

Conceptos básicos

En tu proceso de aprendizaje, ten muy presente los siguientes conceptos para aplicarlos en los ejercicios de programación o desarrollo.

CONCEPTO	EJEMPLO
<p>Promesas: Las promesas en JavaScript son una forma de manejar operaciones asíncronas. Representan un valor que puede estar disponible ahora, en el futuro, o nunca. Una promesa tiene tres estados posibles:</p> <ol style="list-style-type: none">1. Pendiente (pending): Cuando la operación está en progreso.2. Resuelta (fulfilled): Cuando la operación se completa con éxito.3. Rechazada (rejected): Cuando ocurre un error o la operación falla.	<pre>const miPromesa = new Promise((resolve, reject) => { const exito = true; if (exito) { resolve("¡Operación completada con éxito!"); } else { reject("Hubo un error en la operación."); } }); // Usando la promesa miPromesa .then((mensaje) => { console.log(mensaje); // "¡Operación completada con éxito!" }) .catch((error) => { console.error(error); // "Hubo un error en la operación." });</pre>
<p>Encadenamiento de promesa : Las promesas se pueden encadenar para ejecutar varias operaciones en secuencia.</p>	<pre>const sumar = (a, b) => { return new Promise((resolve, reject) => { resolve(a + b); }); };</pre>

	<pre> }; sumar(3, 5) .then((resultado) => { console.log(`El resultado es: \${resultado}`); // El resultado es: 8 return sumar(resultado, 10); // Nueva promesa }) .then((nuevoResultado) => { console.log(`El nuevo resultado es: \${nuevoResultado}`); // El nuevo resultado es: 18 }) .catch((error) => { console.error(error); }); </pre>
<p>Promesas con setTimeout: Las promesas son útiles para manejar temporizadores o tareas asíncronas como peticiones a servidores.</p>	<pre> const esperar = (milisegundos) => { return new Promise((resolve) => { setTimeout(() => { resolve(`Esperado \${milisegundos} ms`); }, milisegundos); }); }; esperar(2000).then((mensaje) => { console.log(mensaje); // "Esperado 2000 ms" }).catch((error) => { console.error(error); }); </pre>
<p>Promesas en paralelo: Promise.all ejecuta varias promesas al mismo tiempo y espera a que todas se resuelvan.</p>	<pre> const promesa1 = Promise.resolve(5); const promesa2 = new Promise((resolve) => setTimeout(resolve, 2000, 10)); const promesa3 = Promise.resolve(15); Promise.all([promesa1, promesa2, promesa3]).then((resultados) => { console.log(resultados); // [5, 10, 15] </pre>

	});
<p>Promesas en paralelo: Promise.race se usa cuando se tienen varias promesas que representan tareas con diferentes tiempos de ejecución, y se quiere saber cuál se completa primero.</p>	<pre> const promesaRapida = new Promise((resolve) => { setTimeout(() => resolve("Promesa rápida resuelta en 1 segundo"), 1000); }); const promesaLenta = new Promise((resolve) => { setTimeout(() => resolve("Promesa lenta resuelta en 3 segundos"), 3000); }); const promesaMuyLenta = new Promise((resolve) => { setTimeout(() => resolve("Promesa muy lenta resuelta en 5 segundos"), 5000); }); Promise.race([promesaRapida, promesaLenta, promesaMuyLenta]) .then((resultado) => { console.log(resultado); // "Promesa rápida resuelta en 1 segundo" }) .catch((error) => { console.error(error); }); </pre>
<p>Consumo asíncrono de una api con Fetch: Fetch es un cliente Http de Javascript que nos permite comunicarnos con servidores web.</p>	<pre> fetch("https://jsonplaceholder.typicod e.com/posts/1") .then((response) => { if (!response.ok) { throw new Error("Error en la solicitud"); } return response.json(); }) .then((data) => { console.log(data); }) .catch((error) => { </pre>

```
console.error("Hubo un problema con la solicitud:", error);
});
```

Funciones asíncronas: Las funciones asíncronas en JavaScript son funciones que permiten manejar tareas asíncronas de manera más sencilla y legible utilizando las palabras clave `async` y `await`.

- **async:** Declara una función como asíncrona. Esto significa que siempre devuelve una **promesa**.
- **await:** Pausa la ejecución de la función asíncrona hasta que una promesa se resuelve o rechaza. Solo puede usarse dentro de funciones declaradas con `async`.

```
//EJEMPLO DE ENVOLVIMIENTO DE UNA PROMESA MEDIANTE async
async function obtenerMensaje() {
  return "Hola desde una función asíncrona";
}

obtenerMensaje().then((mensaje) => {
  console.log(mensaje); // "Hola desde una función asíncrona"
});

//USO DE ASYNC - AWAIT PARA CONVERTIR UN PROCESO ASINCRONO EN SINCRONO
function simularTarea() {
  return new Promise((resolve, reject) => {
    function pendiente() {
      resolve(500);
    }
    setTimeout(pendiente, 4000)
  })
}

async function tareaAsincrona() {
  console.log("Iniciando tarea...");
  const valor = await simularTarea();
  console.log("Tarea completada después de 4 segundos con valor: ", valor);
}

tareaAsincrona();
```

Gráfica que ilustra un proceso asíncrono

