JAVASCRIPT

Funciones

¿Qué es una función?

Las **funciones** nos permiten encapsular comportamientos para **ser reutilizados**.

Luego, estas se ejecutarán cada vez que hayan sido **invocadas**.

Función Declarada

En Javascript podemos declarar una función de diversas formas. La primera que conoceremos es la **declarada**, que se hace de la siguiente manera:

```
function sumar() {
    // Código a ser ejecutado
}
```

Esta es la manera más tradicional y consta de comenzar con la palabra reservada **function**, seguido del nombre de nuestra función con un par de paréntesis y un **bloque de llaves** que **encerrarán el código** que deba encapsular dicha función.

Función Expresada

Otra forma consiste en asignar la declaración de la función a una variable tradicional. Este tipo de **función** se la conoce como anónima ya que al crearla no posee un nombre sino que toma el nombre de la variable a la cual fue asignada.

```
const sumar = function () {
    // Código a ser ejecutado
}
```

La **principal diferencia** con una *función declarada es que estas pueden ser utilizadas incluso antes de su declaración*, mientras que las expresadas contienen el comportamiento de **hoisting**, por lo que no pueden ser llamadas antes de la declaración de la variable.

Uso de una función

Cuando queremos utilizar nuestras funciones debemos **invocarlas** a través de su **nombre** seguidas de un par de paréntesis.

```
function saludar() {
    console.log('Hola mundo!');
}

saludar(); // Hola mundo!
```

Return

Nuestras funciones se pueden resolver de **2 maneras** diferentes.

La primera es ejecutar una serie de **instrucciones que no presenten un resultado específico**, como podría ser eliminar una etiqueta de nuestro HTML.

La segunda y más común es *devolver o retornar el resultado de lo que suceda dentro de la función*, como por ejemplo **la suma de 2 valores**, con el fin de utilizarlo para algo más.

Para ello utilizamos la palabra reservada **return**, que elevará lo que devuelva la función para poder ser capturado desde un **scope superior**.

```
function sumar() {
    let resultado = 33 + 18;

    return resultado;
}

// Guarda el valor 51 en la variable suma
let suma = sumar();

console.log(suma + 7); // 58
```

*todo el código que escribamos dentro de la función luego de return, no será leído o ejecutado por el programa.

Parámetros y argumentos

Nos permiten crear funciones reutilizables, declarando “variables” dentro de los paréntesis de nuestras funciones. A estas variables se las conoce como parámetros y funcionan como **comodines**.

Tomando el ejemplo anterior pero utilizando parámetros, nos quedaría algo así:

```
function sumar(a, b) {
    let resultado = a + b;

    return resultado;
}

// Guarda el valor 51 en la variable suma
let suma = sumar(33,18);
```

El resultado es el mismo, pero nos permite reutilizar la función con otros casos:

```
let resultado1 = sumar(33,18); // 51
let resultado2 = sumar(7, 15); // 22
```

*los valores pasados dentro de los paréntesis al momento de invocar la función, se los conoce como **argumentos**.

Arrow Functions

Desde ES6 contamos con las **arrow functions**, esta sintaxis puede resultar mucho más acotada dependiendo como se use.

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

Esta forma es **muy parecida a una función expresada** solo que no usamos la palabra function y en **lugar de las llaves colocamos una flecha**.

En esa flecha se encuentra de forma implícita la palabra return, por lo cual no debemos colocarla. De esta manera **nuestra función queda en una sola línea de código**.

Arrow Functions

Cuando necesitemos más de una línea seguiremos **utilizando** el bloque de llaves tradicionales luego de la flecha y la palabra return en caso que deseemos retornar un resultado.

```
const sumar = (a,b) => {
    let resultado = a + b;
    console.log('El valor retornado es' + resultado);

    return resultado;
};
```

Callbacks

Se dan cuando pasamos una función como parámetro de otra función.

En **funciones sincrónicas** estas funciones se ejecutan inmediatamente al ejecutar la función principal.

En **funciones asincrónicas** el callback es la forma que tenemos para ejecutar una función una vez terminado un proceso dependiente anterior.

Los callbacks sientan las bases para el manejo del asincronismo que aprenderemos más adelante.

Callbacks Sincrónicos

Tomando el último ejemplo, **usábamos un console.log()** para imprimir el resultado en consola antes de retornarlo, pero... ¿Qué sucede en algunos casos lo queremos imprimir por consola y en otros mediante una alerta? ¿Necesitaríamos 2 funciones casi idénticas?

```
const sumarConsole = (a, b) => {
    let resultado = a + b;
    console.log('El valor retornado es: ', resultado);
}

const sumarAlert = (a, b) => {
    let resultado = a + b;
    alert('El valor retornado es: ', resultado);
}
```

Callbacks Sincrónicos

Para evitar duplicar nuestro código utilizaremos un **callback**, es decir, pasaremos una función como parámetro para ser utilizada dentro de mi otra función.

```
const sumar = (a, b, callback) => {
    let resultado = a + b;

    callback(resultado);
};

sumar(10, 7, function (suma) {
    console.log('El valor retornado es: ', suma);
});

sumar(8, 5, function (suma) {
    alert('El valor retornado es: ', suma);
});
```