PROMESAS

¿Qué es una Promesa en JavaScript?

Las promesas en JavaScript al igual que los Callbacks, son un mecanismo para manejar operaciones asíncronas. Una promesa es un objeto que representa un valor que puede estar disponible ahora, en el futuro, o nunca.

Conceptos Clave:

Estados de una Promesa:

Pending (Pendiente): Es el estado inicial. La promesa está en proceso y no se ha resuelto ni rechazado.

Fulfilled (Resuelta): La operación asíncrona se completó exitosamente y la promesa se resolvió con un valor.

Rejected (Rechazada): La operación falló y la promesa fue rechazada con una razón (error).

Métodos Principales:

.then(onFulfilled, onRejected): Se usa para manejar el resultado de la promesa cuando se resuelve (onFulfilled) o se rechaza (onRejected).

.**catch(onRejected):** Es un atajo para .then(null, onRejected), usado para manejar errores.

.**finally(onFinally):** Ejecuta un código después de que la promesa se haya resuelto o rechazado, independientemente del resultado.

Ejemplos de Promesa

Ejemplo Básico

Vamos a empezar con un ejemplo simple para ilustrar el concepto de una Promesa.

```
let promesaEjemplo = (exito) => {
    return new Promise((resolve, reject) => {
        if (exito) {
            resolve("¡La operación fue exitosa!");
        } else {
            reject("Hubo un error en la operación.");
        }
    });
}
```

```
promesaEjemplo(true)
  .then(resultado => {
    console.log(resultado); // "¡La operación fue exitosa!"
  })
  .catch(error => {
    console.log(error); // "Hubo un error en la operación."
  })
  .finally(() => {
    console.log("Operación completada.");
  });
```

¿Por qué son útiles?

Las promesas permiten trabajar con código asíncrono de una manera más estructurada y manejable que con callbacks anidados (a menudo llamados "callback hell"). Esto facilita la lectura y el mantenimiento del código, especialmente en aplicaciones complejas donde las operaciones asíncronas son comunes, como solicitudes a servidores, temporizadores, o interacciones con APIs.

CODIGO CLASE

```
let empleados = [
    { id: 1, nombre: "Miguel" },
    { id: 2, nombre: "Yenny" },
    { id: 3, nombre: "MCamila" },
  ];

  let salarios = [
    { id: 1, salario: 1000, idempleado: 1 },
    { id: 2, salario: 2000, idempleado: 2 },
  ];

let getEmpleados=(id)=>{
    return new Promise (( resolve, reject   )=>{
        let empleado= empleados.find( item => item.id==id)
        if(empleado==undefined){
            reject("El empleado no existe")
        }else{
            resolve(empleado)
        }
    })
}

let getSalario=(empleado)=>{
    return new Promise((resolve,reject)=>{
```

```
        let salarioEmpleado=salarios.find(item =>
item.idempleado==empleado.id)
        if(salarioEmpleado==undefined){
            reject(`El empleado ${empleado.nombre} no tiene salario `)
        }else{
            resolve(`El empleado ${empleado.nombre} gana
${salarioEmpleado.salario} dolares `)
        }
    })
}


getEmpleados(1)
.then((empleado)=>{
    getSalario(empleado)
    .then((salario)=>{
        console.log(salario);
    })
    .catch((err)=>{
        console.log(err);
    })
})
.catch((error)=>{
    console.log(error);
})
//
//El empleado xx gana yyy
//El empleado yy no tiene salario
//El empleado zz no existe
```

Ejercicio 1 - Saludar a un amigo después de un tiempo

Objetivo: Crear una función que use un callback para saludar a un amigo después de esperar un tiempo determinado.

Instrucciones:

1. Escribe una función llamada saludarDespuesDe que tome dos parámetros:
    - nombre: el nombre de la persona a saludar.
    - callback: una función que se ejecutará después de 2 segundos.

2. Dentro de saludarDespuesDe, usa setTimeout para esperar 2 segundos antes de ejecutar el callback.
3. La función callback debe imprimir en la consola: " ¡Hola, [nombre]! ¿Cómo estás?".

Código base:

```
function saludarDespuesDe(nombre, callback) {
    // Tu código aquí
}

function mostrarSaludo(nombre) {
    console.log(`👋 ¡Hola, ${nombre}! ¿Cómo estás?`);
}

// Llamada a la función (debes completarla)
saludarDespuesDe("Ana", mostrarSaludo);
```

Desafío Extra 🎯

Modifica el código para que el tiempo de espera sea configurable, es decir, el usuario pueda decidir cuántos segundos esperar antes de saludar.

Ejercicio 2: - Preparar un desayuno

Objetivo: Simular la preparación de un desayuno utilizando callbacks.

Instrucciones:

1. Crea una función llamada prepararDesayuno que simule hacer panqueques y luego llame a un callback para servirlos.
2. La función debe:
   - Imprimir " Preparando panqueques..." en la consola.
   - Usar setTimeout para simular que la preparación tarda 3 segundos.
   - Después de 3 segundos, llamar al callback con el mensaje "✅ Panqueques listos para servir."
3. Crea una función llamada servirDesayuno que reciba un mensaje y lo imprima en la consola.

Código base:

```
function prepararDesayuno(callback) {
    console.log(" Preparando panqueques...");
    // Simular que tarda 3 segundos en preparar los panqueques
}

function servirDesayuno(mensaje) {
    console.log(mensaje);
}
```

```
// Llamada a la función (debes completarla)
prepararDesayuno(servirDesayuno);
```

Desafío Extra 🎯

Modifica el código para que después de servir los panqueques, se sirva un jugo de naranja con otro callback.

Ejercicio 3: - Descargar un archivo y procesarlo

Objetivo: Simular la descarga de un archivo y luego procesarlo usando callbacks anidados.

Instrucciones:

4. Crea una función llamada descargarArchivo que reciba un nombre de archivo y un callback.
   - Debe imprimir "📥 Descargando [archivo]...".
   - Usar setTimeout para simular que la descarga tarda 4 segundos.
   - Después de 4 segundos, debe llamar al callback con el mensaje "✅ Archivo [archivo] descargado."
5. Crea otra función llamada procesarArchivo que reciba el mensaje y otro callback.
   - Debe imprimir "🔄 Procesando archivo...".
   - Usar setTimeout para simular que el procesamiento tarda 2 segundos.
   - Luego debe llamar al callback con el mensaje "📂 Archivo procesado exitosamente."
6. Finalmente, crea una función llamada finalizarProceso que reciba el mensaje final y lo imprima en consola.

Código base:

```
function descargarArchivo(nombreArchivo, callback) {
    console.log(`📥 Descargando ${nombreArchivo}...`);
    // Simular 4 segundos de espera antes de llamar al callback
}

function procesarArchivo(mensaje, callback) {
    console.log("🔄 Procesando archivo...");
    // Simular 2 segundos de procesamiento antes de llamar al
callback
}

function finalizarProceso(mensaje) {
    console.log(mensaje);
}

// Llamada a las funciones (debes completarlas)
descargarArchivo("documento.pdf", function(mensajeDescarga) {
```

```
    console.log(mensajeDescarga);
    procesarArchivo(mensajeDescarga, function(mensajeProcesado) {
        console.log(mensajeProcesado);
        finalizarProceso("🚀 Todo el proceso ha finalizado con
éxito.");
    });
});
```

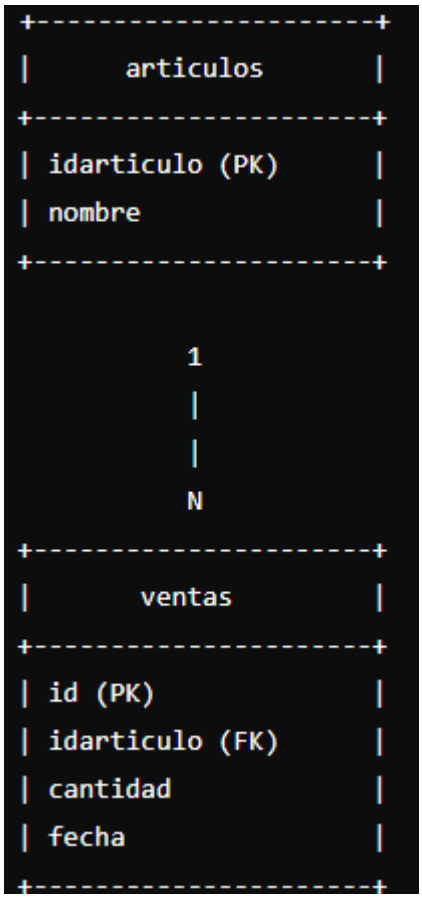
Desafío Extra 🎯

Modifica el código para que:

Después del procesamiento del archivo, se envíe automáticamente un correo de confirmación con otro callback.

Se muestre un mensaje de error si la descarga falla

Ejercicio 4 de Promesas con "Base de Datos" Simulada



Objetivo: Entender el uso de Promesas para manejar operaciones asíncronas, simulando consultas a una base de datos.

Descripción del Ejercicio

Vamos a simular dos tablas:

- `ventas`: Contiene registros de ventas con `idarticulo`, `cantidad` y `fecha`.
- `articulos`: Contiene `idarticulo` y `nombre`.

Tu tarea es escribir una función que:

1. Encuentre los `idarticulos` con más de 3 ventas.
2. Busque los nombres de esos `idarticulos` en la tabla `articulos`.
3. Utilice Promesas para manejar las operaciones asíncronas.
4. Al final la respuesta en un listado con los nombres, id y cantidad de ventas de los ventas cuya cantidad sea superio a 3

Televisor (id=xxxxx) tuvo 3 ventas
Nevera (id=yyyyy) tuvo 7 ventas

```
const ventas = [
  { id: 1, idarticulo: 101, cantidad: 2, fecha: '2024-08-01' },
  { id: 2, idarticulo: 102, cantidad: 1, fecha: '2024-08-01' },
  { id: 3, idarticulo: 103, cantidad: 3, fecha: '2024-08-02' },
  { id: 4, idarticulo: 101, cantidad: 4, fecha: '2024-08-02' },
  { id: 5, idarticulo: 101, cantidad: 1, fecha: '2024-08-03' },
  { id: 6, idarticulo: 104, cantidad: 1, fecha: '2024-08-03' },
  { id: 7, idarticulo: 102, cantidad: 7, fecha: '2024-08-04' },
  { id: 8, idarticulo: 101, cantidad: 1, fecha: '2024-08-04' },
  { id: 9, idarticulo: 102, cantidad: 1, fecha: '2024-08-05' },
  { id: 10, idarticulo: 103, cantidad: 2, fecha: '2024-08-05' }
];

const articulos = [
  { idarticulo: 101, nombre: 'Articulo A' },
  { idarticulo: 102, nombre: 'Articulo B' },
  { idarticulo: 103, nombre: 'Articulo C' },
  { idarticulo: 104, nombre: 'Articulo D' }
];
```

Ejercicio5: Simulación de un Pedido de Alimentos

Imaginee que eres responsable de gestionar pedidos para un restaurante. Cada pedido consiste en una serie de ingredientes que deben ser preparados por diferentes cocineros. Cada cocinero se tarda un tiempo aleatorio para preparar su parte del pedido.

El ejercicio consiste en:

1. **Simular ingredientes**: Cada ingrediente tiene un nombre y un tiempo estimado para ser preparado (en milisegundos).
2. **Simular cocineros**: Cada cocinero es responsable de un grupo de ingredientes. Cada cocinero tarda un tiempo aleatorio en procesar su grupo.
3. **Gestión de Promesas**: Utilizando promesas, cada cocinero devuelve un mensaje indicando que terminó de procesar su grupo de ingredientes.
4. **Composición de las promesas**: Una vez que todos los cocineros han terminado, se debe devolver un mensaje global indicando que el pedido está listo.

Requisitos:

- **Ingredientes**: Un array de objetos que contiene el nombre y el tiempo de preparación de cada ingrediente.

- **Cocineros**: Un array de objetos donde cada cocinero es responsable de un grupo de ingredientes.
- **Uso de Promesas**: Cada cocinero debe devolver una promesa que se resuelve cuando termina de preparar su grupo de ingredientes.

Instrucciones:

1. **Ingredientes**:
   - Crear un array de objetos que represente los ingredientes del pedido, donde cada objeto tiene una propiedad `nombre` (string) y `tiempo` (número de milisegundos).
2. **Cocineros**:
   - Crear un array de objetos de cocineros. Cada cocinero debe ser responsable de preparar un conjunto de ingredientes.
   - Simular el proceso de preparación utilizando una promesa que se resuelve después del tiempo indicado en cada ingrediente.
3. **Simulación de trabajo**:
   - Cada cocinero debe "preparar" sus ingredientes en un tiempo aleatorio entre 1 y 3 segundos. La promesa del cocinero debe resolverse una vez que termine de preparar todos los ingredientes de su grupo.
4. **Composición de Promesas**:
   - Debes esperar a que todos los cocineros hayan terminado de preparar sus ingredientes antes de devolver el mensaje "Pedido listo".

Ejemplo array

```
const ingredientes = [
  { nombre: "Pollo", tiempo: 1500 },
  { nombre: "Arroz", tiempo: 1000 },
  { nombre: "Verduras", tiempo: 1200 },
  { nombre: "Salsa", tiempo: 800 },
  { nombre: "Especias", tiempo: 500 }
];
```