En JavaScript, la asincronía y las promesas son conceptos fundamentales para manejar operaciones no bloqueantes y realizar tareas que pueden llevar tiempo, como cargar datos desde un servidor o leer un archivo.

Asincronía:

La asincronía en JavaScript se refiere a la capacidad de realizar operaciones sin detener la ejecución del programa principal. En un programa síncrono, las operaciones se ejecutan una tras otra en un orden secuencial, lo que significa que una operación debe completarse antes de pasar a la siguiente. En un programa asíncrono, las operaciones pueden ocurrir simultáneamente o en un orden no predecible, lo que permite que el programa continúe ejecutándose mientras se esperan las operaciones asincrónicas.

Ejemplos comunes de operaciones asíncronas en JavaScript incluyen las solicitudes a servidores (por ejemplo, mediante AJAX), la lectura de archivos y las funciones de temporización como **setTimeout** y **setInterva**l.

Promesas:

Las promesas son un patrón de diseño que permite trabajar con operaciones asíncronas de manera más fácil y legible. Una promesa representa un valor que puede no estar disponible aún, pero que se resolverá en el futuro, o bien, puede ser rechazada con un motivo de error.

Una promesa tiene tres estados:

**Pendiente (Pending):** Estado inicial, la promesa está esperando que se cumpla o se rechace.

**Resuelta (Fulfilled):** La operación asíncrona se completó con éxito y la promesa se resuelve con un valor.

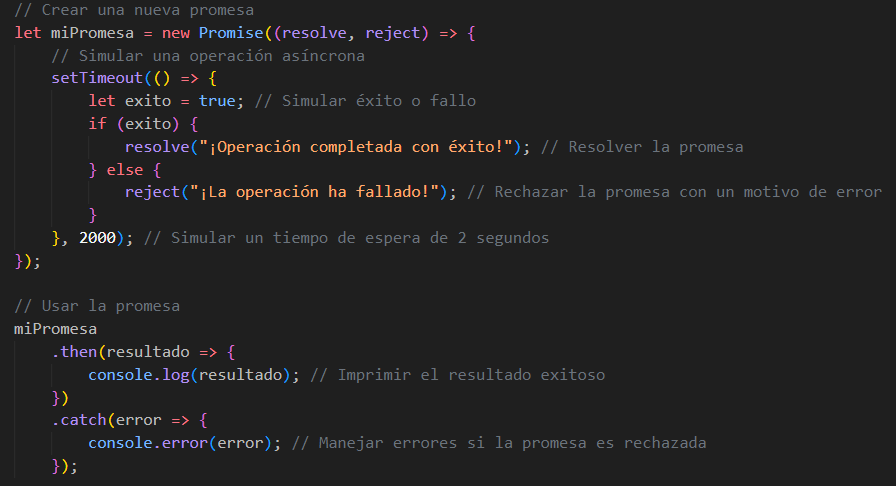
**Rechazada (Rejected):** La operación asíncrona falló y la promesa es rechazada con un motivo de error.

Las promesas tienen dos métodos principales:

**then():** Se usa para manejar el resultado exitoso de una promesa.

**catch():** Se usa para manejar errores cuando una promesa es rechazada.

Aquí hay un ejemplo de cómo se crea y se usa una promesa en JavaScript:

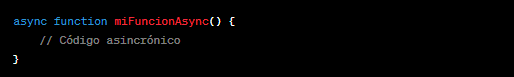


En este ejemplo, la promesa se resuelve después de 2 segundos y se imprime el mensaje de éxito. Si la promesa fuera rechazada, se manejaría el error en el método **catch()**.

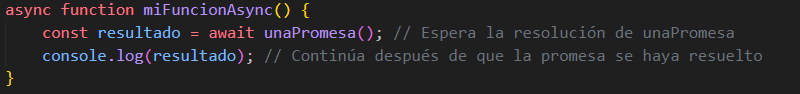
Las promesas hacen que el código asíncrono sea más fácil de leer y entender, y son la base de muchas técnicas modernas de manejo asíncrono en JavaScript, como los **async**/**await**.

**async** y **await** son características de JavaScript que simplifican aún más la escritura de código asíncrono, especialmente cuando se trabaja con promesas. Estas características se introdujeron en ECMAScript 2017 (también conocido como ES8) y han mejorado significativamente la legibilidad y el mantenimiento del código asíncrono. Aquí te explico cómo funcionan:

❶ async: La palabra clave async se utiliza para declarar una función como asincrónica. Cuando defines una función como async, esta función siempre devuelve una promesa, incluso si no contiene explícitamente una promesa. Esto facilita la gestión de la asincronía y el manejo de errores dentro de la función. Ejemplo:

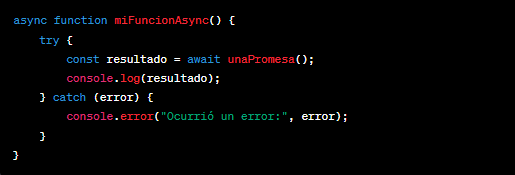


❷ await: La palabra clave await se utiliza dentro de una función async para esperar la resolución de una promesa. Cuando usas await dentro de una función async, le dices a JavaScript que suspenda la ejecución de la función hasta que la promesa se resuelva. Esto permite que el código se vea y se comporte de manera más secuencial, lo que mejora la legibilidad. Ejemplo:



Es importante destacar que **await** solo se puede usar dentro de una función declarada como **async**. Además, el uso de **await** solo tiene sentido con promesas, ya que está diseñado para esperar operaciones asincrónicas.

❸ Manejo de errores: Una ventaja importante de usar **async**/**await** es que facilita el manejo de errores en código asincrónico. Puedes usar **try**/**catch** para capturar errores cuando se rechazan las promesas. Esto hace que el manejo de errores sea más similar al manejo de errores en código síncrono. Ejemplo:



Si la promesa **unaPromesa()** se rechaza, el control se transferirá al bloque catch, donde puedes manejar el error.

***En resumen, async y await son características que hacen que el código asíncrono en JavaScript sea más fácil de leer y mantener al permitir que las operaciones asincrónicas se manejen de manera más secuencial y con un manejo de errores más claro. Esto ha mejorado en gran medida la experiencia de desarrollo en JavaScript para tareas asincrónicas.***