# Lab: Classes

Submit your solutions in the SoftUni judge system at:

<https://judge.softuni.org/Contests/2768/Classes-and-Attributes-Lab>

## Person

Write a **class** that represents a personal record. It has the following properties, all set from the constructor:

* firstName
* lastName
* age
* email

And a method toString(), which prints a summary of the information. See the example for formatting details.

### Input

The constructor function will receive valid parameters.

### Output

The toString()method should **return** a string in the following format:

**`{firstName} {lastName} (age: {age}, email: {email})`**

Submit the class definition as is, **without** wrapping it in any function.

### Example

|  |
| --- |
| Sample Input |
| let person = new Person('Anna', 'Simpson', 22, 'anna@yahoo.com');  console.log(person.toString()); |
| Output |
| Anna Simpson (age: 22, email: anna@yahoo.com) |

## Get Persons

Write a function that returns an array of **Person** objects. Use the class from the previous task, create the following instances, and return them in an array:

|  |  |  |  |
| --- | --- | --- | --- |
| First Name | Last Name | Age | Email |
| Anna | Simpson | 22 | anna@yahoo.com |
| SoftUni |  |  |  |
| Stephan | Johnson | 25 |  |
| Gabriel | Peterson | 24 | g.p@gmail.com |

For any empty cells, do not supply a parameter (call the constructor with fewer parameters).

### Input / Output

There will be **no input**, the data is static and matches the table above. As **output**, **return an array** with **Person** **instances**.

Submit a function that returns the required output.

## Circle

Write a **class** that represents a **Circle**. It has only one data property - its **radius**, and it is set through the **constructor**. The class needs to have **getter** and **setter** methods for its **diameter** - the setter needs to calculate the radius and change it and the getter needs to use the radius to calculate the diameter and return it.

The circle also has a getter area(), which calculates and **returns** its area.

### Input

The constructor function and diameter setter will receive valid parameters.

### Output

The diameter() and area() getters should **return** numbers.

Submit the class definition as is, **without** wrapping it in any function.

### Examples

|  |  |
| --- | --- |
| Sample Input | Output |
| let c = new Circle(2);  console.log(`Radius: ${c.radius}`);  console.log(`Diameter: ${c.diameter}`);  console.log(`Area: ${c.area}`);  c.diameter = 1.6;  console.log(`Radius: ${c.radius}`);  console.log(`Diameter: ${c.diameter}`);  console.log(`Area: ${c.area}`); | Radius: 2  Diameter: 4  Area: 12.566370614359172  Radius: 0.8  Diameter: 1.6  Area: 2.0106192982974678 |

## Point Distance

Write a JS **class** that represents a **Point**. It has **x** and **y** coordinates as properties, that are set through the constructor, and a **static method** for finding the distance between two points, called distance().

### Input

The distance() method should receive two **Point** objects as parameters.

### Output

The distance() method should **return** a number, the distance between the two-point parameters.

Submit the class definition as is, **without** wrapping it in any function.

### Example

|  |  |
| --- | --- |
| Sample Input | Output |
| let p1 = new Point(5, 5);  let p2 = new Point(9, 8);  console.log(Point.distance(p1, p2)); | 5 |

## Auto-Engineering Company

You are tasked to create a register for a company that produces cars. You need to store **how many cars** have been produced from a **specific model** of a **specific brand**.

### Input

The **input** comes as array of strings. Each element holds information in the following format:

"{carBrand} | {carModel} | {producedCars}"

The **carBrand** and **carModel** are **strings**, the **producedCars** are **numbers**. If the **carBrand** you’ve received **already exists**, just add the **new** **carModel** to it with the **produced cars** **as its value**. If even the **carModel** exists, just **add** the **given value** to the **current one**.

### Output

As **output**, you need to print - **for every car brand**, the **car models**, and a **number of cars produced** from that model. The output format is:

`{carBrand}

###{carModel} -> {producedCars}

###{carModel2} -> {producedCars}

...`

The order of printing is the order in which the brands and models first appear in the input. The first brand in the input should be the first printed and so on. For each brand, the first model received from that brand, should be the first printed and so on.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| ['Audi | Q7 | 1000',  'Audi | Q6 | 100',  'BMW | X5 | 1000',  'BMW | X6 | 100',  'Citroen | C4 | 123',  'Volga | GAZ-24 | 1000000',  'Lada | Niva | 1000000',  'Lada | Jigula | 1000000',  'Citroen | C4 | 22',  'Citroen | C5 | 10'] | Audi  ###Q7 -> 1000  ###Q6 -> 100  BMW  ###X5 -> 1000  ###X6 -> 100  Citroen  ###C4 -> 145  ###C5 -> 10  Volga  ###GAZ-24 -> 1000000  Lada  ###Niva -> 1000000  ###Jigula -> 1000000 |

## String Builder \*

You are given the following **JavaScript class**:

|  |
| --- |
| string-builder.js |
| **class** StringBuilder {  constructor(string) {  **if** (string !== ***undefined***) {  StringBuilder.*\_vrfyParam*(string);  **this**.**\_stringArray** = Array.from(string);  } **else** {  **this**.**\_stringArray** = [];  }  }   append(string) {  StringBuilder.*\_vrfyParam*(string);  **for**(**let** i = 0; i < string.**length**; i++) {  **this**.**\_stringArray**.push(string[i]);  }  }   prepend(string) {  StringBuilder.*\_vrfyParam*(string);  **for**(**let** i = string.**length** - 1; i >= 0; i--) {  **this**.**\_stringArray**.unshift(string[i]);  }  }   insertAt(string, startIndex) {  StringBuilder.*\_vrfyParam*(string);  **this**.**\_stringArray**.splice(startIndex, 0, ...string);  }   remove(startIndex, length) {  **this**.**\_stringArray**.splice(startIndex, length);  }   **static** *\_vrfyParam*(param) {  **if** (**typeof** param !== **'string'**) **throw new TypeError**(**'Argument must be a string'**);  }   toString() {  **return this**.**\_stringArray**.join(**''**);  } } |

### Functionality

The above code defines a **class** that holds **characters** (strings with length 1) in an array. An **instance** of the class should support the following operations:

* Can be **instantiated** with a passed in **string** argument or **without** anything
* Functionappend(string) - **converts** the passed in **string** argument to an **array** and adds it to the **end** of the storage
* Function **prepend**(**string**) - **converts** the passed in **string** argument to an **array** and adds it to the **beginning** of the storage
* FunctioninsertAt(string, index) - **converts** the passed in **string** argument to an **array** and adds it at the **given** index (you only need to test the behavior when the index is in range)
* Functionremove(startIndex, length) - **removes** elements from the storage, starting at the given index (**inclusive**), **length** number of characters (you only need to test the behavior when the index is in range)
* FunctiontoString() - **returns** a string with **all** elements joined by an **empty** string
* All passed in **arguments** should be **strings.** If any of them are **not**, **throws** a type **error** with the following message: **'Argument must be a string'**

### Example

This is an example of how this code is **intended to be used**:

|  |  |  |
| --- | --- | --- |
| Sample code usage |  | Corresponding output |
| **let** str = **new** StringBuilder(**'hello'**); str.append(**', there'**); str.prepend(**'User, '**); str.insertAt(**'woop'**,5 ); **console**.log(str.toString()); str.remove(6, 3); **console**.log(str.toString()); | User,woop hello, there  User,w hello, there |

### Your Task

Using **Mocha** and **Chai** write **JS unit tests** to test the entire functionality of the StringBuilder class. Make sure it is **correctly defined as a class** and instances of it have all the required functionality. You may use the following code as a template:

|  |
| --- |
| describe(**"*TODO* …"**, **function**() {  ***it***(**"*TODO …*"**, **function**() {  *//* ***TODO:*** …  });  *//* ***TODO:*** …  }); |