Arrow functions

DigitalHouse >
Coding School

“

Las funciones son uno de los elementos que más vamos a usar a la hora de programar en JavaScript.

Las **funciones arrow** nos permiten escribirlas con una **sintaxis** más **compacta**.

”

Índice

1 Declaración y estructura

Estructura básica

Pensemos en una función simple que podríamos programar de la manera habitual: una suma de dos números.

```
function sumar (a, b) { return a + b }
```

Ahora veamos la versión reducida de esa misma función, al transformarla en una función arrow.

```
let sumar = (a, b) => a + b;
```

Nombre de una función arrow

Las funciones arrow **son siempre anónimas**. Es decir, que no tienen nombre como las funciones normales.

```
{}  (a, b) => a + b;
```

Si queremos nombrarlas, es necesario escribirlas como una función expresada. Es decir, asignarla como valor de una variable.

```
{}  let sumar = (a, b) => a + b;
```

De ahora en más podremos llamar a nuestra función por su nuevo nombre.

Parámetros de una función arrow

Usamos paréntesis para indicar los **parámetros**. Si nuestra función no recibe parámetros, debemos escribirlos igual.

```
let sumar = (a, b) => a + b;
```

Una particularidad de este tipo de funciones es que si recibe un único parámetro, podemos prescindir de los paréntesis.

```
let doble = a => a * 2;
```

La flecha de una función arrow

La usamos para indicarle a JavaScript que vamos a escribir una función (reemplaza a la palabra reservada `function`).

```
let sumar = (a, b) => a + b;
```

Lo que está a la izquierda de la fecha será la entrada de la función (los parámetros) y lo que está a la derecha, la lógica (y el posible retorno).

Lo que está a la izquierda de la fecha será la entrada de la función (los parámetros) y lo que está a la derecha, la salida (el retorno)

Las funciones arrow reciben su nombre por el operador `=>`. Si lo miramos con un poco de imaginación, se parece a una flecha.

En inglés suele llamarse fat arrow (flecha gorda) para diferenciarlo de la flecha simple `->`.

Cuerpo de una función arrow

Como ya vimos, si la función tiene una sola línea de código, y esta misma es la que hay que **retornar**, no hacen falta las llaves ni la palabra reservada return.

```
{}  let sumar = (a, b) => a + b;
```

De lo contrario, vamos a necesitar utilizar ambas. Eso normalmente pasa cuando tenemos más de una línea de código en nuestra función.

```
{}  let esMultiplo = (a, b) => {
        let resto = a % b; // Obtenemos el resto de la div.
        return resto == 0; // Si el resto es 0, es múltiplo
    };
```

2 | Ejemplos

{código}

```
let saludo = () => 'Hola Mundo!';

let dobleDe = numero => numero * 2;

let suma = (a, b) => a + b;

let horaActual = () => {
    let fecha = new Date();
    return fecha.getHours() + ':' +
    fecha.getMinutes();
}
```

Función arrow **sin parámetros**.

Requiere de los paréntesis para iniciarse.

Al tener una sola línea de código, y que esta misma sea la que queremos retornar, el return queda implícito.

{código}

```
let saludo = () => 'Hola Mundo!';

let dobleDe = numero => numero * 2;

let suma = (a, b) => a + b;

let horaActual = () => {
    let fecha = new Date();
    return fecha.getHours() + ':' +
    fecha.getMinutes();
}
```

Función arrow con un **único parámetro** (no necesitamos los paréntesis para indicarlo) y con un return implícito.

{código}

```
let saludo = () => 'Hola Mundo!';

let dobleDe = numero => numero * 2;

let suma = (a, b) => a + b;

let horaActual = () => {
    let fecha = new Date();
    return fecha.getHours() + ':' +
    fecha.getMinutes();
}
```

Función arrow con **dos parámetros**.

Necesita de los paréntesis y tiene un return implícito.

{código}

```
let saludo = () => 'Hola Mundo!';

let dobleDe = numero => numero * 2;

let suma = (a, b) => a + b;
```

```
let horaActual = () => {
    let fecha = new Date();
    return fecha.getHours() + ':' +
    fecha.getMinutes();
}
```

Función arrow **sin parámetros** y con un **return explícito**.

En este caso hacemos uso de las llaves y del return ya que la lógica de esta función se desarrolla en más de una línea de código.


DigitalHouse>
Coding School