

W2D4 - Promises

AGENDA

Async flow with callbacks Recap

Exception handling

Error handling with an async flow with callbacks

Callback Hell

Promises

I switched your hamburger

for a veggie burger

FourGreenSteps.com

Memecreato

Disclaimer: DON'T PANIC

- Async workflow in JavaScript **takes a while** to understand
- Promises are also **hard** to understand initially
- Don't expect that by EOD you will **fully understand** them



Async flow with callbacks Recap

Example: What is the console.log going to print?

```
const processOrder = (customer, callback) => {  
  console.log(`${customer} orders a burger!`);  
  setTimeout(() => {  
    callback(`Burger ready for ${customer}`);  
  }, 3000);  
};  
  
// What is going to be the output of the function execution?  
  
processOrder('Sponge Bob', (message) => console.log(message));  
  
console.log('Sponge Bob eats the burger');
```

Exception Handling

- Errors that are thrown will **stop the execution** of our code.
- To better handle errors in our code, we use a **try catch**.

```
try {  
  processOrder('Sponge Bob', (message) =>  
    console.log(message));  
  
} catch(err) {  
  console.log("Error:", err.message);  
}
```

Try catch allows the code to continue executing.

Handling Errors with Async Code

- Try Catch works well with synchronous code, but it does not work with asynchronous code
- How do we handle errors with async code?

Callback Hell

- When we need to have **multiple nested async function calls**, we get into what we call **callback hell**!



Exercise: <https://gist.github.com/DominicTremblay/311014069b5ce616b5ccf4792a362910>

Promises

Why Promises?

- Promises suggest a **better syntax** to handle callbacks
- Multiple async calls can be **handled more gracefully**

Promises

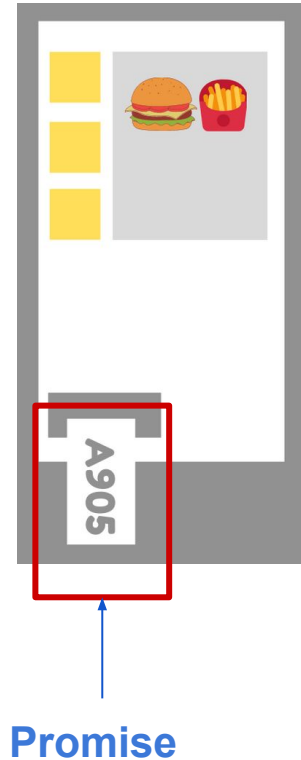
What is a promise?

- A promise is an object that represents the **eventual completion (or failure)** of an asynchronous operation and its resulting value

Promises

What is a promise? An analogy

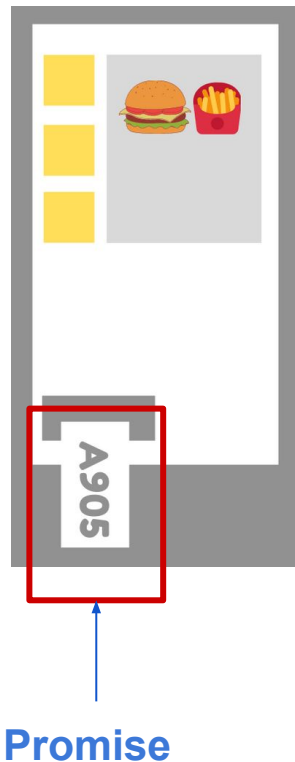
- Placing an order at the kiosk at a restaurant
- You've been given a receipt with a number
- The receipt is a promise that you're going to get a meal when it's ready!



Promises

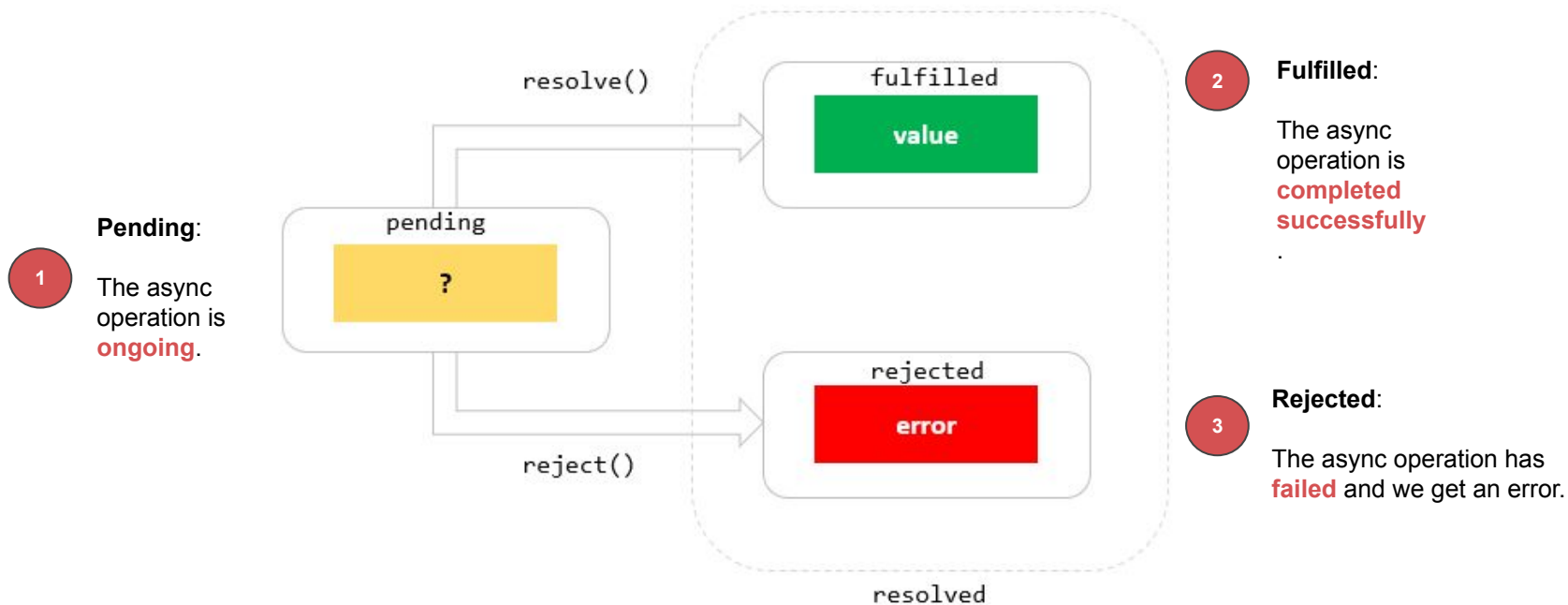
The Promise has 3 potential states:

- **Pending State**: You've been given a receipt with a number.
- **Fulfilled**: Your order is ready! You present the ticket and get your meal :) **OR**
- **Rejected**: They're out of burgers! :(Your order cannot be fulfilled. It's rejected!



Promises

The Promise Object has 3 potential states:



Promises

How to use a promise:

1. Create an **executor function** to create a promise
2. **Consuming** the promise (using it)

Promises

The **Executor** Function:

Some
Async
Function

```
const ExecutorFct = (resolveFct, rejectFct) => {
```

```
  resolveFct(someValue);
```

Success => calling
resolveFct and passing it
the data

```
  // OR
```

```
  rejectFct(someValue);
```

Failed => calling rejectFct
and passing it the error

```
};
```

A promise returns a promise object

```
const promiseObj = new Promise(ExecutorFct);
```

Promises

Consuming the promise:

```
const ExecutorFct = (resolveFct, rejectFct) => {
```

```
  resolveFct(someValue);
```

```
  // OR
```

```
  rejectFct(someValue);
```

```
};
```

```
const promiseObj = new Promise(ExecutorFct);
```

```
promiseObj
```

```
  .then((result) => console.log(result))
```

```
  .catch((err) => console.log(err));
```

Consuming
the promise



Promises

Promise.all

Use Promise.all when we need to wait for all of the promises to be resolved.

```
Promise.all([getUser(), getOrder(), getUser()])
  .then((values) =>
    console.log(`${values[0]} says: ${values[1]} ${values[2]}`)
  )
  .catch((err) => console.log(err));
```

Questions?

