Guía de Trabajo: Funciones y Alcance en JavaScript

Objetivo

El objetivo principal de aprender funciones en JavaScript es desarrollar la capacidad de escribir código modular, reutilizable y eficiente. Esto te permitirá estructurar tus programas de manera organizada, lógica y flexible.

Introducción

Las funciones son un pilar fundamental en la programación. Dominar su uso te facilitará dividir tareas complejas en partes más manejables, reduciendo la redundancia y mejorando la organización del código.

---

1. Conceptos básicos sobre funciones

1.1. ¿Qué es una función?

Una función es un bloque de código diseñado para realizar una tarea específica. Las funciones:

- Pueden ser reutilizadas en cualquier parte del programa.
- Pueden aceptar **parámetros** para operar con diferentes valores.
- Pueden devolver un resultado utilizando la palabra clave return.

1.2. Sintaxis básica

```
function nombreDeLaFuncion(parametro1, parametro2) {
    // Bloque de código
    return resultado; // Opcional
}

// Llamada de la función
nombreDeLaFuncion(arg1, arg2);
```

- **parametro1 y parametro2** son los parámetros de la función (opcional).
- El bloque de código dentro de las llaves **{}** se ejecuta cuando la función es llamada.
- **return** es opcional; si se utiliza, la función devolverá un valor.
- **Arg1 y arg2** son los argumentos.

---

2. Tipos de funciones

2.1. Funciones Declarativas

Son definidas usando la palabra clave **function** y pueden ser llamadas incluso antes de su definición gracias al **hoisting**.

Ejemplo:

```
function sumar(a, b) {
    return a + b;
}
console.log(sumar(3, 5)); // 8
```

2.2. Funciones Expresadas

Se asignan como un valor a una variable. No tienen **hoisting**, por lo que deben definirse antes de ser usadas.

Ejemplo:

```
const multiplicar = function(a, b) {
    return a * b;
};
console.log(multiplicar(4, 2)); // 8
```

2.3. Funciones de Flecha (Arrow Functions)

Introducidas en ES6, tienen una sintaxis más corta y no tienen su propio this.

Ejemplo:

```
const dividir = (a, b) => a / b;
console.log(dividir(10, 2)); // 5
```

---

3. Parámetros y Argumentos

3.1. Parámetros

Son las variables que una función espera recibir cuando se define.

Ejemplo:

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

3.2. Argumentos

Son los valores que pasas a los parámetros al invocar una función.

```javascript
saludar("Juan"); // Hola, Juan
```

3.3. Parámetros Predeterminados

Puedes asignar un valor por defecto a los parámetros.

Ejemplo:

```javascript
function saludar(nombre = "Usuario") {
    console.log("Hola, " + nombre + "!");
}
saludar(); // Hola, Usuario!
saludar("Pedro"); // Hola, Pedro!
```

---

4. Retorno de Valores

Las funciones pueden devolver valores usando return. Esto permite que el resultado de la función sea utilizado más adelante.

Ejemplo:

```
function cuadrado(numero) {
    return numero * numero;
}
const resultado = cuadrado(4);
console.log(resultado); // 16
```

---

5. Alcance (Scope)

El alcance determina dónde se pueden usar las variables.

5.1. Alcance Local

Las variables declaradas dentro de una función solo son accesibles dentro de esa función.

Ejemplo:

```
function ejemploLocal() {
    let local = "Solo dentro de esta función";
    console.log(local);
}
ejemploLocal(); // "Solo dentro de esta función"
// console.log(local); // Error: local no está definida
```

5.2. Alcance Global

Las variables declaradas fuera de cualquier función son accesibles en todo el programa.

Ejemplo:

```
let global = "Disponible en todo el programa";
function mostrarGlobal() {
    console.log(global);
}
mostrarGlobal(); // "Disponible en todo el programa"
```

5.3. Alcance de Bloque

Con let y const, las variables solo son accesibles dentro del bloque donde fueron declaradas.

Ejemplo:

```
if (true) {
    let bloque = "Disponible solo dentro del bloque";
    console.log(bloque);
}
// console.log(bloque); // Error: bloque no está definida
```

---

6. Funciones Anidadas

Son aquellas funciones que se definen dentro de otras funciones. Es decir, tienes una función "principal" que contiene una o más funciones dentro de ella. Estas funciones internas pueden acceder a las variables de la función externa, lo que las hace muy útiles para crear módulos de código.

Ejemplo:

javascript

```javascript
function saludo(nombre) {
    // Función anidada
    function mensaje() {
        console.log("Hola, " + nombre + "!");
    }

    // Llamamos a la función anidada
    mensaje();
}

// Llamamos a la función principal
saludo("Carlos");
```

Ejercicio Práctico 4: Funciones

1. Crea una función principal llamada calculadora que tenga funciones anidadas para sumar, restar, multiplicar y dividir. Usa una estructura condicional para determinar cuál función anidada ejecutar según un operador recibido como argumento.
2. Contar número pares hasta n veces y sumarlos.

3. Verificar si un número es primo
4. Encontrar el número más grande en un arreglo
5. Verificar si un número está en un arreglo.

7. Callbacks

Un **callback** es una función que se pasa como argumento a otra función y que se ejecutará después de que esa función haya completado su operación.

javascript

```javascript
function greet(name, callback) {
  console.log(`Hola, ${name}`);
  callback(); // Llama a la función pasada como argumento
}

function sayGoodbye() {
  console.log("Adiós!");
}

greet("Juan", sayGoodbye);
// Salida:
// Hola, Juan
// Adiós!
```

7.1. Callbacks síncronos

- Ocurren de forma inmediata, dentro del mismo ciclo del Event Loop.

Ejemplo:

javascript

```javascript
function add(a, b, callback) {
  let result = a + b;
  callback(result);
}

add(2, 3, (sum) => {
  console.log(`La suma es: ${sum}`); // Se ejecuta inmediatamente
});
```

7.2. Callbacks asíncronos

- Se ejecutan después de que ocurra una tarea que toma tiempo, como un temporizador o una operación de red.

Ejemplo:

javascript

```javascript
function fetchData(callback) {
  setTimeout(() => {
    callback("Datos recibidos");
  }, 2000);
}

fetchData((data) => {
  console.log(data); // Se ejecuta después de 2 segundos
});
```

7.3. Ventajas de los Callback

1. Son simples de implementar.
2. Funcionan bien para tareas pequeñas y lineales.

7.4. Patrones de uso comunes de Callbacks

7.4.1. Callback en setTimeout:

Se ejecuta despuede un tiempo determinado.

```javascript
setTimeout(() => {
  console.log("Esto se ejecuta después de 2 segundos");
}, 2000);
```

7.4.2. Callback de eventos DOM

Un callback de eventos DOM es una función que se ejecuta cuando ocurre un evento específico en un elemento del DOM, como hacer clic en un botón, pasar el ratón sobre un elemento o presionar una tecla.

```javascript
document.getElementById("boton").addEventListener("click", () => {
  console.log("Botón clickeado!");
});
```

7.4.3. Callback en operaciones de red

Los callbacks en operaciones de red se usan para manejar la respuesta de una solicitud HTTP o cualquier tarea que implique comunicación con un servidor.

.

---

7. Conclusión

Las funciones en JavaScript son herramientas esenciales para escribir código eficiente, modular y claro. Dominar los diferentes tipos de funciones, parámetros, retornos y alcances te permitirá desarrollar programas más organizados y profesionales.