# Homework: Simulating OOP in JavaScript

This document defines the homework assignments from the ["Advanced JavaScript" Course @ Software University](https://softuni.bg/trainings/1099/Advanced-JavaScript-March-2015). Please submit as homework a single zip / rar / 7z archive holding the solutions (source code) of all below described problems.

## Bug fix

A fellow programmer tried to create a simple "class" in JavaScript but he's got a problem getting it to work. Fix any bugs you may find in his code.

function Person(firstName, lastName) {

this.firstName = firstName;

this.lastName = lastName;

return {name: this.firstName + " " + this.lastName};

}

|  |  |
| --- | --- |
| **Sample Code** | **Sample Output** |
| var peter = new Person("Peter", "Jackson");  console.log(peter.name);  console.log(peter.firstName);  console.log(peter.lastName); | Peter Jackson  Peter  Jackson |

## \* Bug fix, level 2

After you've solved your friend's problem, now he's got another one. He wants to be able to change the names. Rewrite the function **Person()** so that it will work according to what his boss wants.

You should be able to get and update the values of the first and last name. If you change the first or last name, the full name will need to change automatically. Also, if you change the full name, the first and last names must change too. Refer to the samples to get a better understanding. Assume that **all data will be valid** (e. g., for person.fullName you will always get two words, separated by a single space).

Note that **firstName**, **lastName** and **fullName** are **properties** (not functions).

**Hint:** You may need to check one of the rather "exotic" ways to work with properties in JavaScript.

For the samples, assume the following variable exists:

var person = new Person("Peter", "Jackson");

|  |  |
| --- | --- |
| **Sample Code** | **Sample Output** |
| // Getting values  console.log(person.firstName);  console.log(person.lastName);  console.log(person.fullName); | Peter  Jackson  Peter Jackson |
| // Changing values  person.firstName = "Michael";  console.log(person.firstName);  console.log(person.fullName);  person.lastName = "Williams";  console.log(person.lastName);  console.log(person.fullName); | Michael  Michael Jackson  Williams  Michael Williams |
| // Changing the full name should work too  person.fullName = "Alan Marcus";  console.log(person.fullName);  console.log(person.firstName);  console.log(person.lastName); | Alan Marcus  Alan  Marcus |

## Array Flattening

Implement a function **flatten()** which works on an array of arrays and returns a new array (**does not modify** the original one). The function extracts all inner arrays and merges all values into a single-dimensional array.

|  |  |
| --- | --- |
| **Sample Code** | **Sample Output** |
| var array = [1, 2, 3, 4];  console.log(array.flatten()); | [1, 2, 3, 4] |
| var array = [1, 2, [3, 4], [5, 6]];  console.log(array.flatten());  console.log(array); // Not changed | [1, 2, 3, 4, 5, 6]  [1, 2, [3, 4], [5, 6]] |
| var array = [0, ["string", "values"], 5.5, [[1, 2, true], [3, 4, false]], 10];  console.log(array.flatten()); | [0, "string", "values", 5.5, 1, 2, true, 3, 4, false, 10] |

## Useful String Methods

Write the following methods which work on strings. All methods should return new strings, and should not modify the existing string instances.

* **string.startsWith(substring)** – returns true if the string starts with the provided substring and false otherwise
* **string.endsWith(substring)** – returns true if the string ends with the provided substring and false otherwise
* **string.left(count)** – returns the first **count** characters of the string. If **count** is greater than the length of the string, returns the whole string
* **string.right(count)** – returns the last **count** characters of the string. If **count** is greater than the length of the string, returns the whole string
* **string.padLeft(count, character)** – returns a new string which contains **count** times the specified **character** at its beginning. **character** is optional and defaults to space
* **string.padRight(count, character)** – returns a new string which contains **count** times the specified **character** at its end. **character** is optional and defaults to space
* **string.repeat(count)** – repeats the provided string **count** times. Do not use the default implementation here, write your own

Try to make all methods as efficient as possible (they should consume little memory and take little time to complete). Assume that **all parameters will be correct** (e. g. **substring** will be a string, **count** will be a number, etc.).

|  |  |
| --- | --- |
| **Sample Code** | **Sample Output** |
| var example = "This is an example string used only for demonstration purposes.";  console.log(example.startsWith("This"));  console.log(example.startsWith("this"));  console.log(example.startsWith("other")); | true  false  false |
| var example = "This is an example string used only for demonstration purposes.";  console.log(example.endsWith("poses."));  console.log(example.endsWith ("example"));  console.log(example.startsWith("something else")); | true  false  false |
| var example = "This is an example string used only for demonstration purposes.";  console.log(example.left(9));  console.log(example.left(90)); | This is a  This is an example string used only for demonstration purposes. |
| var example = "This is an example string used only for demonstration purposes.";  console.log(example.right(9));  console.log(example.right(90)); | purposes.  This is an example string used only for demonstration purposes. |
| // Combinations must also work  var example = "abcdefgh";  console.log(example.left(5).right(2)); | de |
| var hello = "hello";  console.log(hello.padLeft(5));  console.log(hello.padLeft(10));  console.log(hello.padLeft(5, "."));  console.log(hello.padLeft(10, "."));  console.log(hello.padLeft(2, ".")); | hello  hello  hello  .....hello  hello |
| var hello = "hello";  console.log(hello.padRight(5));  console.log(hello.padRight(10));  console.log(hello.padRight(5, "."));  console.log(hello.padRight(10, "."));  console.log(hello.padRight(2, ".")); | hello  hello // There are 5 spaces here  hello  hello.....  hello |
| var character = "\*";  console.log(character.repeat(5));  // Alternative syntax  console.log("~".repeat(3)); | \*\*\*\*\*  ~~~ |
| // Another combination  console.log("\*".repeat(5).padLeft(10, "-").padRight(15, "+")); | -----\*\*\*\*\*+++++ |

## \* Vector

Vectors have a lot of applications in programming, mathematics and science. Your task is to write a relatively simple implementation of a vector class. Refer to the example code for more information on how your class is expected to behave.

All vectors have dimensions. A dimension (also called **component**) is nothing more than a number. So, a three-dimensional (3D) vector consists of three numbers, a one-dimensional (1D) vector consists of one number, and a 10D vector consists of ten numbers. Your class **should contain all dimensions of the vector**.

All vectors have the following methods:

* **vector.add(other)** – returns the sum of the two vectors. It is a new vector with the sum of the components of the original vectors: (1, 2, 3) + (4, 5, 6) = (5, 7, 9)
* **vector.substract(other)** – returns the difference of the two vectors. It is a new vector with the difference of the components of the original vectors: (1, 2, 3) - (4, 5, 6) = (-3, -3, -3)
* **vector.dot(other)** – returns the dot product of the two vectors. It is the sum of the products of the components of the original vectors: (1, 2, 3) . (4, 5, 6) = 1 \* 4 + 2 \* 5 + 3 \* 6 = 32
* **vector.norm()** – returns the square root of the sum of the squared components of the vector. Note that this is a number, not a vector: norm((1, 2, 3)) = sqrt(1 \* 1 + 2 \* 2 + 3 \* 3) = sqrt(14) = 3.7416573867…
* **vector.toString()** – returns a user-friendly representation of the vector. Refer to the example to see how to implement this

The dimensions of **vector** and **other** must always be the same. Throw an exception if this does not happen.

|  |  |
| --- | --- |
| **Sample Code** | **Sample Output** |
| var a = new Vector([1, 2, 3]);  var b = new Vector([4, 5, 6]);  var c = new Vector([1, 1, 1, 1, 1, 1, 1, 1, 1, 1]);  console.log(a.toString());  console.log(c.toString());  // The following throw errors  var wrong = new Vector();  var anotherWrong = new Vector([]); | (1, 2, 3)  (1, 1, 1, 1, 1, 1, 1, 1, 1, 1) |
| var a = new Vector([1, 2, 3]);  var b = new Vector([4, 5, 6]);  var c = new Vector([1, 1, 1, 1, 1, 1, 1, 1, 1, 1]);  var result = a.add(b);  console.log(result.toString());  a.add(c); // Error | (5, 7, 9) |
| var a = new Vector([1, 2, 3]);  var b = new Vector([4, 5, 6]);  var c = new Vector([1, 1, 1, 1, 1, 1, 1, 1, 1, 1]);  var result = a.subtract(b);  console.log(result.toString());  a.subtract(c); // Error | (-3, -3, -3) |
| var a = new Vector([1, 2, 3]);  var b = new Vector([4, 5, 6]);  var c = new Vector([1, 1, 1, 1, 1, 1, 1, 1, 1, 1]);  var result = a.dot(b);  console.log(result.toString());  a.dot(c); // Error | 32 |
| var a = new Vector([1, 2, 3]);  var b = new Vector([4, 5, 6]);  var c = new Vector([1, 1, 1, 1, 1, 1, 1, 1, 1, 1]);  console.log(a.norm());  console.log(b.norm());  console.log(c.norm());  console.log(a.add(b).norm()); | 3.7416573867739413  8.774964387392123  3.1622776601683795  12.449899597988733 |