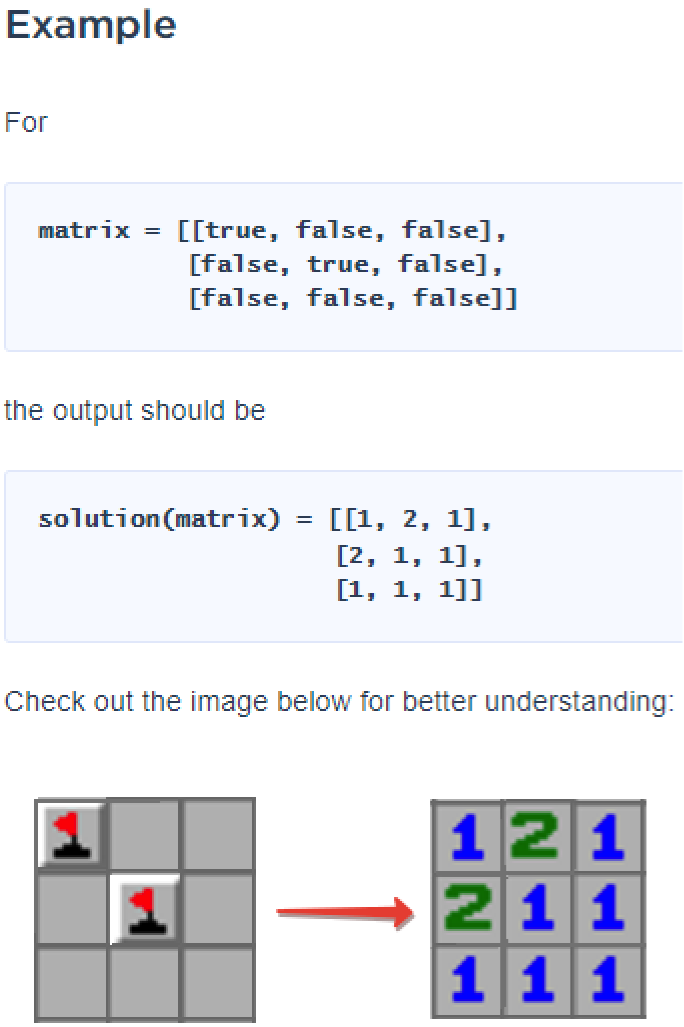
Coding Problem: -

In the popular **Minesweeper** game you have a board with some mines and those cells that don't contain a mine have a number in it that indicates the total number of mines in the neighboring cells. Starting off with some arrangement of mines we want to create a **Minesweeper** game setup.



Example

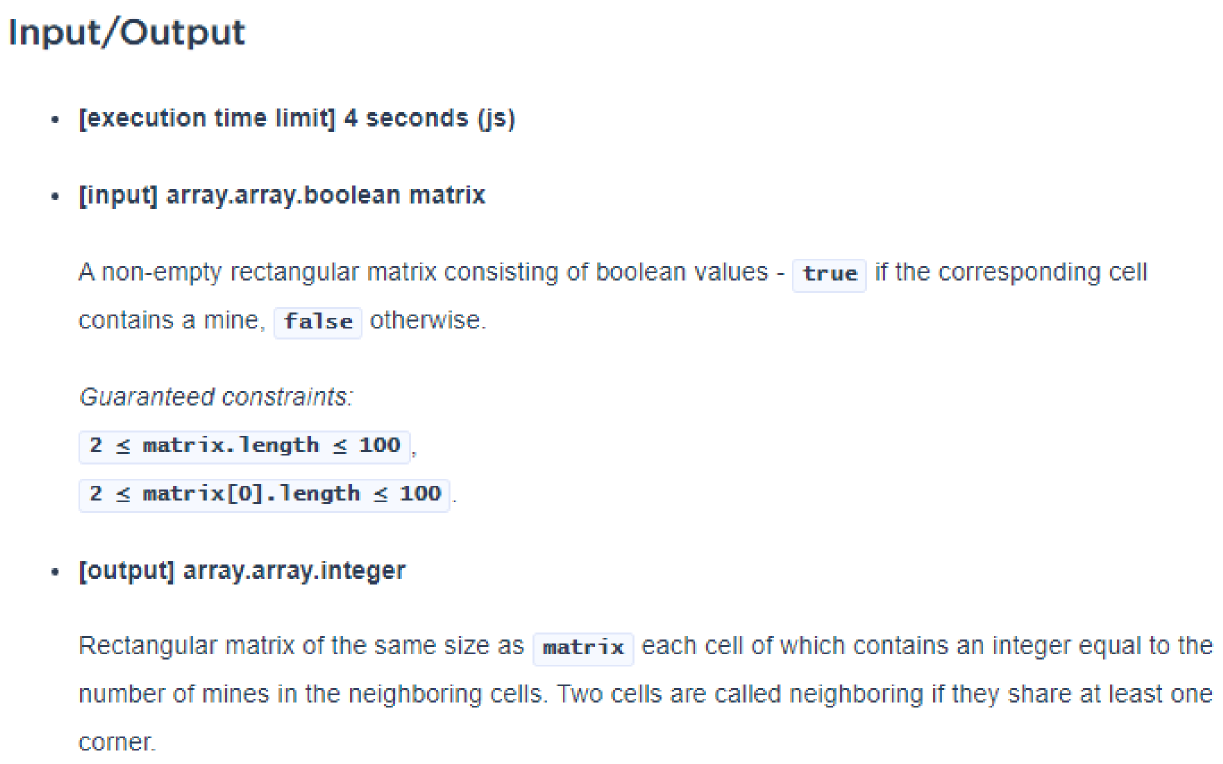
For

```
matrix = [[true, false, false],
          [false, true, false],
          [false, false, false]]
```

the output should be

```
solution(matrix) = [[1, 2, 1],
                    [2, 1, 1],
                    [1, 1, 1]]
```

Check out the image below for better understanding:



Input/Output

- **[execution time limit] 4 seconds (js)**
- **[input] array.array.boolean matrix**

  A non-empty rectangular matrix consisting of boolean values - `true` if the corresponding cell contains a mine, `false` otherwise.

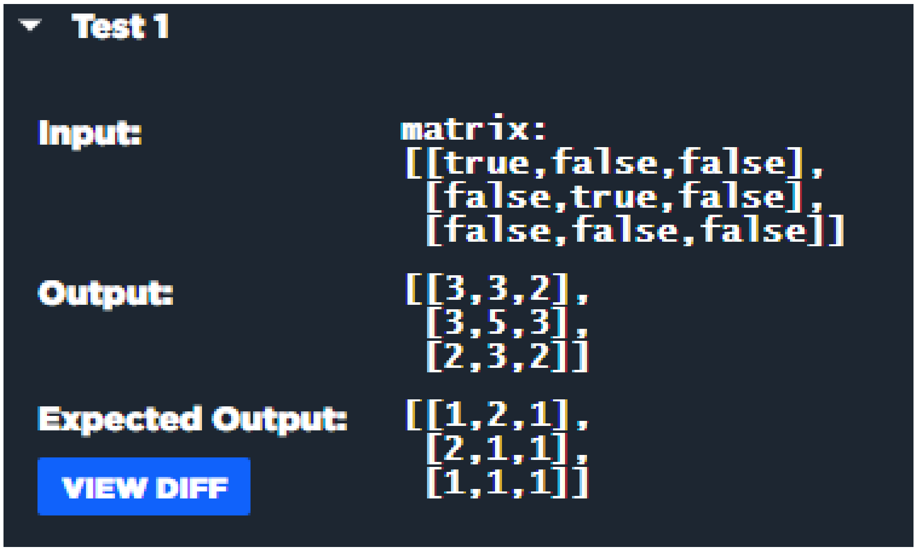
  Guaranteed constraints:
  `2 ≤ matrix.length ≤ 100`,
  `2 ≤ matrix[0].length ≤ 100`.

- **[output] array.array.integer**

  Rectangular matrix of the same size as `matrix` each cell of which contains an integer equal to the number of mines in the neighboring cells. Two cells are called neighboring if they share at least one corner.

Test Cases: -

Test Case 1: -



Input:

```
matrix:
[[true,false,false],
 [false,true,false],
 [false,false,false]]
```

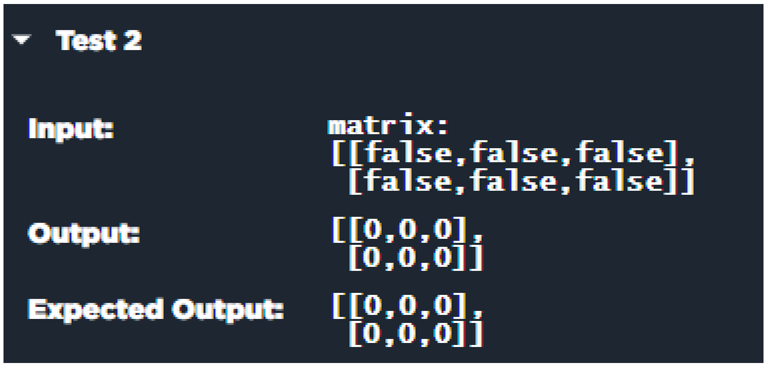
Output:

```
[[3,3,2],
 [3,5,3],
 [2,3,2]]
```

Expected Output:

```
[[1,2,1],
 [2,1,1],
 [1,1,1]]
```

Test Case 2: -



Test 2

Input:

```
matrix:
[[false,false,false],
 [false,false,false]]
```

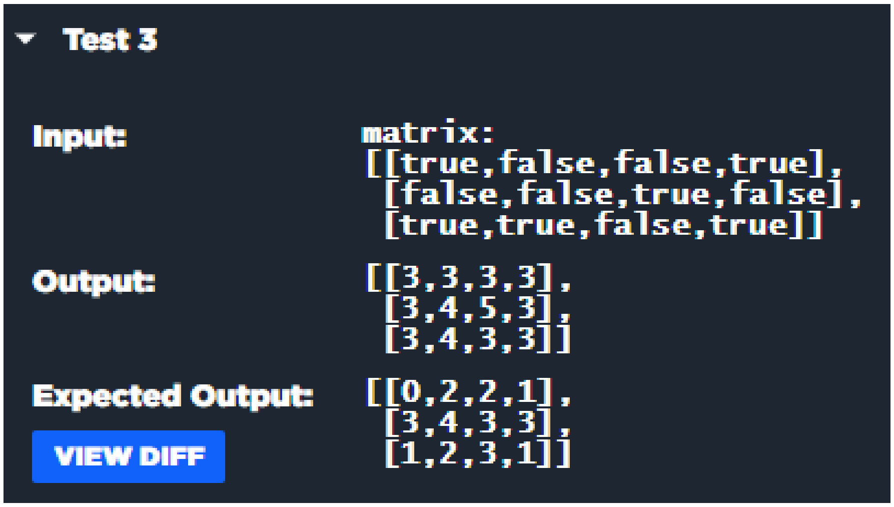
Output:

```
[[0,0,0],
 [0,0,0]]
```

Expected Output:

```
[[0,0,0],
 [0,0,0]]
```

Test Case 3: -



Test 3

Input:

```
matrix:
[[true,false,false,true],
 [false,false,true,false],
 [true,true,false,true]]
```

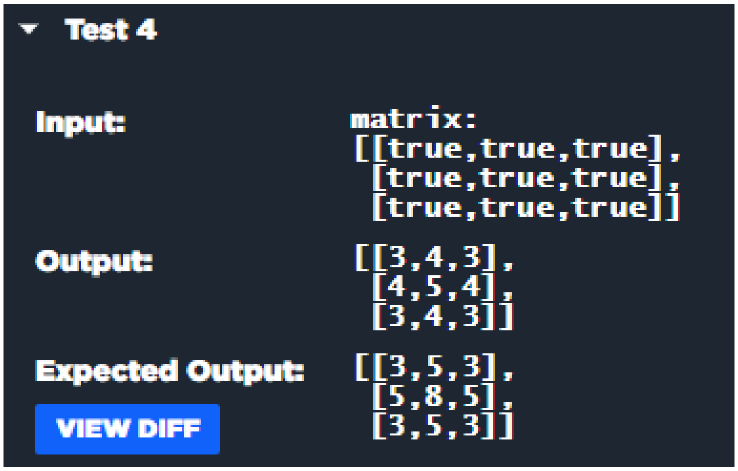
Output:

```
[[3,3,3,3],
 [3,4,5,3],
 [3,4,3,3]]
```

Expected Output:

```
[[0,2,2,1],
 [3,4,3,3],
 [1,2,3,1]]
```

Test Case 4: -



Test 4

Input:

```
matrix:
[[true,true,true],
 [true,true,true],
 [true,true,true]]
```

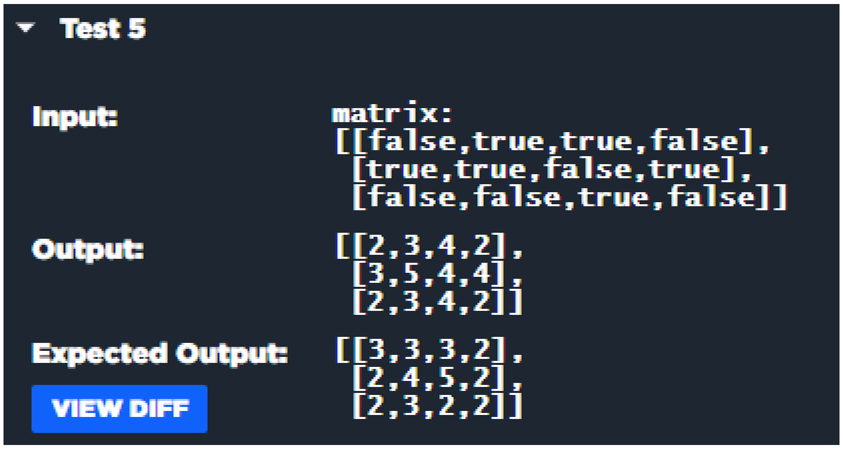
Output:

```
[[3,4,3],
 [4,5,4],
 [3,4,3]]
```

Expected Output:

```
[[3,5,3],
 [5,8,5],
 [3,5,3]]
```

Test Case 5: -



Input:

```
matrix:
[[false,true,true,false],
 [true,true,false,true],
 [false,false,true,false]]
```

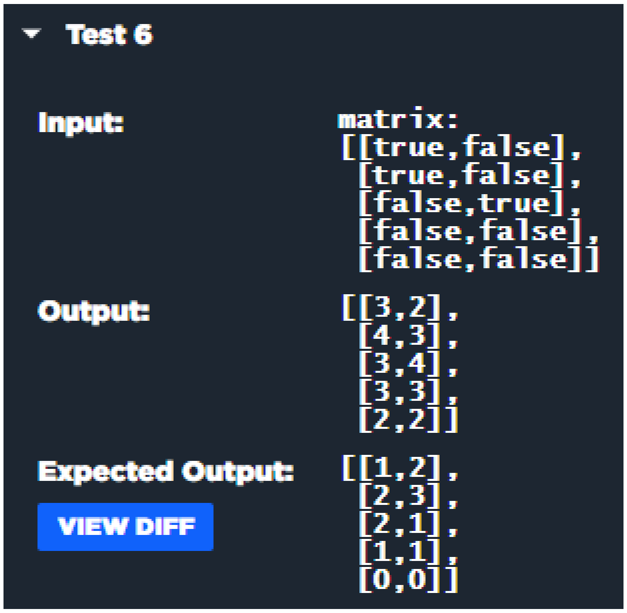
Output:

```
[[2,3,4,2],
 [3,5,4,4],
 [2,3,4,2]]
```

Expected Output:

```
[[3,3,3,2],
 [2,4,5,2],
 [2,3,2,2]]
```

Test Case 6: -



Input:

```
matrix:
[[true,false],
 [true,false],
 [false,true],
 [false,false],
 [false,false]]
```

Output:

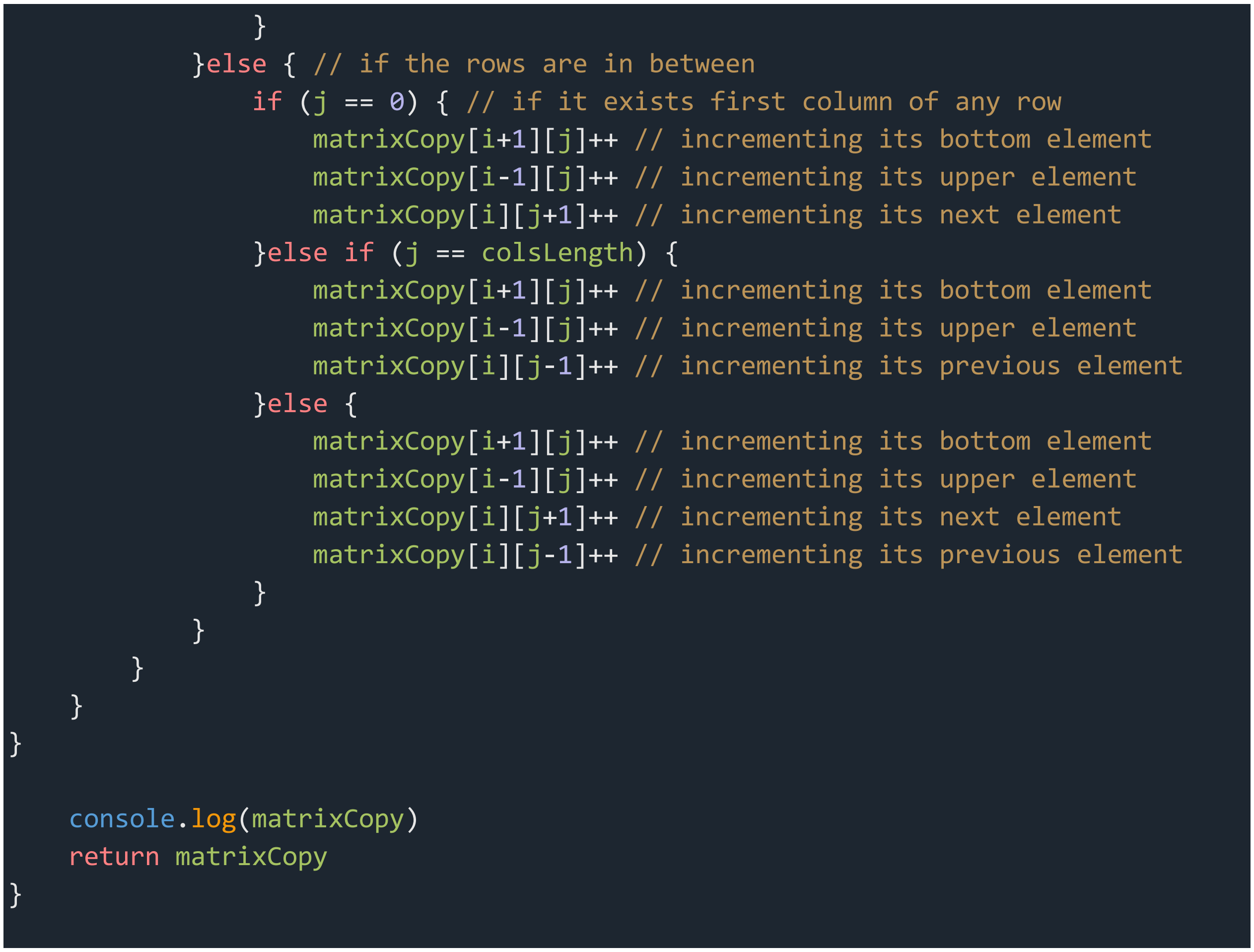
```
[[3,2],
 [4,3],
 [3,4],
 [3,3],
 [2,2]]
```

Expected Output:

```
[[1,2],
 [2,3],
 [2,1],
 [1,1],
 [0,0]]
```

My Solution: -

```
function solution(matrix) {
    let rowsLength = matrix.length -1
    let colsLength = matrix[0].length -1
    let matrixCopy = [...matrix]
    for (let i in matrixCopy){
        for (let j in matrixCopy[i]){
            if(matrixCopy[i][j]){
                matrixCopy[i][j] = 1
            }else{
                matrixCopy[i][j] = 0
            }
        }
    }
    console.log(matrixCopy)
    for (let i = 0; i <= rowsLength; i++) {
    for (let j = 0; j <= colsLength; j++) {
        if (matrixCopy[i][j]) { // if matrix has true value, converts their neighbors with ++
            if (i == 0) { // checking if it is first row
                if( j == 0) { // checking if its first column of row first
                    matrixCopy[i+1][j]++ // incrementing its bottom element
                    matrixCopy[i][j+1]++ // increment its next element
                }else if (j == colsLength) { // checking if its last column of first row
                    matrixCopy[i+1][j]++ // incrementing its bottom element
                    matrixCopy[i][j-1]++ // increment its previous element
                }else { // if the column lies in between the first row
                    matrixCopy[i+1][j]++ // incrementing its bottom element
                    matrixCopy[i][j+1]++ // increment its next element
                    matrixCopy[i][j-1]++ // increment its previous element
                }
            }else if (i == rowsLength) { // checking if it is last row
                if( j == 0) { // checking if its first column of last first
                    matrixCopy[i-1][j]++ // incrementing its upper element
                    matrixCopy[i][j+1]++ // increment its next element
                }else if (j == colsLength) { // checking if its last column of last row
                    matrixCopy[i-1][j]++ // incrementing its upper element
                    matrixCopy[i][j-1]++ // increment its previous element
                }else { // if the column lies in between the first row
                    matrixCopy[i-1][j]++ // incrementing its upper element
                    matrixCopy[i][j+1]++ // increment its next element
                    matrixCopy[i][j-1]++ // increment its previous element
```



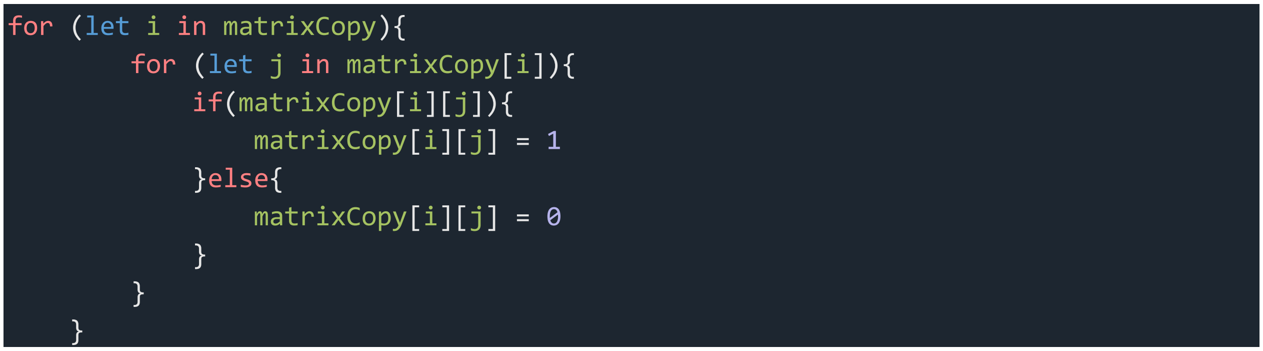
```
                }
            }else { // if the rows are in between
                if (j == 0) { // if it exists first column of any row
                    matrixCopy[i+1][j]++ // incrementing its bottom element
                    matrixCopy[i-1][j]++ // incrementing its upper element
                    matrixCopy[i][j+1]++ // incrementing its next element
                }else if (j == colsLength) {
                    matrixCopy[i+1][j]++ // incrementing its bottom element
                    matrixCopy[i-1][j]++ // incrementing its upper element
                    matrixCopy[i][j-1]++ // incrementing its previous element
                }else {
                    matrixCopy[i+1][j]++ // incrementing its bottom element
                    matrixCopy[i-1][j]++ // incrementing its upper element
                    matrixCopy[i][j+1]++ // incrementing its next element
                    matrixCopy[i][j-1]++ // incrementing its previous element
                }
            }
        }
    }
}

    console.log(matrixCopy)
    return matrixCopy
}
```

What I understand from the Problem: -

There is an input array in the form of “true”, “false”

Assign default value of true => 1 & false => 0



```
for (let i in matrixCopy){
        for (let j in matrixCopy[i]){
            if(matrixCopy[i][j]){
                matrixCopy[i][j] = 1
            }else{
                matrixCopy[i][j] = 0
            }
        }
    }
```

Now we’ve this type of data

True	False	False	➔	1	0	0
False	True	false		0	1	0
False	False	False		0	0	0

Now making increment of 1 to all of those elements that are neighbors of “true”.

For true [1] [1]: -

- ➔ There is no upper row
- ➔ There is no column behind

Output: -

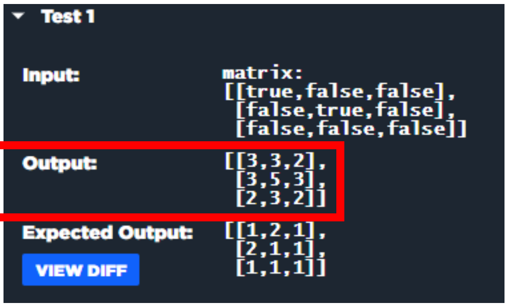
1	1	0
1	1	0
0	0	0

Now for the second time:

1	2	0
2	1	1
0	1	0

Comparison: -

1	2	0
2	1	1
0	1	0



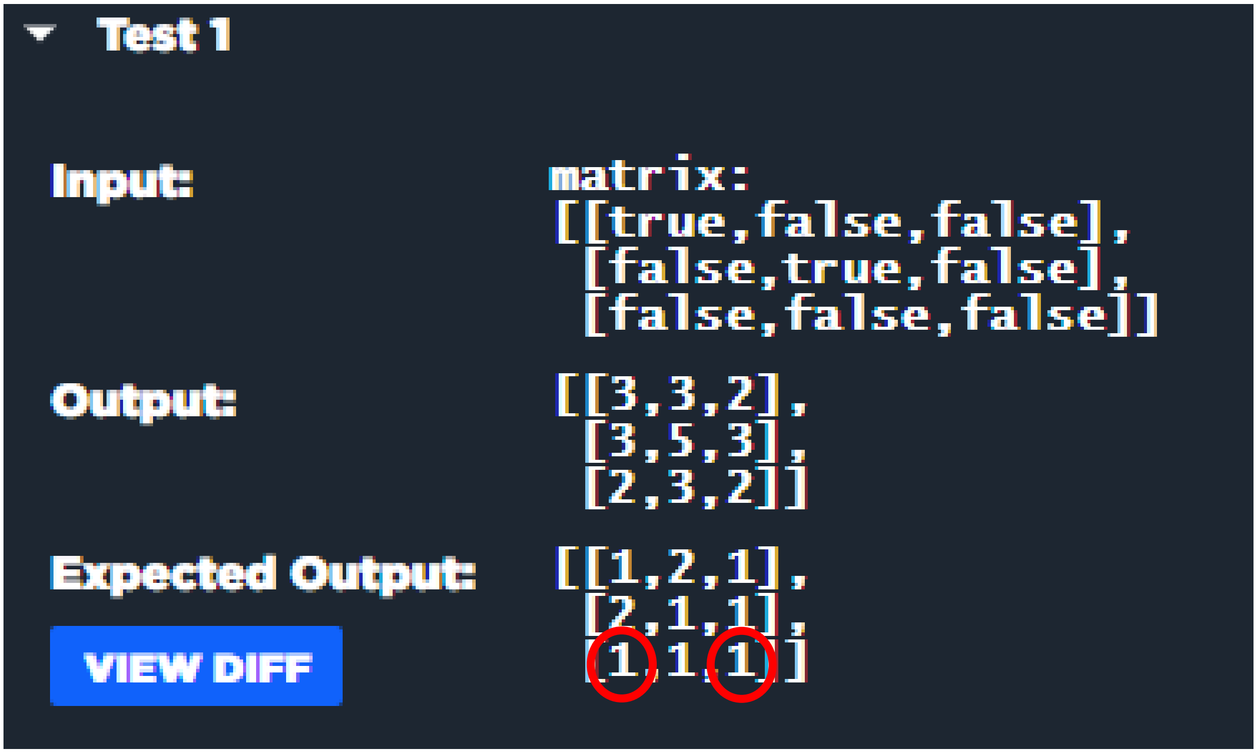
▾ Test 1

```
Input:            matrix:
                  [[true,false,false],
                   [false,true,false],
                   [false,false,false]]

Output:           [[3,3,2],
                   [3,5,3],
                   [2,3,2]]

Expected Output:  [[1,2,1],
                   [2,1,1],
                   [1,1,1]]
```

Another Issue: -



Input:

```
matrix:
[[true,false,false],
 [false,true,false],
 [false,false,false]]
```

Output:

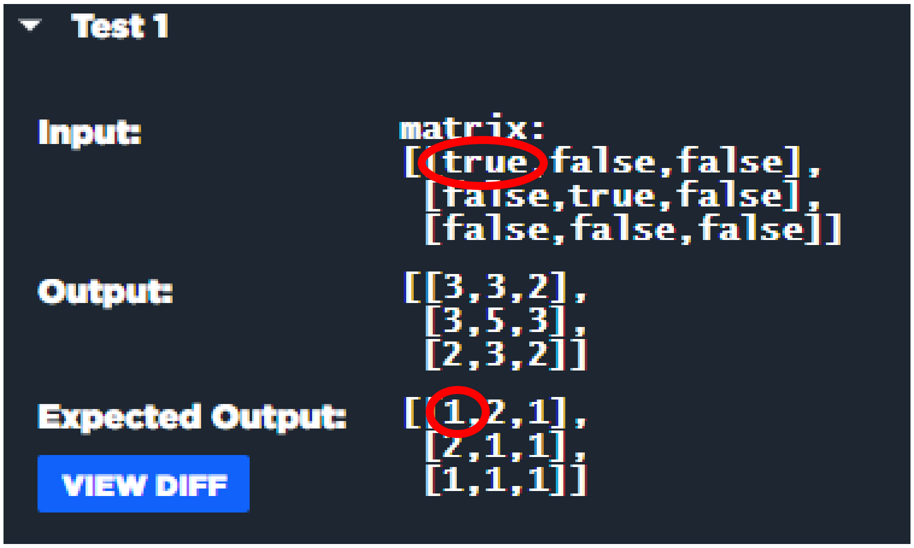
```
[[3,3,2],
 [3,5,3],
 [2,3,2]]
```

Expected Output:

```
[[1,2,1],
 [2,1,1],
 [1,1,1]]
```

Why there are “1” in these positions?

And another misconception in understanding the problem: -

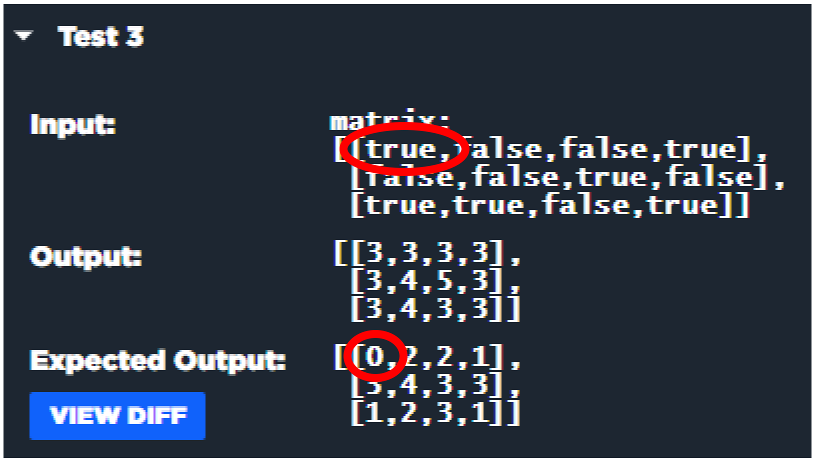


Test 1

```
Input:              matrix:
                    [[true,false,false],
                     [false,true,false],
                     [false,false,false]]

Output:             [[3,3,2],
                     [3,5,3],
                     [2,3,2]]

Expected Output:    [[1,2,1],
                     [2,1,1],
                     [1,1,1]]
```



Test 3

```
Input:              matrix:
                    [[true,false,false,true],
                     [false,false,true,false],
                     [true,true,false,true]]

Output:             [[3,3,3,3],
                     [3,4,5,3],
                     [3,4,3,3]]

Expected Output:    [[0,2,2,1],
                     [3,4,3,3],
                     [1,2,3,1]]
```