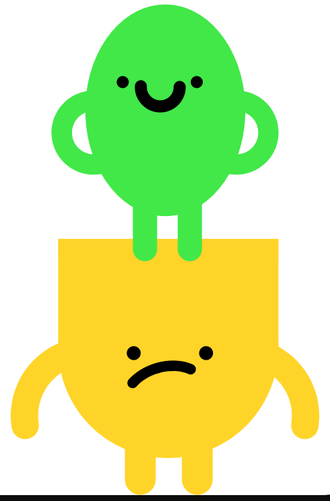
Problem (Valid Sudoku):
leetcode.com/problems/valid-sudoku

Solution:

- **Brute Force & Hash Map**
  - Shivam Slingn,
  - leetcode.com/2000shivam659
  - github.com/2000shivam659

Presentation Template & Elements:
Marketplace Designers, CANVA





Presented by:
Sutisan N. 6213221
Wanitcha S. 6213365

Thank you :D

See you!