

Exercise: Arrays and Matrices

Problems for exercise and homework for the ["JavaScript Fundamentals Course@SoftUni"](https://judge.softuni.bg/Contests/313). Submit your solutions in the SoftUni judge system at <https://judge.softuni.bg/Contests/313>

1. Print an Array with a given Delimiter

Write a JS function that prints a given array.

The **input** comes as an **array of strings**. The last element of the array is the delimiter.

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

Examples

Input	Output	Input	Output
<pre>['One', 'Two', 'Three', 'Four', 'Five', '-']</pre>	One-Two-Three-Four-Five	<pre>['How about no?', 'I', 'will', 'not', 'do', 'it!', '_']</pre>	How about no?_I_will_not_do_it!

Hints

- Let's start by extracting the delimiter from the input array:

```
function main(input) {  
    let delimiter = input[input.length - 1];  
}
```
- Now that we have the element, we need to delete it from the array, because we don't need it in the output. Thankfully, the Array in JavaScript has a **built-in function** for **removing the last element**, which is **Array.pop()**.

```
function main(input) {  
    let delimiter = input[input.length - 1];  
    input.pop();  
}
```
- And last but not least, let's print each element of the array, separated with the next one by the delimiter:

```

let result = "";

for(let i = 0; i < input.length; i++) {
    if(i == 0) {
        result += input[i];
    } else {
        result += delimiter + input[i];
    }
}

console.log(result);

```

- The **result** variable holds our final string. The **if** check in the loop is necessary so that we don't attach an **unneeded delimiter** somewhere in the result string.

2. Print every N-th Element from an Array

Write a JS function that prints every element of an array, on a given step.

The **input** comes as an **array of strings**. The last element 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 print the **1-st**, the **4-th**, the **7-th** ... and so on, until you reach the end of the array. The elements must be printed each on a new line.

Examples

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

- Use what you've seen from the **previous problem** to **extract the last element** of the array.
- Create a **step** variable to hold the **given step** of the array. Then **print all the elements** with a **for** loop, **incrementing the loop variable** with the value of the **step** variable.

3. *Add and Remove Elements from an Array

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 array itself, with each element 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 an **array of strings**. The **last element** of the array 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 a Non-decreasing Subsequence from an Array

Write a JS 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. Each element should be printed on a new line.

Examples

Input	Output
[1, 3, 8, 4, 10, 12, 3, 2, 24]	1 3 8 10 12 24

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

Input	Output
[20, 3, 2, 15, 6, 1]	20

Hints

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

6. Sort an Array by 2 Criteria

Write a JS function that orders a **given array of strings**, by **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 ordered array of strings.

Examples

Input	Output
['alpha', 'beta', 'gamma']	beta alpha gamma

Input	Output
['Isacc', 'Theodor', 'Jack', 'Harrison', 'George']	Jack Isacc George Theodor Harrison

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

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.

7. Magic Matrices

Write a JS 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	Input	Output	Input	Output
[[4, 5, 6], [6, 5, 4], [5, 5, 5]]	true	[[11, 32, 45], [21, 0, 1], [21, 1, 1]]	false	[[1, 0, 0], [0, 0, 1], [0, 1, 0]]	true

8. *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	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	3, 3	1 2 3 8 9 4 7 6 5

9. **Diagonal Attack

Write a JS function that reads a given matrix of numbers, and checks if both **main diagonals have equal sum**. 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 **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 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.

Examples

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

10. *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 which has 5 rows and 5 columns and the star is at coordinates – **0, 0**. Then the following should happen:

1				

1	2			
2	2			

1	2	3	4	5
2	2	3	4	5
3	3	3	4	5
4	4	4	4	5
5	5	5	5	5

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

		1		

	2	2	2	
	2	1	2	
	2	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

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.