Ejercicios de Promesas en Node.js

Estructura básica de una Promesa

Una promesa representa una operación asíncrona que se completará en el futuro. Aquí tienes su estructura básica:

```
const miPromesa = new Promise((resolve, reject) => {
  const todoBien = true;

  if (todoBien) {
    resolve('Todo salió bien');
  } else {
    reject('Algo salió mal');
  }
});
```

Consumir una promesa con .then() y .catch()

```
miPromesa
  .then(resultado => console.log('RESUELTA:', resultado))
  .catch(error => console.error('RECHAZADA:', error))
  .finally(() => console.log('Fin de la operación'));
```

Consumir una promesa con async/await

```
async function ejecutar() {
  try {
    const resultado = await miPromesa;
    console.log('RESUELTA:', resultado);
  } catch (error) {
    console.error('RECHAZADA:', error);
  }
}
```

Ejercicios Sencillos resueltos

Ejercicio 1: Temporizador con Promesa

Crea una función esperar(ms) que devuelva una promesa que se resuelva después de X milisegundos.

```
function esperar(ms) {
  return new Promise(resolve => {
    setTimeout(() => resolve(`Han pasado ${ms} ms`), ms);
  });
}

esperar(2000)
  .then(mensaje => console.log(mensaje))
  .catch(err => console.error(err));
```

Ejercicio 2: Comprobar si un número es mayor que 10

La función debe resolver si el número es mayor que 10 o rechazar si no lo es.

```
function esMayorQueDiez(num) {
  return new Promise((resolve, reject) => {
    if (num > 10) resolve(`${num} es mayor que 10`);
    else reject(`${num} no es mayor que 10`);
  });
}

//Llamadas de ejemplo
// Número mayor que 10 → se resuelve
esMayorQueDiez(15)
  .then((msg) => {
    console.log('OK:', msg);
  })
  .catch((err) => {
    console.error('ERROR:', err);
  });

// Número menor o igual que 10 → se rechaza
esMayorQueDiez(7)
  .then((msg) => {
    console.log('OK:', msg);
  })
  .catch((err) => {
    console.error('ERROR:', err);
  });
```

Ejercicio 3: Simular búsqueda de usuario

Simula una consulta asíncrona usando setTimeout y promesas.

```
const usuarios = [
  { id: 1, nombre: 'Ana' },
  { id: 2, nombre: 'Luis' },
  { id: 3, nombre: 'María' },
];

function buscarUsuarioPorId(id) {
  return new Promise((resolve, reject) => {
    setTimeout(() => {
      const usuario = usuarios.find((u) => u.id === id);
      if (usuario) {
        resolve(usuario);              // Lo encontramos
      } else {
        reject(new Error(`Usuario ${id} no encontrado`)); // No existe
      }
    }, 1000);
  });
}


// Caso 1: id existente → se resuelve

buscarUsuarioPorId(2)

  .then((usuario) => {

    console.log('Usuario encontrado:', usuario);

  })

  .catch((err) => {

    console.error('Error:', err.message);

  });



// Caso 2: id no existente → se rechaza

buscarUsuarioPorId(99)

  .then((usuario) => {

    console.log('Usuario encontrado:', usuario);

  })

  .catch((err) => {

    console.error('Error:', err.message);

  });
```

ENTREGABLES PARA CASA

EJERCICIO 1 – Temporizador con mensaje aleatorio

Crea una función llamada mensajeAleatorio() que:

1. Devuelva una promesa.
2. Espere entre 1 y 3 segundos (tiempo aleatorio con Math.random()).
3. Se resuelva con el mensaje:

   "El mensaje tardó X segundos en aparecer".

Después llama a la función 3 veces usando .then().

EJERCICIO 2 – Validar contraseña

Crea una función validarPassword(pass) que devuelva una promesa que:

- Se resuelva si la contraseña tiene 8 o más caracteres.
- Se rechace si tiene menos.

Realiza dos llamadas a la función:

- Una con contraseña válida.
- Otra con una contraseña demasiado corta.

Maneja los errores con .catch().

EJERCICIO 3 – Buscar producto por nombre

Dado el siguiente array:

```
const productos = [
  { nombre: 'Teclado', precio: 25 },
  { nombre: 'Ratón', precio: 15 },
  { nombre: 'Monitor', precio: 120 }
];
```

Crea una función buscarProducto(nombre) que:

1. Devuelva una promesa.
2. Use setTimeout de 1 segundo para simular una búsqueda.
3. Se resuelva con el producto si existe.
4. Se rechace con un error si NO existe.

Luego llámala con:

- Un nombre de producto que exista.
- Un nombre de producto que no exista.

EJERCICIO 4 — Triple operación encadenada

Crea tres funciones que devuelvan promesas:

1. sumar(x) → devuelve una promesa que resuelve x + 10
2. multiplicar(x) → devuelve una promesa que resuelve x * 3
3. restar(x) → devuelve una promesa que resuelve x - 5

Realiza la siguiente cadena:

sumar(5) → luego multiplicar → luego restar

Muestra el resultado final en consola.
