# More Exercise: Arrays

## Print N-th Element

Write a function that collect each element of an array, on a given **step**.

The **input** comes as **array** **of** **strings**. The last element is **N - the step**.

The **collections** are every element on the **N-th** step **starting** fromthe **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. Then, print elements in a row, **separated** by single space.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| ['5', '20', '31', '4', '20', '2'] | 5 31 20 |
| ['dsa', 'asd', 'test', 'test', '2'] | dsa test |
| ['1', '2', '3', '4', '5', '6'] | 1 |

## Add and Remove

Write a 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.

### Input

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

### Output

Print elements in a row, separated by single space. In case of an empty array, just print "**Empty**".

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| ['add', 'add', 'add', 'add'] | 1 2 3 4 |
| ['add', 'add', 'remove', 'add', 'add'] | 1 4 5 |
| ['remove', 'remove', 'remove'] | Empty |

## Rotate Array

Write a 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 **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 |
| ['Banana', 'Orange', 'Coconut', 'Apple', '15'] | Orange Coconut Apple Banana |
| ['remove', 'remove', 'remove'] | Empty |

### Hints

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

## Non-Decreasing Subsequence

Write a function that extracts only those numbers that forma **non-decreasing subsequence**. In other words, you start from the **first element** and continue to **the end** of the givenarrayof 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.

### Input

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

### Output

The **output** is the processed array after the filtration, which should be a non-decreasing subsequence. The elements should be printed on one line, separated by a **single space**.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| [ 1, 3, 8, 4, 10, 12, 3, 2, 24] | 1 3 8 10 12 24 |
| [ 1, 2, 3, 4] | 1 2 3 4 |
| [ 20, 3, 2, 15, 6, 1] | 20 |

### Hints

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

## Tseam Account

As a gamer, Peter has Tseam Account. He loves to buy new games. You are given Peter's account with all of his games-> **strings**, **separated** by space. Until you receive **"Play!"** you will be receiving commands which Peter does with his account.`

You may receive the following commands:

* **Install {game}**
* **Uninstall {game}**
* **Update {game}**
* **Expansion {game}-{expansion}**

If you receive **Install command**, you should **add** the game at **last** position in the account, but only if it **isn't** installed already.

If you receive **Uninstall command**, **delete** the game if it **exists**.

If you receive **Update command**, you should **update** the game **if it exists** and place it on **last position**.

If you receive **Expansion command**, you should check if the game exists and **insert** after it the expansion in the following format: "**{game}:{expansion}";**

### Input

* On the **first input line** you will receive Peter`s **account** - **sequence** of game names, **separated** by space.
* Until you receive **"Play!"** you will be receiving **commands**.

### Output

* As output you must print Peter`s Tseam **account**.

### Constraints

* The **command will always be valid.**
* The **game** and **expansion** will be strings and will contain any character, except **'-'**.
* Allowed working **time** / **memory**: **100ms** / **16MB**.

### Examples

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comments** |
| ['CS WoW Diablo',  'Install LoL',  'Uninstall WoW',  'Update Diablo',  'Expansion CS-Go',  'Play!'] | CS CS:Go LoL Diablo | We receive the account => CS, WoW, Diablo  We Install LoL => CS, WoW, Diablo, LoL  Uninstall WoW => CS, Diablo, LoL  Update Diablo => CS, LoL, Diablo  We add expansion => CS, CS:Go, LoL, Diablo  We print the account. |
| ['CS WoW Diablo',  'Uninstall XCOM',  'Update PeshoGame',  'Update WoW',  'Expansion Civ-V',  'Play!'] | CS Diablo WoW |  |

# 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.

## 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 everyrow and everycolumn are equal.

### Input

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

### Output

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 |

### Hints

* You can read more about the magic square [here](https://en.wikipedia.org/wiki/Magic_square).

## Spiral Matrix

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

### Input

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

### Output

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 |

## Diagonal Attack

Write a 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.

### Input

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.

### Output

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** |  | **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 |  | ['1 1 1',  '1 1 1',  '1 1 0'] | 1 1 1  1 1 1  1 1 0 |

## 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 info.

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 |  |  |  |  |  | 1 | 2 | 3 | 4 | 5 |
|  |  |  |  |  |  |  | 2 | 2 |  |  |  |  |  | 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:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  | 3 | 3 | 3 | 3 | 3 |
|  |  |  |  |  |  |  |  | 2 | 2 | 2 |  |  |  | 3 | 2 | 2 | 2 | 3 |
|  |  | 1 |  |  |  |  |  | 2 | 1 | 2 |  |  |  | 3 | 2 | 1 | 2 | 3 |
|  |  |  |  |  |  |  |  | 2 | 2 | 2 |  |  |  | 3 | 2 | 2 | 2 | 3 |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  | 3 | 3 | 3 | 3 | 3 |

### Input

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

### Output

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

### Examples

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Input** | **Output** |  | **Input** | **Output** |  | **Input** | **Output** |
| [4, 4, 0, 0] | 1 2 3 4  2 2 3 4  3 3 3 4  4 4 4 4 |  | [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 | [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.