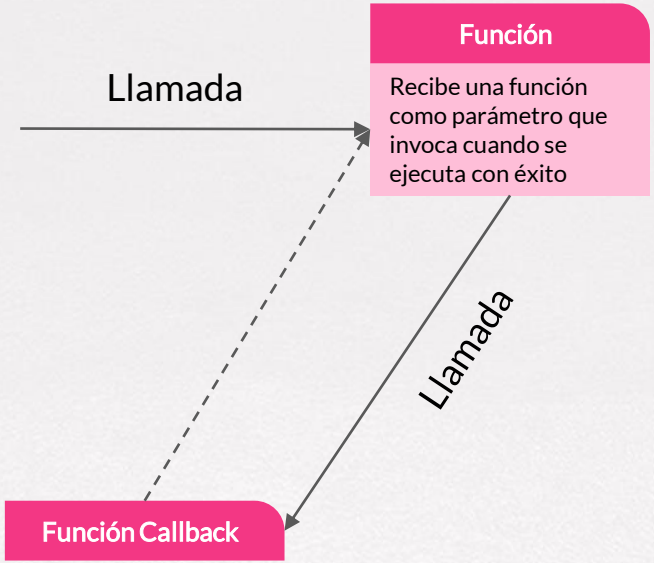
Curso de JavaScript avanzado

Promesas y asincronía

Callbacks y errores

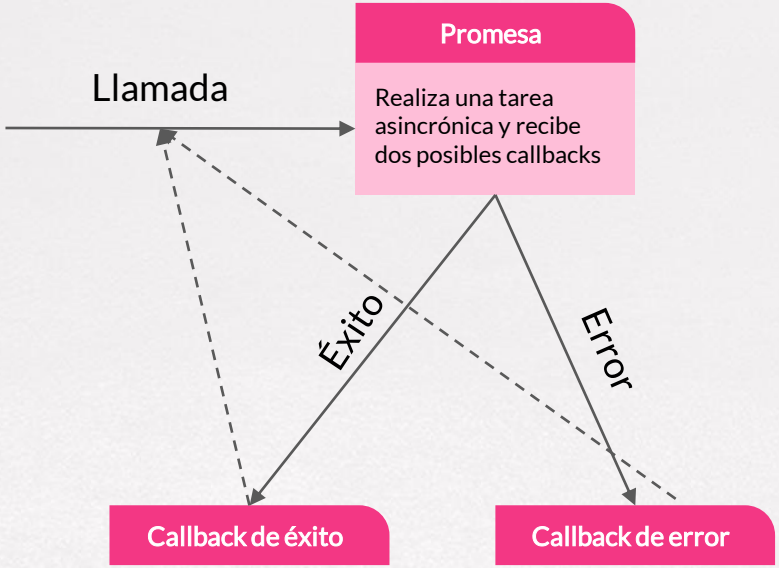
ÍNDICE

Manejadores y callbacks



- Un callback es una función que se provee como parámetro y **es llamada por la función objetivo al completar su tarea con éxito**
- Los callbacks pueden ser **sincrónicos o asincrónicos**
- Los callbacks se utilizan a menudo en operaciones sincrónicas **para continuar tareas**

Uso de callbacks en promesas



- En una promesa una función callback **nunca es llamada antes de que acabe el bucle de eventos**
- Los callbacks pueden ser **sincrónicos o asincrónicos**
- El manejador then() de la promesa será llamada independientemente del rechazo **a menos que se disponga de un bloque catch()**

Uso de callbacks en promesas

```
const miFuncion = (val) => {
  return new Promise((resolve, reject) => {
    if (val) {
      resolve('El valor es true!')
    } else {
      reject('El valor es false!')
    }
  });
}
```

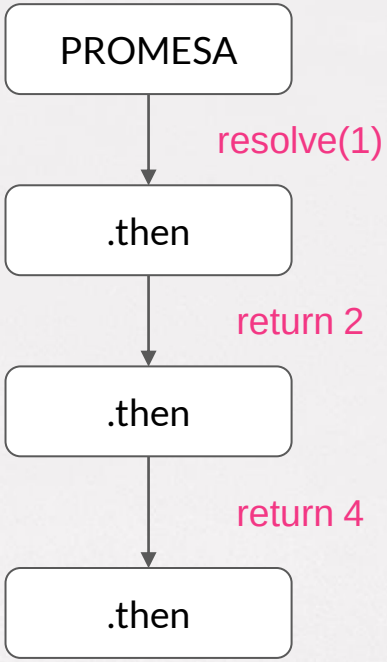
```
const funcExito = (res) => {
  console.log(res)
};
const funcError = (res) => {
  console.error(res)
};

miFuncion(true).then(funcExito, funcError);
// Logueará el mensaje de exito por console.log
miFuncion(true).then(funcExito).catch(funcError);
// Logueará el mensaje de exito por console.log

miFuncion(false).then(funcExito, funcError);
// Logueará el mensaje de error por console.error
miFuncion(false).then(funcExito).catch(funcError);
// Logueará el mensaje de exito por console.error
```

Encadenamiento de promesas

- Cada uso del manejador o handler **then()** devuelve una nueva promesa **con el resultado que haya sido devuelto en el callback anterior**
- Por defecto al devolver un valor en el callback del manejador, este será el valor que se resuelva en la promesa retornada
- También es posible retornar una promesa explícitamente, **lo que hará que todos los manejadores subsiguientes esperen a su resolución**

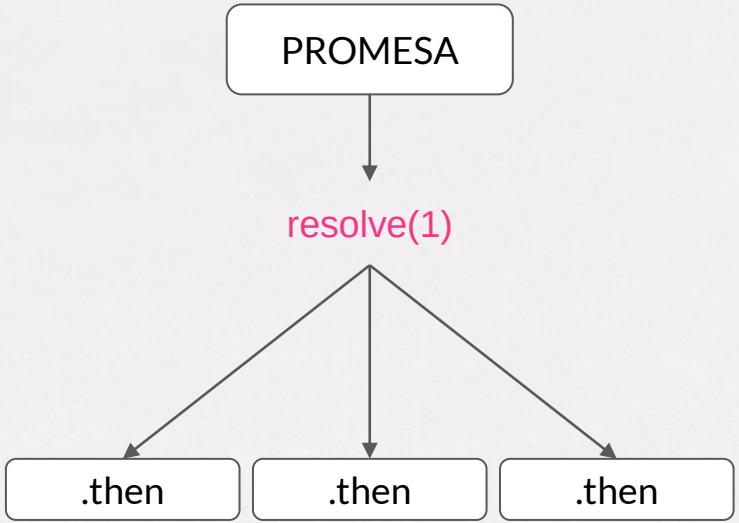
Encadenamiento de promesas

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

  setTimeout(() => resolve(1), 1000);

}).then((result) => {
  alert(result); // 1
  return result * 2;
}).then((result) => {
  alert(result); // 2
  return result * 2;
}).then((result) => {
  alert(result); // 4
  return result * 2;
});
```



Encadenamiento de promesas

```
let promise = new Promise(function(resolve,
reject) {
  setTimeout(() => resolve(1), 1000);
});
promise.then((result) => {
  alert(result); // 1
  return result * 2;
});
promise.then((result) => {
  alert(result); // 1
  return result * 2;
});
promise.then((result) => {
  alert(result); // 1
  return result * 2;
});
```



Encadenamiento de promesas

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

  setTimeout(() => resolve(1), 1000);

}).then((result) => {
  alert(result); // 1
  return new Promise((resolve, reject) => {
    setTimeout(() => resolve(result * 2), 1000);
  });
}).then((result) => {
  // Este manejador esperará 1 segundo a
  // ejecutarse
  alert(result); // 2
  return result * 2;
}).then((result) => {
  alert(result); // 4
  return result * 2;
});
```

- Permitir devolver promesas de manera explícita nos habilita **para crear cadenas de acciones asíncronas**

- Esto es muy útil en el desarrollo diario para, por ejemplo, **usar fetch a la hora de pedir recursos relacionados**

Encadenamiento de promesas

```
fetch('/user.json')
  .then((response) => response.json())
  .then((user) =>
fetch(`https://api.github.com/users/${user.name}`))
  .then((response) => response.json())
  .then((githubUser) => {
    let img = document.createElement('img')
    img.src = githubUser.avatar_url
    img.className = 'promise-avatar-example'
    document.body.append(img)
    setTimeout(() => img.remove(), 3000) // (*)
  })
```

Errores y gestión

```
fetch('/user.json')
  .then((response) => response.json())
  .then((user) =>
fetch(`https://api.github.com/users/${user.name}`))
  .then((response) => response.json())
  .then((githubUser) => {
    let img = document.createElement('img')
    img.src = githubUser.avatar_url
    img.className = 'promise-avatar-example'
    document.body.append(img)
    setTimeout(() => img.remove(), 3000) // (*)
  })
  .catch(error => alert(error.message));
```

- El catch en las promesas no sólo gestiona los reject explícitos, **también actúa como un try...catch**
- Cualquier **throw** de error dentro de cualquier manejador **then** ocasionará el rechazo de la promesa y será capturado por el manejador **catch**

Errores y gestión

```
new Promise((resolve, reject) => {
  throw new Error("Error!");
}).catch((error) => {
  alert("El error ha sido manejado con éxito");
}).then(() => alert("Este manejador se ejecuta"));
```

- Un error capturado por el bloque catch se considerará manejado si este manejador no lanza a su vez un nuevo error, y **continuará la ejecución**

Errores y gestión

```
new Promise((resolve, reject) => {

  throw new Error("Error!");

}).catch((error) => {

  throw error;

}).then(() => alert("Este manejador no se ejecuta"))
.catch((error) => {
  alert("Este manejador se ejecuta con el error")
})
```

- Un manejador de error que lance un nuevo error será manejado por el siguiente **catch** más cercano

PARA RESUMIR

- Un callback es una función como parámetro **llamada por la función objetivo al completar su tarea con éxito** siempre que haya finalizado el bucle de eventos
- El uso del manejador then ejecuta el callback dado e implícita o explícitamente **devuelve una promesa que el resto de manejadores esperarán a ver resuelta**
- **Los manejadores then y catch actúan como un bloque try...catch** que abarca a todo el encadenamiento que quiera hacerse en la promesa