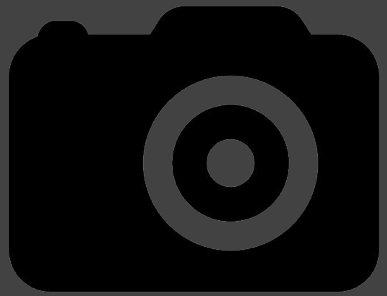
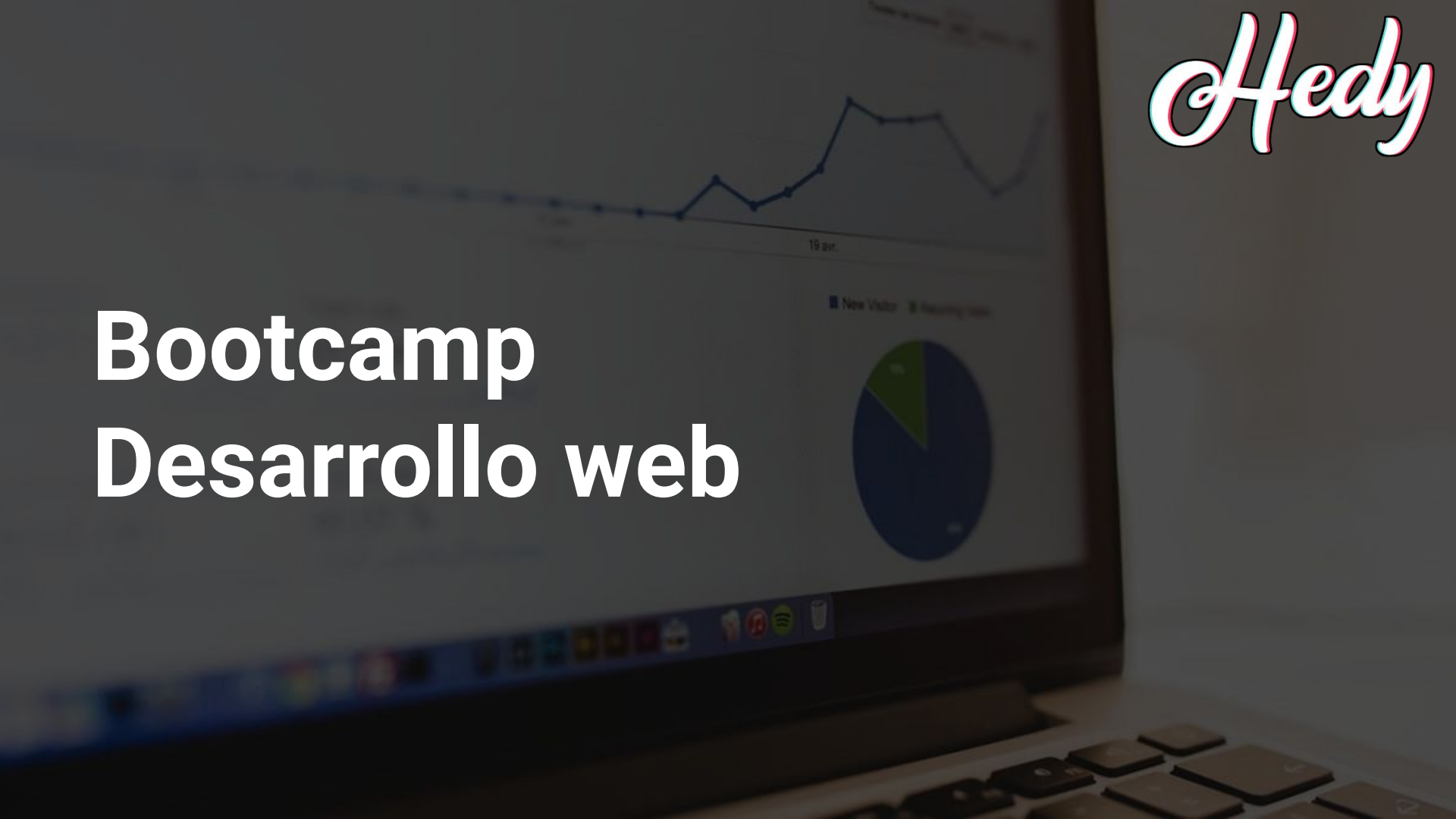
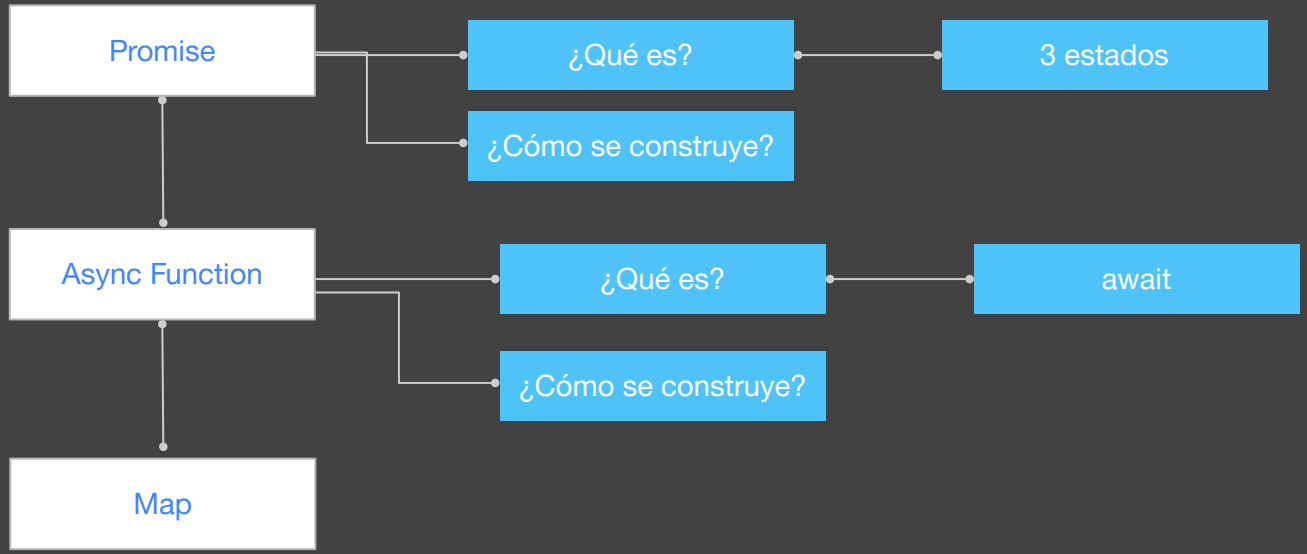
Hedy

Recordá poner a grabar la clase




Hedy
Bootcamp
Desarrollo web

Mapa conceptual clase 7



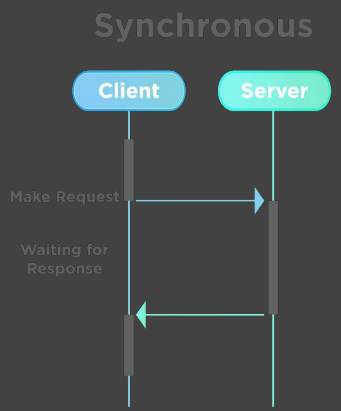
```
fetchPromise
  .then((response) => {
    if (!response.ok) {
      throw new Error(`HTTP error: ${response.status}`);
    }
    return response.json()
  })
  .then((data) => {
    console.log(data[0].name);
  })
  .catch((error) => {
    console.error(`Could not get products: ${error}`);
```

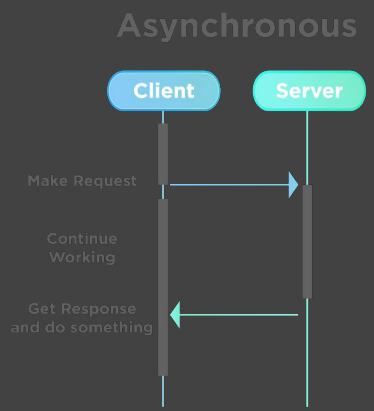
Hedy

Promise

JS Promise

.isFulfilled .isPending
.isRejected





El objeto **Promise** (promesa) representa la eventual finalización o falla de una operación asincrónica y su valor resultante.

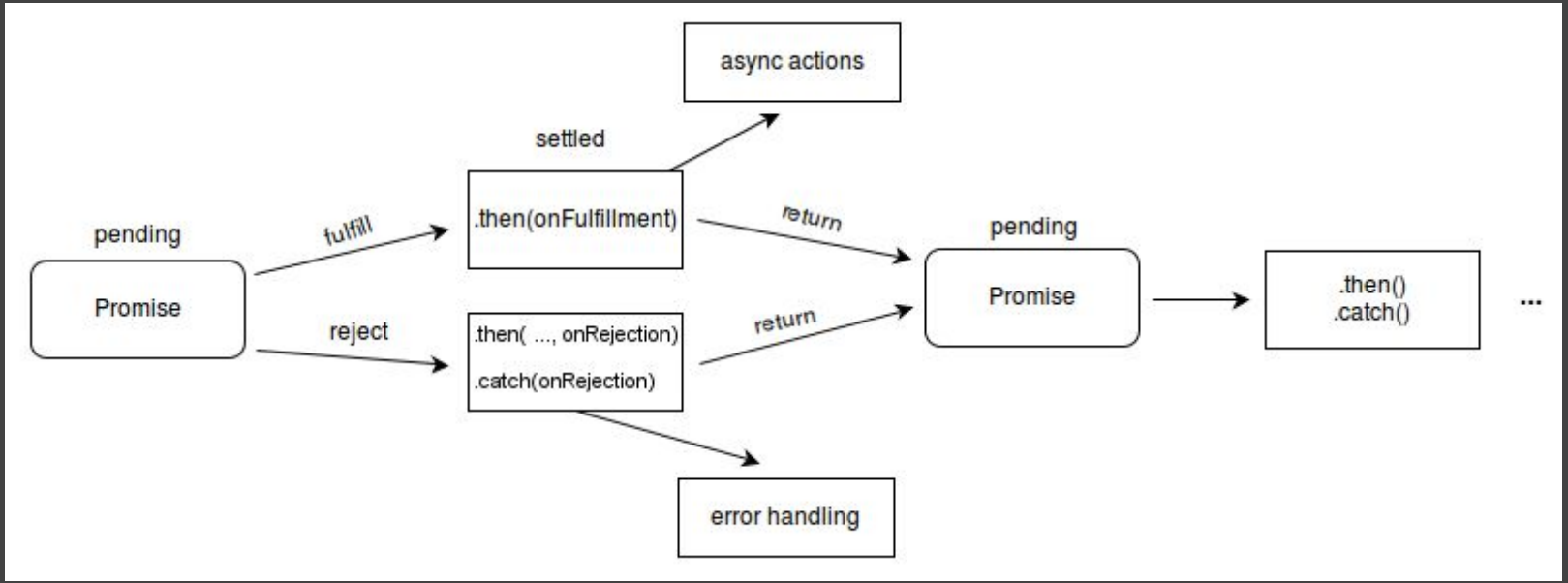
En lugar de devolver inmediatamente el valor final, el método asíncrono devuelve la **promesa** de proporcionar el valor en algún momento en el futuro.

3 estados posibles:

- ***pending*** (pendiente): estado inicial, ni cumplido ni rechazado.
- ***fulfilled*** (cumplida): lo que significa que la operación se completó con éxito.
- ***rejected*** (rechazada): lo que significa que la operación falló.

Una promesa pendiente puede *cumplirse* con un valor
o *rechazarse* con un motivo (error).

Cuando ocurre cualquiera de estas opciones, se llama a los controladores asociados en cola por el método **.then** de una promesa.



Ejemplo:

Hedy

En este ejemplo creamos una promesa, le decimos que es lo que tiene que hacer cuando tiene éxito, y lo que tiene que hacer cuando falla.

Las promesas son muy buenas herramientas cuando tenemos que hacer algo que va a tomar mucho tiempo, cómo descargar una imagen, y lo queremos pasar a segundo plano en vez de hacer que el resto del script espere hasta que la descarga termine.

```
let p = new Promise((resolver, rechazar) => {
let a = 1 + 3;
if (a == 2) {
resolver('- Exito');
}
else{
rechazar('- Fracaso')
}
})

p.then((mensaje)=> {
    console.log('Este mensaje esta en el then ' + mensaje);

}).catch((mensaje)=> {
    console.log('Este mensaje esta en el catch ' + mensaje);
})
```

```
async function loadData() {
  let data = {};
  const api = new FakeAPI();

  data.customer = await api.findCustomer('James', 'Brown');
  data.orders = await api.getOrders(data.customer.id);
  data.mostRecentOrder = data.orders[0];
  data.mostRecentOrder.status = await api.getOrderTrackingStatus(
    data.mostRecentOrder.id
  );
  console.log(data);
}
loadData();
```

Async & Await

Hedy

La palabra clave **async** convierte a una función en **asíncrona**, la cual devuelve una **promesa**.

- Cuando la función **async** devuelve un valor, la promesa se **resolverá** con el valor devuelto.
- Si la función **async** genera una excepción o error, la promesa se rechazará.

Función asíncrona

```
async function myFunction() {
  return "Hello";
}
```

=

Promesa

```
function myFunction() {
  return Promise.resolve("Hello");
}
```

Una función **async** puede contener una expresión **await**, la cual pausa la ejecución de la función asíncrona y espera la resolución de la promesa pasada y, a continuación, reanuda la ejecución de la función async y devuelve el valor resuelto.

```
let value = await promise;
```

La palabra clave **await** solamente se puede usar dentro de una función **async**

La finalidad de las funciones async/await es simplificar el comportamiento del uso síncrono de promesas.

Reescritura de una cadena de promesas con una función async

Función asíncrona

```
async function getProcessedData(url) {
  let v;
  try {
    v = await downloadData(url);
  } catch(e) {
    v = await downloadFallbackData(url);
  }
  return processDataInWorker(v);
}
```

Promesa

```
function getProcessedData(url) {
  return downloadData(url) // returns a promise
    .catch(e => {
      return downloadFallbackData(url)  // returns a
    })
    .then(v => {
      return processDataInWorker(v); // returns a pr
    });
}
```

```
const mathScores = [39, 50, 45, 41, 50];

mathScores.map((currentValue, index, array) => {
    console.log('Current value:' + currentValue);
    console.log('Index:' + index);
    console.log('Array:' + array);
    return currentValue;
})
```

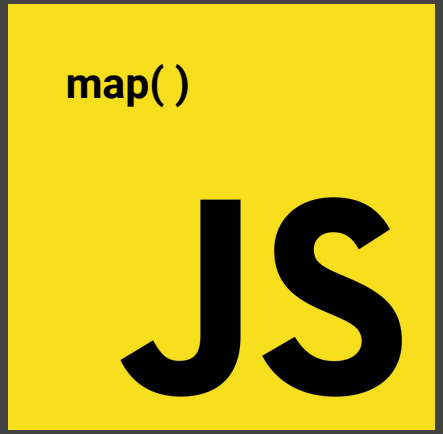
Hedy

Map


```
const mathScores = [39, 50, 45, 41, 50];

mathScores.map((currentValue, index, array) => {
    console.log('Current value:' + currentValue);
    console.log('Index:' + index);
    console.log('Array:' + array);
```

El método map() nos permite generar un nuevo array tomando de base otro array y utilizando una función transformadora.



Ejemplos:

Creamos un nuevo arreglo y le asignamos todos los números del arreglo números pero aplicándoles la función sqrt del objeto map (raíz cuadrada).

```
const numeros = [49, 16, 64, 81];
const nuevoArray = numeros.map(Math.sqrt)

console.log(nuevoArray); //Output: [7, 4, 8, 9]
```

Creamos un nuevo arreglo y le asignamos todos los números del arreglo numeros multiplicados por diez con una función construida por nosotros

```
const numeros = [65, 44, 12, 4];
const nuevoArreglo = numeros.map(multiplicarPorDiez);

function multiplicarPorDiez(numero){
  return numero * 10;
}

console.log(nuevoArreglo); //Output: [650, 440, 120, 40]
```

¿Preguntas?


Hedy
Hedy