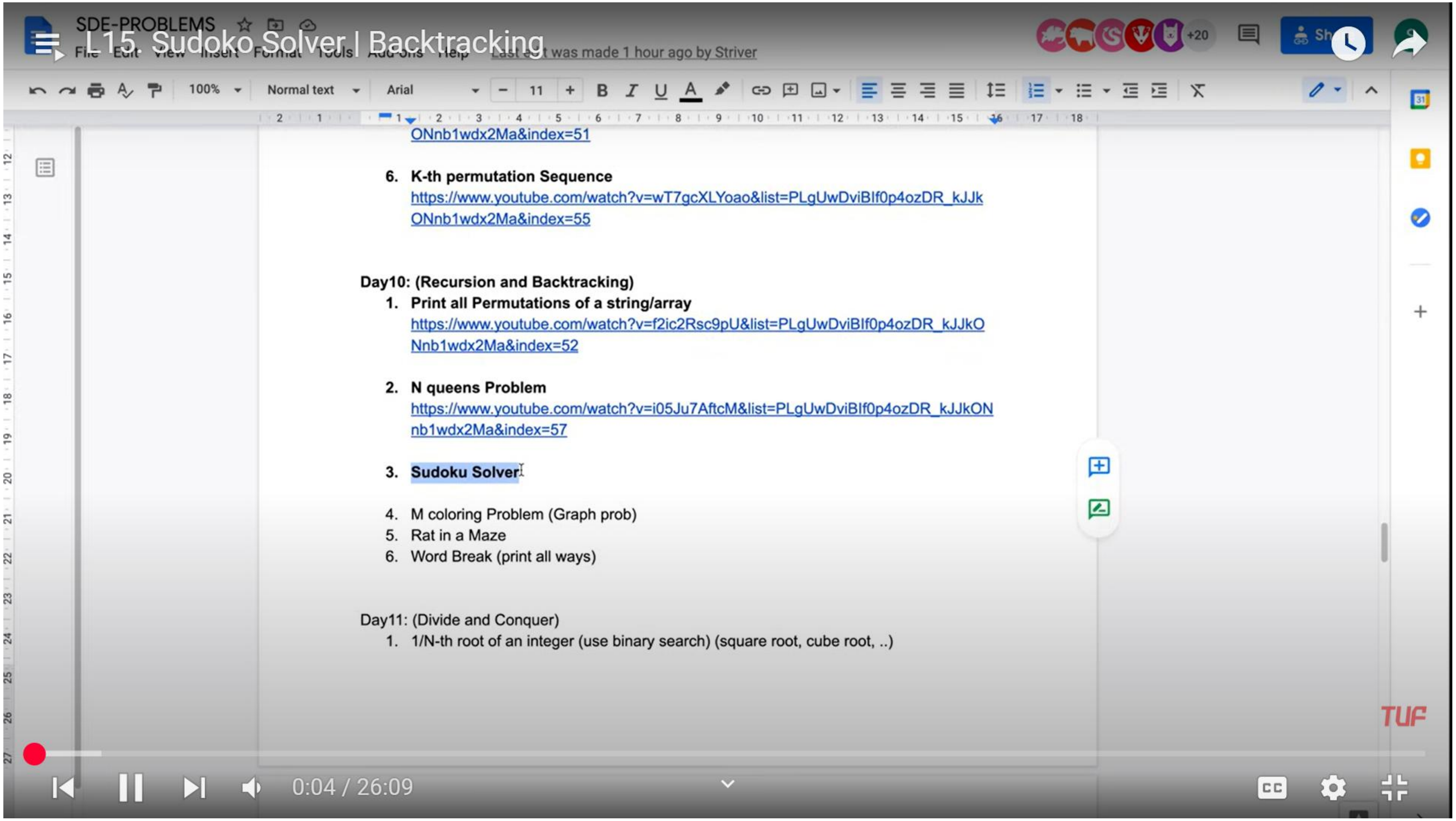

SDE-PROBLEMS
L15. Sudoko Solver | Backtracking
File Edit View Insert Format Tools Add-ons Help
Last edit was made 1 hour ago by Striver
100%
Normal text
Arial
11
TUF
0:04 / 26:09

ONnb1wdx2Ma&index=51

6. **K-th permutation Sequence**
https://www.youtube.com/watch?v=wT7gcXLYoao&list=PLgUwDviBIf0p4ozDR_kJJkONnb1wdx2Ma&index=55

Day10: (Recursion and Backtracking)

1. **Print all Permutations of a string/array**
https://www.youtube.com/watch?v=f2ic2Rsc9pU&list=PLgUwDviBIf0p4ozDR_kJJkONnb1wdx2Ma&index=52

2. **N queens Problem**
https://www.youtube.com/watch?v=i05Ju7AftcM&list=PLgUwDviBIf0p4ozDR_kJJkONnb1wdx2Ma&index=57

3. **Sudoku Solver**

4. M coloring Problem (Graph prob)
5. Rat in a Maze
6. Word Break (print all ways)

Day11: (Divide and Conquer)

1. 1/N-th root of an integer (use binary search) (square root, cube root, ..)

37. Sudoku Solver

Hard 2779 106 Add to List Share

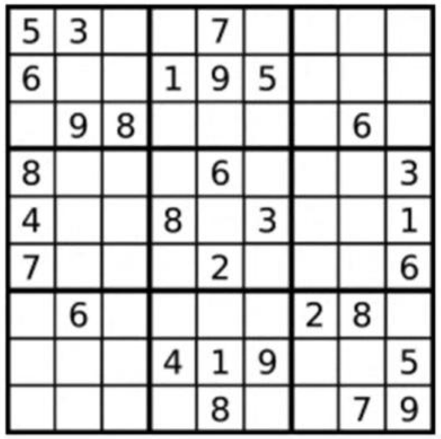
Write a program to solve a Sudoku puzzle by filling the empty cells.

A sudoku solution must satisfy **all of the following rules**:

1. Each of the digits `1-9` must occur exactly once in each row.
2. Each of the digits `1-9` must occur exactly once in each column.
3. Each of the digits `1-9` must occur exactly once in each of the 9 `3x3` sub-boxes of the grid.

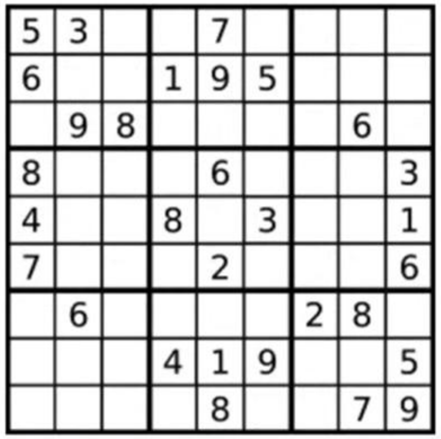
The `'.'` character indicates empty cells.

Example 1:



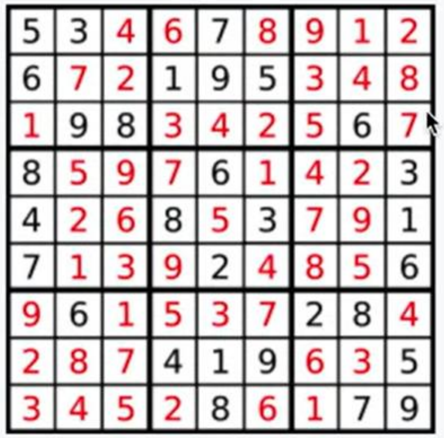
```
Input: board = [["5","3",".",".","7",".",".",".","."],["6",".",".","1","9","5",".",".","."],[".","9","8",".",".",".",".","6","."],
["8",".",".",".","6",".",".",".","3"],["4",".",".","8",".","3",".",".","1"],["7",".",".",".","2",".",".",".","6"],[".","6",".",".",".",".","2","8","."],
[".",".",".","4","1","9",".",".","5"],[".",".",".",".","8",".",".","7","9"]]
Output: [["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"],
```

Example 1:



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

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: [["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 only valid solution is shown below:



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

$\underline{\underline{9 \times 9}}$.

$\underline{9}$ (3

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

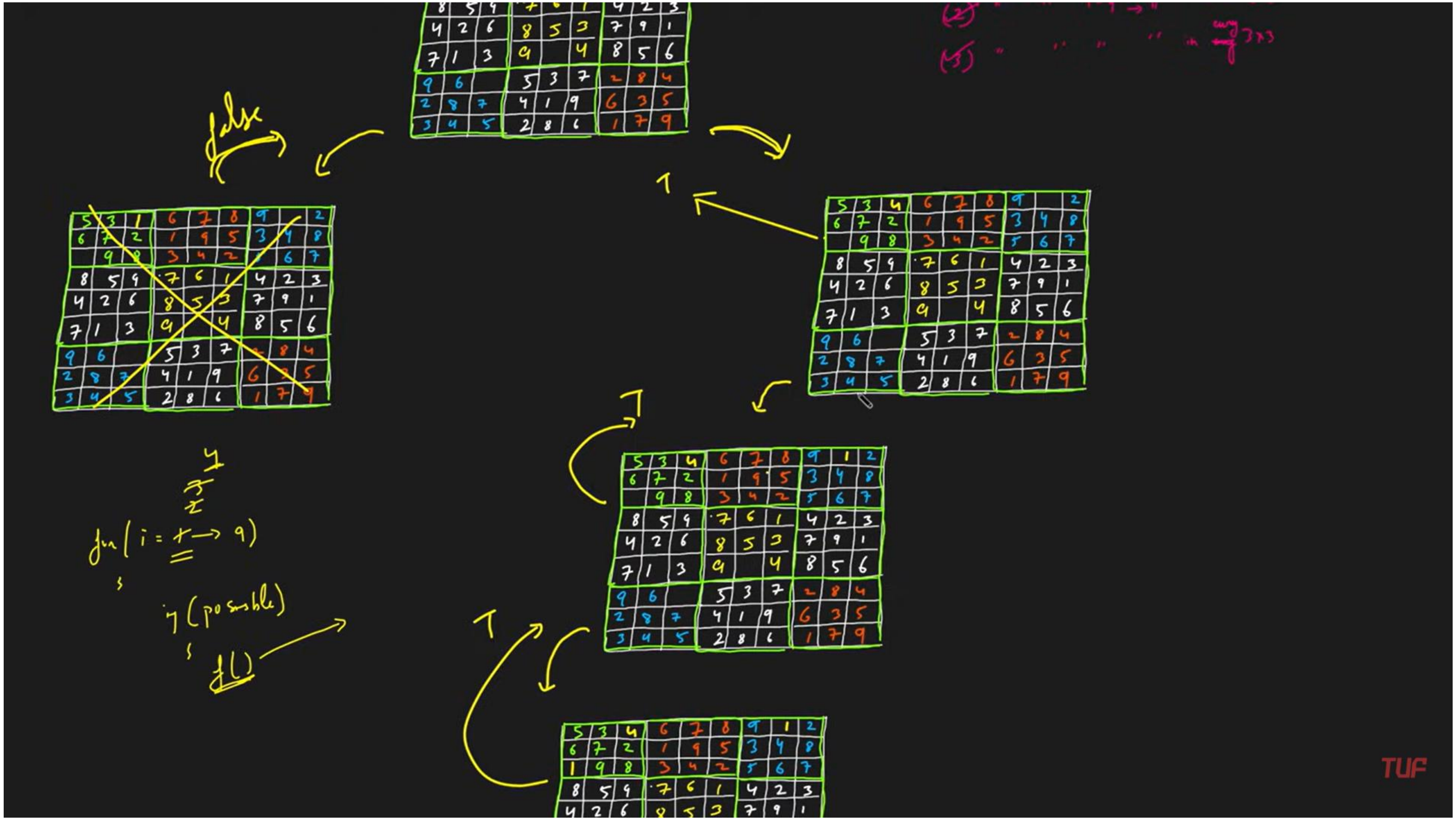
$\underline{\underline{9 \times 9}}$.

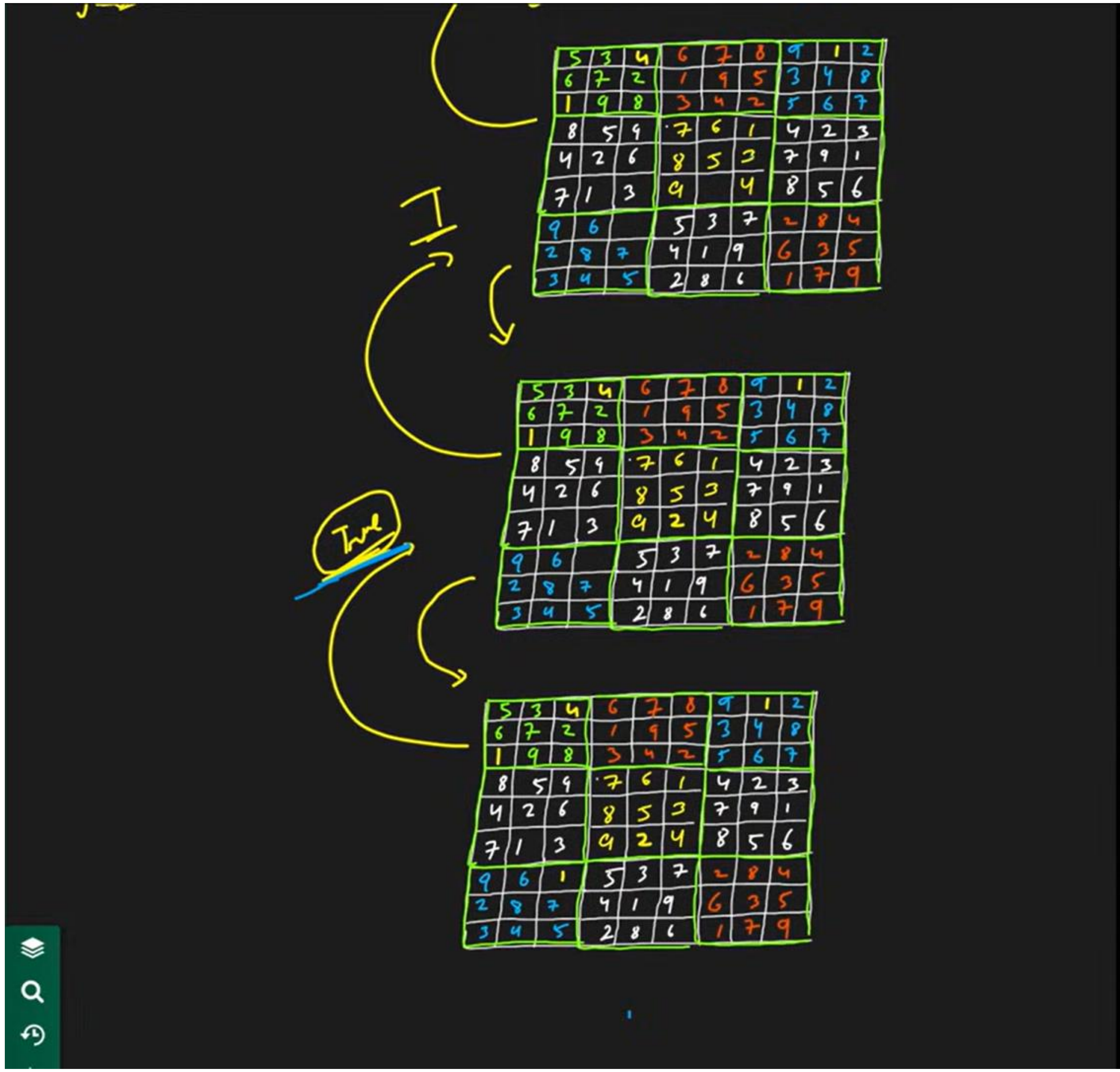
$\underline{\underline{9}}$ $(\underline{\underline{3 \times 3}})$

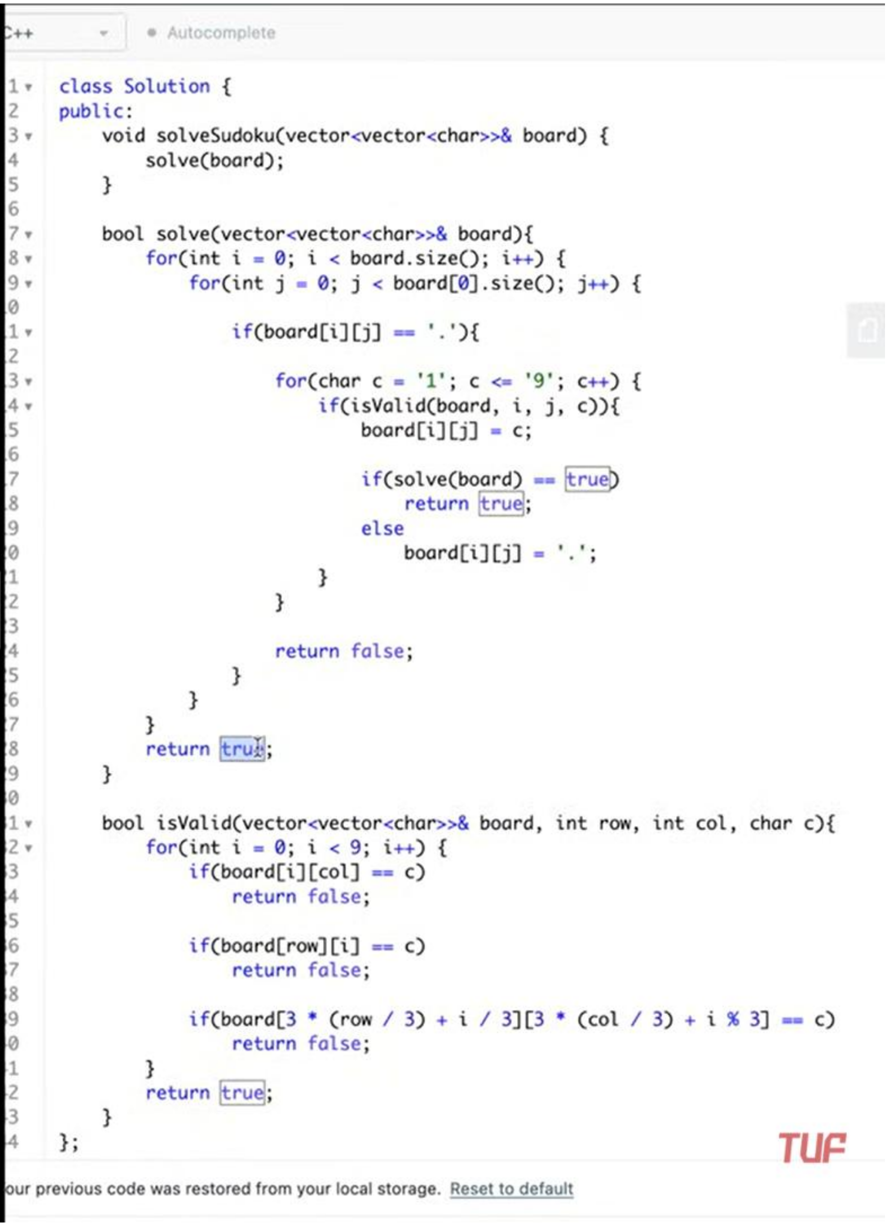
(1) The digit 1-9 → once in ~~any~~ every row

(2) " " 1-9 → " " ~~#~~ every col

(3) " " " " in ~~any~~ every 3×3

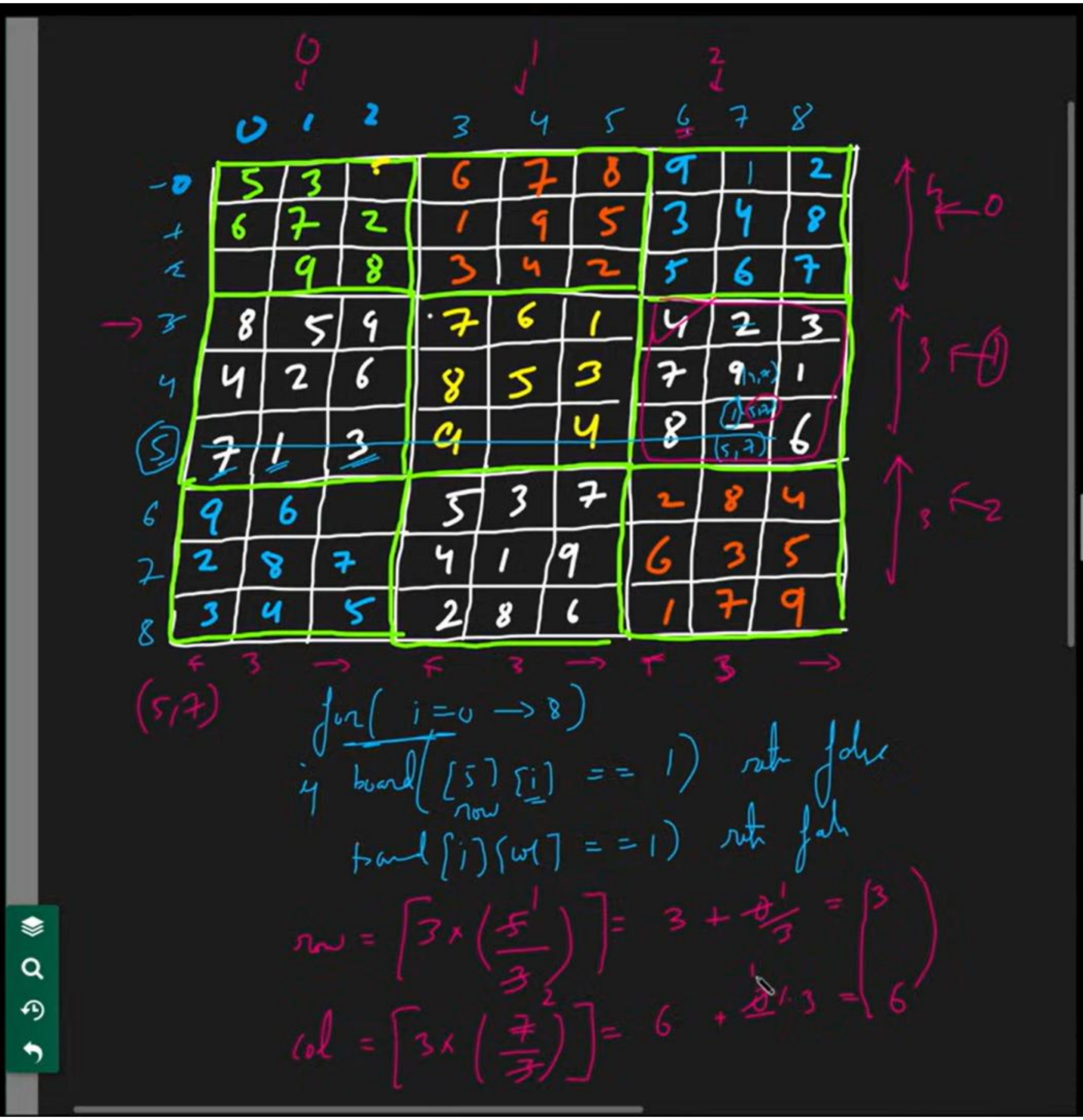

false
T
T
T
4
3
2
fn (i = ~~1~~ → 9)
{
if (possible)
{
f()
(3) " " " " in every 3x3
TUF



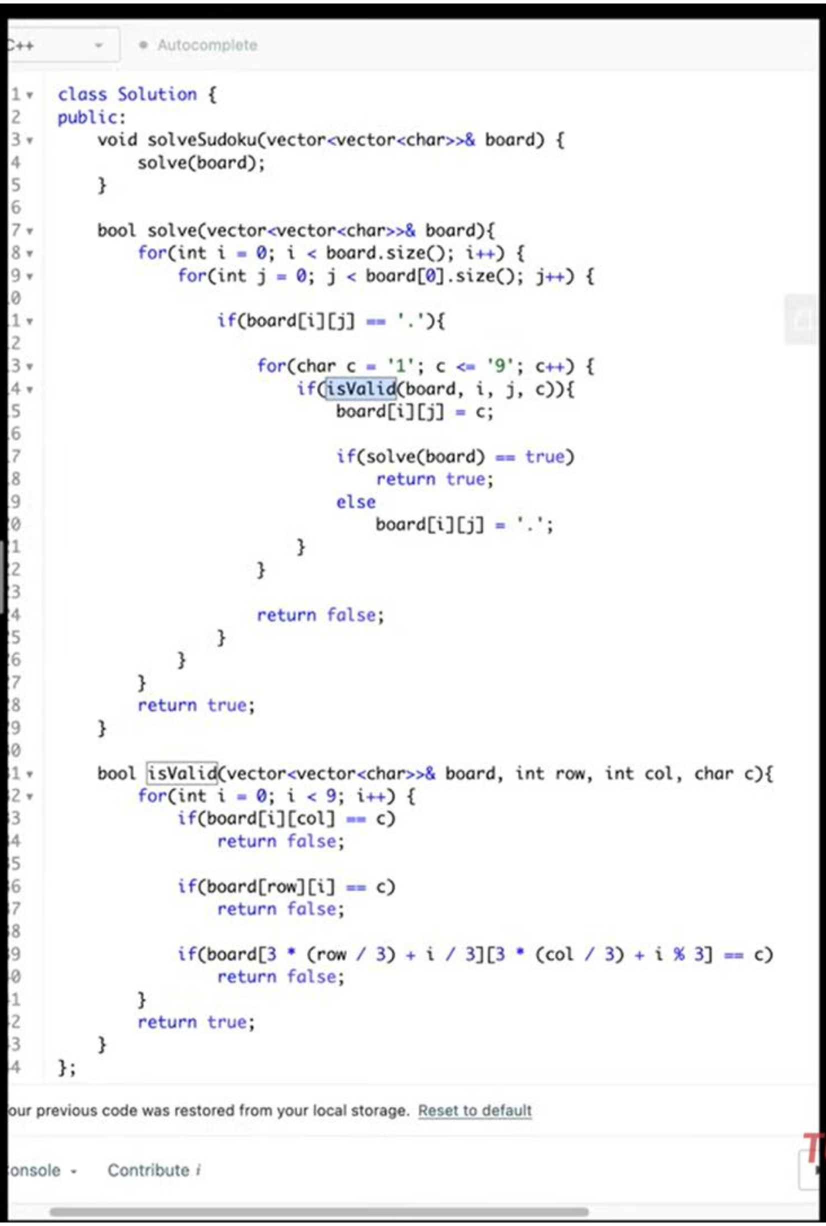


```
class Solution {
public:
    void solveSudoku(vector<vector<char>>& board) {
        solve(board);
    }

    bool solve(vector<vector<char>>& board){
        for(int i = 0; i < board.size(); i++) {
            for(int j = 0; j < board[0].size(); j++) {

                if(board[i][j] == '.'){

                    for(char c = '1'; c <= '9'; c++) {
                        if(isValid(board, i, j, c)){
                            board[i][j] = c;

                            if(solve(board) == true)
                                return true;
                            else
                                board[i][j] = '.';
                        }
                    }

                    return false;
                }
            }
        }
        return true;
    }

    bool isValid(vector<vector<char>>& board, int row, int col, char c){
        for(int i = 0; i < 9; i++) {
            if(board[i][col] == c)
                return false;

            if(board[row][i] == c)
                return false;

            if(board[3 * (row / 3) + i / 3][3 * (col / 3) + i % 3] == c)
                return false;
        }
        return true;
    }
};
```

our previous code was restored from your local storage. Reset to default



for(i=0 → 8)
if board[5][i] == 1) rtn false
board[i][col] == 1) rtn false
row = [3×(5/3)] = 3 + 0/3 = 3
col = [3×(7/3)] = 6 + 0%3 = 6
(5,7)



```
class Solution {
public:
    void solveSudoku(vector<vector<char>>& board) {
        solve(board);
    }

    bool solve(vector<vector<char>>& board){
        for(int i = 0; i < board.size(); i++) {
            for(int j = 0; j < board[0].size(); j++) {

                if(board[i][j] == '.'){

                    for(char c = '1'; c <= '9'; c++) {
                        if(isValid(board, i, j, c)){
                            board[i][j] = c;

                            if(solve(board) == true)
                                return true;
                            else
                                board[i][j] = '.';
                        }
                    }

                    return false;
                }
            }
        }
        return true;
    }

    bool isValid(vector<vector<char>>& board, int row, int col, char c){
        for(int i = 0; i < 9; i++) {
            if(board[i][col] == c)
                return false;

            if(board[row][i] == c)
                return false;

            if(board[3 * (row / 3) + i / 3][3 * (col / 3) + i % 3] == c)
                return false;
        }
        return true;
    }
};
```

our previous code was restored from your local storage. Reset to default