**Lección 7: Javascript VI (Callbacks)**

En esta lección cubriremos:

* Callbacks
* Más métodos de Arrays
* Introducción a la programación funcional

**Callbacks**

Un concepto muy importante en Javascript es la capacidad de pasar una función como argumento a otra función. Estas funciones se denominan callbacks. Estas funciones pueden llamarse en cualquier momento y pasar argumentos dentro de la función. Pronto descubriremos por qué las devoluciones de llamada son tan importantes para Javascript. La convención es usar cb como argumento para la variable que se usará de callback.

**function** decirHolaAlUsuario(usuario) {

**return** 'Hola ' **+** usuario **+** '!';

}

**function** decirAdiosAlUsuario(usuario) {

**return** 'Adiós ' **+** usuario **+** '!';

}

**function** crearSaludo(usuario, cb) {

**return** cb(usuario);

}

crearSaludo('Dan', decirHolaAlUsuario); *// 'Hello Dan!'*

crearSaludo('Dan', decirAdiosAlUsuario); *// 'Goodbye Dan!'*

**Más métodos de Arrays**

Ya conocemos y utilizamos métodos de matriz, .push, .pop, .shift, .unshift y .length. Pero hay muchos más métodos disponibles de forma nativa en un array. Los métodos de los que vamos a hablar aquí se denominan “métodos de orden superior”, porque toman los callbacks como argumentos.

**.forEach**

.forEach es un bucle for integrado en cada array. .forEach toma un callback como su único argumento, e itera sobre cada elemento de la matriz y llama al callback en él. El callback puede tomar dos argumentos, el primero es el elemento en sí, el segundo es el índice del elemento (este argumento es opcional).

**const** autos **=** ['Ford', 'Chevrolet', 'Toyota', 'Tesla'];

*// Podemos escribir el callback en los paréntesis como una función anónima*

autos.forEach(**function**(elemento, indice) {

console.log(elemento);

});

*// O podemos crear una instancia de una función para usarla como callback.*

*// Además, no necesitamos usar el argumento de índice, si no lo necesitas, no dudes en omitirlo.*

**function** mostrarNombres(elemento) {

console.log(elemento);

}

*// And call that function in the forEach parentheses*

autos.forEach(mostrarNombres);

**.reduce**

.reduce ejecutará un bucle en nuestra matriz con la intención de reducir cada elemento en un elemento que se devuelve. Como es el primer argumento, acepta un callback que toma dos argumentos, primero un ‘acumulador’ (el resultado del método de reducción hasta ahora), y el segundo es el elemento en el que se encuentra actualmente. El callback debe contener siempre una declaración de devolución (“return”). .reduce también toma un segundo argumento opcional, que sería el acumulador de arranque (“starting accumulator”). Si no se suministra el acumulador de arranque, la reducción comenzará en el primer elemento de la matriz. .reduce siempre devolverá el acumulador cuando termine de recorrer los elementos.

**const** numeros **=** [ 1, 2, 3, 4, 5, 6, 7, 8, 9];

**const** palabras **=** [ 'Hola,', 'mi', 'nombre', 'es', 'Martin'];

*// Podemos escribir la función anónima directamente en los paréntesis de .reduce*

*// Si omitimos el elemento inicial, siempre comenzará en el primer elemento.*

**const** suma **=** numeros.reduce(**function**(acc, elemento){

**return** acc **+** elemento;

});

*// Podemos escribir una función fuera de los parents de .reduce (para usar varias veces más tarde)*

**function** multiplicarDosNumeros(a, b) {

**return** a **\*** b;

}

**const** productos **=** numeros.reduce(multiplicarDosNumeros);

*// .reduce funciona en cualquier tipo de datos.*

*// En este ejemplo configuramos un acumulador de arranque*

**const** frases **=** palabras.reduce(**function**(acc, elemento) {

**return** acc **+** ' ' **+** elemento;

}, 'Frase completa:');

console.log(suma); *// 45*

console.log(productos); *// 362880*

console.log(frases); *// "Frase completa: Hola, mi nombre es Martin"*

**.map**

.map se usa cuando queremos cambiar cada elemento de una matriz de la misma manera. .map toma una devolución de llamada como único argumento. Al igual que el método .forEach, el callback tiene el elemento y el índice de argumentos opcionales. A diferencia de .reduce, .map devolverá toda la matriz.

**const** numeros **=** [2, 3, 4, 5];

**function** multiplicarPorTres(elemento) {

**return** elemento **\*** 3;

}

**const** doble **=** numeros.map(**function**(elemento) {

**return** elemento **\*** 2;

});

**const** triple **=** numeros.map(multiplicarPorTres)

console.log(doble); *// [ 4, 6, 8, 10 ]*

console.log(triple); *// [ 6, 9, 12, 15 ]*

**Abre la carpeta “homework” y completa la tarea descrita en el archivo README**

[Homework](https://github.com/atralice/Curso.Prep.Henry/tree/master/07-JS-VI/homework)

**Recursos adicionales**

* [Understanding Callback Functions and How to Use Them](http://javascriptissexy.com/understand-javascript-callback-functions-and-use-them/)
* [Eloquent Javascript: Higher Order Functions](https://eloquentjavascript.net/05_higher_order.html)
* [MDN: Callback function](https://developer.mozilla.org/en-US/docs/Glossary/Callback_function)
* [MDN: Array methods](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array)