UNIVERSIDAD NACIONAL DE SAN AGUSTÍN DE AREQUIPA
Laboratorio de Lenguages de programación

CARRERA	CURSO	AMBIENTE
Ingeniería de Sistemas	Lenguages de programación	305

PRACTICA No	NOMBRE DE PRÁCTICA	CODIGO DE LAB.	DURACION (HORAS)
08	Programación funcional Java script	305	2

REVISION	FECHA	DESCRIPCIÓN
1	21/05/2019	Revisión de Guías de Laboratorio

1. OBJETIVOS

- Dar a conocer al estudiante de conceptos básicos programación funcional

2. TEMAS A TRATAR

- Programación funcional con Java script

3. MATERIALES, EQUIPOS, SOFTWARE

- Ordenador
- Navegador google chrome o mozilla
- Editor de texto Sublime o Notepad ++.

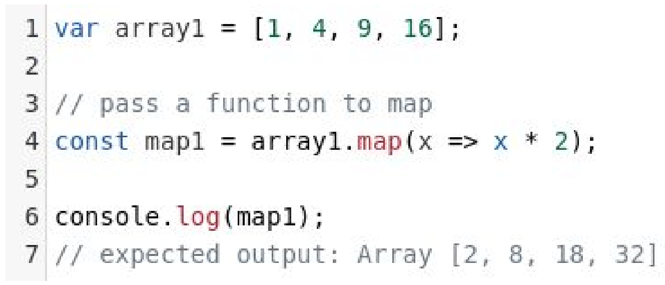
4. CONCEPTOS

MAP, REDUCE, FILTER and ARROW FUNCTIONS

Extraído de la documentación de javascript (https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference)

MAP

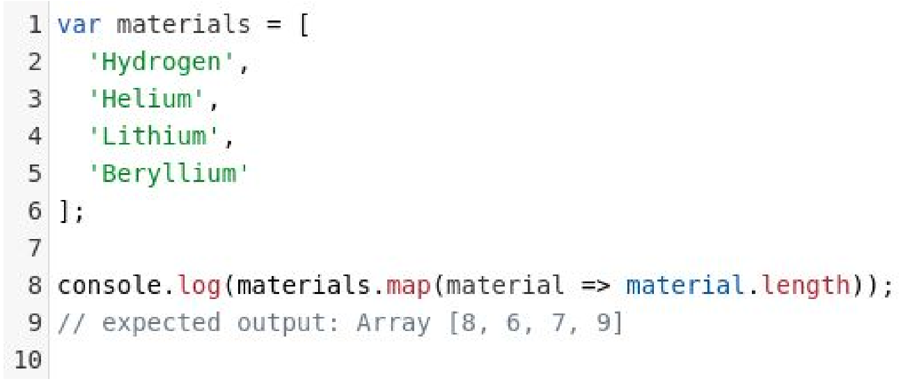
map calls a provided callback function once for each element in an array, in order, and constructs a new array from the results. callback is invoked only for indexes of the array which have assigned values, including undefined. It is not called for missing elements of the array (that is, indexes that have never been set, which have been deleted or which have never been assigned a value). Example:



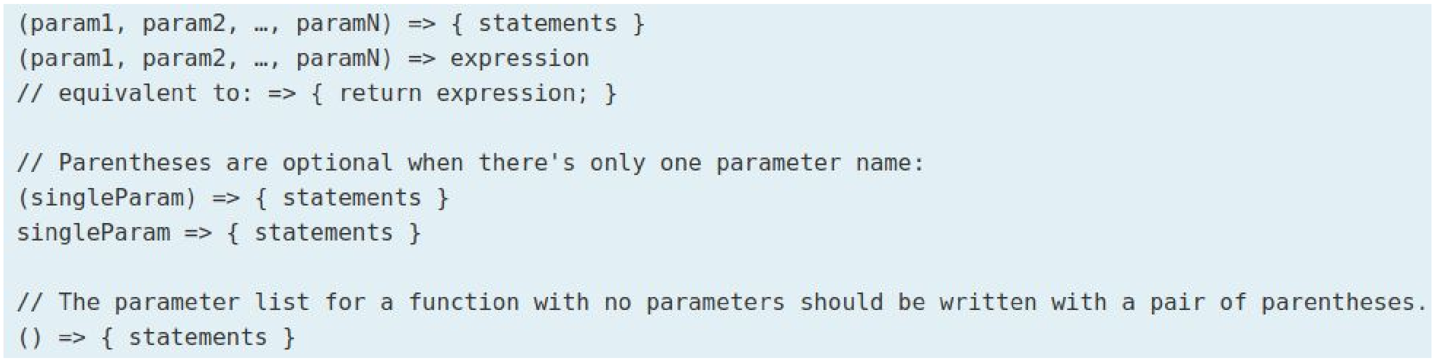
```
var array1 = [1, 4, 9, 16];

// pass a function to map
const map1 = array1.map(x => x * 2);

console.log(map1);
// expected output: Array [2, 8, 18, 32]
```

ARROW FUNCTIONS

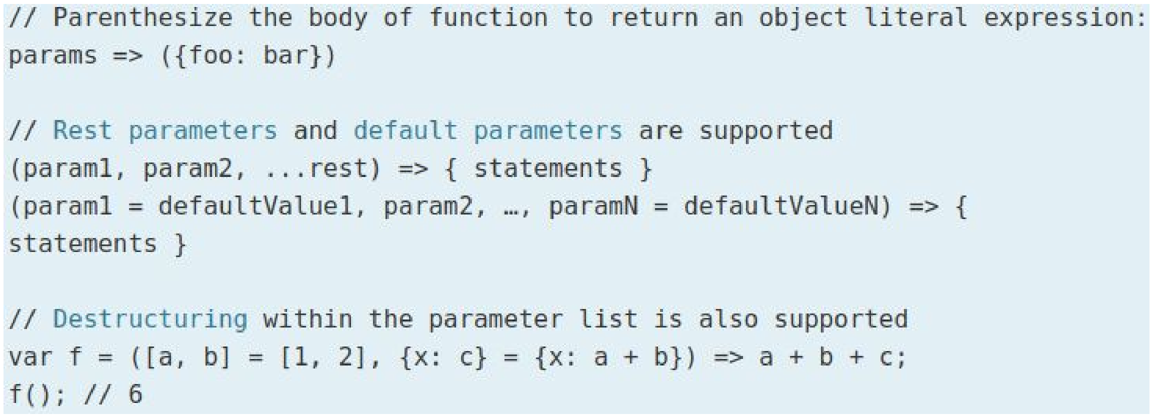
An arrow function expression is a syntactically compact alternative to a regular function expression, although without its own bindings to the this, arguments, super, or new.target keywords. Arrow function expressions are ill suited as methods, and they cannot be used as constructors. Examples:



```
var materials = [
  'Hydrogen',
  'Helium',
  'Lithium',
  'Beryllium'
];

console.log(materials.map(material => material.length));
// expected output: Array [8, 6, 7, 9]

```



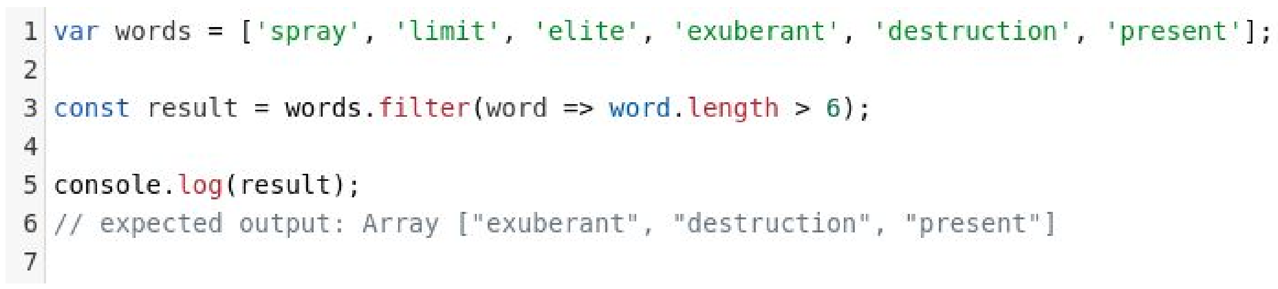
```
(param1, param2, …, paramN) => { statements }
(param1, param2, …, paramN) => expression
// equivalent to: => { return expression; }

// Parentheses are optional when there's only one parameter name:
(singleParam) => { statements }
singleParam => { statements }

// The parameter list for a function with no parameters should be written with a pair of parentheses.
() => { statements }
```



```
// Parenthesize the body of function to return an object literal expression:
params => ({foo: bar})

// Rest parameters and default parameters are supported
(param1, param2, ...rest) => { statements }
(param1 = defaultValue1, param2, …, paramN = defaultValueN) => {
statements }

// Destructuring within the parameter list is also supported
var f = ([a, b] = [1, 2], {x: c} = {x: a + b}) => a + b + c;
f(); // 6
```

FILTER

The filter() method creates a new array with all elements that pass the test implemented by the provided function. Examples:



```
var words = ['spray', 'limit', 'elite', 'exuberant', 'destruction', 'present'];

const result = words.filter(word => word.length > 6);

console.log(result);
// expected output: Array ["exuberant", "destruction", "present"]

```

REDUCE

The reduce() method executes the callback once for each assigned value present in the array, taking four arguments:

- accumulator
- currentValue
- currentIndex
- array

accumulator
The accumulator accumulates the callback's return values. It is the accumulated value previously returned in the last invocation of the callback, or initialValue, if supplied (see below).

currentValue
The current element being processed in the array.

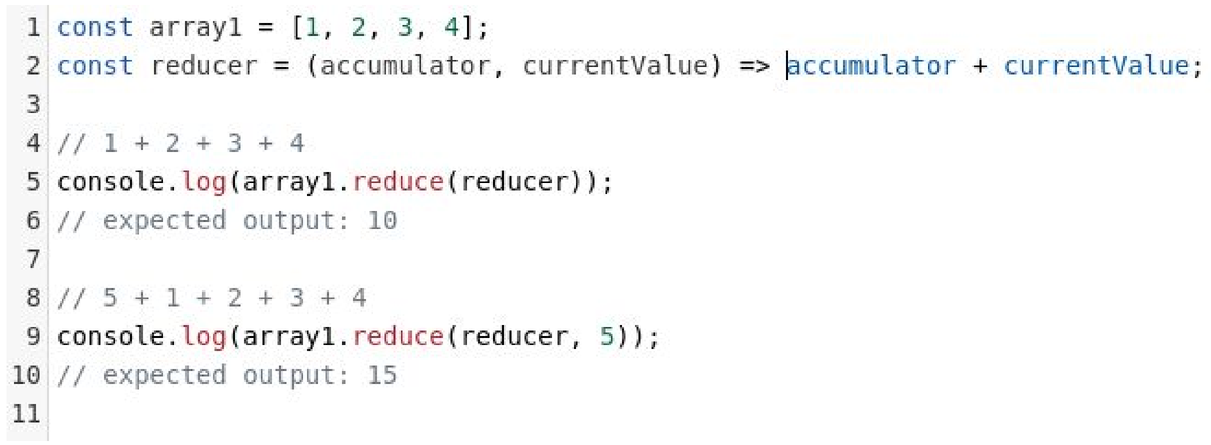
currentIndex Optional

The index of the current element being processed in the array. Starts from index 0 if an initialValue is provided. Otherwise, starts from index 1.

array Optional
The array reduce() was called upon.

The first time the callback is called, accumulator and currentValue can be one of two values. If initialValue is provided in the call to reduce(), then accumulator will be equal to initialValue, and currentValue will be equal to the first value in the array. If no initialValue is provided, then accumulator will be equal to the first value in the array, and currentValue will be equal to the second.

Examples:



```
const array1 = [1, 2, 3, 4];
const reducer = (accumulator, currentValue) => accumulator + currentValue;

// 1 + 2 + 3 + 4
console.log(array1.reduce(reducer));
// expected output: 10

// 5 + 1 + 2 + 3 + 4
console.log(array1.reduce(reducer, 5));
// expected output: 15

```

5. PROBLEMAS PROPUESTOS

1. Display only the tasks names. Use the imperative and functional version (you could use map or filter or reduce for the functional version of your code)

```
let tasks = [
{
  'name'     : 'Buy milk from the shop',
  'duration' : 20,
  'priority' : 1
},
{
  'name'     : 'Clean the house',
  'duration' : 120,
  'priority' : 3
},
{
  'name'     : 'Study JS functions',
  'duration' : 180,
  'priority' : 1
},
];
```

2. For the last exercise, write a program that return only the task with priority = 1. Write the imperative and the functional version (you could use map or filter or reduce for the functional version of your code)

3. For the last exercise, show the total amount of estimated time of all tasks. Write the imperative and the functional version (you could use map or filter or reduce for the functional version of your code)

4. Write a program that show the mean price of type car = ‘suv’. Write the imperative and the functional version (you could use map or filter or reduce for the functional version of your code)

```
const vehicles = [
       { make: 'Honda', model: 'CR-V', type: 'suv', price: 24045 },
      { make: 'Honda', model: 'Accord', type: 'sedan', price: 22455 },
      { make: 'Mazda', model: 'Mazda 6', type: 'sedan', price: 24195 },
      { make: 'Mazda', model: 'CX-9', type: 'suv', price: 31520 },
      { make: 'Toyota', model: '4Runner', type: 'suv', price: 34210 },
      { make: 'Toyota', model: 'Sequoia', type: 'suv', price: 45560 },
      { make: 'Toyota', model: 'Tacoma', type: 'truck', price: 24320 },
      { make: 'Ford', model: 'F-150', type: 'truck', price: 27110 },
      { make: 'Ford', model: 'Fusion', type: 'sedan', price: 22120 },
      { make: 'Ford', model: 'Explorer', type: 'suv', price: 31660 }
    ];
```

5. Write a program that show the total score of force users only. Write the imperative and the functional version (you could use map or filter or reduce for the functional version of your code)

```
var personnel = [
 {
   id: 5,
   name: "Luke Skywalker",
   pilotingScore: 98,
   shootingScore: 56,
   isForceUser: true,
 },
 {
   id: 82,
   name: "Sabine Wren",
   pilotingScore: 73,
   shootingScore: 99,
   isForceUser: false,
 },
 {
   id: 22,
   name: "Zeb Orellios",
   pilotingScore: 20,
   shootingScore: 59,
   isForceUser: false,
 },
 {
   id: 15,
   name: "Ezra Bridger",
   pilotingScore: 43,
   shootingScore: 67,
   isForceUser: true,
 },
 {
   id: 11,
   name: "Caleb Dume",
   pilotingScore: 71,
   shootingScore: 85,
   isForceUser: true,
 },
];
```