BOOTCAMP ’20

SESIONI #19

What will be covered?

- Promises
- Async / Await
- Exercises

Promises

A **Promise** is a proxy for a value not necessarily known when the promise is created.

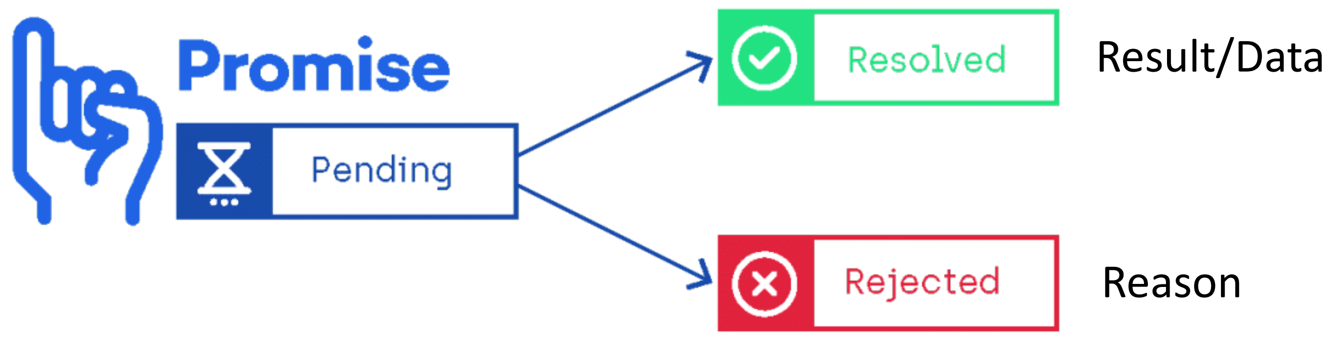
It allows you to associate handlers with an asynchronous action's eventual success value or failure reason.

This lets asynchronous methods return values like synchronous methods: instead of immediately returning the final value, the asynchronous method returns a *promise* to supply the value at some point in the future.

Promises

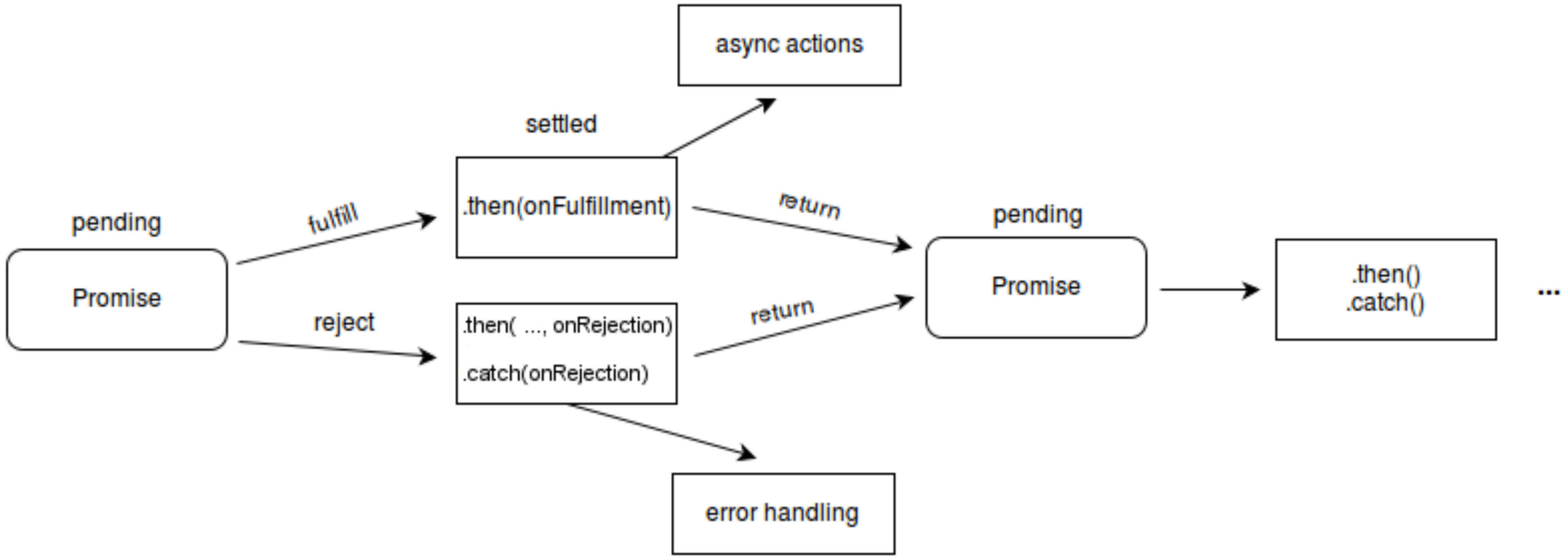
A Promise is in one of these states:

- *pending*: initial state, neither fulfilled nor rejected.
- *fulfilled*: meaning that the operation was completed successfully.
- *rejected*: meaning that the operation failed.



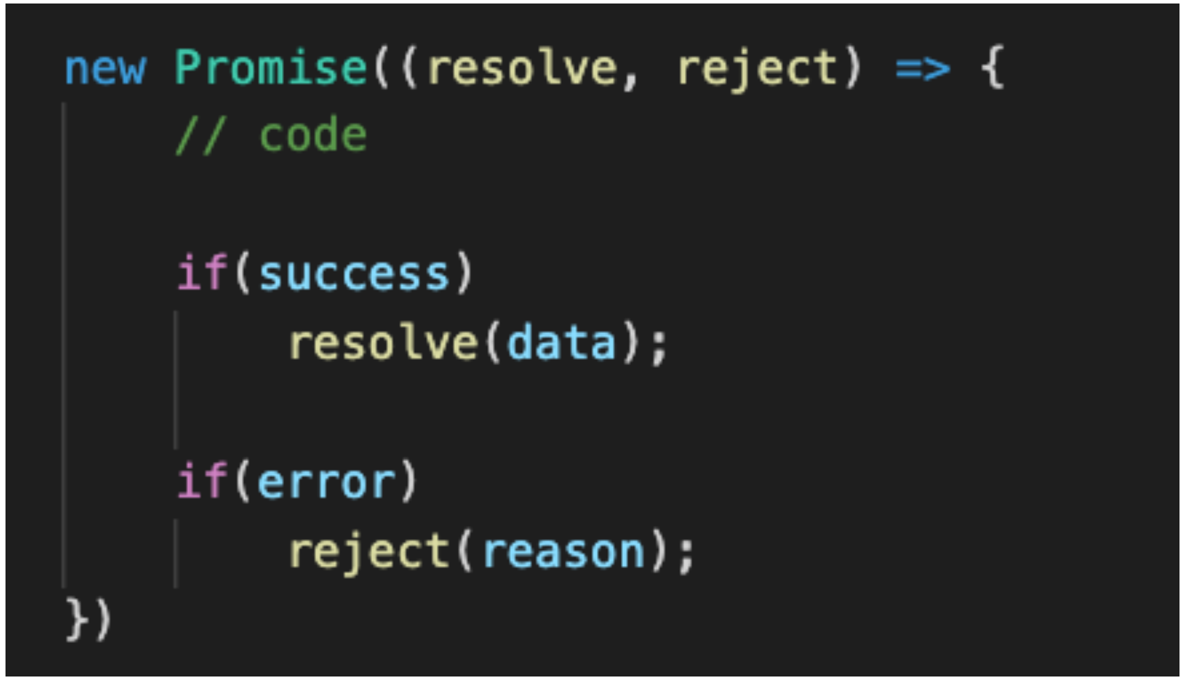
Promises

As the Promise.prototype.then() and Promise.prototype.catch() methods return promises, they can be chained.



Promises

How we can define a promise?



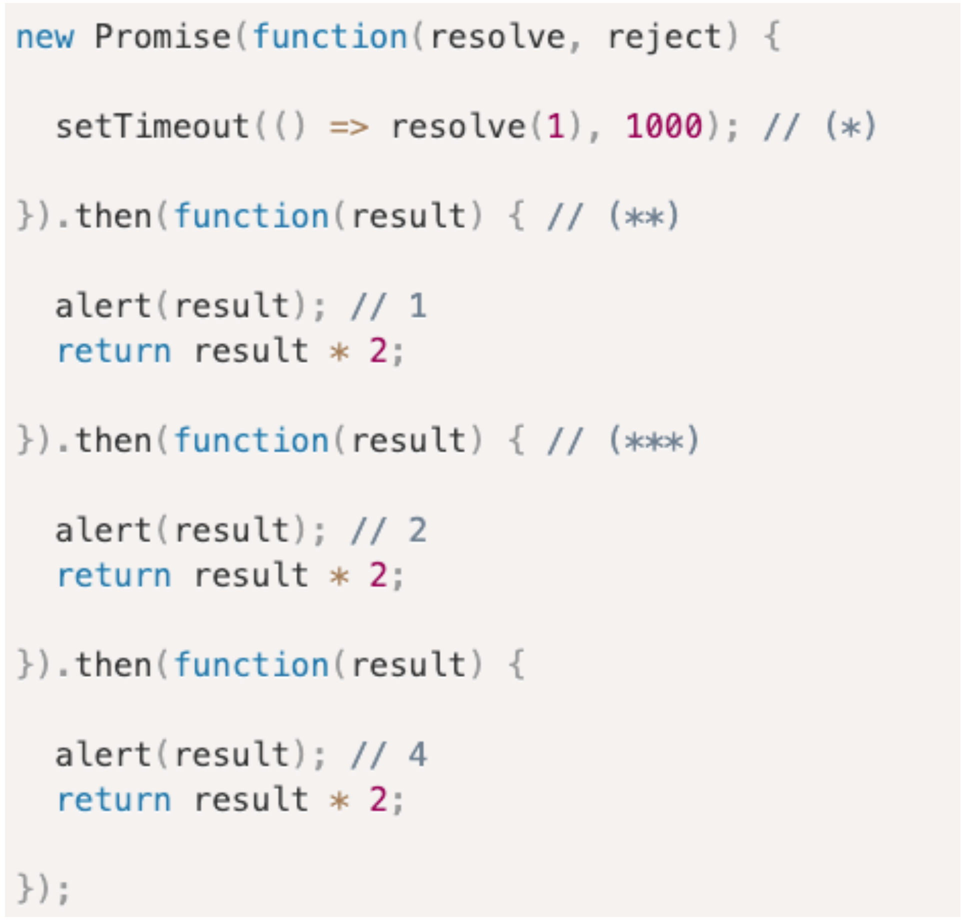
```
new Promise((resolve, reject) => {
    // code

    if(success)
        resolve(data);

    if(error)
        reject(reason);
})
```

Promises

```
const promise = new Promise((resolve, reject) => {
  const x = 10;

  if (x == 10)
    resolve('Success!');
  else
    reject("Error!");
});
```

```
promise.then(res => {
  console.log(res);
}).catch(e => {
  console.log(e);
});
```

Promises



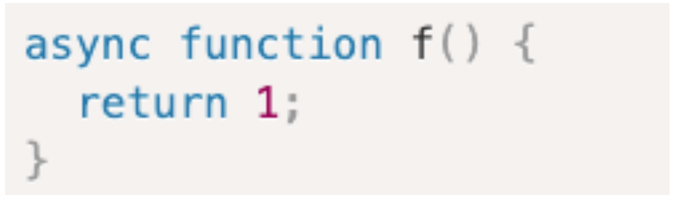
```
new Promise(function(resolve, reject) {

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

}).then(function(result) { // (**)

  alert(result); // 1
  return result * 2;

}).then(function(result) { // (***)

  alert(result); // 2
  return result * 2;

}).then(function(result) {

  alert(result); // 4
  return result * 2;

});
```

Promises

Real-life example

Async / Await

- There’s a special syntax to work with promises in a more comfortable fashion, called “async/await”.

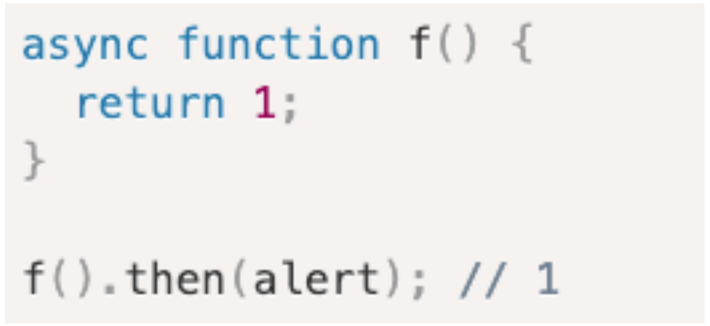


```
async function f() {
  return 1;
}
```

- The word “async” before a function means one simple thing: a function always returns a promise.

Async / Await

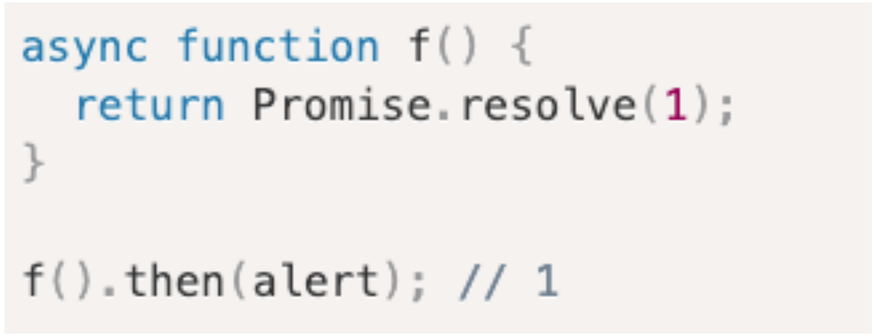
For instance, this function returns a resolved promise with the result of 1; let’s test it:



```
async function f() {
  return 1;
}

f().then(alert); // 1
```

Async / Await

…We could explicitly return a promise, which would be the same:

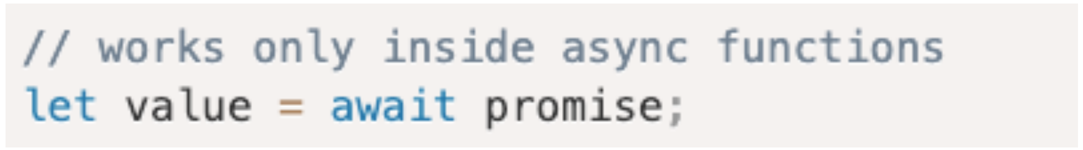


```javascript
async function f() {
  return Promise.resolve(1);
}

f().then(alert); // 1
```

async ensures that the function returns a promise, and wraps non-promises in it.

Async / Await

Another keyword, **await**, that works only inside **async** functions

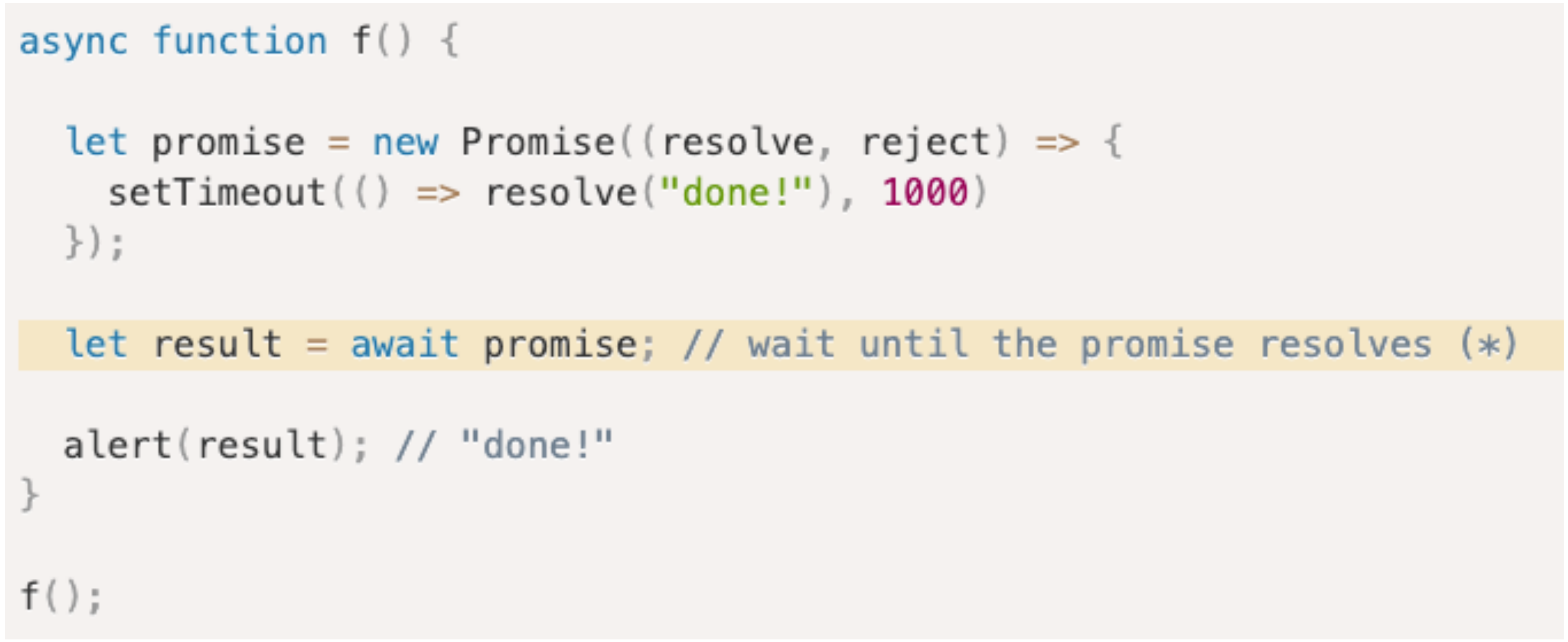


```
// works only inside async functions
let value = await promise;
```

The keyword **await** makes JavaScript wait until that promise settles and returns its result.

Async / Await

- Here’s an example with a promise that resolves in 1 second:



```
async function f() {

  let promise = new Promise((resolve, reject) => {
    setTimeout(() => resolve("done!"), 1000)
  });

  let result = await promise; // wait until the promise resolves (*)

  alert(result); // "done!"
}

f();
```

Async / Await

- The function execution “pauses” at the line (*) and resumes when the promise settles, with result becoming its result. So the code above shows “done!” in one second.
- Let’s emphasize: await literally suspends the function execution until the promise settles, and then resumes it with the promise result. That doesn’t cost any CPU resources, because the JavaScript engine can do other jobs in the meantime: execute other scripts, handle events, etc.
- It’s just a more elegant syntax of getting the promise result than promise.then, easier to read and write.

Async / Await

Real-life example

QUESTIONS