Lab: Advanced Functions

Problems for in-class lab for the "JavaScript Advanced" course @ SoftUni. Submit your solutions in the SoftUni judge system at https://judge.softuni.bg/Contests/2764/Advanced-Functions-Lab.

- Area and Volume Calculator

Write a function that **calculates** the **area** and the **volume** of a figure, which is **defined** by its **coordinates** (**x**, **y**, **z**).

area()

```
function area() {
   return Math.abs(this.x * this.y);
};
```

vol()

```
function vol() {
   return Math.abs(this.x * this.y * this.z);
};
```

solve()

```
function solve(area, vol, input) {
   //ToDo....
}
```

Input

You will receive **3** parameters - the **functions area** and **vol** and a **string**, which contains the figures' coordinates.

For more information check the examples.

Output

The output should be **returned** as an **array of objects**. Each object has **two properties**: the figure's **area** and **volume**.

[
{ area: ${area1}, volume: ${volume1} },
{ area: ${area2}, volume: ${volume2} },
. . .
]

Note:

Submit only the solve function.

Examples

Sample Input	Output
area, vol, `[** **{"x":"1","y":"2","z":"10"},** **{"x":"7","y":"7","z":"10"},** **{"x":"5","y":"2","z":"10"}** **]`	**[** **{ area: 2, volume: 20 },** **{ area: 49, volume: 490 },** **{ area: 10, volume: 100 }** **]**
area, vol, `[** **{"x":"10","y":"-22","z":"10"},** **{"x":"47","y":"7","z":"-5"},** **{"x":"55","y":"8","z":"0"},** **{"x":"100","y":"100","z":"100"},** **{"x":"55","y":"80","z":"250"}** **]`	**[** **{ area: 220, volume: 2200 },** **{ area: 329, volume: 1645 },** **{ area: 440, volume: 0 },** **{ area: 10000, volume: 1000000 },** **{ area: 4400, volume: 1100000 }** **]**

- Add

Write a program that keeps a number inside its context and **returns** a new function that adds a given number to the previous one.

Input

Check the examples below to see how your code will be executed.

Output

Your function should return the final result.

Examples

Sample Input	Output
let add5 = solution(5); **console.log(add5(2));** **console.log(add5(3));**	**7** **8**
let add7 = solution(7); **console.log(add7(2));** **console.log(add7(3));**	**9** **10**

- Currency Format

Write a **higher-order** function createFormatter that fixes some of the parameters of another function. Your program will **receive four parameters**: **three values** and a **function** that **takes 4 parameters** and **returns a formatted string** (a monetary value with currency symbol).
Your task is to **return a partially applied function**, based on the input function that has its **first three** parameters fixed and only **takes one parameter**.
You will receive the following function:

currencyFormatter

```
function currencyFormatter(separator, symbol, symbolFirst, value) {
   let result = Math.trunc(value) + separator;
   result += value.toFixed(2).substr(-2,2);
   if (symbolFirst) return symbol + ' ' + result;
   else return result + ' ' + symbol;
}
```

Receive and set the following parameters to fixed values:
separator
symbol
symbolFirst
The final parameter value is the one that the returned function must receive.
Input
You will receive four parameters:

- separator (string)
- symbol (string)
- symbolFirst (Boolean)
- formatter (function)

Output
You need to return a function that takes one parameter - value
Examples

Sample Input
let dollarFormatter = createFormatter(',', '$', true, currencyFormatter); **console.log(dollarFormatter(5345));** ***// $ 5345,00*** **console.log(dollarFormatter(3.1429));** ***// $ 3,14*** **console.log(dollarFormatter(2.709));** ***// $ 2,71***

- Filter Employees

Write a program that filters the employees of your company. You should print the result in a specific format. You will receive **2** parameters (**data**, **criteria**). You should **parse** the input, find all employees that fulfill the criteria, and print them.
Input
You will receive a **string** with all the employees, and **criteria** by which you should sort the employees. If the criteria are "**all**" print all the employees in the given format.
Output
The output should be printed on the console.
For more information check the examples.
Examples

Sample Input	Output

`[{
"id": "1",
"first_name": "Ardine",
"last_name": "Bassam",
"email": "abassam0@cnn.com",
"gender": "Female"
},{
"id": "2",
"first_name": "Kizzee",
"last_name": "Jost",
"email": "kjost1@forbes.com",
"gender": "Female"
},
{
"id": "3",
"first_name": "Evanne",
"last_name": "Maldin",
"email": "emaldin2@hostgator.com",
"gender": "Male"
}]`,
'gender-Female'
	0. Ardine Bassam - abassam0@cnn.com
	1. Kizzee Jost - kjost1@forbes.com

`[{
"id": "1",
"first_name": "Kaylee",
"last_name": "Johnson",
"email": "k0@cnn.com",
"gender": "Female"
},{
"id": "2",
"first_name": "Kizzee",
"last_name": "Johnson",
"email": "kjost1@forbes.com",
"gender": "Female"
},{
"id": "3",
"first_name": "Evanne",
"last_name": "Maldin",
"email": "emaldin2@hostgator.com",
"gender": "Male"
},{
"id": "4",
"first_name": "Evanne",
"last_name": "Johnson",
"email": "ev2@hostgator.com",
"gender": "Male"
}]`,
'last_name-Johnson'
	0. Kaylee Johnson - k0@cnn.com
	1. Kizzee Johnson - kjost1@forbes.com
	2. Evanne Johnson - ev2@hostgator.com

- Command Processor

Write a program that keeps a string inside its context and can execute different commands that modify or print the string on the console.
append(string) - append the given string at the end of the internal string
removeStart(n) - remove the first n characters from the string, n is an integer
removeEnd(n) - remove the last n characters from the string, n is an integer
print - print the stored string on the console

Input
Check the examples below to see how your code will be executed.
Output
Whenever you receive the command print, the output should be printed on the console.
Examples

Sample Input	Output
let firstZeroTest = solution(); **firstZeroTest.append('hello');** **firstZeroTest.append('again');** **firstZeroTest.removeStart(3);** **firstZeroTest.removeEnd(4);** **firstZeroTest.print();**	**loa**
let secondZeroTest = solution(); **secondZeroTest.append('123');** **secondZeroTest.append('45');** **secondZeroTest.removeStart(2);** **secondZeroTest.removeEnd(1);** **secondZeroTest.print();**	**34**

- 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 george', 'add peter', 'remove peter','print']	**pesho,george**

- 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