**Promesas**

¿POR QUÉ necesitas entender las Promesas JavaScript? 👇

Porque son la pieza clave para poder trabajar con operaciones **asíncronas** en JavaScript, ya que permiten empezar una acción que se mantiene en espera de un resultado, sin paralizar la ejecución del resto del código de tu programa.

Las promesas JavaScript fueron introducidas con la mega actualización ES2015 y reforzadas aún más con las recientes funciones **async.**

**Historia de una promesa**

Imagina que un cliente te encarga un nuevo proyecto. Tras pactar un resultado, la fase de desarrollo comienza.

Aunque no te hayas dado cuenta **acabas de crear una promesa y**, como todas tienen 4 partes que debes identificar.

**Acción**: algo que llevara tiempo (el desarrollo del proyecto)

**Consumidor**: quien espera el resultado de la acción (el cliente).

**Resultado**: lo que espera el consumidor a través de la acción (el proyecto finalizado)

**Estado**: la fase en la que se encuentra la acción (pendiente, cumplida o incumplida).

Ahora llevemos este ejemplo a JavaScript….

**Creación de una promesa**

Las promesas se crean construyendo un nuevo objeto Promise JavaScript.

La función (acción) que se le pasa al constructor se llama ejecutora. **Contiene el código que eventualmente debería producir el resultado** prometido.

Función ejecutora (executor)

let desarrollo = new Promise(function(...){

    //Acciones necesarias para desarrollar

   //el proyecto para el cliente

})

Con el operador new creamos un nuevo objeto de la clase Promise y lo guardamos en la variable proyecto.

Antes de llegar a un resultado, el estado de una promesa es **pendiente**(pending).

**Satisfacer una promesa**

Entregando el proyecto a tiempo habrás satisfecho la promesa con tu cliente, ¿verdad?

En JavaScript, **satisfacer una promesa se llama resolver**. Para hacerlo usamos el primer argumento que le podemos pasar a la función ejecutora: el callback resolve.

let desarrollo = new Promise(function(resolve){

    const proyecto = "algo "

    resolve(proyecto)

})

El callback resolve finaliza la promesa (acción desarrollo) satisfactoriamente y retorna el resultado(proyecto) al consumidor (cliente).

**Rechazar una promesa**

Sin embargo, si no llegas a tiempo o hay cualquier error en el proyecto, habrás incumplido tu promesa.

En JavaScript, **incumplir una promesa se llama rechazar,** para hacerlo usamos el segundo argumento de la función ejecutora: el callback reject.

let desarrollo = new Promise(function(resolve, reject){

    const error = "Proyecto incompleto"

    reject(error)

El callback reject finaliza la promesa (desarrollo) insatisfactoriamente y retorna el error al consumidor (cliente).

Al rechazar la promesa, su estado pasa de pendiente a **rechazar** (rejected).

**Consumir una promesa**

¿La promesa es el enlace entre la acción (proyecto) y el consumidor (cliente) pero, como se cómo se expresa este último en JavaScript?

Para consumir una promesa tenemos 2 métodos: **then y catch, que sirven para suscribirse a un resultado determinado.**

then se ejecuta cuando la promesa se resuelve satisfactoriamente, recibiendo el resultado enviado por resolver(proyecto).

let desarrollo = new Promise(function(resolve, reject){

   desarrollo

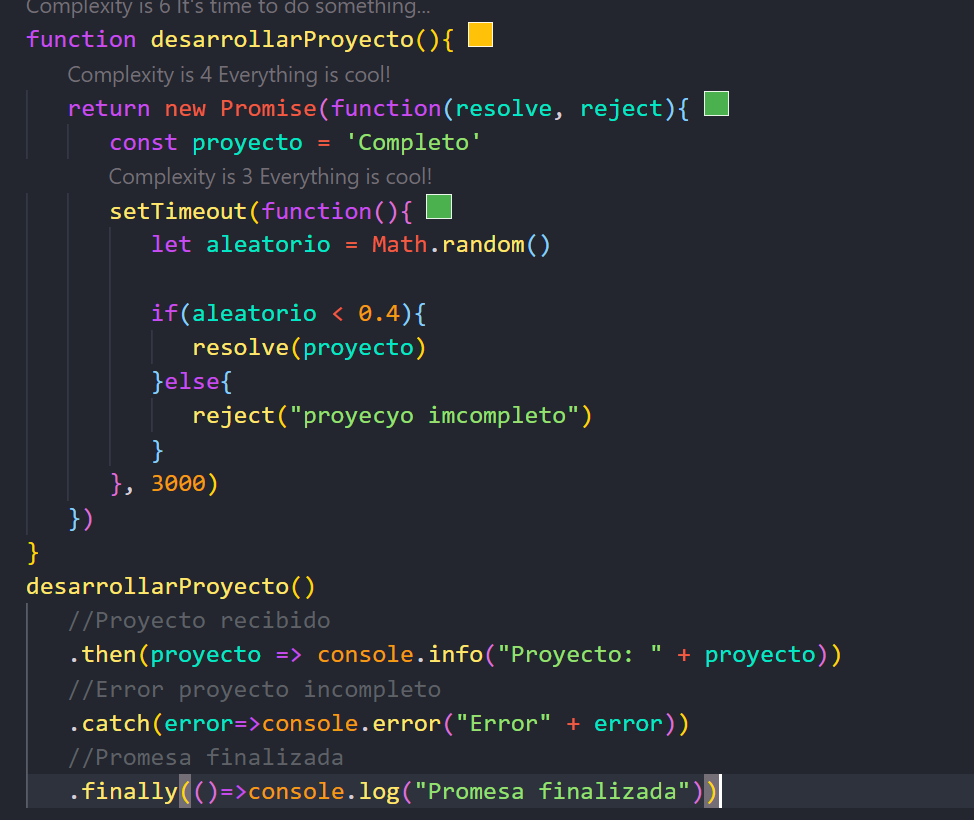
   .then(proyecto => console.info(proyecto))

   .catch(error=> console.error(error))

})

Catch se ejecuta si la promesa se rechaza, **recibiendo el error enviando por reject.**

**Ejemplo completo**



Al consumir la promesa,

Recibimos su resultado: el Proyecto o un error.

Recuerde que una promesa

Suele ser una acción asíncrona (que tarda un tiempo en finalizar).

En base al número aleatorio, resolvemos o rechazamos.

**¿Qué pasa si tenemos muchas promesas?**

**async/await**

A veces, cuando nuestro código involucra muchas operaciones asincrónicas, se nos puede armar un lio de .then() anidados, lo cual da como resultado código difícil de entender y de mantener. Es por eso que javascript permite otra forma de lidiar con promesas, evitándonos el uso de los métodos then y catch, y es con el async y await.

function desarrollarProyecto(){

   return new Promise(function(resolve, reject){

      const proyecto = 'Completo'

      setTimeout(function(){

         let aleatorio = Math.random()

         if(aleatorio < 0.4){

            resolve(proyecto)

         }else{

            reject("proyecto imcompleto")

         }

      }, 3000)

   })

}

 /\*desarrollarProyecto()

   //Proyecto recibido

   .then(proyecto => console.info("Proyecto: " + proyecto))

   //Error proyecto incompleto

Con esto ya evito poner el .then() y .catch()

   .catch(error=>console.error("Error" + error))

   //Promesa finalizada

   .finally(()=>console.log("Promesa finalizada"))

\*/

async function programa(){

   const resultado = await desarrollarProyecto()

Si queremos hacer varias funciones asincronas una tras de otra y queremos tener el resultado siempre disponible y de una forma mas limpia y más fácil .

   console.log("El resultado es: " + resultado);

}

programa();

async function programa(){

try{

   const resultado1 = await desarrollarProyecto();

No olvidarnos de try{..} catch{..}

Para que poder agarrar los errores de las promesas si no se cumplen.

   console.log("El resultado es: " + resultado1);

   const resultado2 = await desarrollarProyecto();

   console.log("El resultado es: " + resultado2);

   const resultado3 = await desarrollarProyecto();

   console.log("El resultado es: " + resultado3);

   const resultado4 = await desarrollarProyecto();

   console.log("El resultado es: " + resultado4);

} catch(error){

   console.log("lo sentimos ocurrio un error: " + error);

    }

}

programa();

/\*

[Running] node "d:\CoderHood\javascript\asincronimos\_promesas\promesas.js"

El resultado es: Completo

El resultado es: Completo

lo sentimos ocurrio un error: proyecto imcompleto

[Done] exited with code=0 in 9.241 seconds

\*/