



# Arrow functions & Array Methods

# Índice

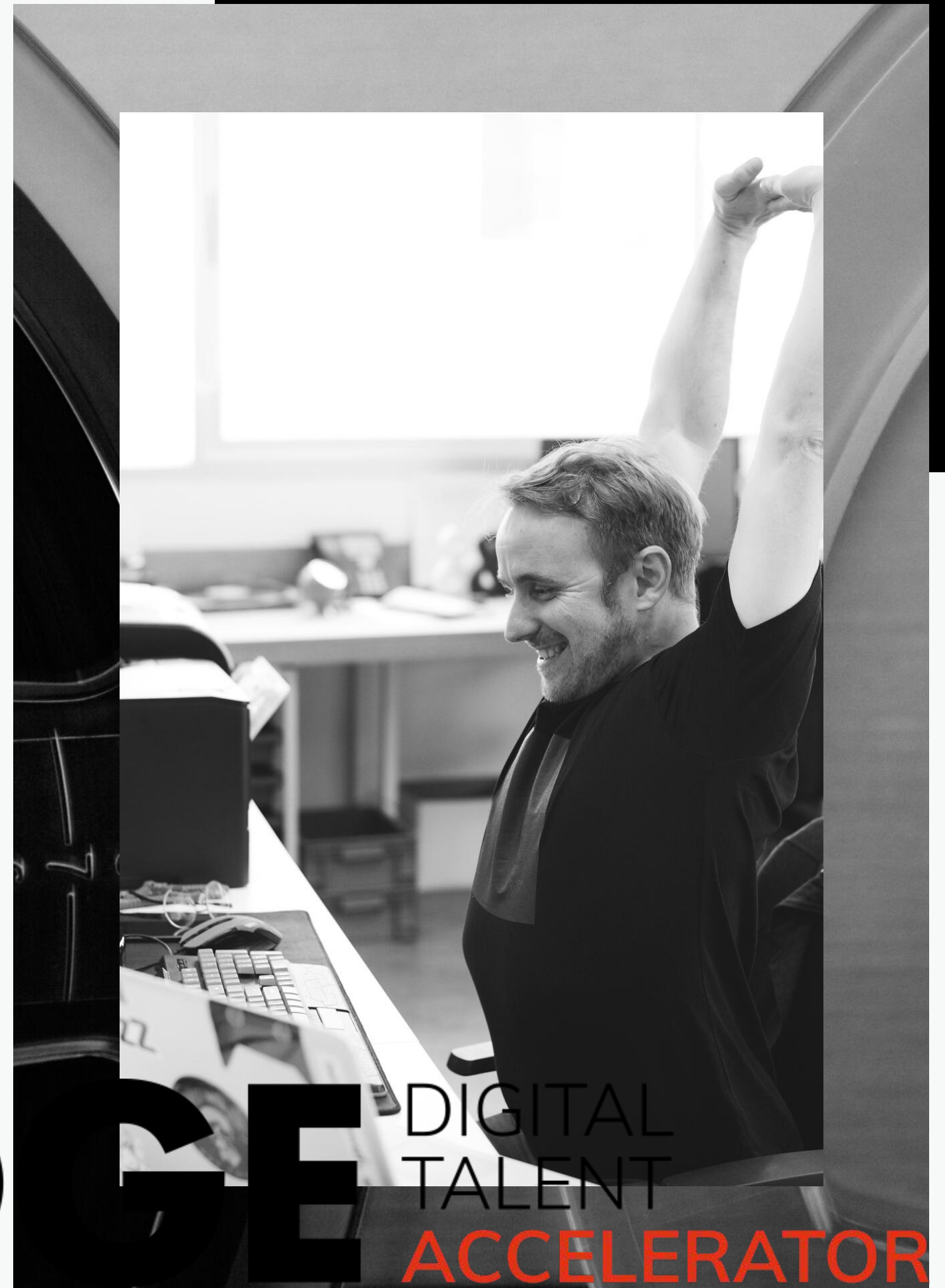
¿Que es ES6?

Arrow functions (funciones flecha)

Callbacks

Foreach, Map, Reduce, Filter

**THE**  **BRIDGE** **DIGITAL  
TALENT  
ACCELERATOR**





¿Qué es ES6?

# ES6

Cuando hablamos de **ECMAScript**, nos referimos al **estándar que ya va por la versión ES6 y determina cómo emplear el lenguaje Javascript**. Tiene algunos cambios significativos en su sintaxis que **permite escribir código de forma más rápida y limpia**.



¿Qué es una función flecha  
(arrow function)?

# Funciones flecha

Las **funciones flecha** son una forma de escribir funciones en **ES6**. Ahora se usará la palabra reservada `const` y seguirá de:

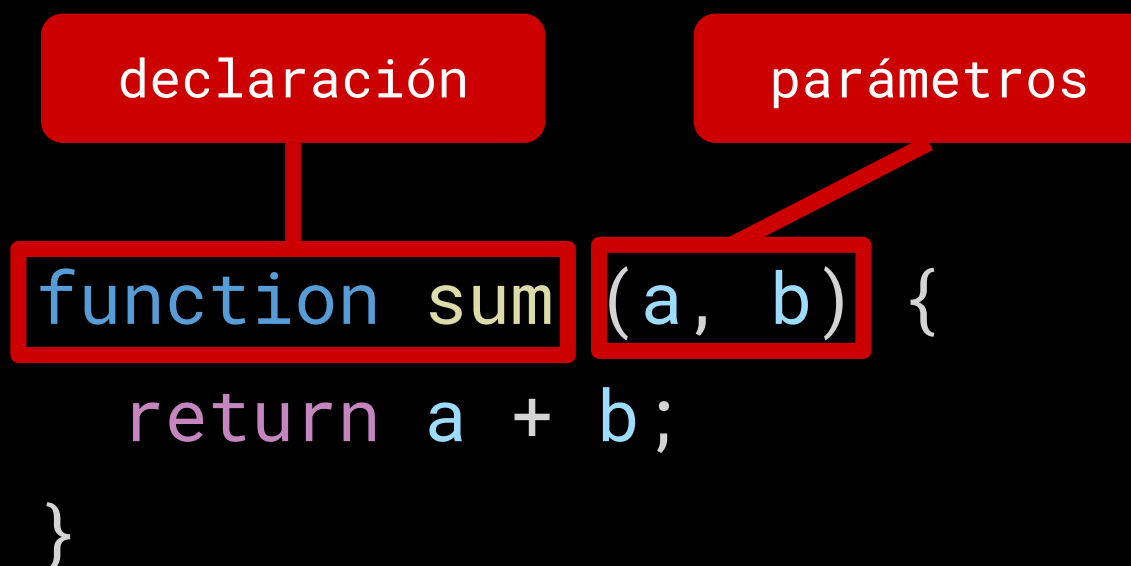


Diagram illustrating a regular function declaration in ES6. The code is: `function sum(a, b) {  
 return a + b;  
}`. Red boxes highlight `function sum` and `(a, b)`. Red labels with arrows point to these boxes: `declaración` points to `function sum`, and `parámetros` points to `(a, b)`.

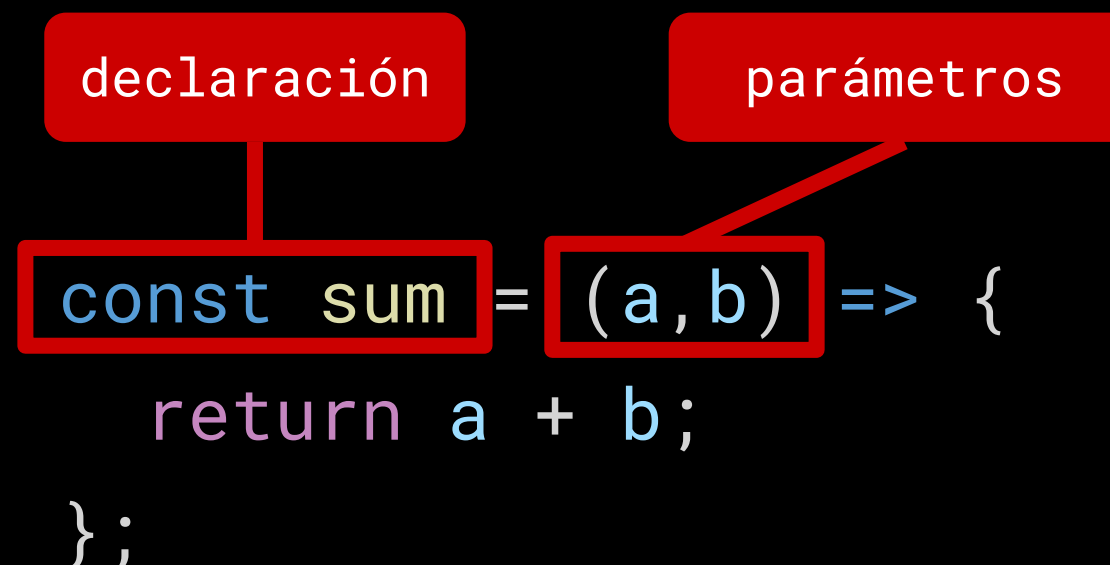


Diagram illustrating a function declaration using `const` and arrow syntax in ES6. The code is: `const sum = (a, b) => {  
 return a + b;  
};`. Red boxes highlight `const sum` and `(a, b)`. Red labels with arrows point to these boxes: `declaración` points to `const sum`, and `parámetros` points to `(a, b)`.



## Funciones flecha en línea

Con la función flecha se puede omitir la estructura de bloque y escribir todo en la misma línea.

```
const sum = (a, b) => a + b;
```



## Funciones flecha con un solo parámetro

Con una función flecha que solo tenga un parámetro podemos escribirla de la siguiente forma:

```
const test = a => a;
```





¿Qué es un callback?

# Callback

Una función de callback es una **función que se pasa a otra función como un argumento**, que luego se invoca dentro de la función externa para completar algún tipo de rutina o acción.

# Callback

Un **callback** (llamada hacia atrás) es pasar una **función B (test2)** por parámetro a una **función A(test1)**, de modo que la **función A puede ejecutar esa función B de forma genérica desde su código**, y nosotros podemos definirlas desde fuera de dicha función:

```
function test2() {  
    console.log("Función test2 ejecutada.");  
}  
  
function test1(callback) {  
    callback(); // función test2  
}  
  
test1(test2);
```

“

Foreach

# Foreach

El bucle foreach en js es un bucle que simplemente **recorre los objetos de un array**.

# Foreach

Lo que hacemos con un **foreach** también lo podemos hacer con un bucle **for** normal.

```
for (let i = 0; i < fruits.length; i++) {  
  console.log(fruits[i]);  
}
```

```
fruits.forEach(fruit => {  
  console.log(fruit)  
});
```

# Foreach

Al igual que otros iteradores de arrays tales como **map** y **filter**, la función callback puede recibir tres parámetros.

- **El elemento actual:** Este es el elemento del arreglo por el cual se está haciendo la iteración.
- **Su índice:** Este es el índice de la posición que tiene el elemento dentro del arreglo.
- **El array objetivo:** Este es el array que estamos iterando

```
const canciones = ["Hang onto yourself", "Go your own way", "Modern Love"];

canciones.forEach((cancion, indice, array) => {
  console.log('Elemento actual', cancion, 'índice', indice, 'array', array);
});
```

“

Map



# Map

A veces tenemos una **array de objetos que deseamos modificar/agregar propiedades de cada objeto**, otras veces podemos tener una array de strings que debemos convertir todas en minúsculas.

En realidad, puede haber innumerables situaciones en las que el map sea su salvador y es muy fácil de usar.



# Map

Esta es la forma más básica de llamar un map:

```
const myArray = [1,2,3,4]

const myArrayTimesTwo = myArray.map((value)=>{
  return value * 2
})
```

## Map

Dado este array de objetos,  
creamos un array de strings  
utilizando un **for of**:

```
const songs = [  
  { name: "Curl of the Burl", artist: "Mastodon" },  
  { name: "Oblivion", artist: "Mastodon" },  
  { name: "Flying Whales", artist: "Gojira" },  
  { name: "Euralio y sus cabras", artist: "Euralio" },  
];  
  
const allSongNames = []  
  
for (const song of songs) {  
  allSongNames.push(song.name)  
}
```

## Map

Dado este array de objetos,  
creamos un array de strings  
como el anterior ejemplo  
utilizando map:

```
const songs = [  
  {name:"Curl of the Burl", artist:"Mastodon"},  
  {name:"Oblivion", artist:"Mastodon"},  
  {name:"Flying Whales", artist:"Gojira"},  
  {name:"Euralio y sus cabras", artist:"Euralio"}  
]  
  
const allSongNames = songs.map(song =>{  
  return song.name  
})
```

# Filter

La función filter permite **filtrar los elementos de un array y generar uno nuevo** con todos los elementos que cumplen una determinada condición.

# Filter

Filter acepta como parámetro una función (callback) la cual se ejecutará por cada elemento del array y deberá de retornar true para indicar que el elemento actual debe de ser incluido en el nuevo array o false en caso contrario. Veamos un ejemplo:

```
let numeros = [1,2,3,4,5,6,7,8,9,10]
```

```
let numerosFiltrados = numeros.filter(numero => numero > 5)
```

condición

# Filter

Veamos el siguiente array,  
queremos crear un array  
que contenga las palabras  
que tengan más de 6  
caracteres:

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

let result =[]
for (const word of words) {
  if(word.length > 6){
    result.push(word)
  }
}
```



## Filter

Lo mismo que hemos hecho con un **for of** y un **if** lo podemos hacer en una sola línea con **filter**:

```
const words = ['spray', 'limit', 'elite', 'exuberant', 'destruction', 'present'];  
  
const result = words.filter(word => word.length > 6);
```



# Reduce

Por defecto esta función se encarga de “reducir” un conjunto de elementos a un único resultado:

# Reduce

- **resultado** es una variable que recogerá el valor después de haber aplicado la función reduce().
- reduce recibe **dos parámetros**: una **función callback** y el **valor inicial**.
- **callback()** recibe **dos parámetros**: el **valor anterior** y el **valor actual**. Lo que va a hacer reduce es recorrer todos los elementos de la lista y aplicar la función callback. **El valor anterior será el resultado devuelto por la ejecución anterior de callback(). Si es la primera vez, será el valorInicial**. Para el valor actual, asignará el elemento que esté evaluando en ese momento.
- **valorInicial** es el valor que reduce le pasará a callback como valorAnterior la primera vez que lo ejecute. Este valor es opcional y **si no le pasas nada, se utilizará el primer elemento de la lista** y se saltará al segundo.

```
const resultado = lista.reduce(function callback(valorAnterior, valorActual) {  
    return; /* resultado de la función callback */  
}, valorInicial);
```

## Reduce

Veamos un ejemplo simple, donde sumamos todos los valores de un array de números utilizando un for:

```
const numbers = [10, 5, 7];  
let sum = 0;  
  
for (let i = 0; i < numbers.length; i++) {  
    sum += numbers[i];  
}
```



# Reduce

Veamos el mismo ejemplo anterior utilizando reduce:

```
const numbers = [10, 5, 7]
```

```
const sum = numbers.reduce((a, b) => a + b)
```