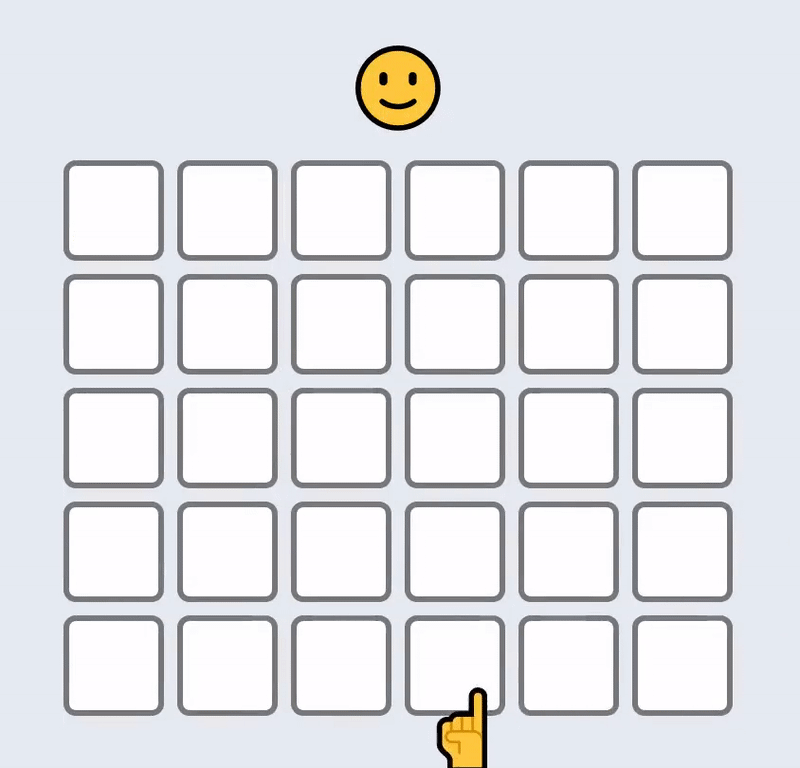
Minesweeper is a popular single-player computer game. The goal is to locate mines within a rectangular grid of cells. At the start of the game, all of the cells are concealed. On each turn, the player clicks on a blank cell to reveal its contents, leading to the following result:

* If there's a mine on this cell, the player loses and the game is over;
* Otherwise, a number appears on the cell, representing how many mines there are within the 8 neighbouring cells (up, down, left, right, and the 4 diagonal directions);
* If the revealed number is **0**, each of the 8 neighbouring cells are automatically revealed in the same way.



You are given a boolean matrix **field** representing the distribution of bombs in the rectangular field. You are also given integers **x** and **y**, representing the coordinates of the player's first clicked cell - **x** represents the row index, and **y** represents the column index, both of which are **0**-based.

Your task is to return an integer matrix of the same dimensions as **field**, representing the resulting field after applying this click. If a cell remains concealed, the corresponding element should have a value of **-1**.

**It is guaranteed that the clicked cell does not contain a mine.**

Example

* For

**field = [[false, true, true],**

**[true, false, true],**

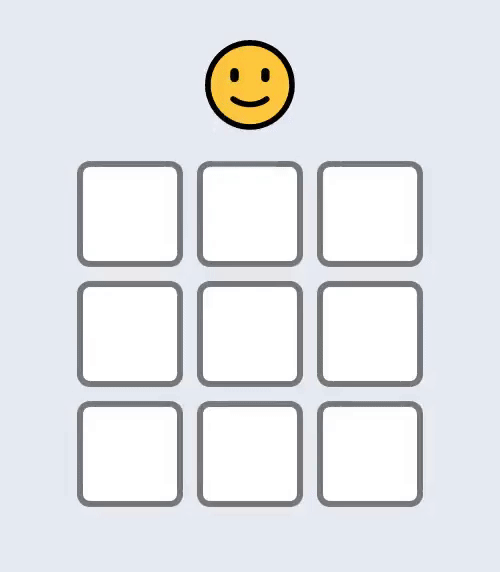
**[false, false, true]]**

**x = 1**, and **y = 1**, the output should be

**solution(field, x, y) = [[-1, -1, -1],**

**[-1, 5, -1],**

**[-1, -1, -1]]**



There are **5** neighbors of the cell **(1, 1)** which contain a mine, so the value in **(1, 1)** should become **5**, and the other elements of the resulting matrix should be **-1** since no other cell would be expanded.

* For

**field = [[true, false, true, true, false],**

**[true, false, false, false, false],**

**[false, false, false, false, false],**

**[true, false, false, false, false]]**

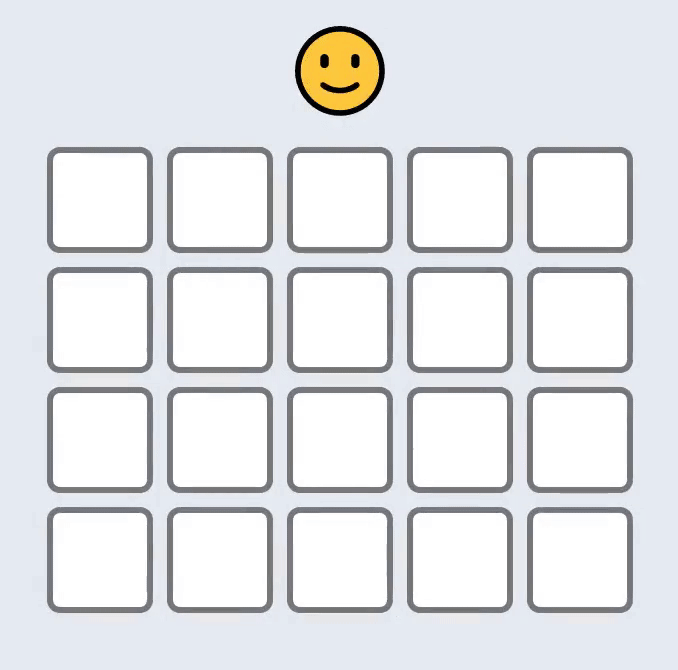
**x = 3**, and **y = 2**, the output should be

**solution(field, x, y) = [[-1, -1, -1, -1, -1],**

**[-1, 3, 2, 2, 1],**

**[-1, 2, 0, 0, 0],**

**[-1, 1, 0, 0, 0]]**



Since the value in the cell **(3, 2)** is **0**, all of its neighboring cells (**(2, 1)**, **(2, 2)**, **(2, 3)**, **(3, 1)**, and **(3, 3)**) are also revealed. Since the value in the cell **(2, 2)** is also **0**, its neighbouring cells **(1, 1)**, **(1, 2)** and **(1, 3)** are revealed, and since the value in cell **(2, 3)** is **0**, its neighbours **(1, 4)**, **(2, 4)**, and **(3, 4)** are also revealed. The cells **(3, 3)**, **(2, 4)**, and **(3, 4)** also contain the value **0**, but since all of their neighbours have already been revealed, no further action is required.

Input/Output

* **[execution time limit] 4 seconds (py3)**
* **[input] array.array.boolean field**

A rectangular matrix representing the locations of the mines in the game field.

*Guaranteed constraints:*  
**2 ≤ field.length ≤ 100**,  
**2 ≤ field[i].length ≤ 100**.

* **[input] integer x**

The row number of the cell which is clicked (**0**-based).

*Guaranteed constraints:*  
**0 ≤ x < field.length**.

* **[input] integer y**

The column number of the cell which is clicked (**0**-based).

*Guaranteed constraints:*  
**0 ≤ y < field[0].length**.

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

The expanded matrix after the click.

**[Python 3] Syntax Tips**

**# Prints help message to the console**

**# Returns a string**

**def helloWorld(name):**

**print("This prints to the console when you Run Tests")**

**return "Hello, " + name**

10/23/2022 Solution:

def solution(field, x, y):

    def check\_mines(field, x, y):

        m = len(field)

        n = len(field[0])

        if x == 0:

            x\_vector = [0, 1]

        elif x == m - 1:

            x\_vector = [-1, 0]

        else:

            x\_vector = [-1, 0, 1]

        if y == 0:

            y\_vector = [0, 1]

        elif y == n - 1:

            y\_vector = [-1, 0]

        else:

            y\_vector = [-1, 0, 1]

        mines = 0

        adj = []

        for i in x\_vector:

            for j in y\_vector:

                if i == 0 and j == 0:

                    continue

                adj.append((x + i, y + j))

                if field[x + i][y + j] is True:

                    mines += 1

        return [mines, adj]

    m = len(field)

    n = len(field[0])

    output = [[-1 for \_ in range(n)] for \_ in range(m)]

    searched = set()

    bfs\_arr = [(x, y)]

    searched.add((x, y))

    while len(bfs\_arr) != 0:

        temp = []

        for point in bfs\_arr:

            x = point[0]

            y = point[1]

            arr = check\_mines(field, x, y)

            searched.add(point)

            output[x][y] = arr[0] # updates the number of mines

            if arr[0] == 0: # if number of mines is zero

                for p in arr[1]:

                    if p not in searched:

                        temp.append(p)

                        searched.add(p) # without this line, only line 43 is acting, so there are a lot of duplicate points that are searched in each iteration of bfs\_arr, so the runtime is very slow. Each iteration of bfs\_arr should include unique points, and previously searched points should not be searched again !!!!!!!

        bfs\_arr = temp

    return output