**Diferencia entre Callbacks y Funciones Síncronas**

|  |  |
| --- | --- |
| **Callbacks** | **Funciones Síncronas** |
| **Definición:** Es una función que se pasa como argumento a otra función y se ejecuta en respuesta a un evento o cuando una tarea asincrónica se completa. | **Definición:** Son funciones que se ejecutan inmediatamente y bloquean el flujo de ejecución hasta que se completan. |
| **Ventajas**   * Facilitan el manejo de tareas asincrónicas. * Evitan el bloqueo de código, lo que mantiene la aplicación receptiva. | **Ventajas**   * Son útiles para tareas que no involucran operaciones asincrónicas o demoras significativas |
| **Ejemplo de uso:** Si se realiza una solicitud AJAX para obtener datos, el callback se ejecutará cuando los datos estén listos, permitiendo que el resto del código siga funcionando sin interrupciones. | **Ejemplo de uso:** Cuando se realizan cálculos secuenciales, donde se necesita que una función termine antes de continuar. |

**Promesas en JavaScript**

Las promesas son objetos que representan el resultado de una operación asincrónica. Es una forma de manejar operaciones asincrónicas de manera más estructurada y legible en comparación con los callbacks tradicionales, permiten que el código se continue ejecutando sin bloquearse. El constructor de Promise toma 2 funciones callback como parámetro.

**Estados de una promesa**

* **Pendiente:** Este es el estado inicial de una promesa. La operación asíncrona no ha terminado aún y la promesa está esperando que se resuelva o se rechace.
* **Cumplida:** La operación asíncrona se ha completado exitosamente, y la promesa tiene un valor de resultado disponible. En este estado, se ejecutan las funciones proporcionadas en el método .then().
* **Rechazada:** La operación asíncrona ha fallado por alguna razón, y la promesa tiene una razón (generalmente un objeto de error) que explica el fallo. En este estado, se ejecutan las funciones proporcionadas en el método .catch().

**Creación de una Promise:** Para crear una Promise, utilizamos el constructor Promise, pasando una función que recibe dos parámetros: **resolve** (resolver) y **reject** (rechazar).

**Texto

Descripción generada automáticamente**

**Métodos de Promesas**

* **.then(onFulfilled, onRejected):** El método .then() se usa para especificar qué hacer cuando la promesa se resuelve (es decir, cuando la operación asíncrona completa con éxito) o cuando se rechaza (es decir, cuando ocurre un error). Se puede encadenar múltiples llamadas a .then() para manejar diferentes fases de la operación asíncrona.

1. **Primer parámetro 🡪 Manejo del éxito:** El primer argumento de .then() es una función de callback que se ejecuta cuando la promesa se resuelve exitosamente. Esta función recibe el resultado de la operación asíncrona como parámetro.
2. **Segundo parámetro 🡪 Manejo del error:** El segundo argumento de .then() (opcional) es una función de callback que se ejecuta cuando la promesa se rechaza. Esta función recibe el error como parámetro.
3. **Tercer parámetro 🡪 Encadenamiento:** .then() devuelve una nueva promesa, lo que permite encadenar múltiples llamadas a .then(). Esto es útil para realizar una serie de operaciones asíncronas en secuencia.

* **.catch(onRejected):** El método .catch() es esencialmente un atajo para el manejo de errores en promesas. Equivale a usar el segundo parámetro en el método .then() para manejar los errores. Por lo tanto, solo acepta una función de manejo de errores y no permite pasar más de un parámetro.
* **.finally(onFinally):** Agrega un manejador que siempre se ejecuta, independientemente del resultado de la promesa (ya sea que se haya cumplido o rechazado).

Texto

Descripción generada automáticamente

**Asincronía**

La **asincronía** en JavaScript permite que el código se ejecute sin bloquear el hilo principal. Esto es crucial en entornos como el navegador, donde queremos que la interfaz de usuario siga siendo interactiva mientras se llevan a cabo tareas en segundo plano, como solicitudes de red.

Las Promises son una de las herramientas que JavaScript utiliza para manejar operaciones asincrónicas. Otras formas incluyen **callbacks** y el evento **setTimeout**.

**Hay que tener cuidado en el manejo de formularios con asincronía.**

**Async / Await**

Son características de JavaScript que facilitan la escritura y lectura del código asincrónico. Es una forma más moderna y legible de gestionar la asincronía junto con las promesas evitando complicaciones con el uso de callbacks anidados.

**Uso de Async:** La palabra clave async se usa para declarar una función asincrónica. Esta función siempre devuelve una Promise.

**Texto

Descripción generada automáticamente**

**Uso de Await:** La palabra clave await se utiliza dentro de una función async para esperar el resultado de una Promise. La ejecución de la función se pausa hasta que la Promise se resuelva o se rechace.

**Texto

Descripción generada automáticamente**

**Manejo de Errores:** Puedes manejar errores en funciones async usando try...catch, lo cual hace que el código sea más limpio y fácil de entender.

**Texto

Descripción generada automáticamente**