**Repaso de Arrays**

Los arrays son un tipo de dato de JavaScripts, que nos permite guardar múltiples valores en una sola variable. Podemos verlo como una colección.

La forma de crear un **array** es utilizando los corchetes y separar los valores por comas.

let playlist = ['Smells Like Teen Spirit', 'Come As You Are', 'Heart-Shaped Box', 'Lithium'];

En este caso creamos un array que contiene 4 canciones de una playlist.

Para obtener los datos de un array podemos utilizar el índice, que empiezan en 0. Entonces, por ejemplo, podemos obtener el primer elemento con el índice 0.

const playlist = ['Smells Like Teen Spirit', 'Come As You Are', 'Heart-Shaped Box', 'Lithium'];

let primerCancion = playlist[0];

let segundaCancion = playlist[1];

**Recorrer un array**

La mayoría de las veces, los ciclos **for** son utilizados para recorrer/**iterar** sobre cada uno de los elementos que tiene un array.

Usando el acceso mediante el índice, vamos a ver cada elemento.

Para poder hacer este recorrido, vamos a utilizar la propiedad **length** de los arrays.

var villanosDeBatman = ['Joker', 'Bane', "Ra's Al Ghul", 'Scarecrow'];

for (var i = 0; i < villanosDeBatman.length; i++) {

console.log(villanosDeBatman[i]);

}

// Joker

// Bane

// Ra's Al Ghul

// Scarecrow

**.forEach()**

Para iterar un array, también podemos utilizar el método **forEach**. Este método acepta una función como parámetro. La función que le pasamos a este método recibe como parámetro cada uno de los elementos del array.

var villanosDeBatman = ['Joker', 'Bane', "Ra's Al Ghul", 'Scarecrow'];

villanosDeBatman.forEach(function(villain) {

console.log(villain);

});

// Joker

// Bane

// Ra's Al Ghul

// Scarecrow

Incluso, utilizando las arrow functions que vimos en la anterior clase:

var villanosDeBatman = ['Joker', 'Bane', "Ra's Al Ghul", 'Scarecrow'];

villanosDeBatman.forEach(villain => console.log(villain));

// Joker

// Bane

// Ra's Al Ghul

// Scarecrow

En este ejemplo usamos el método **forEach** para mostrar todos los villanos de Batman que tenemos en el array. Le pasamos una función como parámetro. El parámetro que recibe esta función lo llamamos **villain** ya que estamos recorriendo un array de “villanos”, pero le podemos poner cualquier nombre. Siempre es mejor poner un nombre con contexto que explique de forma fácil de que es la colección que estamos iterando y los elementos que estamos utilizando.

Podemos obtener otro parámetro más en la función que se ejecuta por cada elemento que es el índice del elemento:

var playlist = ['Smells Like Teen Spirit', 'Come As You Are', 'Heart-Shaped Box', 'Lithium'];

playlist.forEach((cancion, indice) => {

console.log('indice', indice, cancion);

});

// indice 0 Smells Like Teen Spirit

// indice 1 Come As You Are

// indice 2 Heart-Shaped Box

// indice 3 Lithium

**.map()**

El método **map** es otra forma de iterar sobre cada elemento del array, creando un nuevo array con el resultado de la función que le pasamos como parámetro.

Podemos utilizar este método para cambiar los valores que tenemos en un array.

En la función que le pasamos como parámetro tenemos que retornar el elemento que queremos en el nuevo array.

**El array original queda intacto**.

var numeros = [1, 2, 3, 4];

var triples = numeros.map(function(numero) {

return numero \* 3;

});

console.log(triples) // [ 3, 6, 9, 12 ] todos los numeros multiplicados por 3

console.log(numeros) // [ 1, 2, 3, 4 ] este array quedó igual que antes

El método **map** se puede usar para transformar de cualquier tipo de dato, a cualquier otro tipo de dato. Si tenemos un array de objetos, podemos transformarlo en un array de strings.

const personas = [

{ nombre: 'Ada Lovelace', edad: 27 },

{ nombre: 'Grace Hopper', edad: 32 },

{ nombre: 'Hedy Lamarr', edad: 37 }

];

const nombres = personas.map(function (persona) {

return persona.nombre;

})

console.log(nombres);

// [ 'Ada Lovelace', 'Grace Hopper', 'Hedy Lamarr' ]

Incluso, utilizando arrow functions, podemos conseguir una forma mucho mejor de escribir lo mismo.

const personas = [

{ nombre: 'Ada Lovelace', edad: 27 },

{ nombre: 'Grace Hopper', edad: 32 },

{ nombre: 'Hedy Lamarr', edad: 37 }

];

const nombres = personas.map(persona => persona.nombre)

console.log(nombres);

// [ 'Ada Lovelace', 'Grace Hopper', 'Hedy Lamarr' ]

**.filter()**

El método **filter** retorna un nuevo array utilizando un filtro. A este método le vamos a pasar una función que retorne verdadero (si queremos añadir el elemento al nuevo array) o falso (si no queremos que el elemento se incluya).

var notas = [1, 2, 3, 4, 10, 5, 6, 3, 9];

var aprobadas = notas.filter(function(nota) {

return nota >= 6;

});

console.log(aprobadas); // [ 10, 6, 9 ]

console.log(notas); // [ 1, 2, 3, 4, 10, 5, 6, 3, 9 ]

Al igual que **map**, podemos utilizar el método **filter** para filtrar un array de cualquier tipo de dato. Por ejemplo, podríamos filtrar un array de strings u objetos.

const personas = [

{ nombre: 'Ada Lovelace', edad: 27 },

{ nombre: 'Grace Hopper', edad: 32 },

{ nombre: 'Hedy Lamarr', edad: 37 }

];

const mayoresDe30 = personas.filter(persona => persona.edad > 30)

console.log(mayoresDe30);

// mayoresDe30 = [

// { nombre: 'Grace Hopper', edad: 32 },

// { nombre: 'Hedy Lamarr', edad: 37 }

// ]

**.reduce()**

El método **reduce** nos permite recorrer el array y obtener un resultado reducido en base a cada elemento del array.

Acepta dos parámetros:

* Como primer parámetro una función con dos parámetros: un acumulador y el elemento del array. El acumulador puede ser de cualquier tipo (un objeto, un array, un número, un string, etc).
* Como segundo parámetro, opcional, es el valor inicial del acumulador.

Podemos, por ejemplo, utilizar **reduce** para conseguir la suma total de varios números.

var notas = [1, 2, 3, 4, 10, 5];

var sumaDeNotas = notas.reduce(function(total, nota) {

return total + nota;

});

console.log(sumaDeNotas); // 25 resultado final de sumar todas las notas

Podemos ver una animación de como funciona el método **reduce** en el siguiente link: <http://reduce.surge.sh/>

En otro ejemplo, supongamos que tenemos un array de personas con los años de experiencia programando y queremos saber el total de años de experiencia entre todos.

const personas = [

{ nombre: 'Ada Lovelace', aniosExperiencia: 4 },

{ nombre: 'Grace Hopper', aniosExperiencia: 3 },

{ nombre: 'Hedy Lamarr', aniosExperiencia: 7 }

];

const totalAniosExperiencia = personas.reduce(function (totalAnios, persona) {

return totalAnios + persona.aniosExperiencia;

}, 0);

console.log(totalAniosExperiencia); // 14

Vean como en el ejemplo, cuando utilicé el método **reduce**, inicié el acumulador con 0. Veamos como nos quedaría el código si usamos arrow functions:

const totalAniosExperiencia = personas.reduce((totalAnios, persona) => totalAnios + persona.aniosExperiencia, 0);

Y supongamos que ahora queremos conocer la persona con más experiencia. Para eso podemos utilizar también el método **reduce**.

const masExperimentada = personas.reduce(function (mayor, persona) {

return mayor.aniosExperiencia > persona.aniosExperiencia ? mayor : persona

});

console.log(masExperimentada);

// { nombre: 'Hedy Lamarr', aniosExperiencia: 7 }

En este ejemplo, no estamos pasando ningún valor inicial al acumulador. Cuando hacemos esto, lo que le estamos indicando es que tome el primer valor del array como valor inicial.

**Encadenando .map(), .filter() y .reduce()**

Una de las cosas más potentes que tienen estas funciones, es que podemos “encadenarlas” para llegar a un objetivo final.

Por ejemplo, supongamos que tenemos una lista de gastos (tanto en pesos argentinos como dólares).

const gastos = [

{ detalle: 'Dolores nonumy magna.', moneda: 'ARS', monto: 405 },

{ detalle: 'Takimata clita ipsum.', moneda: 'USD', monto: 5 },

{ detalle: 'Sadipscing duo et.', moneda: 'ARS', monto: 357 },

{ detalle: 'Aliquyam dolor gubergren.', moneda: 'USD', monto: 13 },

{ detalle: 'Stet ea sadipscing.', moneda: 'ARS', monto: 109 }

];

Y queremos obtener el monto total de los gastos que hicimos en dólares, y mostrar el resultado final en pesos.

Una posible solución, utilizando un **for**, sería:

let total = 0;

for (let i = 0; i < gastos.length; i++) {

if (gastos[i].moneda === 'USD') {

total += gastos[i].monto \* 42

}

}

console.log(total); // 756

Si bien esta solución cumple con el enunciado, otra forma de resolverlo utilizando **map, filter** y **reduce**, sería:

let total = gastos

.filter(gasto => gasto.moneda === 'USD')

.map(gasto => gasto.monto \* 42)

.reduce((total, monto) => total + monto);

console.log(total); // 756