▓▓▓▓▓▓▓▓▓▓▓▓ EXPLICACION DE ESTE CÓDIGO ▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓

const promise = new Promise(function (resolve, reject) {

resolve('hey!');

});

▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓

Claro, vamos a desglosar esa línea de código paso a paso:

**Código:**

javascript

Copiar código

const promise = new Promise(function (resolve, reject) {

resolve('hey!');

});

**Explicación:**

1. **const promise**:
   * Aquí estás creando una constante llamada promise para almacenar el valor que devuelve la función Promise.
   * Las constantes (const) en JavaScript son variables cuyo valor no puede ser reasignado.
2. **new Promise(...)**:
   * Promise es un constructor en JavaScript que se utiliza para crear promesas.
   * Al usar new Promise(), estás creando una nueva instancia de una promesa. Una **promesa** representa una operación asincrónica que eventualmente se completará o fallará, devolviendo un valor o un error.
   * El constructor Promise toma una función como argumento, y esa función recibe dos parámetros: **resolve** y **reject**.
3. **function (resolve, reject)**:
   * Esta es una función anónima (no tiene nombre) que se pasa al constructor de Promise.
   * **resolve** y **reject** son dos funciones que JavaScript proporciona automáticamente a la función anónima. Estas funciones son clave para manejar el éxito o el fracaso de la promesa:
     + **resolve**: Cuando se llama, marca la promesa como "resuelta" (fulfilled) y pasa un valor.
     + **reject**: Cuando se llama, marca la promesa como "rechazada" (rejected) y pasa una razón o error.
4. **resolve('hey!')**:
   * Dentro del cuerpo de la función, se llama a resolve con el argumento 'hey!'.
   * Esto significa que la promesa se "resuelve" inmediatamente con el valor 'hey!'. En este caso, la promesa ha sido exitosa y el valor resultante de la promesa será el string 'hey!'.
   * No se utiliza reject en este ejemplo, lo que significa que la promesa no falla; se resuelve correctamente.

**Analogía:**

Imagina que prometes entregar un paquete a un amigo. **resolve** es como decir "entregué el paquete con éxito", y el contenido del paquete es 'hey!'. En cambio, **reject** sería como decir "no pude entregar el paquete" y proporcionar una explicación.

**Resumen:**

Esta línea crea una **promesa** que se resuelve con el valor 'hey!' de inmediato. La promesa está lista para ser utilizada en otro momento, por ejemplo, en un .then() para obtener el valor cuando esté disponible.

▓▓▓▓▓▓▓▓▓▓▓▓ EXPLICACION DE ESTE CÓDIGO ▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓

countCows.then((result) => {

▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓

Claro, desglosaré la línea de código countCows.then((result) => { paso a paso para que puedas entender su funcionamiento:

**Código:**

javascript

Copiar código

countCows.then((result) => {

**Explicación detallada:**

1. **countCows**:
   * **Tipo**: Es una promesa. Este objeto countCows ya ha sido declarado anteriormente en el código, y seguramente está relacionado con alguna operación asincrónica (por ejemplo, contar vacas en una base de datos o algo similar).
   * **Función**: Como es una promesa, puede estar en uno de estos tres estados:
     + *Pendiente* (pending): Todavía no se ha completado ni rechazado.
     + *Resuelta* (fulfilled): La operación asincrónica se completó con éxito.
     + *Rechazada* (rejected): Hubo un error o fallo en la operación.
2. **.then(...)**:
   * **Función**: Es un método que se usa con promesas. Sirve para manejar el resultado cuando la promesa se resuelve correctamente.
   * **Sintaxis**: El método .then() recibe una función como argumento que se ejecutará cuando la promesa se resuelva.
   * **Propósito**: Indica lo que debe suceder cuando la promesa countCows se resuelva con éxito (es decir, cuando la operación asincrónica termine y devuelva un valor).
3. **(result)**:
   * **Tipo**: Es un parámetro.
   * **Función**: result representa el valor que se obtiene cuando la promesa se resuelve. Este valor es el que devuelve la promesa cuando ha terminado su ejecución satisfactoriamente.
   * **Ejemplo**: Si countCows es una promesa que cuenta vacas, entonces result podría ser el número de vacas, por ejemplo 5.
4. **=> {**:
   * **Función**: Esto es el comienzo de una función flecha. Se trata de una función anónima que se ejecutará cuando la promesa se resuelva.
   * **Propósito**: Dentro de este bloque, puedes especificar lo que quieres que ocurra cuando la promesa se haya resuelto, utilizando el valor que te devolvió, en este caso, almacenado en result.

**Resumiendo:**

* Cuando la promesa countCows se resuelve con éxito, la función pasada al método .then() se ejecuta. El valor resuelto por la promesa (el resultado de la operación asincrónica) es accesible a través del parámetro result, y dentro del bloque de código { ... }, puedes utilizar ese valor para hacer algo, como mostrarlo en la consola o manipularlo de alguna forma.

Si el código completo fuera así:

javascript

Copiar código

countCows.then((result) => {

console.log(result);

});

En este caso, cuando la promesa countCows se resuelva, imprimirá el valor de result en la consola.

¿Te ha quedado más claro el funcionamiento de esta línea de código?

▓▓▓▓▓▓▓▓▓▓▓▓ EXPLICACION DE ESTE CÓDIGO ▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓

).catch((error) => {

▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓

Claro, te explicaré la línea ).catch((error) => { paso a paso para que entiendas cómo funciona:

**Código:**

javascript

Copiar código

).catch((error) => {

**Explicación detallada:**

1. **)**:
   * Esta es simplemente el paréntesis de cierre del método .then() que has visto antes. El código anterior era algo como countCows.then((result) => { ... }), y ahora, con este paréntesis, cierras la llamada a .then().
2. **.catch(...)**:
   * **Función**: El método .catch() es parte de la estructura de promesas en JavaScript. Se utiliza para manejar errores o excepciones cuando la promesa **no se resuelve** correctamente, es decir, cuando falla o es rechazada.
   * **Propósito**: Este método es especialmente útil cuando quieres capturar lo que sucede si la operación asincrónica falla, como cuando el servidor no responde o ocurre algún error inesperado.
   * **Sintaxis**: .catch() toma una función como argumento, la cual se ejecuta cuando la promesa es rechazada (fallida).
3. **(error)**:
   * **Tipo**: Es un parámetro.
   * **Función**: error es el valor que recibe la función cuando la promesa es rechazada. Este valor representa la razón por la cual la promesa falló.
   * **Ejemplo**: Si la operación asincrónica es rechazada porque no se puede conectar al servidor, error podría ser algo como "No se pudo conectar al servidor" o algún objeto con información detallada del problema.
4. **=> {**:
   * **Función**: Esto es el comienzo de una función flecha, igual que en el .then(). Esta función anónima se ejecutará cuando la promesa sea rechazada.
   * **Propósito**: Dentro de este bloque de código { ... }, defines lo que debe ocurrir en caso de que la promesa falle. Usualmente, se utiliza para registrar el error en la consola, mostrar un mensaje al usuario, o intentar alguna otra acción de recuperación.

**Resumiendo:**

* El método .catch() se encarga de manejar los errores de la promesa. Si la promesa falla o es rechazada, el valor del error se pasa a la función que le pasamos a .catch(), y dentro de esta función puedes decidir qué hacer con ese error.

Ejemplo completo de cómo podría usarse:

javascript

Copiar código

countCows

.then((result) => {

console.log(result); // Si la promesa se resuelve

})

.catch((error) => {

console.error(error); // Si la promesa es rechazada

});

En este ejemplo, si la promesa countCows se resuelve correctamente, se ejecuta la función en el .then(), pero si algo falla (por ejemplo, no hay conexión al servidor), se ejecuta la función en el .catch(), y el error se imprime en la consola.

¿Te ha quedado claro cómo funciona .catch() y su uso en la promesa?

▓▓▓▓▓▓▓▓▓▓▓▓ EXPLICACION DE ESTE CÓDIGO ▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓

}).finally(() => console.log('Finally'));

▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓

Claro, te explicaré la línea }).finally(() => console.log('Finally')); de la misma manera que antes, desglosando cada parte para que entiendas cómo funciona.

**Código:**

javascript

Copiar código

}).finally(() => console.log('Finally'));

**Explicación detallada:**

1. **})**:
   * **Cierre**: Esta parte simplemente cierra el bloque de la función anónima que se usó en el .catch(). Antes de esta línea, teníamos algo como catch((error) => { ... }). Aquí se termina el código que maneja los errores de la promesa.
2. **.finally(...)**:
   * **Función**: El método .finally() es otro método de las promesas en JavaScript, que se usa para ejecutar código una vez que la promesa ha terminado, sin importar si se resolvió (con éxito) o fue rechazada (con error).
   * **Propósito**: Se ejecuta **después** de que la promesa haya sido manejada, es decir, después del bloque de .then() o .catch(). Es útil para realizar acciones que deben ocurrir **siempre**, independientemente del resultado, como limpiar recursos, detener una animación de carga, o registrar algún mensaje.
   * **Sintaxis**: .finally() recibe una función como argumento, que es la que se ejecutará una vez que la promesa haya finalizado.
3. **(() => ... )**:
   * **Función Flecha**: De nuevo, estamos usando una **función flecha**. En este caso, es una función que no recibe parámetros y se ejecuta automáticamente cuando la promesa ha finalizado.
   * **Propósito**: Define lo que debe ocurrir cuando la promesa ya ha terminado su ejecución (independientemente de si fue exitosa o fallida).
4. **console.log('Finally')**:
   * **Función**: Aquí usamos console.log() para imprimir un mensaje en la consola. En este caso, el mensaje será 'Finally'.
   * **Propósito**: Imprime el mensaje 'Finally' para indicar que la promesa ha finalizado, ya sea que se haya resuelto correctamente o haya fallado. En un contexto más amplio, podrías hacer tareas como liberar memoria, parar animaciones, o cerrar conexiones.

**Ejemplo completo:**

El código completo podría verse así:

javascript

Copiar código

countCows

.then((result) => {

console.log(result); // Si la promesa se resuelve

})

.catch((error) => {

console.error(error); // Si la promesa es rechazada

})

.finally(() => console.log('Finally')); // Siempre se ejecuta, pase lo que pase

**Resumiendo:**

* **.finally()** se ejecuta una vez que la promesa termina, sin importar si fue exitosa o fallida.
* El propósito es realizar tareas comunes que deben ocurrir en cualquier caso (limpieza, mensajes, detener animaciones).
* En este caso, lo único que hace es imprimir 'Finally' en la consola, lo que significa que la promesa terminó su ciclo.

Espero que te haya quedado claro cómo funciona .finally() en este contexto. ¿Hay alguna parte que quieras repasar más a fondo?

▓▓▓▓▓▓▓▓▓▓▓▓ EXPLICACION DE ESTE CÓDIGO ▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓

let xhttp = new XMLHttpRequest();

▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓

▓▓▓▓▓▓▓▓▓▓▓▓ EXPLICACION DE ESTE CÓDIGO ▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓

let xhttp = new XMLHttpRequest();

▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓