Lab: Objects & Composition

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/2758/Objects-and-Composition-Lab.

1. City Record

You will receive a city's **name** (string), **population** (number), and **treasury** (number) as arguments, which you will need to set as **properties** of an **object** and **return** it.

Examples

Input	Output

`'Tortuga',`
`7000,`
`15000`
	`{`
	`  name: 'Tortuga',`
	`  population: 7000,`
	`  treasury: 15000`
	`}`

`'Santo Domingo',`
`12000,`
`23500`
	`{`
	`  name: 'Santo Domingo',`
	`  population: 12000,`
	`  treasury: 23500`
	`}`

2. Town Population

You have been tasked to create a registry for different **towns** and their **population**.

Input

The **input** comes as array of strings. Each element will contain data for a town and its population in the following format: **"{townName} <-> {townPopulation}"**

If you receive the same town twice, **you should add** the **given population** to the **current one**.

Output

As **output**, you must print all the towns and their population.

Examples

Input	Output

`['Sofia <-> 1200000',`
`'Montana <-> 20000',`
`'New York <-> 10000000',`
`'Washington <-> 2345000',`
`'Las Vegas <-> 1000000']`
	`Sofia : 1200000`
	`Montana : 20000`
	`New York : 10000000`
	`Washington : 2345000`
	`Las Vegas : 1000000`

`['Istanbul <-> 100000',`
`'Honk Kong <-> 2100004',`
`'Jerusalem <-> 2352344',`
`'Mexico City <-> 23401925',`
`'Istanbul <-> 1000']`
	`Istanbul : 101000`
	`Honk Kong : 2100004`
	`Jerusalem : 2352344`
	`Mexico City : 23401925`

Follow us:

3. City Taxes

This task is an extension of Problem 1, you may use your solution from that task as a base.

You will receive a city's **name** (string), **population** (number), and **treasury** (number) as arguments, which you will need to set as **properties** of an **object** and **return** it. In addition to the input parameters, the object must have a property **taxRate** with an initial value of **10**, and three **methods** for managing the city:

- **collectTaxes()** - Increase **treasury** by **population** * **taxRate**
- **applyGrowth(percentage)** - Increase population by **given percentage**
- **applyRecession(percentage)** - Decrease treasury by **given percentage**

Round down the values after each calculation.

Input

Your solution will receive three **valid** parameters. The methods that expect parameters will be tested with valid input.

Output

Return an **object** as described above. The methods of the object modify the object and don't return anything.

Input	Output
```const city = cityTaxes('Tortuga', 7000, 15000); console.log(city);```	```{ name: 'Tortuga', population: 7000, treasury: 15000, taxRate: 10, collectTaxes: [Function: collectTaxes], applyGrowth: [Function: applyGrowth], applyRecession: [Function: applyRecession] }```

Testing with code	
Input	**Output**
```const city = cityTaxes('Tortuga', 7000, 15000); city.collectTaxes(); console.log(city.treasury); city.applyGrowth(5); console.log(city.population);```	85000 7350

4. Object Factory

Create a function that can compose objects by copying functions from a given library of functions. You will receive **two parameters** – a **library** of functions as an associative array (object) and an **array of orders**, represented as objects**. You must **return** a new array – the fulfilled orders.

The **first parameter** will be an object where each property is a **function**. You will use this **library of functions** to compose new objects.

The **second parameter** is an **array of orders**. Each order is an **object** with the following shape:

```
{
  template: [Object],
  parts: string[]
}
```

A **template** is an object that must be **copied**. The **parts array** contains the names of **required functions** as **strings**.

You must **create and return a new array**, by fulfilling all orders from the **orders array**. To fulfill an order, create a copy of the object's template and then add to it all functions, listed in the **parts array** of the order, by taking them from the **function library** (the first parameter to your solution).

Input

You will receive two parameters:

- **library** – an object
- **orders** – an array of objects

Output

Your solution must **return an array** of objects.

Example

Input
| ```
const library = {
  print: function () {
    console.log(`${this.name} is printing a page`);
  },
  scan: function () {
    console.log(`${this.name} is scanning a document`);
  },
  play: function (artist, track) {
    console.log(`${this.name} is playing '${track}' by ${artist}`);
  },
};
const orders = [
  {
    template: { name: 'ACME Printer'},
    parts: ['print']
  },
  {
    template: { name: 'Initech Scanner'},
    parts: ['scan']
  },
  {
    template: { name: 'ComTron Copier'},
    parts: ['scan', 'print']
  },
  {
    template: { name: 'BoomBox Stereo'},
    parts: ['play']
  }
];
``` |

```
const products = factory(library, orders);
console.log(products);
```

Output
```
[
  {
    name: 'ACME Printer',
    print: [Function: print]
  },
  {
    name: 'Initech Scanner',
    scan: [Function: scan]
  },
  {
    name: 'ComTron Copier',
    scan: [Function: scan],
    print: [Function: print]
  },
  {
    name: 'BoomBox Stereo',
    play: [Function: play]
  }
]
```

5. Assembly Line

Create a function that **returns** a **library of decorator functions**. They can be used to **compose** different functionality in a **car object** that they receive as an argument.

Your solution must **return an object**, containing **three decorator functions**:

hasClima – compose air conditioning controls into the passed-in object. This function takes an **object as a parameter** and adds to it the following properties:

- **temp** – **number** with default value **21**;
- **tempSettings** – **number** with default value **21**;
- **adjustTemp** – **function** which takes **no arguments**. If **temp** is less than **tempSettings**, this function adds 1 to **temp**. If **temp** is more than **tempSettings**, it decreases **temp** by 1. If **temp** and **tempSettings** are equal, the function does nothing.

hasAudio – compose audio player functionality into the passed-in object. This function takes an **object as a parameter** and adds to it the following properties:

- **currentTrack** – **object** with properties **name** (string) and **artist** (string). The default value is **null**;
- **nowPlaying** – **function**, which **prints** on the console the text:
    `Now playing '${currentTrack.name}' by ${currentTrack.artist}` , where **name** and **artist** are properties of the **currentTrack** object. If **currentTrack** is **null**, this function does nothing.

hasParktronic – compose parking aid functionality into the passed in object. This function takes an **object as a parameter** and adds to it the following properties:

- **checkDistance** – **function**, which takes a **single argument distance** (number) and **prints** a message on the console, depending on its value:

> **distance** < 0.1 – **"Beep! Beep! Beep!"**
>
> 0.1 <= **distance** < 0.25 – **"Beep! Beep!"**
>
> 0.25 <= **distance** < 0.5 – **"Beep!"**

In any other case, print an **empty string**.

Input

Your **solution** will receive **no arguments**. All the methods in the returned library must take an **object as an argument**. Any methods that you compose into this object must meet the input requirements listed in the description above.

Output

Your **solution** must **return an object** containing the **three decorators** described above.

Example

Setup

```const assemblyLine = createAssemblyLine();```
 	
 ```const myCar = {```
`    make: 'Toyota',`
`    model: 'Avensis'`
`};`

Input	Output
assemblyLine.hasClima(**myCar**); console.log(**myCar**.temp); **myCar**.tempSettings = 18; **myCar**.adjustTemp(); console.log(**myCar**.temp);	21 20

Input	Output

assemblyLine.hasAudio(**myCar**);
myCar.currentTrack = {
`    name: 'Never Gonna Give You Up',`
`    artist: 'Rick Astley'`
`};`
myCar.nowPlaying();
	Now playing 'Never Gonna Give You Up' by Rick Astley

Input	Output
assemblyLine.hasParktronic(**myCar**); **myCar**.checkDistance(0.4); **myCar**.checkDistance(0.2);	Beep! Beep! Beep!

Input	Output

console.log(**myCar**);	{
	`  make: 'Toyota',`
	`  model: 'Avensis',`
	`  temp: 20,`
	`  tempSettings: 18,`
	`  adjustTemp: [Function],`
	`  currentTrack: {`
	`    name: 'Never Gonna Give You Up',`
	`    artist: 'Rick Astley'`
	`  },`

	nowPlaying: [Function], checkDistance: [Function] }

6. From JSON to HTML Table

You're tasked with creating an HTML table of students and their scores. You will receive a single string representing an **array of objects**, the **table's headings** should be equal to the **object's keys**, while **each object's values** should be a **new entry** in the table. Any **text values** in an object should be **escaped**, to avoid introducing dangerous code into the HTML.

Input

The **input** comes with a **single string argument** (the array of objects).

Output

The **output** should be printed on the console – for each **entry row** in the input print the **object representing it**.

Note:

Objects' **keys** will always be the **same.** Check more information for the **HTML Entity here.**

HTML

You are provided with an HTML file to test your table in the browser.

index.html

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>FromJSONToHTMLTable</title>
    <style>
        table,th{
            border: groove;
            border-collapse: collapse;
        }
        td{
            border: 1px solid black;
        }
        td,th{
            padding: 5px;
        }
    </style>
</head>
<body>
    <div id="wrapper">
    </div>
    <script>
        function fromJSONToHTMLTable(input){
            //Write your code here
        }
        window.onload = function(){
            let container = document.getElementById('wrapper');
            container.innerHTML =
fromJSONToHTMLTable(['[{"Name":"Stamat","Price":5.5},{"Name":"Rumen","Price":6}]']);
        };
    </script>
</body>
</html>
```

Examples

Input	Output
`` `[{"Name":"Stamat",` `    "Score":5.5},` `  {"Name":"Rumen",` `   "Score":6}]` ``	`<table>` `   <tr><th>Name</th><th>Score</th></tr>` `   <tr><td>Stamat</td><td>5.5</td></tr>` `   <tr><td>Rumen</td><td>6</td></tr>` `</table>`
`` `[{"Name":"Pesho",` `    "Score":4,` `    " Grade":8},` `  {"Name":"Gosho",` `   "Score":5,` `   " Grade":8},` `  {"Name":"Angel",` `   "Score":5.50,` `   " Grade":10}]` ``	`<table>` `   <tr><th>Name</th><th>Score</th><th>Grade</th></tr>` `   <tr><td>Pesho</td><td>4</td><td>8</td></tr>` `   <tr><td>Gosho</td><td>5</td><td>8</td></tr>` `   <tr><td>Angel</td><td>5.5</td><td>10</td></tr>` `</table>`