**Taller: Concurrencia y Asincronismo en JavaScript**

**Objetivo del taller**

Al finalizar el taller, los estudiantes podrán:

1. Comprender cómo JavaScript gestiona la concurrencia y el asincronismo.
2. Usar correctamente setTimeout, async/await y Promesas.
3. Resolver problemas prácticos aplicando conceptos de asincronismo.

**Sección 1: Introducción a la concurrencia en JavaScript**

**1.1. ¿Qué es la concurrencia?**

* En programación, la concurrencia es la capacidad de manejar múltiples tareas a la vez.
* JavaScript es *single-threaded*, lo que significa que tiene un único hilo de ejecución. Sin embargo, puede manejar tareas concurrentes usando un modelo basado en eventos y un bucle de eventos.

**Conceptos clave:**

* **Call Stack (Pila de llamadas):** Lugar donde se ejecutan las funciones.
* **Heap:** Donde se almacena la memoria asignada dinámicamente.
* **Event Loop (Bucle de eventos):** Gestiona las tareas asincrónicas, monitoreando la cola de tareas y moviéndolas al stack cuando está libre.
* **Web APIs:** Funcionalidades del navegador (o entorno de ejecución) que manejan tareas como setTimeout, fetch, etc.

**Diagrama ilustrativo:**

* Call Stack <-> Web APIs <-> Task Queue <-> Event Loop

**1.2. Ejemplo básico del Event Loop**

javascript

Copiar código

console.log("Inicio");

setTimeout(() => {

console.log("Timeout");

}, 1000);

console.log("Fin");

**Explicación:**

1. console.log("Inicio") se ejecuta inmediatamente en el stack.
2. setTimeout delega su función a las Web APIs.
3. console.log("Fin") se ejecuta mientras setTimeout espera.
4. Después de 1 segundo, la función del setTimeout pasa a la Task Queue y espera a que el stack esté libre.

**Sección 2: Asincronismo en JavaScript**

**2.1. Técnicas de asincronismo**

1. **Callbacks:**
   * Funciones pasadas como argumento para ejecutarse más tarde.
   * Problemas: Callback Hell.

javascript

Copiar código

setTimeout(() => {

console.log("Primero");

setTimeout(() => {

console.log("Segundo");

setTimeout(() => {

console.log("Tercero");

}, 1000);

}, 1000);

}, 1000);

1. **Promesas:**
   * Un objeto que representa la eventual resolución (o rechazo) de una operación asincrónica.

javascript

Copiar código

const promesa = new Promise((resolve, reject) => {

setTimeout(() => resolve("¡Promesa resuelta!"), 2000);

});

promesa.then((resultado) => console.log(resultado));

1. **Async/Await:**
   * Azúcar sintáctico para manejar Promesas.
   * Más legible y parecido a código síncrono.

javascript

Copiar código

const obtenerDatos = async () => {

const resultado = await new Promise((resolve) => {

setTimeout(() => resolve("Datos obtenidos"), 2000);

});

console.log(resultado);

};

obtenerDatos();

**Sección 3: Ejemplos prácticos**

**3.1. Uso de setTimeout**

Simula una operación que tarda tiempo, como una llamada a una API.

javascript

function simularOperacion() {

console.log("Operación iniciada...");

setTimeout(() => {

console.log("Operación completada después de 2 segundos.");

}, 2000);

}

simularOperacion();

**3.2. Promesas**

Encadenar múltiples pasos asincrónicos.

javascript

function tareaAsincrona(mensaje, tiempo) {

return new Promise((resolve) => {

setTimeout(() => resolve(mensaje), tiempo);

});

}

tareaAsincrona("Paso 1 completado", 1000)

.then((resultado) => {

console.log(resultado);

return tareaAsincrona("Paso 2 completado", 2000);

})

.then((resultado) => console.log(resultado));

**3.3. async/await**

Simplificando el código anterior.

Javascript

async function ejecutarTareas() {

const paso1 = await tareaAsincrona("Paso 1 completado", 1000);

console.log(paso1);

const paso2 = await tareaAsincrona("Paso 2 completado", 2000);

console.log(paso2);

}

ejecutarTareas();

**Sección 4: Ejercicios**

**Actividad Síncrona:** Tienes una lista de nombres de archivos y debes procesarlos uno por uno simulando que cada archivo tarda 2 segundos en procesarse.

Javascript

function procesarArchivo(nombre) {

console.log(`Procesando archivo: ${nombre}`);

const inicio = Date.now();

while (Date.now() - inicio < 2000) {} // Bloqueo de 2 segundos (simulación)

console.log(`Archivo procesado: ${nombre}`);

}

const archivos = ["archivo1.txt", "archivo2.txt", "archivo3.txt"];

archivos.forEach((archivo) => procesarArchivo(archivo));

**Objetivo:** Convierte el código a uno concurrente para procesar todos los archivos al mismo tiempo usando:

* Promesas.
* async/await.

**Sección 5: Recursos adicionales**

* [MDN Web Docs: Asynchronous JavaScript](https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Asynchronous)
* JavaScript.info: Promises, async/await

**Sección 6: Conclusión y cierre**

* Repasar conceptos clave.
* Resolver dudas de los estudiantes.
* Reflexionar sobre casos prácticos de asincronismo en el desarrollo web.