

# Asincronía en JS

## Promesas, Callbacks y Async/Await

```
async function getCharacter() { // ← Aysnc/await
  const res = await fetch('https://swapi.dev/api/people')
  const character = await res.json()
  console.log(character)
}

fetch('https://swapi.dev/api/people')
  .then(res => res.json())
  .then(characterInfo => {
    console.log(characterInfo)
  })
}
```

¡Por fin las entiendo!



# Asincronía Promesas

JS

Una promesa en JavaScript son como las de la vida real. Es la expresión que tendremos un valor futuro o el fin de una tarea.

```
// ¡Te prometo que te saludaré en 2 segundos!
const saludo = new Promise((resolve, reject) => {
  setTimeout(() => {
    resolve("¡Hola, Miguel!")
  }, 2000)
})
```

En **resolve** le pasamos el valor que le llegará como **mensaje**.

```
// Cuando me saludes en 2 segundos,
// entonces, yo te saludaré de vuelta
saludo.then(mensaje => {
  console.log('¡Hola, Baby Yoda!')
})
```

¡LAS PROMESAS SÓLO SE RESUELVEN UNA VEZ!



# Asincronía

## Promesas con Await

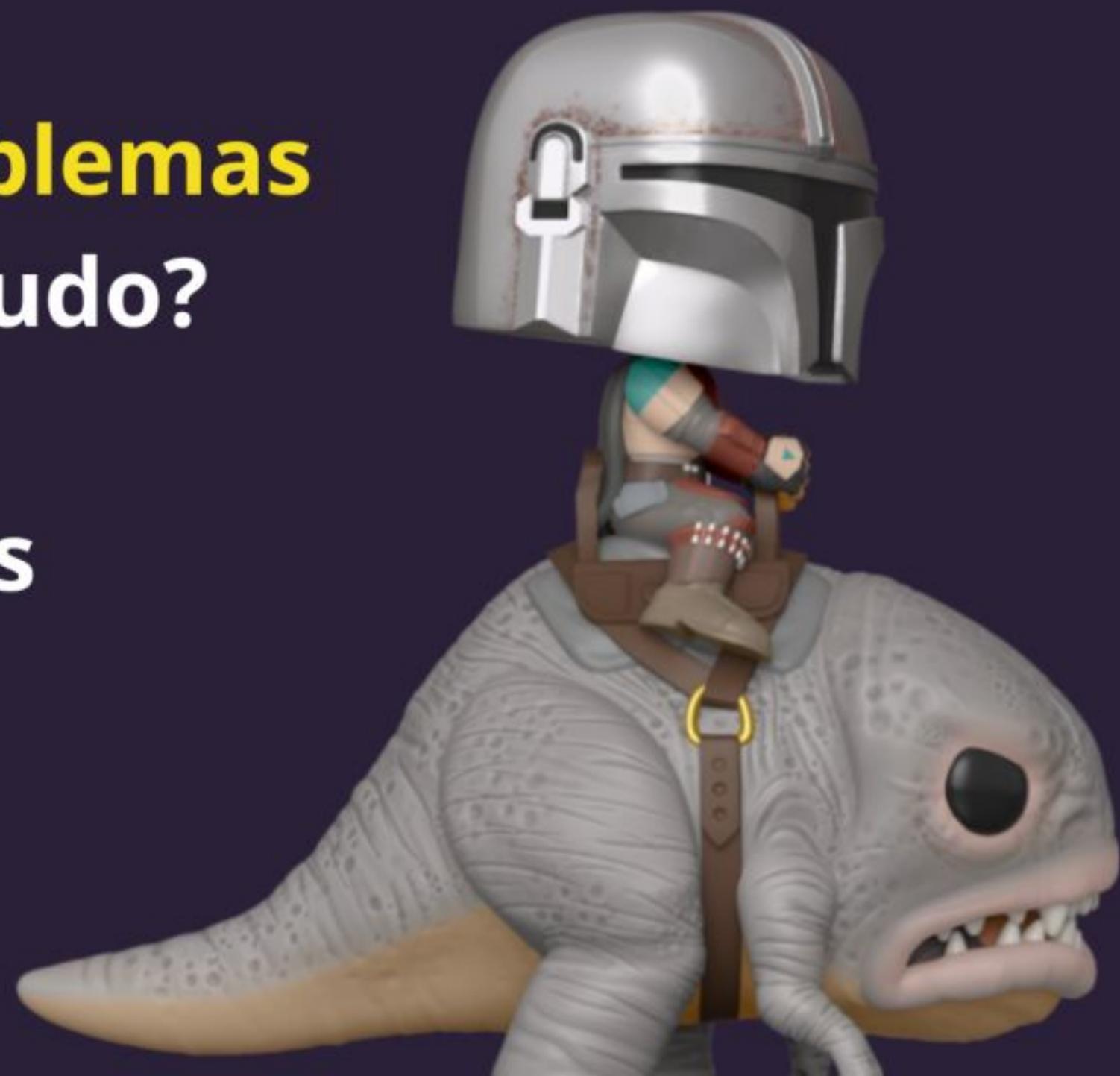
JS

Podemos esperar las promesas usando la palabra reservada **await** si estamos en el nivel superior de un módulo de JavaScript.

```
// Voy a esperar hasta que me salude
await saludo
// Han pasado 2 segundos y me ha saludado
console.log('¡Hola, Baby Yoda!')
```

¿Y qué pasa si hay **problemas** para mandarnos el saludo?

A ver cómo manejamos los **errores**...



# Asincronía Error en Promesas

JS

A veces, las **Promesas** no se cumplen como esperábamos por **problemas**:

```
const saludo = new Promise((resolve, reject) => {
  setTimeout(() => {
    // A los 5 segundos, me di cuenta de un error
    reject("Estoy sin batería")
  }, 5000)
})
```

R2D2 no pudo cumplir la promesa.  
No la resolvió y la rechazo.

```
saludo // Estoy pendiente de tu saludo ...
// Cuando llegue, te saludaré de vuelta
.then(mensaje => {
  console.log('Hola, R2D2!')
})
// Ah, parece que no pudiste cumplir lo
// que prometiste. ¡A ver qué pasó!
.catch(error => {
  // error → "Estoy sin batería"
})
```



# Asincronía Error con `await`

JS

Para manejar un error con `await` debemos envolver la promesa con un `try/catch`.

```
try {  
    // Voy a esperar hasta que me salude ...  
    await saludo  
    // ¡Ya llegó el saludo!  
    console.log('¡Hola, R2D2!')  
} catch (error) { // A ver si hay algún error ...  
    // error → "Estoy sin batería"  
}
```

Recuerda manejar los errores de las promesas.

Como en la vida real,  
no siempre todo  
sale como esperamos.



# Asincronía **async/await**

JS

Para poder usar **await** dentro de funciones,  
debemos indicar que son funciones asíncronas.

```
async function esperarSaludo () {  
  try {  
    await saludo  
    console.log('¡Hola, R2D2!')  
  } catch (error) {  
    console.error(error)  
  }  
}  
  
// También en arrow functions  
const esperarSaludo = async () => {  
  await saludo  
  console.log('Hey, hey!')  
}
```



# Asincronía Callbacks

JS

Un **callback** es una función que se pasa como argumento a otra función y se llama dentro de esa función.

La idea es poder ejecutar una función cuando se complete una tarea.

¡Esto es un callback!



```
element.addEventListener('click', function () {  
    // Esta función es un callback que  
    // se ejecuta cada vez que el usuario  
    // hace clic en un elemento del DOM  
})
```



# Asincronía Callbacks

JS

**Cuando los callbacks responden a un evento pueden ser llamados más de una vez.**

**También es la forma más antigua de manejar operaciones asíncronas en JavaScript.**

```
fs.readFile('archivo.txt', 'utf-8', (err, content) => {  
    // este callback se ejecuta sólo una vez  
    // al terminar de leer el archivo.txt  
    // por lo tanto, responde a una  
    // operación asíncrona  
})
```

**Ahora Node.js está usando las Promesas para este tipo de operaciones.**



# Asincronía Callbacks

JS

¡Los callbacks se usan más veces de lo que piensas en JavaScript!

En este código hay 2 callbacks...

```
const saludo = new Promise((resolve, reject) => {
  setTimeout(() => {
    resolve("¡Hola, Miguel!")
  }, 2000)
})
```

↑  
iToda la función que pasamos al setTimeout!

```
// Cuando me saludes en 2 segundos,
// entonces, yo te saludaré de vuelta
saludo.then(mensaje => {
  console.log('¡Hola, Baby Yoda!')
})
```

↑  
Sí, técnicamente lo que pasamos al .then de una Promesa... ¡es un callback!