

TEMA 10:

Promesas y fetch

A. PROMESAS

1. Punto de partida

- Tenemos una función en JavaScript que devuelve un objeto

```
function datos() {  
  const obj={  
    nombre:"Yo",  
    email:"daf@as.com"  
  };  
  return obj;  
}
```

1. Punto de partida

- Podemos hacer un `console.log` de esa función sin problema y nos devolverá el objeto.

```
console.log(datos());
```



```
▼ {nombre: 'Yo', email: 'daf@as.com'} ⓘ  
  email: "daf@as.com"  
  nombre: "Yo"  
  ► [[Prototype]]: Object
```

- Hasta aquí todo bien ...

2. Problema

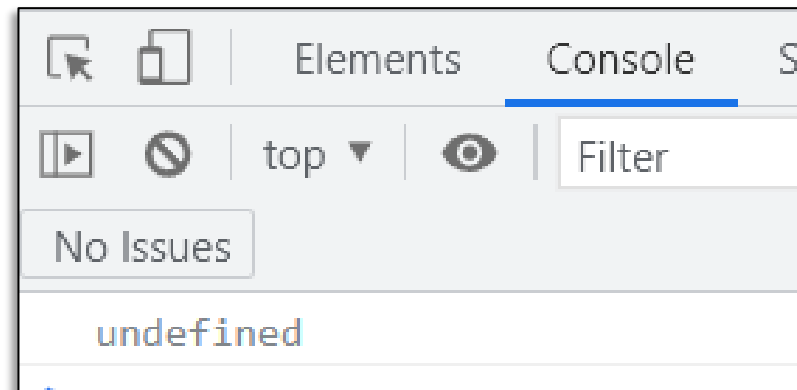
- Puede ser que la página que nos devuelva el resultado (que está en Internet) tenga algo de retardo (que es lo más común).
- Podemos simular un retardo de 3 ms introduciendo la función **setTimeout**.

```
function datos() {  
  setTimeout(function () {  
    const obj = {  
      nombre: "Yo",  
      email: "daf@as.com"  
    };  
    return obj;  
  }, 3); // retraso de 3 ms  
}
```

2. Problema

- A pesar de ser un retraso relativamente pequeño, JavaScript va a toda mecha y no se espera a que esté el resultado.
- Por tanto, si repetimos el `console.log` de dicha función (igual que en el primer ejemplo) veremos que JS no ha esperado a que esté listo el resultado y nos devuelve un valor **undefined**.

```
console.log(datos());
```

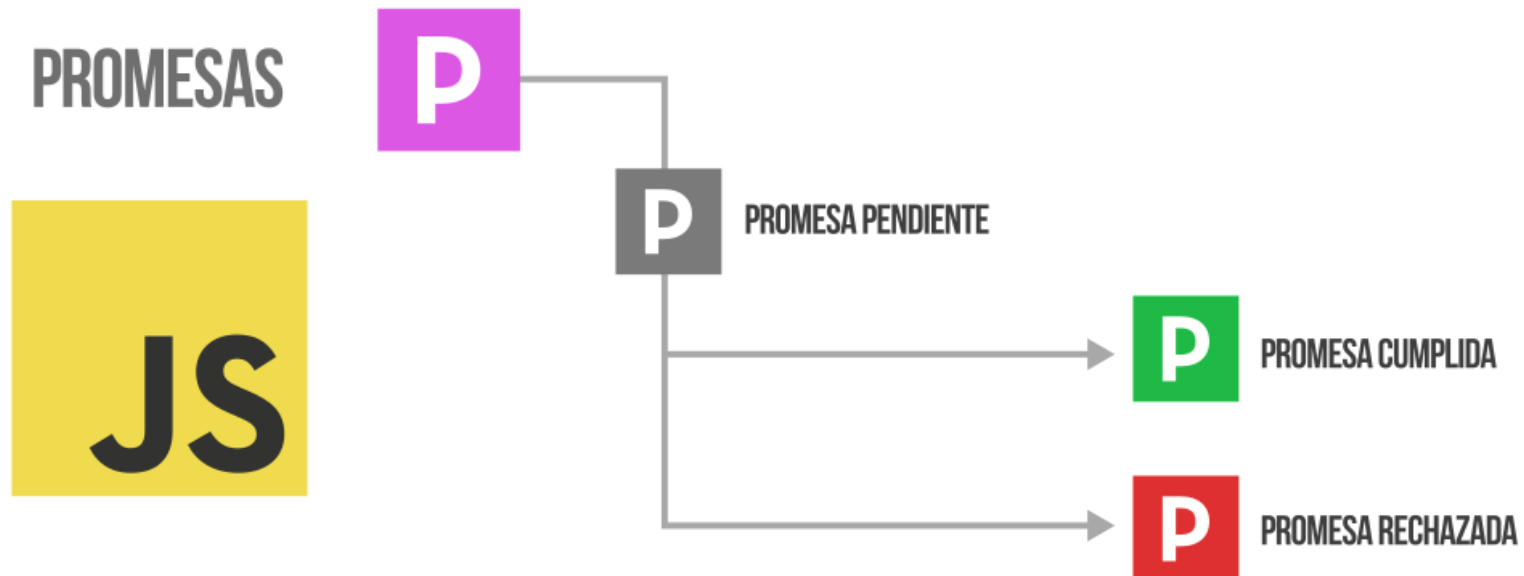


2. Problema

- Si esto pasa en local, con un retraso de 3 ms podemos imaginar que sucederá con paginas muy saturadas, por ejemplo: la API de *Star Wars* hay veces que tarda entre 400 y 500 ms en proporcionar el resultado.

3. Solución: Promesas

- Como su propio nombre indica, una **promesa** es algo que, en principio pensamos que se cumplirá, pero en el futuro pueden ocurrir varias cosas:



3. Solución: Promesas

- La promesa **se cumple** (promesa resuelta)
- La promesa **no se cumple** (promesa se rechaza)
- La promesa se queda en un **estado incierto** indefinidamente (promesa pendiente)

3. Solución: Promesas

- Las **promesas** en JavaScript se representan a través de un Objeto, y cada **promesa** estará en un estado concreto: **pendiente**, **aceptada** o rechazada.
- Además, cada **promesa** tiene los siguientes métodos, que podremos utilizar para utilizarla:

3. Solución: Promesas

Métodos	Descripción
.then (function resolve)	Ejecuta la función callback resolve cuando la promesa se cumple.
.catch(function reject)	Ejecuta la función callback reject cuando la promesa se rechaza.
.then(function resolve, function reject)	Método equivalente a las dos anteriores en el mismo .then() .
.finally(fucntion end)	Ejecuta la función callback end tanto si se cumple como si se rechaza.

3. Solución: Promesas

- Vamos a reconvertir nuestro código en promesas. Partimos de la función original.

```
function datos() {  
  setTimeout(function() {  
    const obj={  
      nombre: "Yo",  
      email: "dfa@asdf.com"  
    };  
    return obj;  
  }, 3);  
}
```

3. Solución: Promesas

- Nuestra función devolverá una Promesa de ese dato, dentro de esa promesa irá todo el código de la anterior función.

```
function datos() {  
  return new Promise((resolve, reject) => {
```

- Nuestra promesa tiene dos estados: **resolve** (cumplida) o **reject** (no cumplida).

3. Solución: Promesas

- La función flecha es similar a una anónima con dos parámetros.

```
function datos() {  
  return new Promise(function (resolve, reject) {
```

3. Solución: Promesas

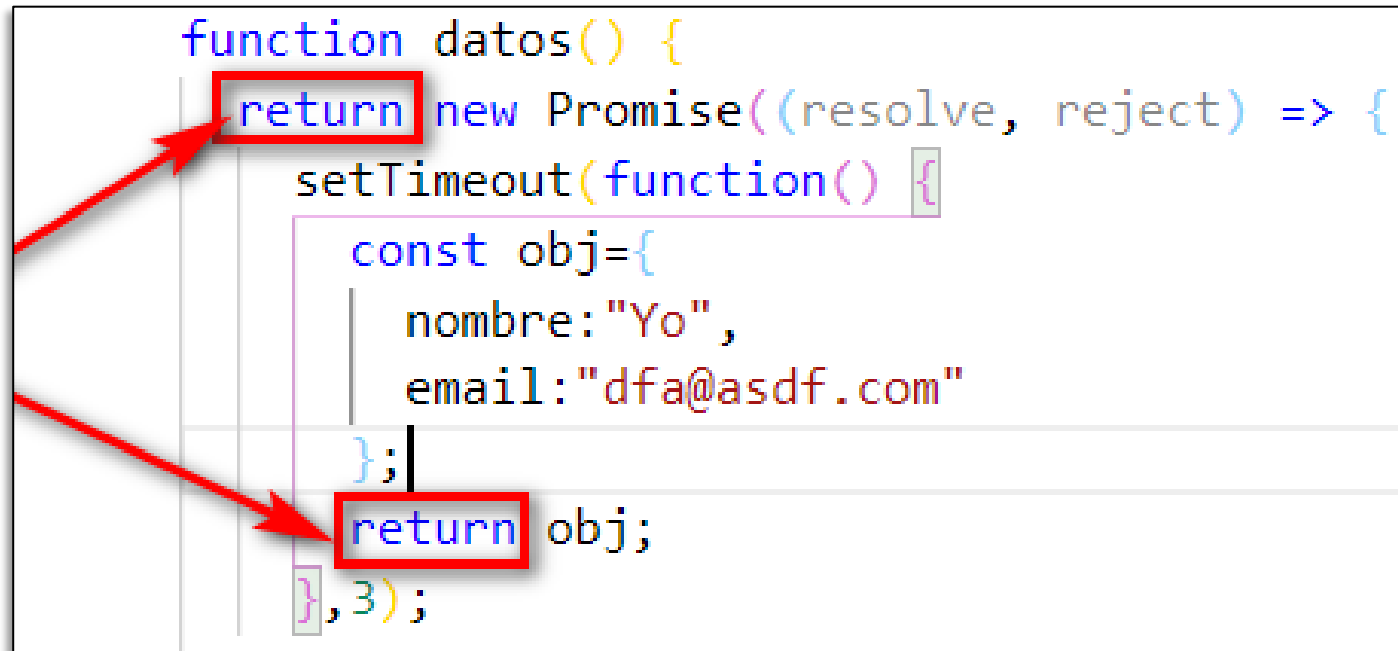
- Insertamos el código original de la función dentro de la promesa (apañando llaves { } y demás)

```
function datos() {  
  return new Promise((resolve, reject) => {  
    setTimeout(function() {  
      const obj={  
        nombre:"Yo",  
        email:"dfa@asdf.com"  
      };  
      return obj;  
    },3);  
  });  
}
```

3. Solución: Promesas

- En la función ahora tenemos el **return** de la promesa y un segundo **return** (de la función original)

```
function datos() {  
  return new Promise((resolve, reject) => {  
    setTimeout(function() {  
      const obj={  
        nombre:"Yo",  
        email:"dfa@asdf.com"  
      };  
      return obj;  
    }, 3);  
  });  
}
```



3. Solución: Promesas

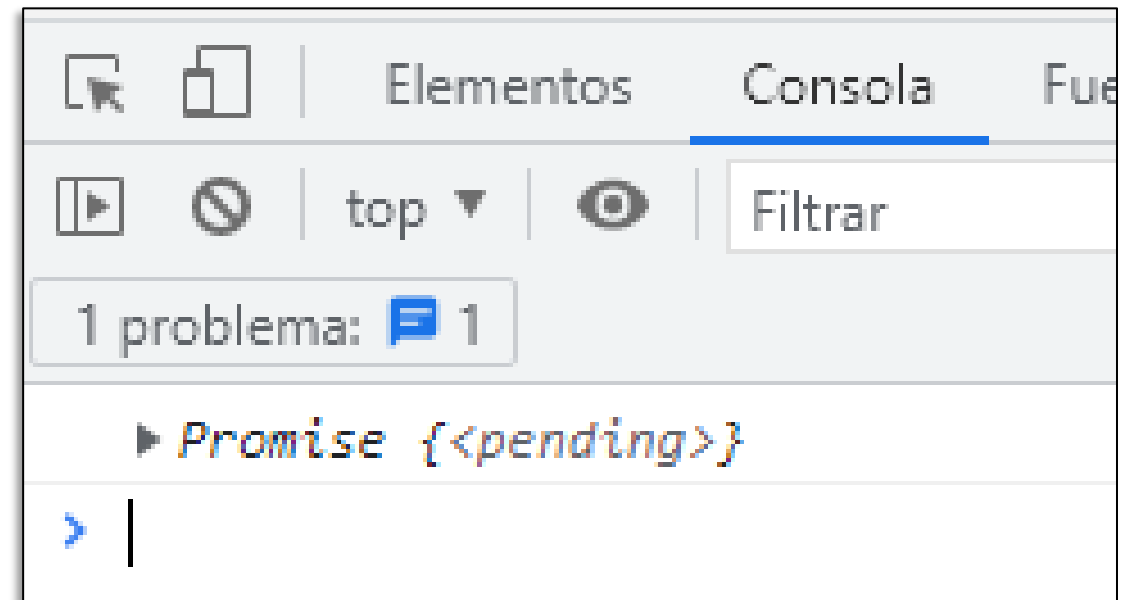
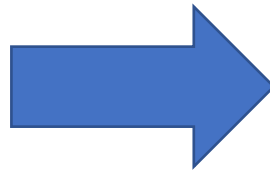
- Ese segundo **return** lo debemos cambiar por **resolve()** (promesa cumplida).
- Con eso tendríamos la función preparada.

```
function datos() {  
  return new Promise((resolve, reject) => {  
    setTimeout(function() {  
      const obj={  
        nombre:"Yo",  
        email:"dfa@asdf.com"  
      };  
      resolve(obj);  
    },3);  
  });  
}
```


3. Solución: Promesas

- Ahora, notaremos que el valor de `datos()` ha cambiado, hemos pasado de **undefined** a **Promise {<pending>}**

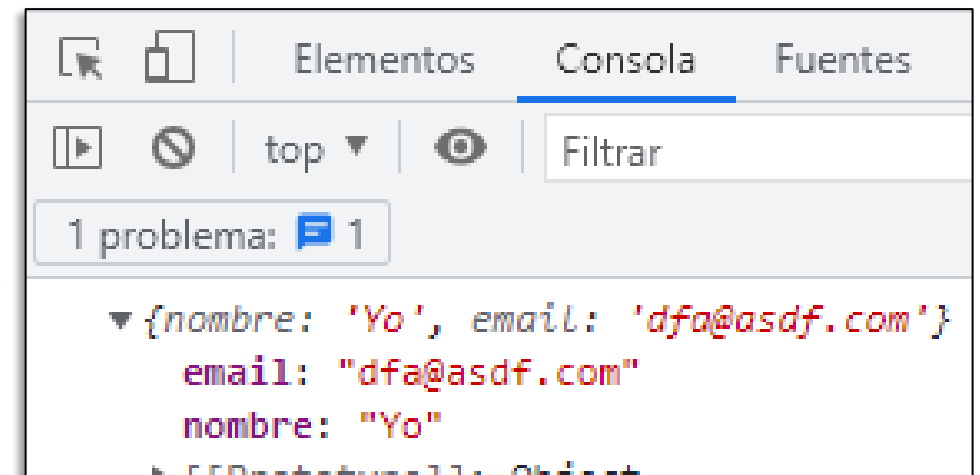
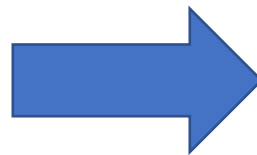
```
console.log(datos());|
```



3. Solución: Promesas

- Para obtener el valor deseado necesitamos resolver la promesa de la función `datos()` para ello debemos llamar a la función con el método **.then**
- **.then** incorporará una función anónima donde el parámetro que recibe es el valor devuelto por la función una vez satisfecha la promesa.

```
datos().then(function (valor) {  
  console.log(valor);  
})
```



3. Solución: Promesas

- Generalmente esta función incluida en el `.then` nos la encontraremos con una función flecha (ya que es mucho más corta de escribir).

```
datos()  
.then(valor => console.log(valor));
```