
SENPAI
ACADEMY

Promesas

SENPAI
ACADEMY

Promesas

```
const multiplicarPositivos = function (numero1, numero2) {
  return new Promise( (resolve, reject) => {
    // Aca estamos forzando a que demore 3 segundos
    // Para simular una operación larga
    setTimeout(() => {

      if (numero1 > 0 && numero2 > 0) {
        const resultado = numero1 * numero2;
        resolve(resultado);
      }
      else {
        reject('Error, alguno de los numeros no es positivo')
      }
    }, 3000);
  })
}

multiplicarPositivos(15, -4).then((res) => {
  console.log('La promesa se resolvio con:', res);
}, (error) => {
  console.log('La promesa se fallo con:', error);
})
```

Las promesas son objetos de JavaScript que nos sirven para representar un valor que puede estar disponible **ahora**, en el **futuro**, o **nunca**.

Sintaxis:

```
new Promise( /* ejecutor */ function(resolver, rechazar) { ... } );
```

El “ejecutor” es una función que recibe 2 argumentos: *resolver* y *rechazar* (que son a su vez funciones callback).

La función “ejecutor” es ejecutada inmediatamente por la implementación de la Promesa, pasándole las funciones *resolver* y *rechazar*.

Normalmente en “ejecutor” realizamos un trabajo asincrónico y luego, una vez completado, llamamos a “resolver” o “rechazar” según el resultado.

Uso de Promesas (MDN)

Promesas

```
const multiplicarPositivos = function (numero1, numero2) {
  return new Promise( (resolve, reject) => {
    // Aca estamos forzando a que demore 3 segundos
    // Para simular una operación larga
    setTimeout(() => {

      if (numero1 > 0 && numero2 > 0) {
        const resultado = numero1 * numero2;
        resolve(resultado);
      }
      else {
        reject('Error, alguno de los numeros no es positivo')
      }
    }, 3000);
  })
}

multiplicarPositivos(15, -4).then((res) => {
  console.log('La promesa se resolvio con:', res);
})
.catch(error => {
  console.log('Se ejecuto el catch:', error);
})
```

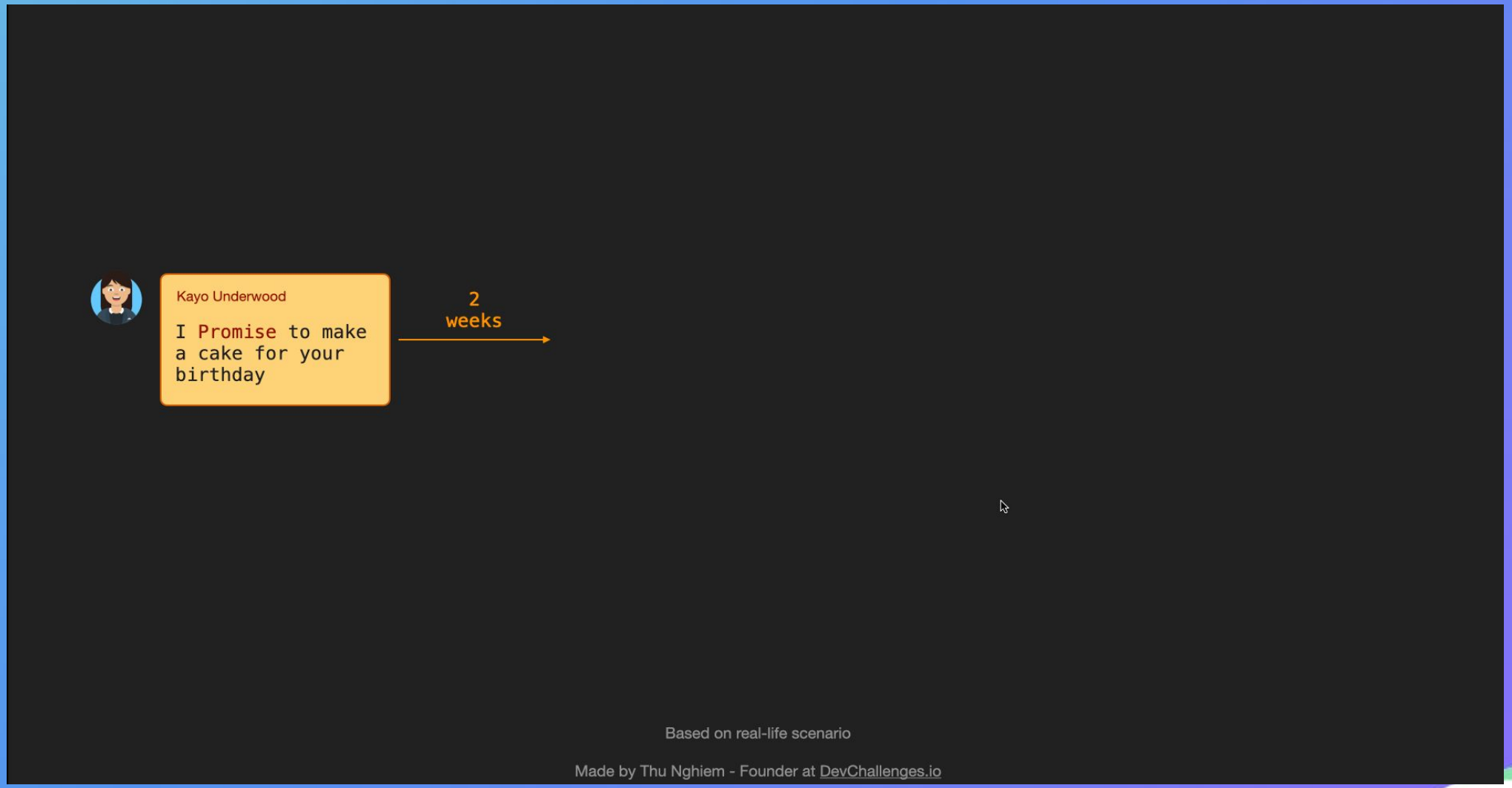
Cuando una función o método nos retorna una promesa, esa promesa se va a encontrar en uno de estos estados:

- pendiente (pending): estado inicial
- cumplida (fulfilled): significa que la operación se completó con éxito.
- rechazada (rejected): significa que la operación falló.

Cuando una promesa se completa, o se rechaza, se invoca el método asociado ***then***, y si ocurre una excepción, se invoca el método asociado ***catch*** (es importante controlar los errores)

Tanto then cómo catch retornan promesas, lo que básicamente significa que los podemos encadenar.

Promesas - Ejemplos



SENPAI ACADEMY

Promesas - Ejemplos

```
const onMyBirthday = (isKayoSick) => {
  return new Promise((resolve, reject) => {
    setTimeout(() => {
      if (!isKayoSick) {
        resolve(2);
      } else {
        reject(new Error("Estoy triste"));
      }
    }, 2000);
  });
};
```

```
// Si invocamos onMyBirthday con false:
onMyBirthday(false)
  .then((result) => {
    console.log(`Tengo ${result} tortas`);
  })
  .catch((error) => {
    console.log(error); // Esta linea no corre
  })
  .finally(() => {
    // Aparece en consola no importa si fallo o no: Party
    console.log("Party");
  });
```

```
// Si invocamos onMyBirthday con true:
onMyBirthday(true)
  .then((result) => {
    console.log(`Tengo ${result} tortas`); // Esta linea no corre
  })
  .catch((error) => {
    console.log(error); // Esto imprime en consola
  })
  .finally(() => {
    // Aparece en consola no importa si fallo o no: Party
    console.log("Party");
  });
```

SENPAI ACADEMY

async / await

Node.js - async / await

async / **await** es una sintaxis que nos va a facilitar el uso de las Promesas.

Si ponemos la palabra clave ***async*** delante de la definición de una función, automáticamente la transformamos en una función asincrónica, permitiendo que dentro utilicemos ***await*** para invocar código asíncrono.

La palabra clave ***await*** se puede poner delante de cualquier función que retorne una promesa, y esto pausa nuestro código para esperar el resultado, en esa misma línea:

```
async function main () {
  const resultado = await multiplicarPositivos(15, 3)
  console.log('resultado', resultado);
}

main();
```

SENPAI ACADEMY

Node.js - async / await

Cuando usamos **async** / **await**, no estamos bloqueando la ejecución del programa, sino que simplemente le indicamos a Node.js que las siguientes líneas de ejecución precisan esperar el resultado de la operación asíncrona.

Para manejar los errores, osea, que la promesa sea rechazada o falle, podemos envolver nuestro bloque de código con una estructura de ***try ... catch***

```
async function main () {

  try {
    const resultado = await multiplicarPositivos(15, -3)
    console.log('resultado', resultado);
  }
  catch (exception) {
    console.log('Ocurrió un error', exception);
  }

}

main();
```

SENPAI ACADEMY

Node.js - Ejercicio 1

Utilizar la librería node-fetch (deben instalarla) y escribir un pequeño programa en Node que haga un request a la Api de Github, e imprima el resultado en la consola.

Deben escribir 2 soluciones, una utilizando la sintaxis de Promesas para manejar el resultado, y otra utilizando async / await.

Endpoint de la API: https://api.github.com/users/gustavguez (pueden probar otros usuarios, como el de ustedes)

Tip: La respuesta de fetch es un stream de datos, por lo que para procesarla es necesario llamar a el método ***.json()*** que también va a retornar una promesa (como vieron en las clases de HTTP).

SENPAI ACADEMY

Node.js - Ejercicio 2

Partiendo del ejercicio 1, ahora queremos listar el nombre de todos los repositorios del usuario consultado de Github. Si se fijan, en la respuesta del perfil del usuario les retorna un url en ***'repos_url'*** => utilizar ese valor para realizar una nueva llamada con fetch y luego iterar en el resultado para mostrar solo los nombres de los repositorios.

Este ejercicio pueden completarlo con Promesas o con async / await.

SENPAI
ACADEMY

SENPAI
ACADEMY

gustavguez

gustavguez

GUSTAVO RODRIGUEZ

FULL STACK DEVELOPER
SOLCRE