**Ejercicio 3.1: Promesa Exitosa**

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

setTimeout(() => {

resolve("¡Operación exitosa!");

}, 2000);

});

promesaExitosa.then(resultado => {

console.log("Éxito:", resultado);

});

**Ejercicio 3.2: Promesa Fallida**

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

setTimeout(() => {

reject("¡Algo salió muy mal!");

}, 1500);

});

promesaFallida.catch(error => {

console.error("Error:", error);

});

**Ejercicio 3.3: .then y .catch juntos**

console.log("Consumiendo promesa exitosa:");

promesaExitosa

.then(resultado => {

console.log("Éxito:", resultado);

})

.catch(error => {

console.log("Este catch no se ejecutará:", error);

});

console.log("\nConsumiendo promesa fallida:");

promesaFallida

.then(resultado => {

console.log("Este then no se ejecutará:", resultado);

})

.catch(error => {

console.error("Error:", error);

});

**Ejercicio 3.4: .finally**

promesaExitosa

.then(resultado => {

console.log("Éxito:", resultado);

})

.catch(error => {

console.error("Error:", error);

})

.finally(() => {

console.log("La operación ha finalizado (exitosa).");

});

promesaFallida

.then(resultado => {

console.log("Resultado:", resultado);

})

.catch(error => {

console.error("Error:", error);

})

.finally(() => {

console.log("La operación ha finalizado (fallida).");

});

**Sección 4: Crear Promesas**

**4.1: verificarNumero(numero)**

function verificarNumero(numero) {

return new Promise((resolve, reject) => {

if (numero >= 10) {

resolve("El número es válido.");

} else {

reject("El número no es válido.");

}

});

}

verificarNumero(15)

.then(msg => console.log("✔️", msg))

.catch(err => console.error("❌", err));

verificarNumero(5)

.then(msg => console.log("✔️", msg))

.catch(err => console.error("❌", err));

**4.2: esperaAleatoria()**

function esperaAleatoria() {

return new Promise((resolve) => {

const tiempoEspera = Math.floor(Math.random() \* 4000) + 1000;

setTimeout(() => {

resolve(tiempoEspera);

}, tiempoEspera);

});

}

console.log("Esperando un tiempo aleatorio...");

esperaAleatoria().then(tiempo => {

console.log(`⏱️ Esperé ${tiempo} milisegundos.`);

});

**Sección 5: Encadenamiento de Promesas**

**5.1: paso1 → paso2 → paso3**

function paso1() {

return new Promise(resolve => {

setTimeout(() => resolve(1), 1000);

});

}

function paso2(valor) {

return new Promise(resolve => {

setTimeout(() => resolve(valor + 1), 1000);

});

}

function paso3(valor) {

return new Promise(resolve => {

setTimeout(() => resolve(valor + 1), 1000);

});

}

paso1()

.then(valor => {

console.log("Paso 1:", valor);

return paso2(valor);

})

.then(valor => {

console.log("Paso 2:", valor);

return paso3(valor);

})

.then(valor => {

console.log("Paso 3:", valor);

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

});

**5.2: Encadenamiento con error**

Promise.resolve(10)

.then(num => {

return num \* 2; // 20

})

.then(num => {

if (num > 30) {

throw new Error("Número demasiado grande");

}

return num;

})

.then(num => {

console.log("Proceso exitoso con el número:", num);

})

.catch(err => {

console.error("⚠️ Error:", err.message);

});

**Sección 6: async / await**

**6.1: Versión async/await de 5.1**

function paso1() {

return new Promise(resolve => setTimeout(() => resolve(1), 1000));

}

function paso2(valor) {

return new Promise(resolve => setTimeout(() => resolve(valor + 1), 1000));

}

function paso3(valor) {

return new Promise(resolve => setTimeout(() => resolve(valor + 1), 1000));

}

async function ejecutarPasos() {

const valor1 = await paso1();

console.log("Paso 1:", valor1);

const valor2 = await paso2(valor1);

console.log("Paso 2:", valor2);

const valor3 = await paso3(valor2);

console.log("Paso 3:", valor3);

console.log("Resultado final:", valor3);

}

ejecutarPasos();

**6.2: obtenerDatosCompletos()**

function obtenerUsuario() {

return new Promise(resolve => {

setTimeout(() => resolve({ id: 1, nombre: "Leanne Graham" }), 1000);

});

}

function obtenerPosts(userId) {

return new Promise(resolve => {

setTimeout(() => resolve([

{ userId, title: "Post 1" },

{ userId, title: "Post 2" }

]), 1500);

});

}

async function obtenerDatosCompletos() {

try {

console.log("Obteniendo usuario...");

const usuario = await obtenerUsuario();

console.log(`Usuario obtenido: ${usuario.nombre}`);

console.log("Obteniendo posts...");

const posts = await obtenerPosts(usuario.id);

console.log(`Posts obtenidos: ${posts.length}`);

console.log(`${usuario.nombre} tiene ${posts.length} posts.`);

} catch (error) {

console.error("Error:", error);

}

}

obtenerDatosCompletos();

**6.3: Manejo de errores con async/await**

function operacionQueFalla() {

return new Promise((\_, reject) => {

setTimeout(() => reject("¡Falló la operación!"), 1000);

});

}

async function intentarOperacionRiesgosa() {

try {

const resultado = await operacionQueFalla();

console.log("Resultado:", resultado);

} catch (error) {

console.error("Error capturado:", error);

}

}

intentarOperacionRiesgosa();

**Sección 7: Métodos estáticos de Promise**

**7.1: Promise.all**

const descarga1 = new Promise(res => setTimeout(() => res("Archivo 1"), 1000));

const descarga2 = new Promise(res => setTimeout(() => res("Archivo 2"), 2000));

const descarga3 = new Promise(res => setTimeout(() => res("Archivo 3"), 1500));

Promise.all([descarga1, descarga2, descarga3])

.then(resultados => {

console.log("✅ Todas las descargas completadas.");

console.log("Resultados:", resultados);

})

.catch(err => console.error("❌ Una descarga falló:", err));

**7.2: Promise.race**

const servidor1 = new Promise(res => setTimeout(() => res("Servidor 1"), 1000));

const servidor2 = new Promise(res => setTimeout(() => res("Servidor 2"), 2000));

Promise.race([servidor1, servidor2])

.then(resultado => {

console.log("⚡ El más rápido fue:", resultado);

});

**7.3: Promise.allSettled**

const p1 = Promise.resolve("Éxito A");

const p2 = Promise.reject("Fallo B");

const p3 = Promise.resolve("Éxito C");

Promise.allSettled([p1, p2, p3]).then(resultados => {

resultados.forEach((resultado, index) => {

console.log(`Promesa ${index + 1}:`, resultado.status, resultado.value || resultado.reason);

});

});

**7.4: Promise.any**

const servidorA = Promise.reject("Falla A");

const servidorB = Promise.resolve("Respuesta de B");

const servidorC = Promise.resolve("Respuesta de C");

Promise.any([servidorA, servidorB, servidorC])

.then(resultado => {

console.log("✅ Primer éxito:", resultado);

})

.catch(error => {

console.error("❌ Todas fallaron:", error);

});