~~Lab~~Exercises: Object Composition

~~Problems for exercises and homework for the "JavaScript Advanced" course @ SoftUni. Submit your solutions in the SoftUni judge system at https://judge.softuni.bg/Contests/1545.~~

~~2.~~1. Order Rectangles

You will be passed a few pairs of **widths** and **heights** of rectangles, create **objects** to represent the rectangles. The objects should additionally have two functions **`area`** - that returns the area of the rectangle and **`compareTo`** - that compares the current rectangle with another and produces a number -signifying if the current rectangle is **smaller** (negative number), **equal** (0) or **larger** (positive number) than the other rectangle.

Input

The input will come as an **array of arrays** - every nested array will contain exactly 2 numbers the **`width`** and the **`height`** of the rectangle.

Output

The output must consist of an array of **rectangles** (objects) sorted by their **`area`** in **descending** order as a **first** criteria and by their **`width`** in **descending** order as a **second** criteria.

Examples

Input	Output
`[[10,5],[5,12]]`	`[{width:5, height:12, area:function(), compareTo:function(other)},` `{width:10, height:5, area:funciton(),compareTo:function(other)}]`

`[[10,5], [3,20], [5,12]]`	`[{width:5, height:12, area:function(), compareTo:function(other)},`
	`{width:3, height:20, area:funciton(),compareTo:function(other)},`
	`{width:10, height:5, area:funciton(),compareTo:function(other)}]]`

~~3.~~2. Fibonacci

Write a JS function that when called, returns the next Fibonacci number, starting at 0, 1. Use a **closure** to keep the current number.

Formatted: Font: Bold

Input

There will be no input.

Output

The **output** must be a Fibonacci number and must be **returned** from the function.

Examples

Sample exectuion

`let fib = getFibonator();`
`console.log(fib()); // 1`
`console.log(fib()); // 1`
`console.log(fib()); // 2`
`console.log(fib()); // 3`
`console.log(fib()); // 5`
`console.log(fib()); // 8`
`console.log(fib()); // 13`

Formatted: Italian (Italy)

~~4.~~3. List Processor

Using a closure, create an inner object to process list commands. The commands supported should be the following:

- **add <string>** - adds the following string in an inner collection.
- **remove <string>** - removes all occurrences of the supplied **<string>** from the inner collection.
- **print** - prints all elements of the inner collection joined by **","**.

Input

The **input** will come as an **array of strings** - each string represents a **command** to be executed from the command execution engine.

Output

For every print command - you should print on the console the inner collection joined by **","**

Examples

Input	Output
`['add hello', 'add again', 'remove hello', 'add again', 'print']`	`again,again`
`['add pesho', 'add gosho', 'add pesho', 'remove pesho','print']`	`gosho`

~~5.~~4. Cars

Write a closure that can create and modify objects. All created objects should be **kept** and be accessible by **name**. You should support the following functionality:

- **`create <name>`** - creates an object with the supplied **<name>**
- **`create <name> inherits <parentName>`** - creates an object with the given **<name>**, that inherits from the parent object with the **<parentName>**
- **`set <name> <key> <value>`** - sets the property with key equal to **<key>** to **<value>** in the object with the supplied **<name>**.
- **`print <name>`** - prints the object with the supplied **<name>** in the format **`"<key1>:<value1>,<key2>:<value2>…"`** - the printing should also print all **inherited properties** from parent objects. Inherited properties should come after own properties.

Input

The **input** will come as an **array of strings** - each string represents a **command** to be executed from your closure.

Output

For every **`print`** command - you should print on the console all properties of the object in the above mentioned format.

Constraints

- **All commands will always be valid, there will be no nonexistent or incorrect input.**

Examples

Input	Output

`['create c1',`
`'create c2 inherit c1',`
`'set c1 color red',`
`'set c2 model new',`
`'print c1',`
`'print c2']`
	`color:red`
	`model:new, color:red`

~~6.~~5. Sum

Create a function which returns an object that can modify the DOM. The returned object should support the following functionality:

- **`init(selector1, selector2, resultSelector)`** - initializes the object to work with the elements corresponding to the supplied selectors.
- **`add()`** - **adds** the numerical value of the element corresponding to **`selector1`** to the numerical value of the element corresponding to **`selector2`** and then writes the result in the element corresponding to **resultSelector**
- **`subtract()`** - **subtracts** the numerical value of the element corresponding to **`selector2`** from the numerical value of the element corresponding to **`selector1`** and then writes the result in the element corresponding to **`resultSelector`**

Input

There will be no input your function must only provide an object.

Output

Your function should return an object that meets the specified requirements.

Constraints

- **All commands will always be valid, there will be no nonexistent or incorrect input.**
- **All selectors will point to single textbox elements.**

HTML

You are given the following HTML for testing purposes:

sum.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
<input type="text" id="num1" />
<input type="text" id="num2" />
<input type="text" id="result" readonly />
<br>
<button id="sumButton">
    Sum</button>
<button id="subtractButton">
    Subtract</button>
</body>
</html>
```