Promesas 🤝

Las **Promesas** (Promises) en JavaScript son un concepto fundamental para manejar la **asincronía**. Una Promesa es un **objeto** que representa la eventual finalización (o fracaso) de una operación asíncrona y su valor resultante. 🤝

En esencia, una Promesa actúa como un marcador de posición para un valor que aún no está disponible, pero que se espera que lo esté en el futuro.

🧭 ¿Por qué son necesarias?

Antes de las Promesas, las operaciones asíncronas (como leer un archivo o hacer una petición a una API) se manejaban principalmente con **callbacks** anidados. Esto a menudo llevaba al llamado **"Callback Hell"** (Infierno de *Callbacks*), donde el código se volvía difícil de leer y mantener.

Las Promesas ofrecen una alternativa más limpia y estructurada al permitirte **encadenar** múltiples operaciones asíncronas de forma secuencial.

🚦 Estados de una Promesa

Una Promesa existe siempre en uno de estos tres estados mutuamente excluyentes:

Estado	Descripción	Resultado
Pending (Pendiente) ⏳	Es el estado inicial. La operación asíncrona aún no se ha completado.	El proceso está en curso.
Fulfilled / Resolved (Cumplida / Resuelta) ✅	La operación asíncrona terminó con éxito y produjo un valor.	El método .then() maneja este estado.
Rejected (Rechazada) ❌	La operación falló, lo que generalmente resulta en un error.	El método .catch() maneja este estado.

📜 Consumo de Promesas (.then() y .catch())

Para obtener el valor o el error de una Promesa, se utilizan los métodos de instancia .then() y .catch():

Ejemplo Clásico (Encadenamiento)

JavaScript

```
hacerPeticion()
  .then((datos) => {
    // Se ejecuta si la Promesa se resuelve (estado Fulfilled)
    console.log("Datos obtenidos:", datos);
    return procesarDatos(datos); // Se puede retornar otra Promesa
  })
  .then((resultadoProcesado) => {
    // Continúa el encadenamiento con el resultado anterior
    console.log("Resultado final:", resultadoProcesado);
  })
  .catch((error) => {
    // Se ejecuta si cualquier Promesa en la cadena es rechazada (estado Rejected)
    console.error("Ocurrió un error:", error);
  });
```

✨ El Enfoque Moderno (async/await)

Aunque .then() y .catch() son fundamentales para entender las Promesas, el estándar moderno de JavaScript utiliza las palabras clave **async** y **await** como una "azúcar sintáctica" para trabajar con Promesas de una manera que parece síncrona.

- **async**: Declara una función como asíncrona, lo que le permite contener la palabra clave await. Una función async siempre devuelve implícitamente una Promesa.
- **await**: Solo se puede usar dentro de una función async. **Pausa la ejecución** de la función hasta que la Promesa a su derecha se resuelva o se rechace, y devuelve el valor resultante.

Ejemplo con async/await

JavaScript

```
async function obtenerYProcesar() {
  try {
    // await pausa la ejecución hasta que la Promesa 'hacerPeticion' termine.
    const datos = await hacerPeticion();

    const resultadoProcesado = await procesarDatos(datos);

    console.log("Resultado final:", resultadoProcesado);

  } catch (error) {
    // El bloque try/catch maneja los errores (Rejected) de forma nativa.
    console.error("Ocurrió un error:", error);
  }
}

obtenerYProcesar();
```

✨ El Enfoque Moderno ACTUALIZADO (async/await)

```
// 1. Definición de la función hacerPeticion (devuelve una Promesa)
function hacerPeticion() {
    return new Promise((resolve, reject) => {
        // Simulación de una operación asíncrona exitosa después de 1 segundo
        setTimeout(() => {
            resolve({ id: 1, nombre: "Producto A" }); // Resuelve la Promesa con datos
        }, 1000);
        // Si fallara, llamaríamos a reject(new Error("Fallo de red"));
    });
}

// 2. Definición de la función procesarDatos (devuelve otra Promesa)
function procesarDatos(datos) {
    return new Promise((resolve) => {
        // Simulación de procesamiento
        const datosProcesados = {
            ...datos,
            procesado: true
        };
        resolve(datosProcesados);
    });
}

// --- Tu código original ahora funcionará: ---

hacerPeticion()
.then((datos) => {
```

```
    console.log("Datos obtenidos (THEN):", datos);
    return procesarDatos(datos);
})
.then((resultadoProcesado) => {
    console.log("Resultado final (THEN):", resultadoProcesado);
})
.catch((error) => {
    console.error("Ocurrió un error (THEN):", error);
});

// Otra forma de consumo con async/await
async function obtenerYProcesar() {
    try {
        const datos = await hacerPeticion();
        const resultadoProcesado = await procesarDatos(datos);
        console.log("Resultado final (ASYNC/AWAIT):", resultadoProcesado);
    } catch (error) {
        console.error("Ocurrió un error (ASYNC/AWAIT):", error);
    }
}
obtenerYProcesar();
```