# Lab: Objects & Composition

Problems for in-class lab for the ["JavaScript Advanced" course @ SoftUni](https://softuni.bg/trainings/3217/js-advanced-january-2021). Submit your solutions in the SoftUni judge system at <https://judge.softuni.bg/Contests/2758/Objects-and-Composition-Lab>.

## 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  } |

## 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 |

## 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 initial value **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 |

## 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[]

}

The **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** (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]  },  ] |