20. Temporizadores (`setTimeout` y `setInterval`)

JavaScript ofrece dos funciones esenciales para ejecutar código de forma **asincrónica** después de un retraso o repetidamente: `setTimeout` y `setInterval`. Estos temporizadores son clave para implementar animaciones, retrasar acciones, o ejecutar tareas periódicas.

---

1. `setTimeout` : Ejecución Única con Retraso

Ejecuta una función **una vez**, después de un tiempo especificado en milisegundos.

Sintaxis:

```
setTimeout(función, retraso, arg1, arg2, ...);
```

Ejemplo:

```
setTimeout((msg)⇒console.log(msg), 3000, "Hola");
```

Al ejecutar esta línea en la consola devuelve el id del hilo que se está ejecutando. Espera 3 segundos y dice hola.

Si quiero pararlo antes de que termine usaré `clearTimeout` usando ese id:

```
// Detener un setTimeout
const timer = setTimeout(() ⇒ {}, 1000);
clearTimeout(timer);
```

```
// Mostrar un mensaje después de 2 segundos (2000 ms)
const timerID = setTimeout(() ⇒ {
  console.log("¡Han pasado 2 segundos!");
}, 2000);

// Cancelar el temporizador antes de que se ejecute
clearTimeout(timerID);
```

---

2. `setInterval` : Ejecución Repetida

Ejecuta una función **repetidamente**, con un intervalo fijo entre cada ejecución.

Sintaxis:

```
setInterval(función, intervalo, arg1, arg2, ...);
```

Ejemplo:

```
setInterval((msg)⇒console.log(msg), 3000, "Hola");
```

Al ejecutar esta línea en la consola devuelve el id del hilo que se está ejecutando. Cada 3 segundos dirá hola.

El hilo puede detenerse con `clearInterval` dándole el id del hilo:

```
// Detener un setInterval
const interval = setInterval(() ⇒ {}, 1000);
clearInterval(interval);
```

```
// Contador que se incrementa cada segundo
let contador = 0;
const intervalID = setInterval(() ⇒ {
   console.log(`Contador: ${contador}`);
   contador++;
   if (contador === 5) {
      clearInterval(intervalID); // Detener después de 5 iteraciones
   }
}, 1000)
```

---

3. Ejemplos Prácticos

Ejemplo 1: Cronómetro Simple

```
let segundos = 0;
const cronometro = setInterval(() ⇒ {
   segundos++;
   console.log(`Han pasado ${segundos} segundos`);
   if (segundos >= 10) {
      clearInterval(cronometro);
      console.log("¡Tiempo finalizado!");
   }
}, 1000);
```

4. Buenas Prácticas

1. **Evitar Acumulación de Intervalos**:
   - Si no limpias un `setInterval`, seguirá ejecutándose incluso si el usuario navega fuera de la página.
   - **Ejemplo**:

     ```
     let intervalID;

     function iniciar() {
     ```

```
    intervalID = setInterval(() => {}, 1000);
}

function detener() {
    clearInterval(intervalID);
}
```

2. **Retrasos No Exactos**:
   - El tiempo en `setTimeout/setInterval` no es garantizado. El evento se añade a la cola del **event loop** y puede retrasarse si el hilo principal está ocupado.
3. **Usar Arrow Functions para `this`**:
   - En clases o métodos, usa funciones flecha para preservar el contexto:

```
class Temporizador {
    constructor() {
        this.contador = 0;
    }

    iniciar() {
        setInterval(() => {
            this.contador++;
            console.log(this.contador); // `this` se refiere a la instancia
        }, 1000);
    }
}
```

---

5. Casos de Uso Comunes

Función	Casos de Uso
`setTimeout`	Retrasar una acción, cargar datos después de un tiempo.
`setInterval`	Actualizar un reloj, verificar notificaciones, animaciones.

Con `setTimeout` y `setInterval`, puedes crear comportamientos dinámicos y asincrónicos. Recuerda siempre limpiar los temporizadores cuando no sean necesarios para evitar fugas de memoria.