**Ejercicio 1: Simulación de una consulta a la base de datos**

**Descripción**: Simula una consulta a una base de datos que devuelve un listado de productos. Si la base de datos tiene productos, la promesa debe resolverse con los datos; si no hay productos, la promesa debe ser rechazada.

**Instrucciones**:

1. Crea una función consultarProductos() que devuelva una promesa.
2. La promesa debe resolver con un arreglo de productos si hay productos, o rechazar con un mensaje de error si no hay productos.
3. Utiliza .then() para manejar el resultado de la promesa y mostrar los productos en la consola. Si hay un error, utiliza .catch() para mostrar el mensaje de error.

**Ejercicio 2: Simulación de la validación de una contraseña**

**Descripción**: Simula un proceso de validación de una contraseña. Si la contraseña es válida, la promesa se resuelve con un mensaje de éxito. Si la contraseña es incorrecta, la promesa se rechaza.

**Instrucciones**:

1. Crea una función validarContraseña() que reciba una contraseña.
2. Si la contraseña es "12345", la promesa debe resolverse con el mensaje "Contraseña válida".
3. Si la contraseña no es "12345", la promesa debe rechazar con el mensaje "Contraseña incorrecta".
4. Utiliza .then() para mostrar el mensaje de éxito y .catch() para mostrar el mensaje de error.

**Ejercicio 3: Simulación de la descarga de un archivo**

**Descripción**: Simula la descarga de un archivo. La promesa debe resolverse después de 2 segundos si la descarga es exitosa, o rechazarse si ocurre algún error en la descarga.

**Instrucciones**:

1. Crea una función descargarArchivo() que devuelva una promesa.
2. La promesa debe resolverse con el mensaje "Archivo descargado exitosamente" después de 2 segundos si la descarga es exitosa.
3. Si ocurre un error durante la descarga, la promesa debe rechazarse con el mensaje "Error al descargar el archivo".
4. Utiliza .then() para mostrar el mensaje de éxito y .catch() para mostrar el mensaje de error.

**Ejercicio 4: Simulación de una consulta a la base de datos**

**Descripción**: Simula una consulta a una base de datos que devuelve un listado de usuarios. Si la base de datos tiene usuarios, la promesa debe resolverse con los datos; si no hay usuarios, la promesa debe ser rechazada.

**Instrucciones**:

1. Crea una función consultarUsuarios() que devuelva una promesa.
2. La promesa debe resolver con un arreglo de usuarios si hay usuarios, o rechazar con un mensaje de error si no hay usuarios.
3. Usa async y await para resolver la promesa dentro de una función.
4. Muestra los usuarios en la consola si la promesa se resuelve, o un mensaje de error si la promesa se rechaza.

**Ejercicio 5: Simulación de la validación de una contraseña**

**Descripción**: Simula un proceso de validación de una contraseña. Si la contraseña es válida, la promesa se resuelve con un mensaje de éxito. Si la contraseña es incorrecta, la promesa se rechaza.

**Instrucciones**:

1. Crea una función validarContraseña() que reciba una contraseña.
2. Si la contraseña es "abc123", la promesa debe resolverse con el mensaje "Contraseña válida".
3. Si la contraseña no es "abc123", la promesa debe rechazarse con el mensaje "Contraseña incorrecta".
4. Usa async y await para manejar la promesa dentro de una función asincrónica.

**Ejercicio 6: Simulación de la descarga de un archivo**

**Descripción**: Simula la descarga de un archivo. La promesa debe resolverse después de 2 segundos si la descarga es exitosa, o rechazarse si ocurre algún error en la descarga.

**Instrucciones**:

1. Crea una función descargarArchivo() que devuelva una promesa.
2. La promesa debe resolverse con el mensaje "Archivo descargado exitosamente" después de 2 segundos si la descarga es exitosa.
3. Si ocurre un error durante la descarga, la promesa debe rechazarse con el mensaje "Error al descargar el archivo".
4. Usa async y await para manejar la promesa dentro de una función asincrónica.

**Ejercicio 7: Consulta de Usuarios desde una API**

### ****Descripción****:

Vas a realizar una petición a un servidor de prueba utilizando la API fetch() para obtener una lista de usuarios. Deberás manejar correctamente la respuesta con promesas y mostrar los datos en la consola.

### ****Instrucciones****:

1. Crea una función obtenerUsuarios() que haga una petición fetch() a la API https://jsonplaceholder.typicode.com/users.
2. La función debe devolver una promesa que:
   * Se resuelve si la respuesta es exitosa (status 200) y devuelve los datos en formato JSON.
   * Se rechaza si ocurre un error o la respuesta no es válida.
3. Usa el método .then() para manejar la respuesta exitosa y mostrar los usuarios en la consola.
4. Usa .catch() para manejar posibles errores y mostrarlos en la consola.

### ****Ejercicio 8.**** Promise.all()

**Descripción:** Crea tres promesas que representen **peticiones a una API ficticia** con diferentes tiempos de respuesta. Usa Promise.all() para esperar a que **todas** se resuelvan y muestra el resultado en consola.

### ****Ejercicio 9.**** Promise.allSettled()

**Descripción:** Simula un sistema de descarga de archivos donde **algunos pueden fallar**. Usa Promise.allSettled() para esperar a todas y mostrar **tanto las exitosas como las fallidas**.

### ****Ejercicio 10.**** Promise.race()

**Descripción:** Simula una **carrera entre tres servidores**. El primero en responder es el ganador. Usa Promise.race() para mostrar qué servidor respondió más rápido.

### ****Ejercicio 11.**** Promise.any()

**Descripción:** Simula **varios intentos de conexión a una API** donde algunas fallan y otras tienen éxito. Usa Promise.any() para obtener la **primera promesa resuelta**.