Let's define the diagonal neighbors of a given cell as the surrounding cells, with precisely one corner touching a corner of the given cell. The picture below shows the diagonal neighbors for the **A** cell.

Given a board of cells containing a bubble of a specific color, your task is to emulate a bubble-popping game. In this game, the player can click a cell every turn to pop bubbles. After clicking the cell, the following happens:

* The bubble in the clicked cell and bubbles *of the same color* among its diagonal neighbors are "popped" and removed, resulting in empty cells.
* After bubbles are removed, the remaining bubbles in cells above the empty cells drop down to fill all empty cells.
* Nothing happens if the clicked cell is empty (it does not contain a bubble).

The input consists of two arrays:

* An initial board of cells **bubbles** - a multidimensional array of integers representing cells containing different colored bubbles;
* A set of player turns **operations** - 2-element integer arrays describing the coordinates (the row and column) of the cell that the player clicked on during each turn.

Return the state of the game board after all **operations** are processed. The output should be a multidimensional array of integers with the same size as **bubbles**, but replace the integers in all empty cells (without bubbles) with **0**.

*Note: You are not expected to provide the most optimal solution, but a solution with time complexity not worse than****O(bubbles.length2 · bubbles[0].length2 · operations.length)****will fit within the execution time limit.*

Example

For

**bubbles = [[1, 1, 1, 4, 3],**

**[4, 1, 2, 3, 3],**

**[1, 5, 1, 1, 2],**

**[4, 3, 2, 2, 4]]**

and

**operations = [[1, 1],**

**[3, 3],**

**[2, 2],**

**[3, 0]]**

the output should be

**solution(bubbles, operations) = [[0, 0, 0, 0, 0],**

**[0, 1, 0, 4, 3],**

**[0, 5, 0, 3, 3],**

**[4, 3, 2, 1, 4]]**

Expand to see the example video.

*Note*: If you cannot view the video properly, please use [this link](https://codesignal-assets.s3.amazonaws.com/uploads/1673636966868/example.mp4) to access it.

Input/Output

* **[execution time limit] 4 seconds (js)**
* **[memory limit] 1 GB**
* **[input] array.array.integer bubbles**

A multidimensional array of integers representing the initial game board of cells containing bubbles. Integers represent the colors of bubbles within cells, and it is guaranteed that all cells will have bubbles initially.

*Guaranteed constraints:*  
**1 ≤ bubbles.length ≤ 30**,  
**1 ≤ bubbles[0].length ≤ 30**,  
**1 ≤ bubbles[i][j] ≤ 900**.

* **[input] array.array.integer operations**

An array of 2-element arrays representing the player's moves - integers represent the coordinates of the row and column that the player has clicked on each turn.

*Guaranteed constraints:*  
**1 ≤ operations.length ≤ 200**,  
**operations[i].length = 2**,  
**0 ≤ operations[i][0] ≤ bubbles.length**,  
**0 ≤ operations[i][1] ≤ bubbles[0].length**.

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

Return the final state of the game board after all operations are processed. The output should be a multidimensional array of integers with the same size as **bubbles**, with **0** representing empty cells without bubbles.

**[JavaScript] Syntax Tips**

**// Prints help message to the console**

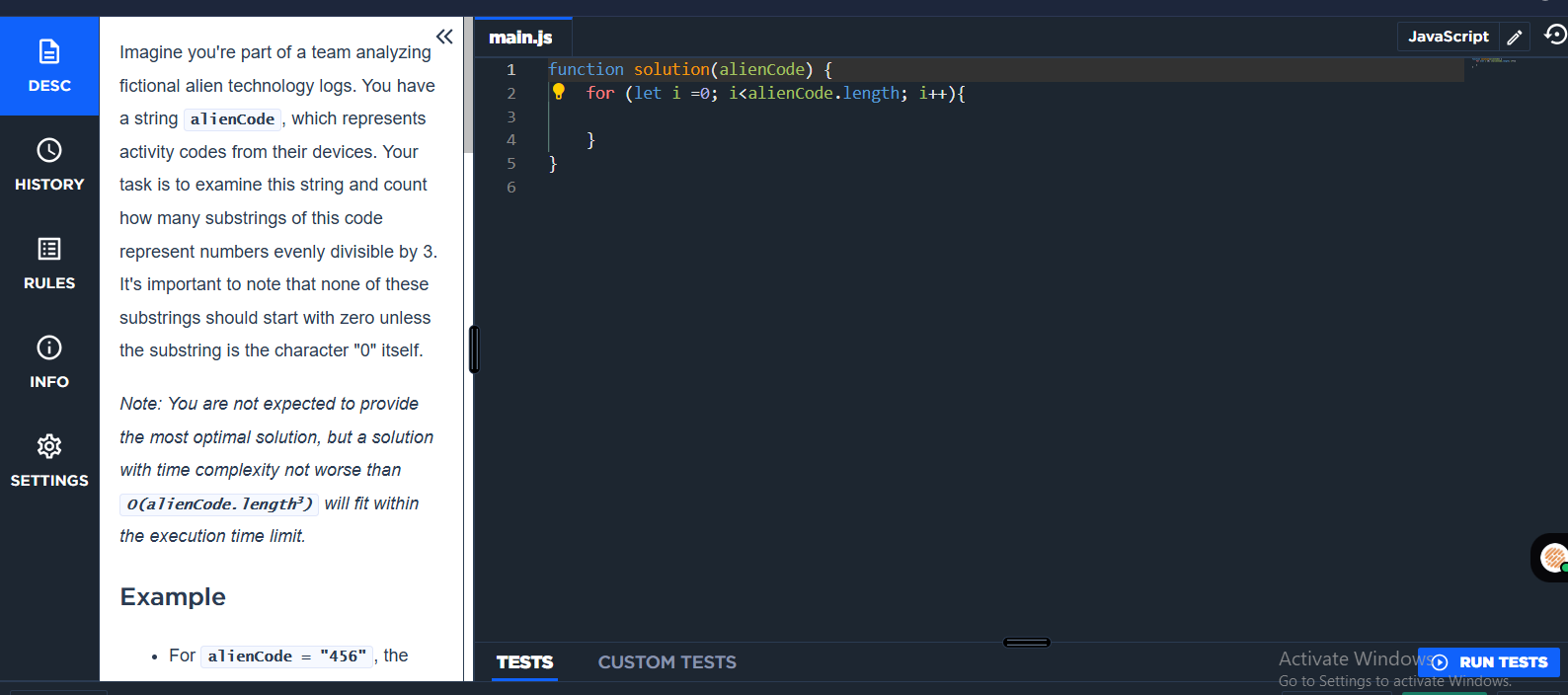
**// Returns a string**

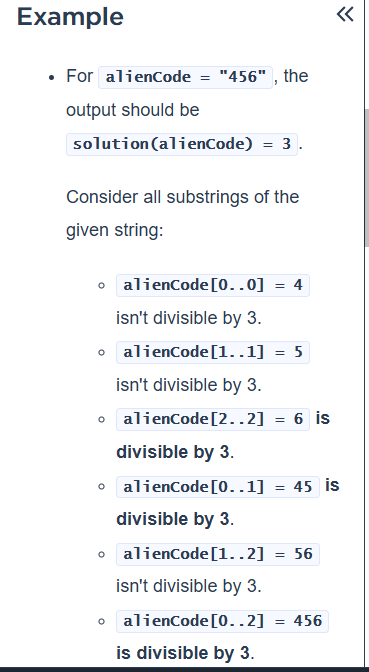
**function helloWorld(name) {**

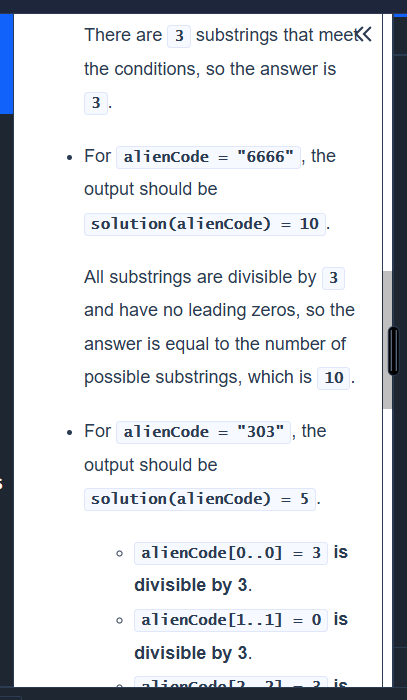
**console.log("This prints to the console when you Run Tests");**

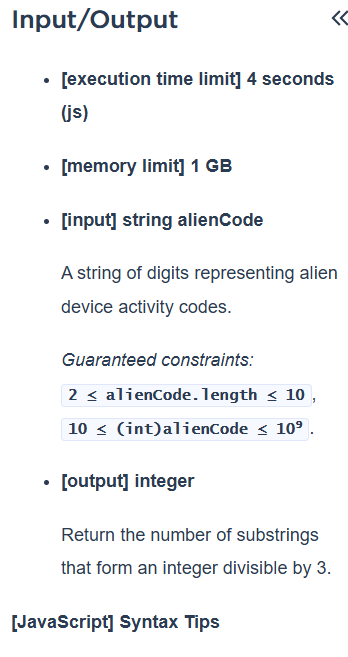
**return "Hello, " + name;**

**}**









Imagine you're part of a team analyzing fictional alien technology logs. You have a string **alienCode**, which represents activity codes from their devices. Your task is to examine this string and count how many substrings of this code represent numbers evenly divisible by 3. It's important to note that none of these substrings should start with zero unless the substring is the character "0" itself.

*Note: You are not expected to provide the most optimal solution, but a solution with time complexity not worse than****O(alienCode.length3)****will fit within the execution time limit.*

Example

* For **alienCode = "456"**, the output should be **solution(alienCode) = 3**.

Consider all substrings of the given string:

* + **alienCode[0..0] = 4** isn't divisible by 3.
  + **alienCode[1..1] = 5** isn't divisible by 3.
  + **alienCode[2..2] = 6** **is divisible by 3**.
  + **alienCode[0..1] = 45** **is divisible by 3**.
  + **alienCode[1..2] = 56** isn't divisible by 3.
  + **alienCode[0..2] = 456** **is divisible by 3**.

There are **3** substrings that meet the conditions, so the answer is **3**.

* For **alienCode = "6666"**, the output should be **solution(alienCode) = 10**.

All substrings are divisible by **3** and have no leading zeros, so the answer is equal to the number of possible substrings, which is **10**.

* For **alienCode = "303"**, the output should be **solution(alienCode) = 5**.
  + **alienCode[0..0] = 3** **is divisible by 3**.
  + **alienCode[1..1] = 0** **is divisible by 3**.
  + **alienCode[2..2] = 3** **is divisible by 3**.
  + **alienCode[0..1] = 30** **is divisible by 3**.
  + **alienCode[1..2] = 03** is divisible by 3, but it has leading zeroes, so we don't count it.
  + **alienCode[0..2] = 303** **is divisible by 3**.

There are **5** substrings that meet the conditions, so the answer is **5**.

Input/Output

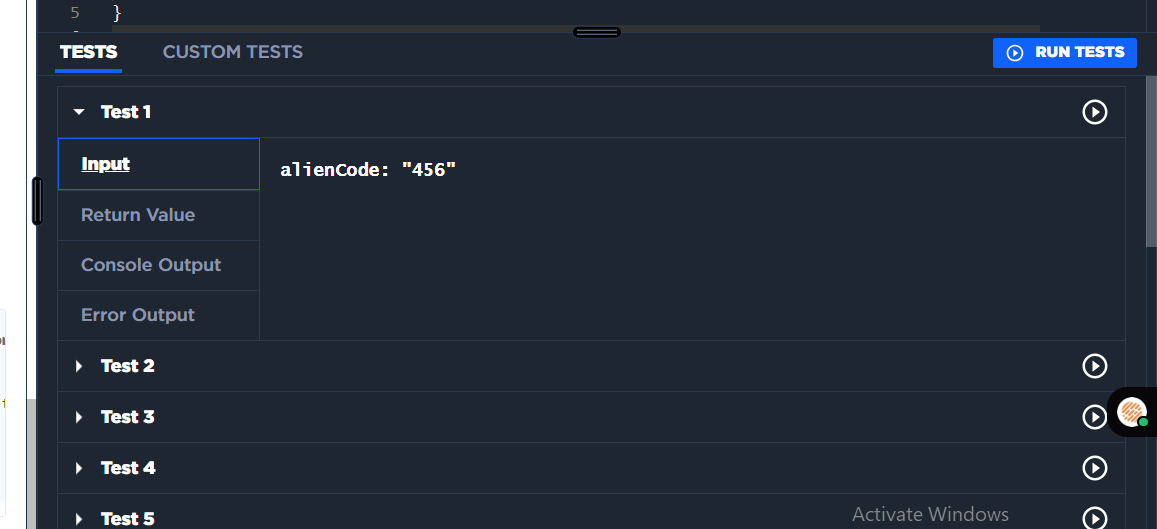
* **[execution time limit] 4 seconds (js)**
* **[memory limit] 1 GB**
* **[input] string alienCode**

A string of digits representing alien device activity codes.

*Guaranteed constraints:*  
**2 ≤ alienCode.length ≤ 10**,  
**10 ≤ (int)alienCode ≤ 109**.

* **[output] integer**

Return the number of substrings that form an integer divisible by 3.



# QUESTION 1:

You are given an array **a** containing a list of strings. Your task is to construct an array of the same length, where the **ith** element is a 2-character string formed by the first character of **a[i]** concatenated with the last character of **a[i + 1]**.

If there is no **a[i + 1]**, cycle back to the beginning of the array. In other words, for the final element, concatenate the first character of **a[a.length - 1]** with the last character of **a[0]**.

Return the resulting array of 2-character strings.

*Note: You are not expected to provide the most optimal solution, but a solution with time complexity not worse than****O(a.length × sum(a[i].length))****will fit within the execution time limit.*

Example

* For **a = ["cat", "dog", "ferret", "scorpion"]**, the output should be **solution(a) = ["cg", "dt", "fn", "st"]**.

Expand to see the example video.

*Note*: If you are not able to see the video, use [this link](https://codesignal.s3.amazonaws.com/uploads/180121600729/32630converted.mp4) to access it.

* For **a = ["I", "have", "a", "nice", "surprise"]**, the output should be **solution(a) = ["Ie", "ha", "ae", "ne", "sI"]**.

Expand to see the example video.

*Note*: If you are not able to see the video, use [this link](https://codesignal.s3.amazonaws.com/uploads/19346259334933/17011converted.mp4) to access it.

* For **a = ["singularity"]**, the output should be **solution(a) = ["sy"]**.

Input/Output

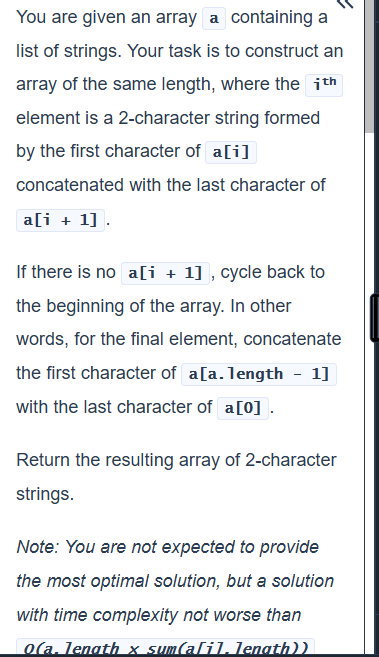
* **[execution time limit] 4 seconds (js)**
* **[memory limit] 1 GB**
* **[input] array.string a**

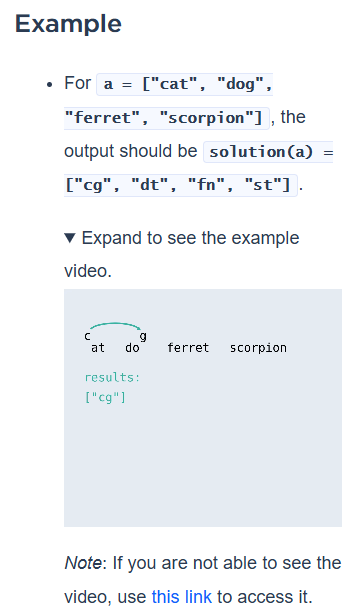
An array of strings consisting of alphanumeric characters.

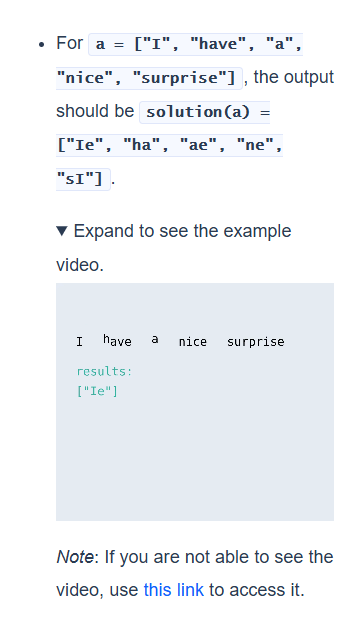
*Guaranteed constraints:*  
**1 ≤ a.length ≤ 100**,  
**1 ≤ a[i].length ≤ 100**.

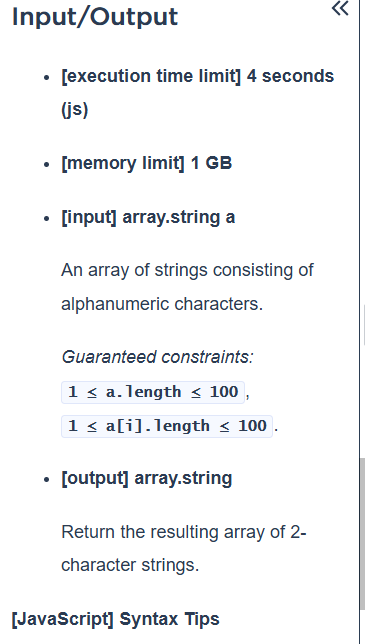
* **[output] array.string**

Return the resulting array of 2-character strings.

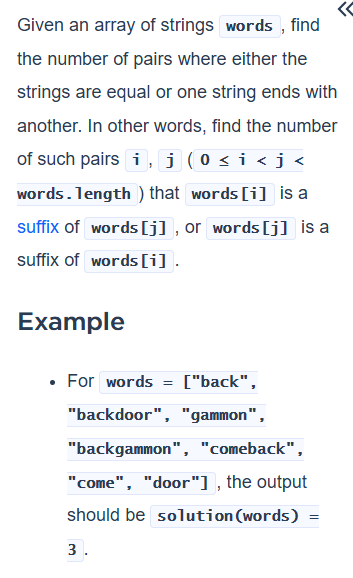


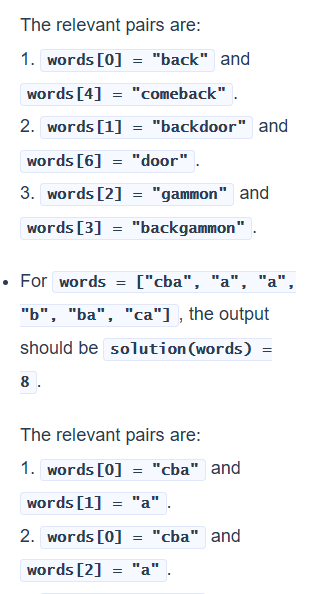


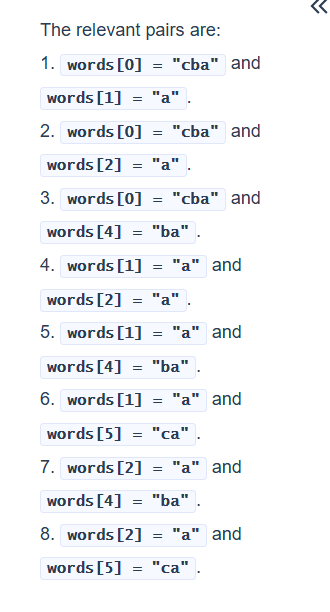


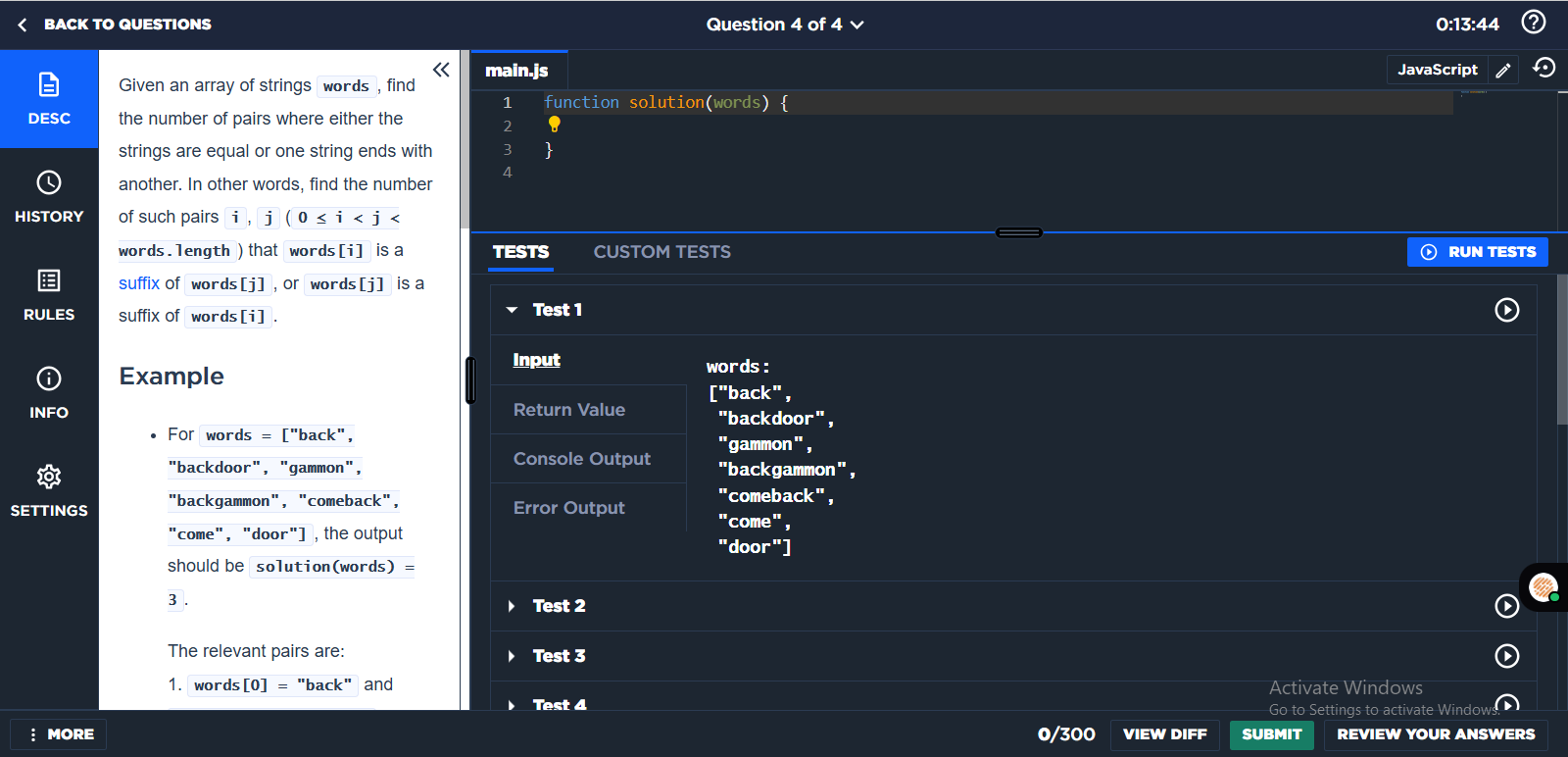


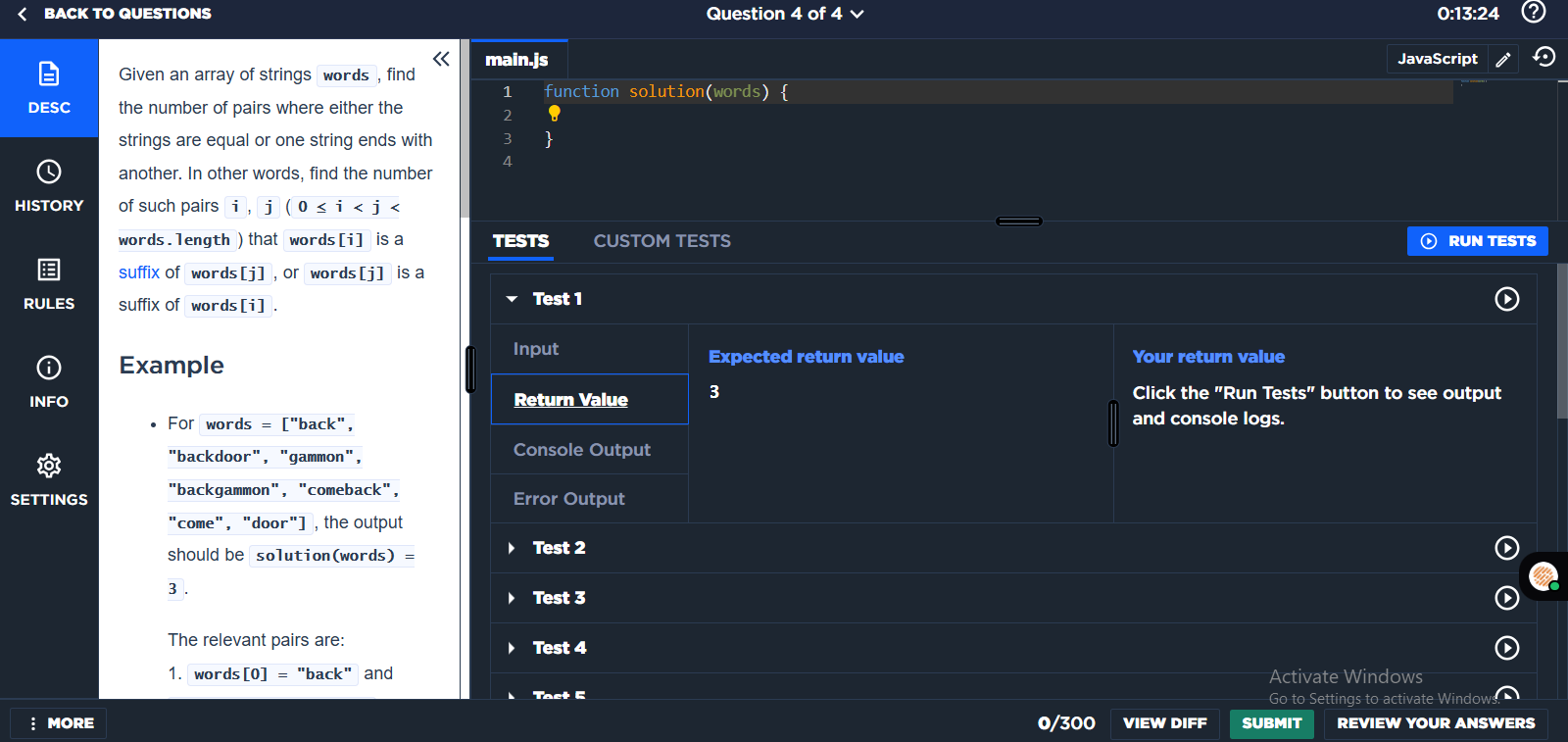
# Question 4



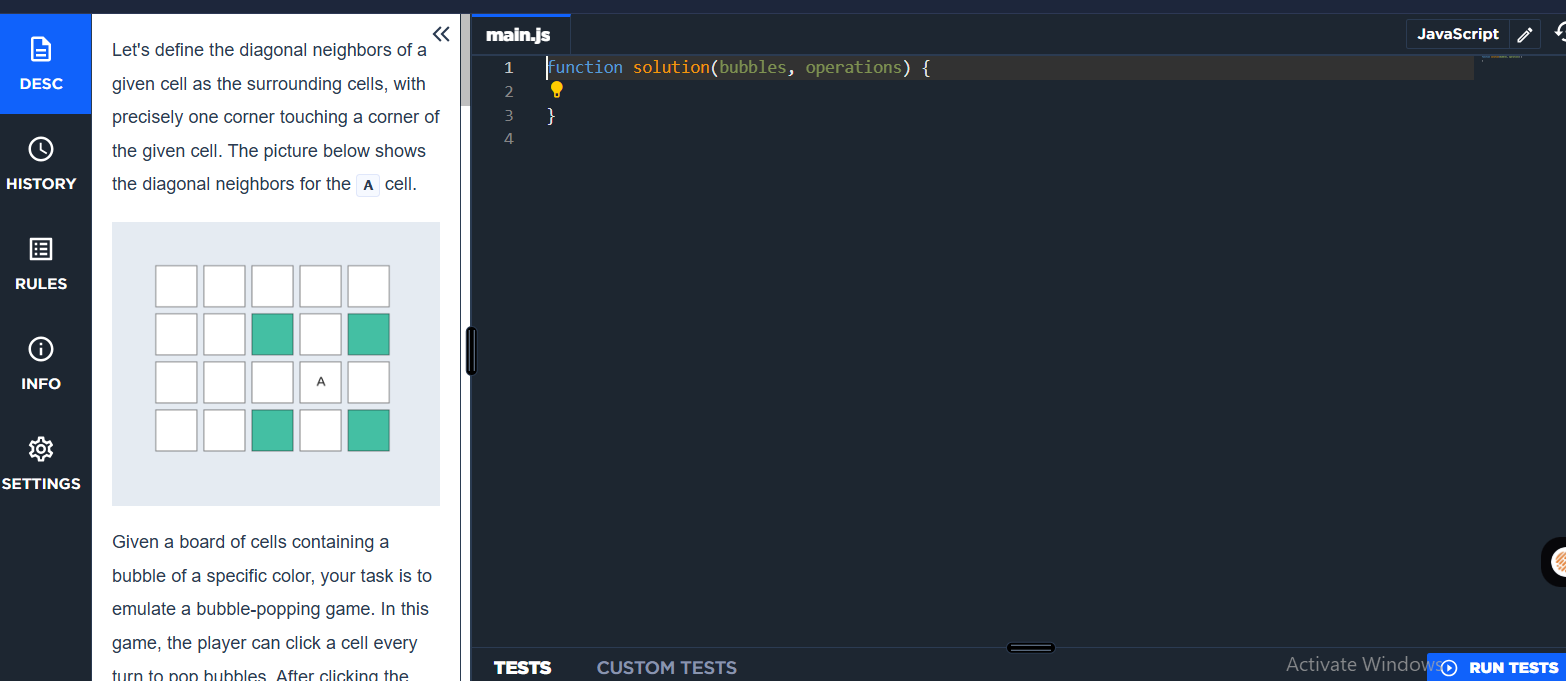


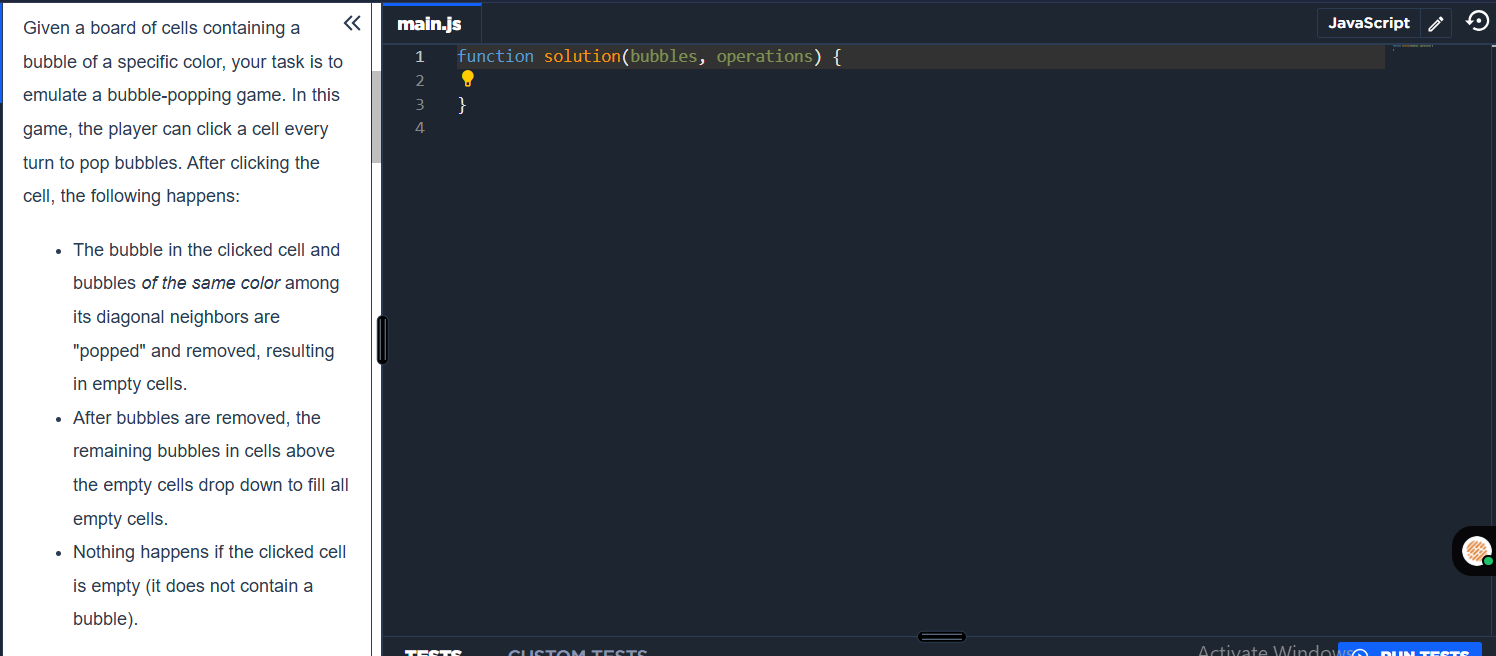


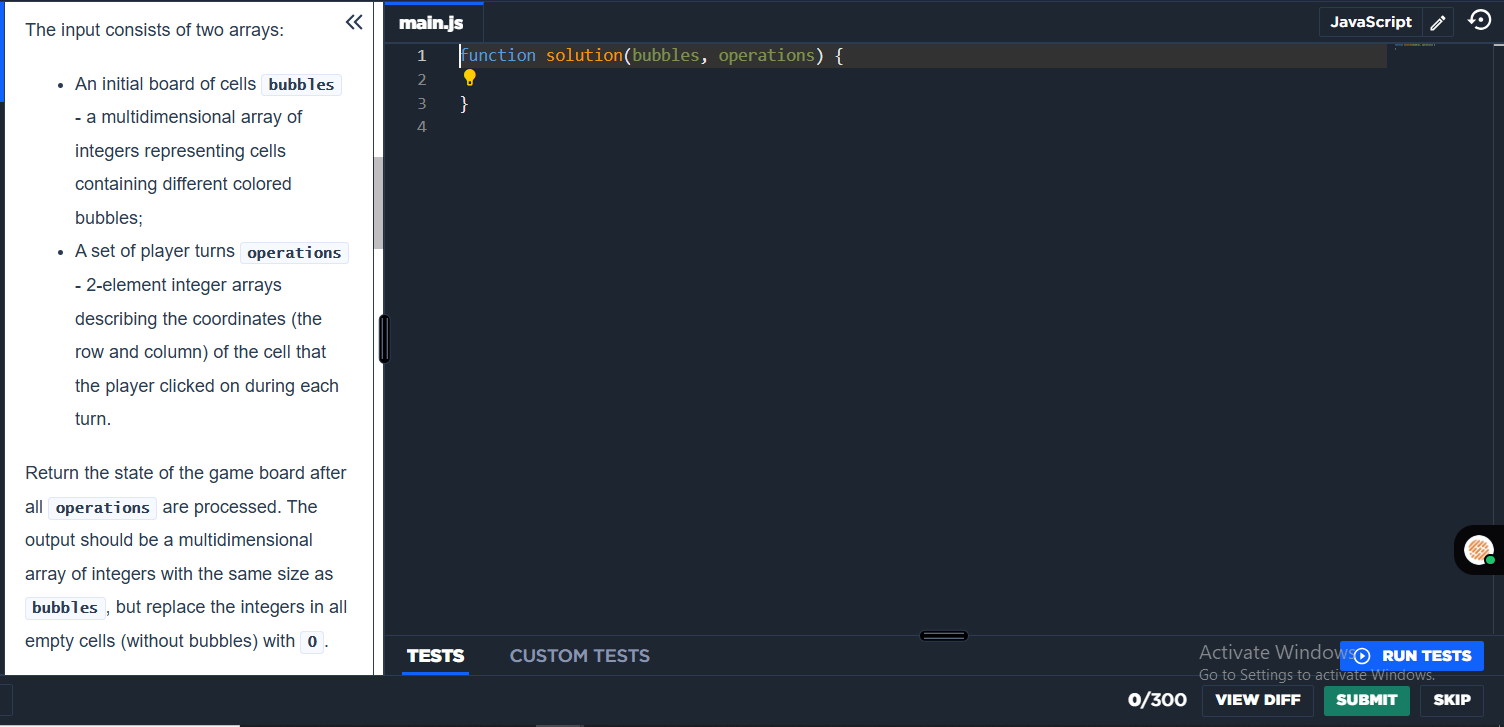


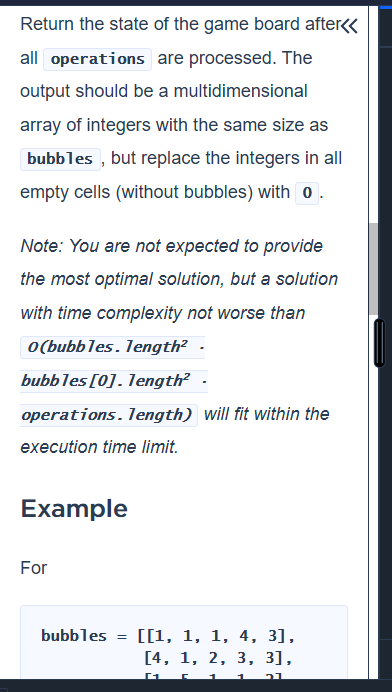


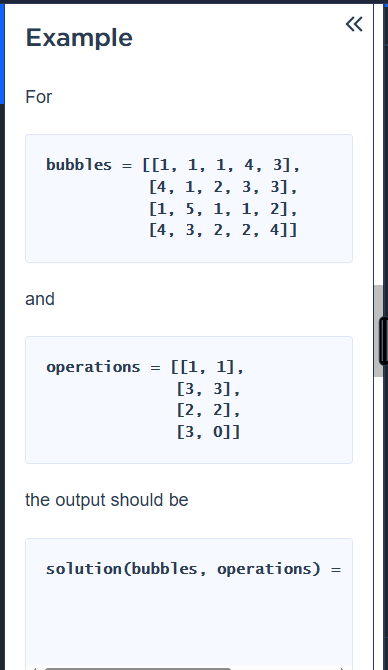
# Question 3:

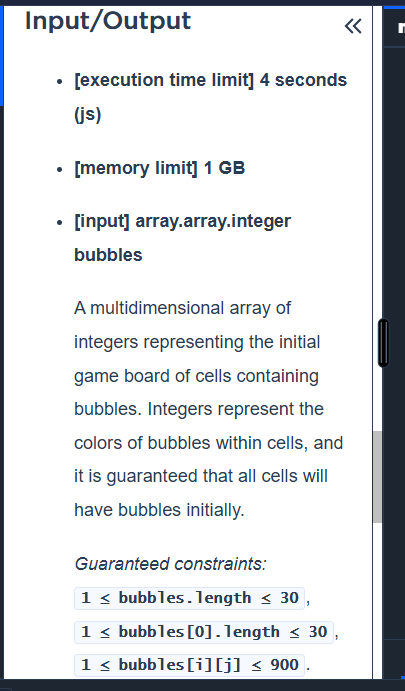


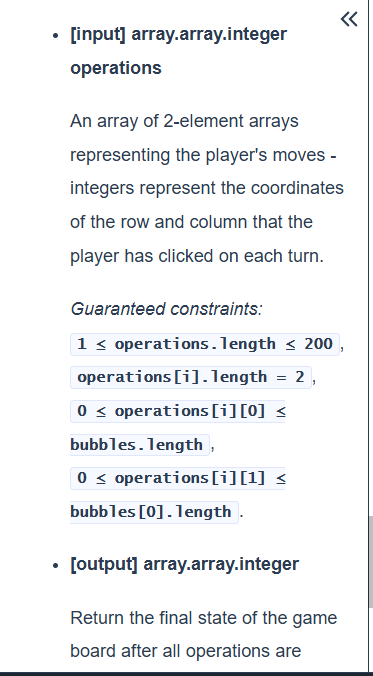


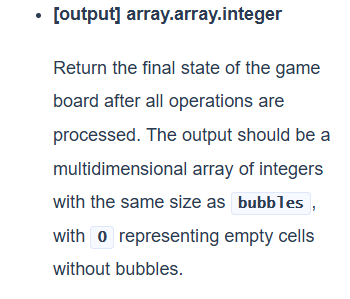




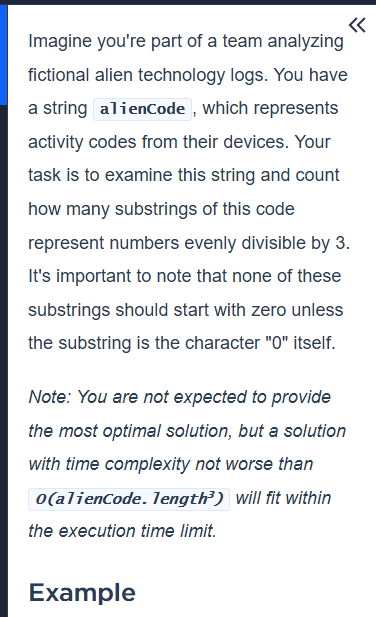


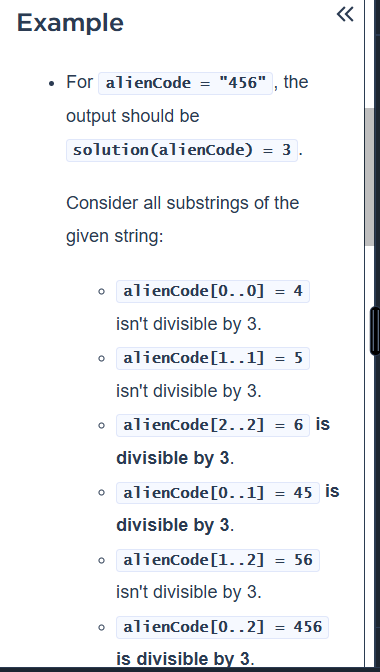


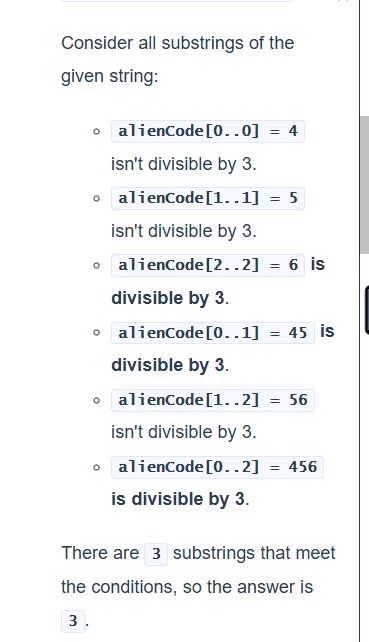


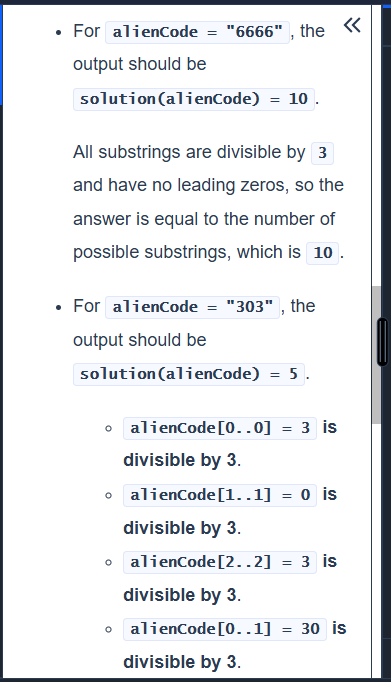


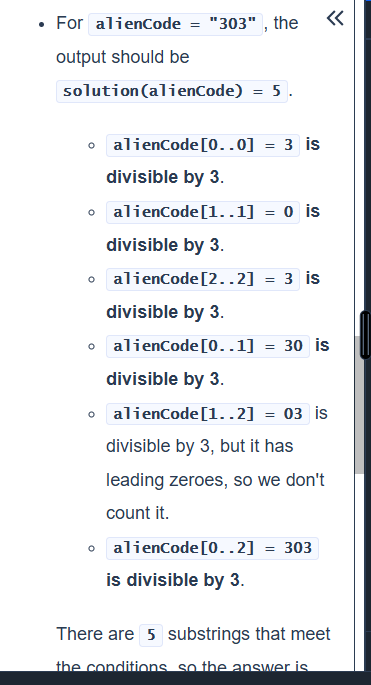
# Question 2:

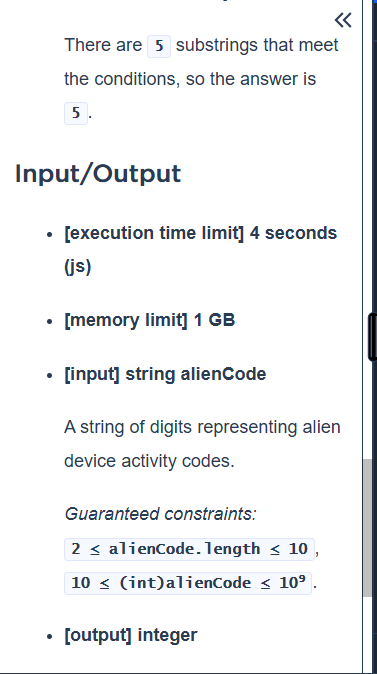


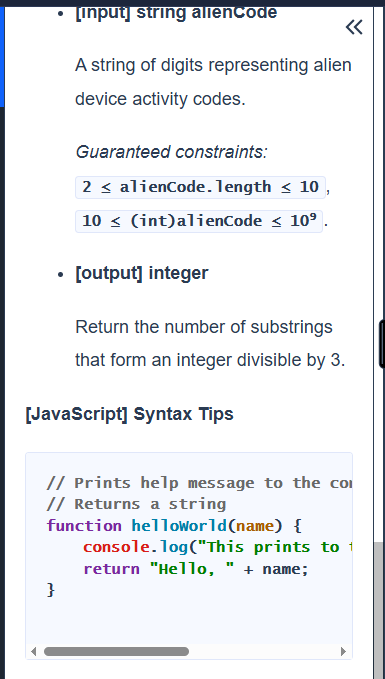












# Answer tried:

