Ejercicios promesas

Ejercicio 1: Consulta de Usuarios a un API Simulada

Enunciado:
Tu tarea es simular una llamada a un servidor para obtener una lista de usuarios. Crea una función obtenerUsuarios que devuelva una promesa. La promesa debe resolverse después de 2 segundos con un array de objetos que representen usuarios. Si ocurre un error (simula una probabilidad del 30%), la promesa debe rechazarse con un mensaje de error.

// Ejemplo del resultado esperado:

```
[
  { id: 1, nombre: "Juan Pérez" },
  { id: 2, nombre: "Ana García" },
  { id: 3, nombre: "Luis Fernández" }
]
```

Requisitos:

1. Muestra el resultado en consola utilizando .then() y maneja el error con .catch().
2. Muestra un mensaje en la consola indicando que la consulta está en proceso antes de que se resuelva la promesa.

Ejercicio 2: Cadena de Promesas para Procesamiento de Datos

Enunciado:
Has recibido una lista de productos desde un servidor. Debes crear una función obtenerProductos que devuelva una promesa con el siguiente array de productos después de 1 segundo.

```
[
  { id: 1, nombre: "Teclado", precio: 30 },
  { id: 2, nombre: "Ratón", precio: 20 },
  { id: 3, nombre: "Monitor", precio: 200 }
]
```

Luego, encadena otra promesa para aplicar un descuento del 10% a cada producto y muestra el resultado final. Si ocurre un error durante el proceso, captura y muestra el error.

Requisitos:

1. Encadena las promesas utilizando .then() para aplicar el descuento.
2. Utiliza .catch() para manejar errores en cualquier parte de la cadena.

Ejercicio 3: Simulación de Envío de Mensajes (Usando Promise.all)

Enunciado:
Estás desarrollando una aplicación de mensajería. Necesitas enviar mensajes a varios usuarios al mismo tiempo. Crea una función enviarMensaje que acepte el nombre del usuario y devuelva una promesa que se resuelva en 2 segundos con un mensaje de confirmación.

```
enviarMensaje("Juan Pérez")
// Resultado esperado después de 2 segundos:
// "Mensaje enviado a Juan Pérez"
```

Luego, utiliza Promise.all para enviar mensajes a una lista de usuarios al mismo tiempo. Si uno de los mensajes falla (simula un fallo aleatorio), la promesa principal debe ser rechazada.

Requisitos:

1. Implementa la función para varios usuarios: ["Juan", "Ana", "Luis", "María"].
2. Si todos los mensajes se envían correctamente, muestra una lista con las confirmaciones.
3. Si al menos uno falla, captura el error y muestra qué mensaje falló.

Ejercicio 4: Simulación de Carga de Imágenes con Retrasos (Usando Promise.race)

Enunciado:
Estás desarrollando una galería de imágenes para una página web. Crea una función cargarImagen que acepte el nombre de una imagen y devuelva una promesa que simule la carga de la imagen con un tiempo aleatorio entre 1 y 5 segundos.

Utiliza Promise.race para mostrar la primera imagen que se cargue correctamente.

Requisitos:

1. Implementa la función para las imágenes ["foto1.jpg", "foto2.jpg", "foto3.jpg"].
2. Muestra en la consola cuál fue la primera imagen que se cargó.

Ejercicio 5: Consulta a un API Pública con fetch y Manejo de Errores

Enunciado:
Utiliza el API pública de JSONPlaceholder para obtener una lista de publicaciones.

1. Realiza una petición fetch a https://jsonplaceholder.typicode.com/posts.
2. Muestra en la consola los títulos de las primeras 5 publicaciones.
3. Si la petición falla, muestra un mensaje de error en la consola.

Requisitos:

4. Encadena las promesas utilizando .then() para procesar la respuesta.
5. Maneja los errores con .catch().