7. Funciones

Las **funciones** son uno de los pilares fundamentales de JavaScript. Son bloques de código reutilizables que permiten encapsular lógica específica y ejecutarla cuando sea necesario. En este capítulo, exploraremos qué son las funciones, cómo declararlas, los diferentes tipos de funciones disponibles en JavaScript y cómo trabajar con parámetros, argumentos y valores de retorno.

¿Qué Son las Funciones?

Una función es un conjunto de instrucciones agrupadas bajo un nombre que puede ser invocado (llamado) en cualquier momento durante la ejecución del programa. Las funciones permiten organizar el código de manera modular, lo que facilita su mantenimiento, reutilización y comprensión.

Beneficios de Usar Funciones

- **Reutilización**: Evita repetir código al encapsular lógica común.
- **Modularidad**: Divide el código en bloques más pequeños y manejables.
- **Facilidad de Mantenimiento**: Cambios en una función afectan solo a esa parte del código.
- **Encapsulamiento**: Oculta detalles de implementación detrás de una interfaz clara.

Declaración de Funciones

En JavaScript, hay varias formas de declarar funciones. A continuación, exploraremos las más comunes:

1. Funciones Declarativas

Son funciones definidas usando la palabra clave `function`, seguida del nombre de la función.

```
function saludar(nombre) {
  console.log(`¡Hola, ${nombre}!`);
}

saludar("Ana"); // ¡Hola, Ana!
```

Características:

- Pueden ser llamadas antes de su declaración debido al **hoisting**.
- Tienen un nombre asociado, lo que facilita su identificación.

---

2. Funciones Expresivas (Expresiones de Función)

Las funciones también pueden asignarse a variables. Estas se conocen como funciones expresivas o anónimas.

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

console.log(sumar(5, 3)); // 8
```

Características:

- No están sujetas al hoisting (deben declararse antes de usarse).
- Son útiles para pasar funciones como argumentos a otras funciones.

---

3. Funciones Flecha (Arrow Functions)

Introducidas en ES6, las funciones flecha ofrecen una sintaxis más concisa y manejan de manera diferente el valor de `this`.

```
const multiplicar = (a, b) => a * b;

console.log(multiplicar(4, 5)); // 20
```

Sintaxis Compacta:

- Si la función tiene una sola expresión, se puede omitir el cuerpo `{}` y la palabra `return`.
- Para funciones sin parámetros, se usan paréntesis vacíos `()`.

```
const saludar = () => console.log("¡Hola!");
saludar(); // ¡Hola!
```

Diferencia en `this`:

- Las funciones flecha no tienen su propio `this`; heredan el `this` del contexto donde fueron definidas.

Parámetros y Argumentos

Los **parámetros** son variables que se definen en la declaración de una función, mientras que los **argumentos** son los valores que se pasan al llamar la función.

Ejemplo:

```
function calcularPromedio(a, b, c) {
  return (a + b + c) / 3;
}

let promedio = calcularPromedio(10, 20, 30); // 10, 20, 30 son argumentos
console.log(promedio); // 20
```

Parámetros por Defecto

Desde ES6, puedes asignar valores predeterminados a los parámetros en caso de que no se proporcionen argumentos.

```
function saludar(nombre = "Invitado") {
  console.log(`¡Hola, ${nombre}!`);
}

saludar(); // ¡Hola, Invitado!
saludar("Juan"); // ¡Hola, Juan!
```

Parámetros Rest

El operador `...` permite recibir un número indefinido de argumentos como un array.

```
function sumar(...numeros) {
  return numeros.reduce((total, num) => total + num, 0);
}

console.log(sumar(1, 2, 3, 4)); // 10
```

Retorno de Valores

Una función puede devolver un valor utilizando la palabra clave `return`. Si no se especifica un valor de retorno, la función devuelve `undefined`.

Ejemplo:

```
function cuadrado(numero) {
  return numero * numero;
}

console.log(cuadrado(5)); // 25
```

> Nota: Una vez que se ejecuta return, la función termina inmediatamente, incluso si hay código después.

Ámbito de las Variables en Funciones

El **ámbito** define dónde están disponibles las variables dentro de una función.

1. **Variables Locales**:
   - Se declaran dentro de una función.
   - Solo son accesibles dentro de esa función.

```
function ejemplo() {
    let mensaje = "Soy local";
    console.log(mensaje);
}

ejemplo(); // Soy local
console.log(mensaje); // ReferenceError: mensaje is not defined
```

1. **Variables Globales**:
    - Se declaran fuera de cualquier función.
    - Son accesibles desde cualquier parte del código.

```
let mensajeGlobal = "Soy global";

function ejemplo() {
    console.log(mensajeGlobal);
}

ejemplo(); // Soy global
```

Funciones Anónimas y Autoinvocadas

Funciones Anónimas

Son funciones sin nombre, generalmente utilizadas como callbacks o asignadas a variables.

```
setTimeout(function() {
    console.log("Esta función se ejecuta después de 1 segundo.");
}, 1000);
```

Funciones Autoinvocadas (IIFE)

Son funciones que se ejecutan inmediatamente después de ser definidas.

```
(function() {console.log("Esta función se ejecuta automáticamente.");})();
```

Callbacks

Un **callback** es una función que se pasa como argumento a otra función y se ejecuta después de que esta última termine su tarea.

```
function procesarDatos(datos, retrollamada) {
    console.log("Procesando datos...");
    retrollamada(datos);
}
```

```
const funcion_a_enviar1 = function(entrada) {
      let entrada_mayus = entrada.toUpperCase()
      console.log(entrada_mayus);
};


const funcion_a_enviar2 = function(entrada) {
      let entrada_minus = entrada.toLowerCase()
      console.log(entrada_minus);
};

procesarDatos("Hola", funcion_a_enviar1);
procesarDatos("Hola", funcion_a_enviar2);
// Salida:
// Procesando datos...
// Callback ejecutado: Hola
```