Exercise: Arrays

Problems for exercises and homework for the "JavaScript Advanced" course @ SoftUni. Submit your solutions in the SoftUni judge system https://judge.softuni.bg/Contests/2753/Arrays-and-Nested-Arrays-Exercise.

1. Print an Array with a Given Delimiter

The **input** comes as two parameters – an **array of strings** and a **string**. The second parameter is the delimiter.

The **output** is the elements of the array, printed on the console, each element **separated** from the others by the **given delimiter**.

Examples

Input	Output
['One', 'Two', 'Three', 'Four', 'Five'], '-'	One-Two-Three-Four-Five

Input	Output
['How about no?', 'I', 'will', 'not', 'do', 'it!'], '_'	How about no?_I_will_not_do_it!

2. Print Every N-th Element from an Array

The **input** comes as two parameters – an **array of strings** and a **number**. The second parameter is **N** – **the step**.

The **output** is every element on the **N-th** step **starting from the first one**. If the step is **3**, you need to return the **1-st**, the **4-th**, the **7-th** … and so on, until you reach the end of the array.

The **output** is the **return** value of your function and must be an **array**.

Example

Input	Output
['5', '20', '31', '4', '20'], 2	['5', '31', '20']

Input	Output
['dsa', 'asd', 'test', 'tset'], 2	['dsa', 'test']

Input	Output
['1', '2', '3', '4', '5'], 6	['1']

Hints

- **Return all the elements** with `for` loop, **incrementing** the **loop variable** with the value of the `step` variable.

3. Add and Remove Elements

Write a JS function that **adds** and **removes** numbers **to/from** an array. You will receive a command which can either be "**add**" or "**remove**".

The **initial number** is **1**. Each input command should **increase that number**, regardless of what it is.
Upon receiving an "**add**" command you should add the current number to your array.
Upon receiving the "**remove**" command you should remove the last entered number, currently existent in the array.

The **input** comes as an **array of strings**. Each element holds a **command**.

The **output** is the element of the array, each printed on a new line. In case of an empty array, just print "**Empty**".

Examples

Input	Output
['add', 'add', 'add', 'add']	1 2 3 4

Input	Output
['add', 'add', 'remove', 'add', 'add']	1 4 5

Input	Output
['remove', 'remove', 'remove']	Empty

4. Rotate Array

Write a JS function that rotates an array. The array should be rotated **to the right side**, meaning that the last element should become the first, upon rotation.

The **input** comes as two parameters – an **array of strings** and a **number**. The **second parameter** is the amount of rotation you need to perform.

The **output** is the resulted array after the rotations. The elements should be printed on one line, separated by a **single space**.

Examples

Input	Output
['1', '2', '3', '4'], 2	3 4 1 2

Input	Output
['Banana', 'Orange', 'Coconut', 'Apple'], 15	Orange Coconut Apple Banana

Hints

- Check if there is a **built-in function** for inserting elements **at the start** of the array.

5. Extract Increasing Subsequence from Array

Write a function that extracts only those numbers that **form a non-decreasing subsequence**. In other words, you start from the **first element** and continue to **the end** of the **given array of numbers**. Any number which is **LESS THAN** the **current biggest one** is **ignored**, alternatively if it's equal or higher than the **current biggest one** you set it as the **current biggest one** and you continue to the next number.

The **input** comes as an **array of numbers**.

The **output** is the processed array after the filtration, which should be a non-decreasing subsequence. Return the **array of numbers**.

Examples

Input	Output
[1, 3, 8, 4,	[1, 3, 8, 10, 12, 24]

Input	Output
[1, 2, 3, 4]	[1, 2, 3, 4]

Input	Output
[20, 3, 2, 15,	[20]

10, 12, 3, 2, 24]	

6, 1]	

Hints

- The **`Array.reduce()`** built-in function might help you a lot with this problem.

6. List of Names

You will receive an **array of names**. Sort them **alphabetically in ascending order** and print a numbered list of all the names, each on a new line.

Example

Input	Output
["John", "Bob", "Christina", "Ema"]	1.Bob 2.Christina 3.Ema 4.John

Hints

- The **`sort` function** rearranges the array in ascending order

7. Sorting Numbers

Write a function that sorts an **array of numbers** so that the first element is the **smallest** one, the second is the **biggest** one, the third is the **second smallest** one, the fourth is the **second biggest** one and so on.

Return the resulting array.

Example

Input	Output
[1, 65, 3, 52, 48, 63, 31, -3, 18, 56]	[-3, 65, 1, 63, 3, 56, 18, 52, 31, 48]

8. Sort an Array by 2 Criteria

Write a function that orders a **given array of strings**, by a **length** in **ascending order** as **primary criteria**, and by **alphabetical value** in **ascending order** as **second criteria**. The comparison should be **case-insensitive**.

The **input** comes as an **array of strings**.

The **output** is the elements of the ordered array of strings, printed each on a new line.

Examples

Input	Output
['alpha', 'beta',	beta alpha

Input	Output
['Isacc', 'Theodor',	Jack Isacc

Input	Output
['test', 'Deny',	Deny omen

'gamma']	gamma

'Jack', 'Harrison', 'George']	George Theodor Harrison

'omen', 'Default']	test Default

Hints

- An array can be sorted by passing a comparing function to the **`Array.sort()`** function.
- Creating a comparing function by 2 criteria can be achieved by first comparing by the **main criteria**, if the 2 items are different (the result of the compare is not 0) - return the result as the result of the comparing function. If the two items are the same by the **main criteria** (the result of the compare is 0), we need to compare by the **second criteria** and the result of that comparison is the result of the comparing function.
- You can check more about **`Array.sort()`** here - https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/sort

Multidimensional Arrays (Matrices)

We will mainly work with 2-dimensional arrays. The concept is as simple as working with a simple 1-dimensional array. It is just an array of arrays.

9. Magic Matrices

Write a function that checks if a given matrix of numbers is **magical**. A matrix is magical if the **sums of the cells** of **every row** and **every column** are **equal**.

The **input** comes as an **array of arrays**, containing numbers (number 2D matrix). The input numbers will **always be positive**.

The **output** is a Boolean result indicating whether the matrix is magical or not.

Examples

Input	Output
[[4, 5, 6], [6, 5, 4], [5, 5, 5]]	true

Input	Output
[[11, 32, 45], [21, 0, 1], [21, 1, 1]]	false

Input	Output
[[1, 0, 0], [0, 0, 1], [0, 1, 0]]	true

10. *Tic-Tac-Toe

Make a tic-tac-toe console application.

You will receive an array of arrays. As you know there are two players in this game, so the first element of the input will be first player's chosen coordinates, the second element will be the second player's turn coordinates and so on.

The initial state of the dashboard is

```
[[false, false, false],
[false, false, false],
[false, false, false]]
```

The first player's mark is X and the second player's mark is O.

Input

One parameter:

- An array - **the moves** in row that players make

Output

- There are two players - X and O
- If a player tries to make his turn on already taken place, he should take his turn again and you should print the following message:

  "This place is already taken. Please choose another!"

- If there are no free spaces on the dashboard and nobody wins the game should end and you should print the following message:

  "The game ended! Nobody wins :("

- If someone wins you should print the following message and the **current state** of the **dashboard**:

  "Player {x} wins!"

  Note: When printing the state of the dashboard the elements of each row the dashboard should be separated by **"\t"** and each row should be on new line.

Constraints

The elements in the input array will always be enough to end the game.

Examples

Input	Output
["0 1", "0 0", "0 2", "2 0", "1 0", "1 1", "1 2", "2 2", "2 1", "0 0"]	Player O wins! O X X X O X O false O

["0 0", "0 0", "1 1", "0 1", "1 2", "0 2", "2 2", "1 2", "2 2", "2 1"]	This place is already taken. Please choose another! Player X wins! X X X false O O false false false
["0 1", "0 0", "0 2", "2 0", "1 0", "1 2", "1 1", "2 1", "2 2", "0 0"]	The game ended! Nobody wins :(O X X X X O O O X

11. **Diagonal Attack

Write a JS function that reads a given matrix of numbers and checks if both **main diagonals** have **equal sums**. If they do, set every element that is **NOT** part of **the main diagonals** to that sum, alternatively just print the matrix unchanged.

The **input** comes as an **array of strings**. Each element represents a **string of numbers**, with **spaces** between them. Parse it into a **matrix of numbers**, so you can work with it.

The **output** is either the new matrix, with all cells not belonging to a main diagonal are changed to the diagonal sum or the original matrix, if the two diagonals have different sums. You need to print **every row on a new line**, with cells **separated by a space**. Check the examples below.

Example

Input	Output
['5 3 12 3 1', '11 4 23 2 5', '101 12 3 21 10', '1 4 5 2 2', '5 22 33 11 1']	5 15 15 15 1 15 4 15 2 15 15 15 3 15 15 15 4 15 2 15 5 15 15 15 1

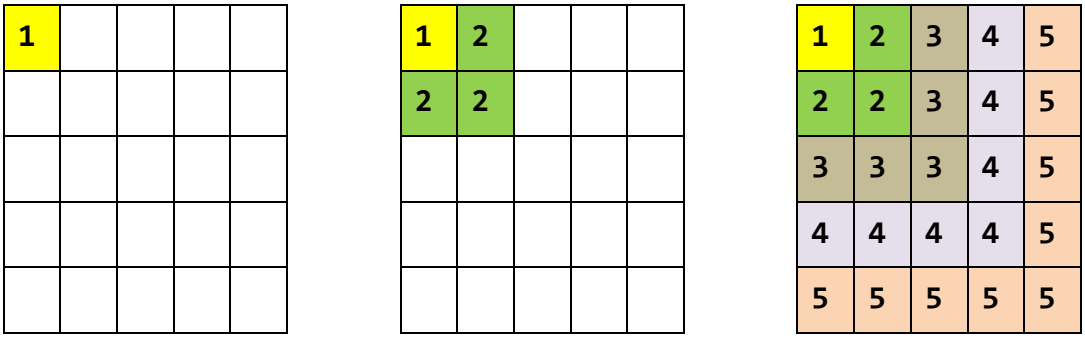
Input	Output
['1 1 1', '1 1 1', '1 1 0']	1 1 1 1 1 1 1 1 0

12. *Orbit

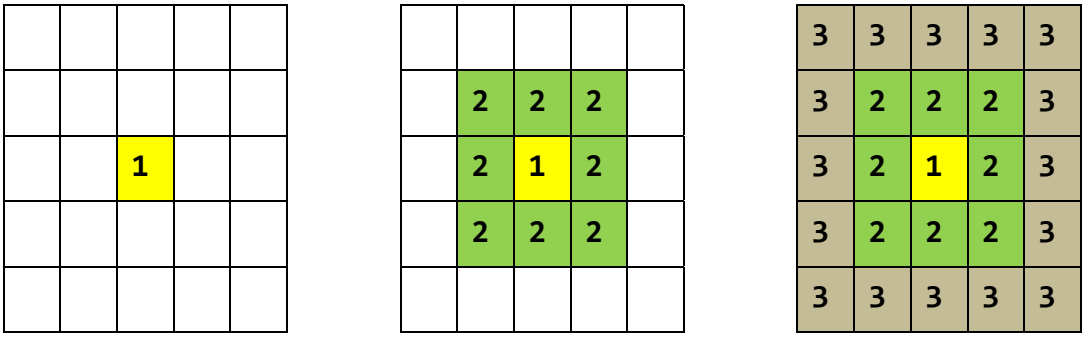
You will be given an empty rectangular space of cells. Then you will be given the position of a star. You need to build the orbits around it.

You will be given a coordinate of a cell, which will **always be inside the matrix**, on which you will put the value - **1**. Then you must set the values of the cells **directly surrounding that cell**, including the **diagonals**, **to 2**. After which you must set the values of the next surrounding cells to 3 and so on. Check the pictures for more information.

For example, we are given a matrix that has 5 rows and 5 columns, and the star is at coordinates - **0, 0**. Then the following should happen:



If the coordinates of the star are somewhere in the middle of the matrix for example - **2, 2**, then it should look like this:



The **input** comes as an **array of 4 numbers [width, height, x, y]** which represents the **dimensions of the matrix** and the **coordinates of the star.**

The **output** is the filled matrix, with the cells **separated by a space**, each **row on a new line**.

Examples

Input	Output

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

Input	Output

`[5, 5, 2, 2]`	`3 3 3 3 3`
	`3 2 2 2 3`
	`3 2 1 2 3`
	`3 2 2 2 3`
	`3 3 3 3 3`

Input	Output

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

Hints

- Check if there is some **dependency** or **relation** between the **position of the numbers** and the **rows** and **columns** of those positions.

13. *Spiral Matrix

Write a JS function that generates a **Spirally-filled** matrix with numbers, with given dimensions.

The **input** comes as **2 numbers** that represent the **dimension of the matrix**.

The **output** is the matrix filled spirally starting from **1**. You need to print **every row on a new line**, with the cells **separated by a space**. Check the examples below.

Examples

Input	Output
5, 5	1 2 3 4 5 16 17 18 19 6 15 24 25 20 7 14 23 22 21 8 13 12 11 10 9

Input	Output
3, 3	1 2 3 8 9 4 7 6 5