1. ¿Qué es una función?

Una función es un bloque de código diseñado para realizar una tarea específica. Se puede reutilizar tantas veces como sea necesario, lo que ayuda a mantener el código modular y limpio.

---

2. Sintaxis básica de una función

Hay varias formas de declarar una función en JavaScript:

2.1 Declaración de función

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

- **`function`**: Palabra clave que define la función.
- **`saludar`**: Nombre de la función.
- **`()`**: Paréntesis donde se pueden definir parámetros.
- **`{}`**: Cuerpo de la función, donde se escribe el código que ejecutará.

Uso:

```
saludar(); // Output: Hola, mundo!
```

2.2 Función con parámetros

Las funciones pueden aceptar parámetros para hacerlas más dinámicas.

```
function saludar(nombre) {
    console.log("Hola, " + nombre + "!");
}
```

Uso:

```
saludar("Juan"); // Output: Hola, Juan!
saludar("María"); // Output: Hola, María!
```

2.3 Función con valor de retorno

Una función puede devolver un valor usando `return`.

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

Uso:

```
let resultado = sumar(5, 3);
console.log(resultado); // Output: 8
```

2.4 Función anónima

Una función sin nombre se denomina **anónima**. Suelen asignarse a variables.

```
const multiplicar = function(a, b) {
    return a * b;
};
```

Uso:

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

2.5 Función flecha (Arrow Function)

Introducidas en ES6, son una forma más corta de escribir funciones.

```
const dividir = (a, b) => {
    return a / b;
};
```

Si el cuerpo de la función tiene una sola línea, se puede simplificar aún más:

```
const dividir = (a, b) => a / b;
```

Uso:

```
console.log(dividir(10, 2)); // Output: 5
```

---

3. Scope (Alcance) de las funciones

El **scope** define dónde puedes acceder a variables dentro del programa. Hay dos tipos principales de alcance en funciones:

- **Local**: Variables definidas dentro de una función sólo son accesibles en esa función.

```
function ejemplo() {
    let mensaje = "Esto es local";
    console.log(mensaje); // Output: Esto es local
}
console.log(mensaje); // Error: mensaje no está definido
```

- **Global**: Variables definidas fuera de cualquier función son accesibles en todo el programa.

```
let globalVar = "Esto es global";
function ejemplo() {
    console.log(globalVar); // Output: Esto es global
}
ejemplo();
```

---

4. Funciones como valores

En JavaScript, las funciones son "ciudadanos de primera clase", lo que significa que se pueden:

- Asignar a variables.
- Pasar como argumentos.
- Devolver desde otras funciones.

```
function operar(fn, a, b) {
    return fn(a, b);
}

const suma = (x, y) => x + y;
const resta = (x, y) => x - y;
```

```
console.log(operar(suma, 5, 3)); // Output: 8
console.log(operar(resta, 5, 3)); // Output: 2
```

---

5. 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 procesarUsuario(nombre, callback) {
    console.log("Procesando usuario: " + nombre);
    callback();
}

procesarUsuario("Carlos", () => {
    console.log("Tarea completada!");
});
```

---

6. Funciones asíncronas

En JavaScript, las funciones pueden manejar operaciones asíncronas usando **promesas** o **async/await**.

Promesas:

```
function obtenerDatos() {
    return new Promise((resolve, reject) => {
        setTimeout(() => resolve("Datos recibidos"), 2000);
    });
}

obtenerDatos().then((mensaje) => console.log(mensaje));
```

Async/Await:

```
async function mostrarDatos() {
    const datos = await obtenerDatos();
    console.log(datos);
}

mostrarDatos();
```