기초 웹

22 Winter CNU 기초 스터디

21 남정연
21 박준서

Javascript Promise

Why Promise?

```
function handleCallButton(evt) {
  setStatusMessage("Calling...");
  navigator.mediaDevices.getUserMedia({video: true, audio: true})
    .then(chatStream => {
      selfViewElem.srcObject = chatStream;
      chatStream.getTracks().forEach(track => myPeerConnection.addTrack(track, chatStream));
      setStatusMessage("Connected");
    }).catch(err => {
      setStatusMessage("Failed to connect");
    });
}
```

Problematic Callbacks

```
chooseToppings(function(toppings) {
  placeOrder(toppings, function(order) {
    collectOrder(order, function(pizza) {
      eatPizza(pizza);
    }, failureCallback);
  }, failureCallback);
}, failureCallback);
```

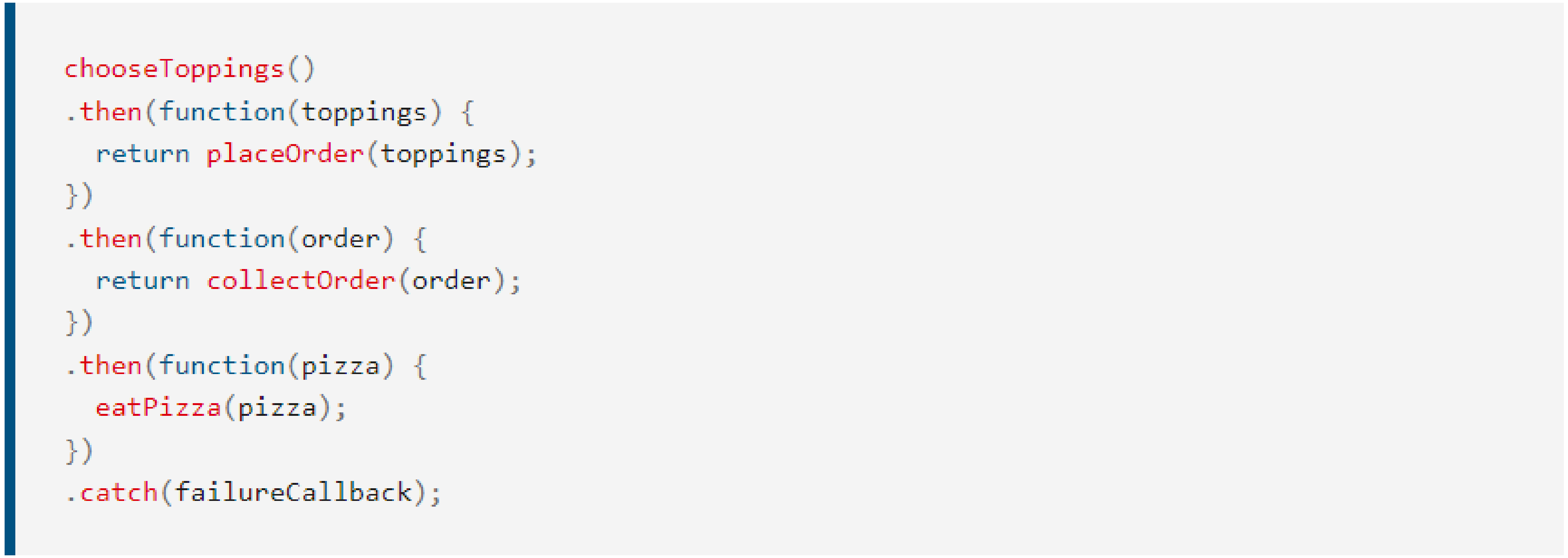
Problematic Callbacks

What is "*callback hell*"?

Asynchronous JavaScript, or JavaScript that uses callbacks, is hard to get right intuitively. A lot of code ends up looking like this:

```
fs.readdir(source, function (err, files) {
  if (err) {
    console.log('Error finding files: ' + err)
  } else {
    files.forEach(function (filename, fileIndex) {
      console.log(filename)
      gm(source + filename).size(function (err, values) {
        if (err) {
          console.log('Error identifying file size: ' + err)
        } else {
          console.log(filename + ' : ' + values)
          aspect = (values.width / values.height)
          widths.forEach(function (width, widthIndex) {
            height = Math.round(width / aspect)
            console.log('resizing ' + filename + 'to ' + height + 'x' + height)
            this.resize(width, height).write(dest + 'w' + width + '_' + filename, function(err) {
              if (err) console.log('Error writing file: ' + err)
            })
          }.bind(this))
        }
      })
    })
  }
})
```

Javascript Promise



```
chooseToppings()
.then(function(toppings) {
  return placeOrder(toppings);
})
.then(function(order) {
  return collectOrder(order);
})
.then(function(pizza) {
  eatPizza(pizza);
})
.catch(failureCallback);
```

Javascript Promise

```
chooseToppings()
.then(toppings =>
  placeOrder(toppings)
)
.then(order =>
  collectOrder(order)
)
.then(pizza =>
  eatPizza(pizza)
)
.catch(failureCallback);
```

```
chooseToppings()
.then(toppings => placeOrder(toppings))
.then(order => collectOrder(order))
.then(pizza => eatPizza(pizza))
.catch(failureCallback);
```

```
chooseToppings().then(placeOrder).then(collectOrder).then(eatPizza).catch(failureCallback);
```

Javascript Promise

```
chooseToppings()
.then(toppings =>
  placeOrder(toppings)
)
.then(order =>
  collectOrder(order)
)
.then(pizza =>
  eatPizza(pizza)
)
.catch(failureCallback);
```

```
chooseToppings()
.then(toppings => placeOrder(toppings))
.then(order => collectOrder(order))
.then(pizza => eatPizza(pizza))
.catch(failureCallback);
```

```
chooseToppings().then(placeOrder).then(collectOrder).then(eatPizza).catch(failureCallback);
```

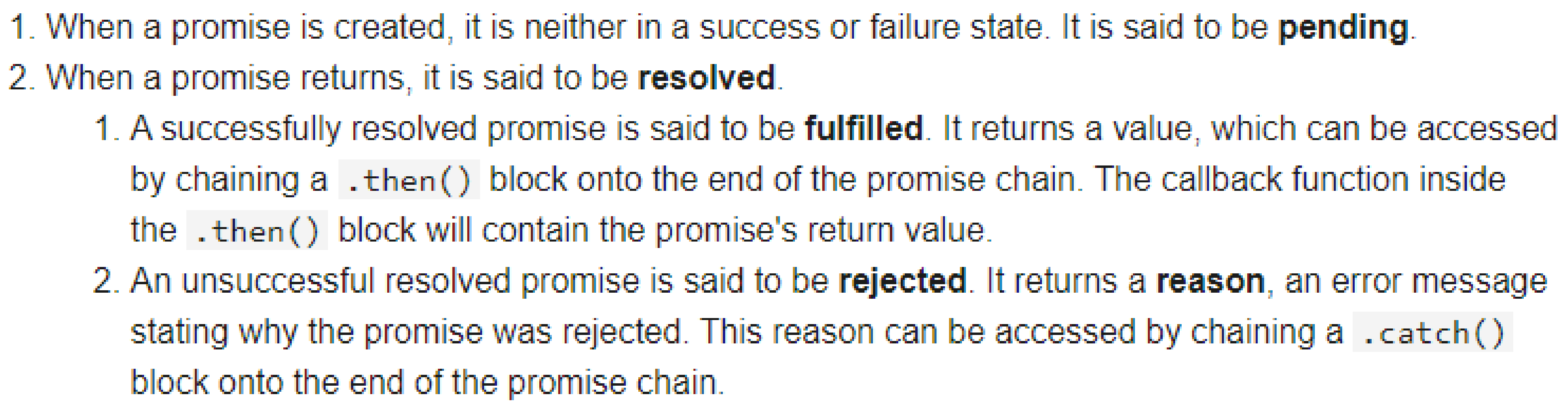
Javascript Promise

```
fetch('coffee.jpg')
.then(response => {
  if (!response.ok) {
    throw new Error(`HTTP error! status: ${response.status}`);
  } else {
    return response.blob();
  }
})
.then(myBlob => {
  let objectURL = URL.createObjectURL(myBlob);
  let image = document.createElement('img');
  image.src = objectURL;
  document.body.appendChild(image);
})
.catch(e => {
  console.log('There has been a problem with your fetch operation: ' + e.message);
});
```

Javascript Promise

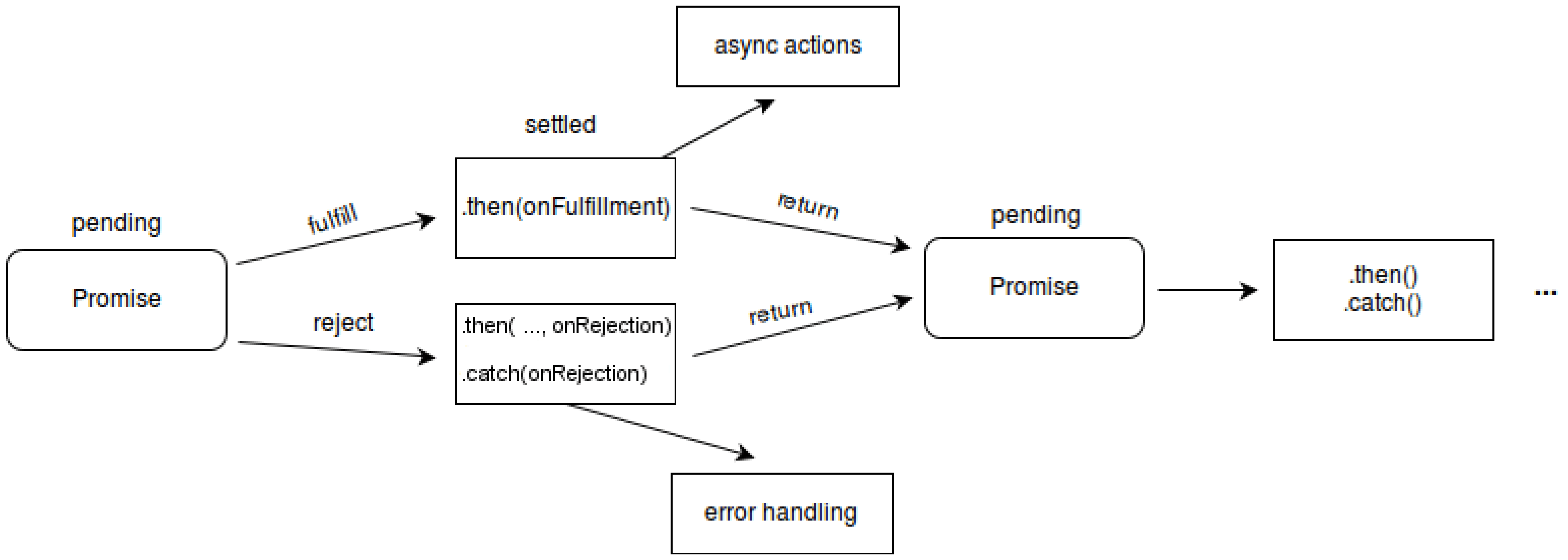
```
myPromise
.then(response => {
  doSomething(response);
})
.catch(e => {
  returnError(e);
})
.finally(() => {
  runFinalCode();
});
```

Javascript Promise



1. When a promise is created, it is neither in a success or failure state. It is said to be **pending**.
2. When a promise returns, it is said to be **resolved**.
    1. A successfully resolved promise is said to be **fulfilled**. It returns a value, which can be accessed by chaining a `.then()` block onto the end of the promise chain. The callback function inside the `.then()` block will contain the promise's return value.
    2. An unsuccessful resolved promise is said to be **rejected**. It returns a **reason**, an error message stating why the promise was rejected. This reason can be accessed by chaining a `.catch()` block onto the end of the promise chain.

Javascript Promise



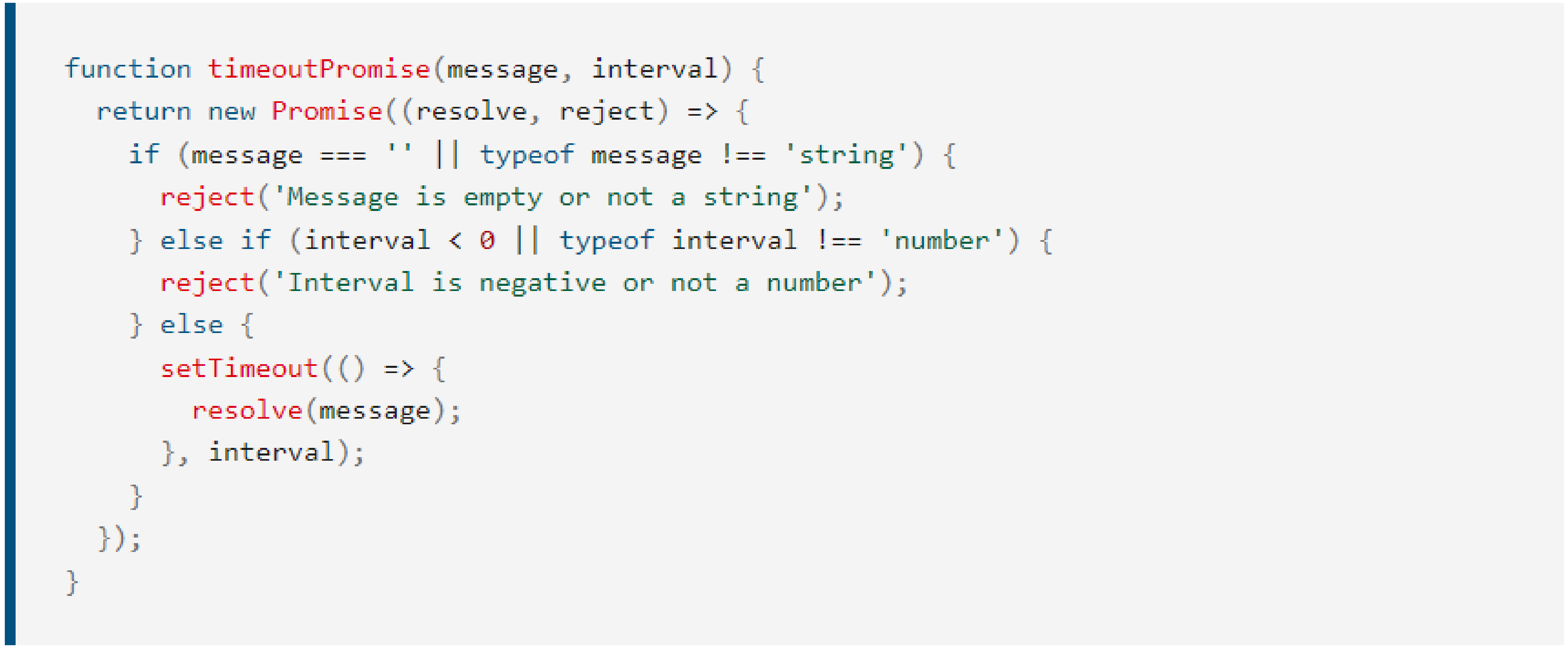
Javascript Promise

```
let timeoutPromise = new Promise((resolve, reject) => {
  setTimeout(() => {
    resolve('Success!');
  }, 2000);
});
```

```
timeoutPromise
.then((message) => {
   alert(message);
})
```

```
timeoutPromise.then(alert);
```

Javascript Promise



```javascript
function timeoutPromise(message, interval) {
  return new Promise((resolve, reject) => {
    if (message === '' || typeof message !== 'string') {
      reject('Message is empty or not a string');
    } else if (interval < 0 || typeof interval !== 'number') {
      reject('Interval is negative or not a number');
    } else {
      setTimeout(() => {
        resolve(message);
      }, interval);
    }
  });
}
```

Promise.All()

```
var p1 = Promise.resolve(3);
var p2 = 1337;
var p3 = new Promise((resolve, reject) => {
  setTimeout(() => {
    resolve("foo");
  }, 100);
});

Promise.all([p1, p2, p3]).then(values => {
  console.log(values); // [3, 1337, "foo"]
});
```

Async / Await

For example, the following:

```
async function foo() {
    return 1
}
```

...is similar to:

```
function foo() {
    return Promise.resolve(1)
}
```

Async / Await

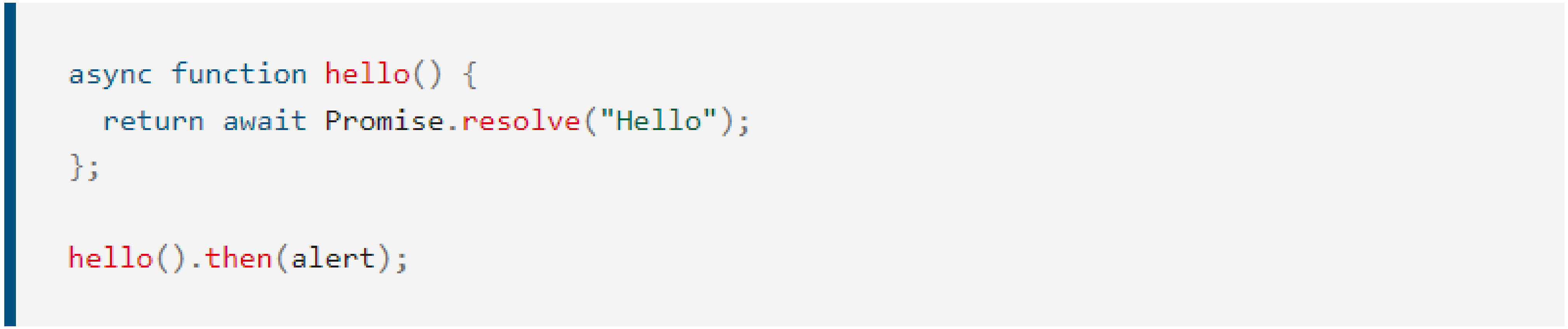
```
function hello() { return "Hello" };
hello();
```

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

```
let hello = async function() { return "Hello" };
hello();
```

```
let hello = async () => "Hello";
```

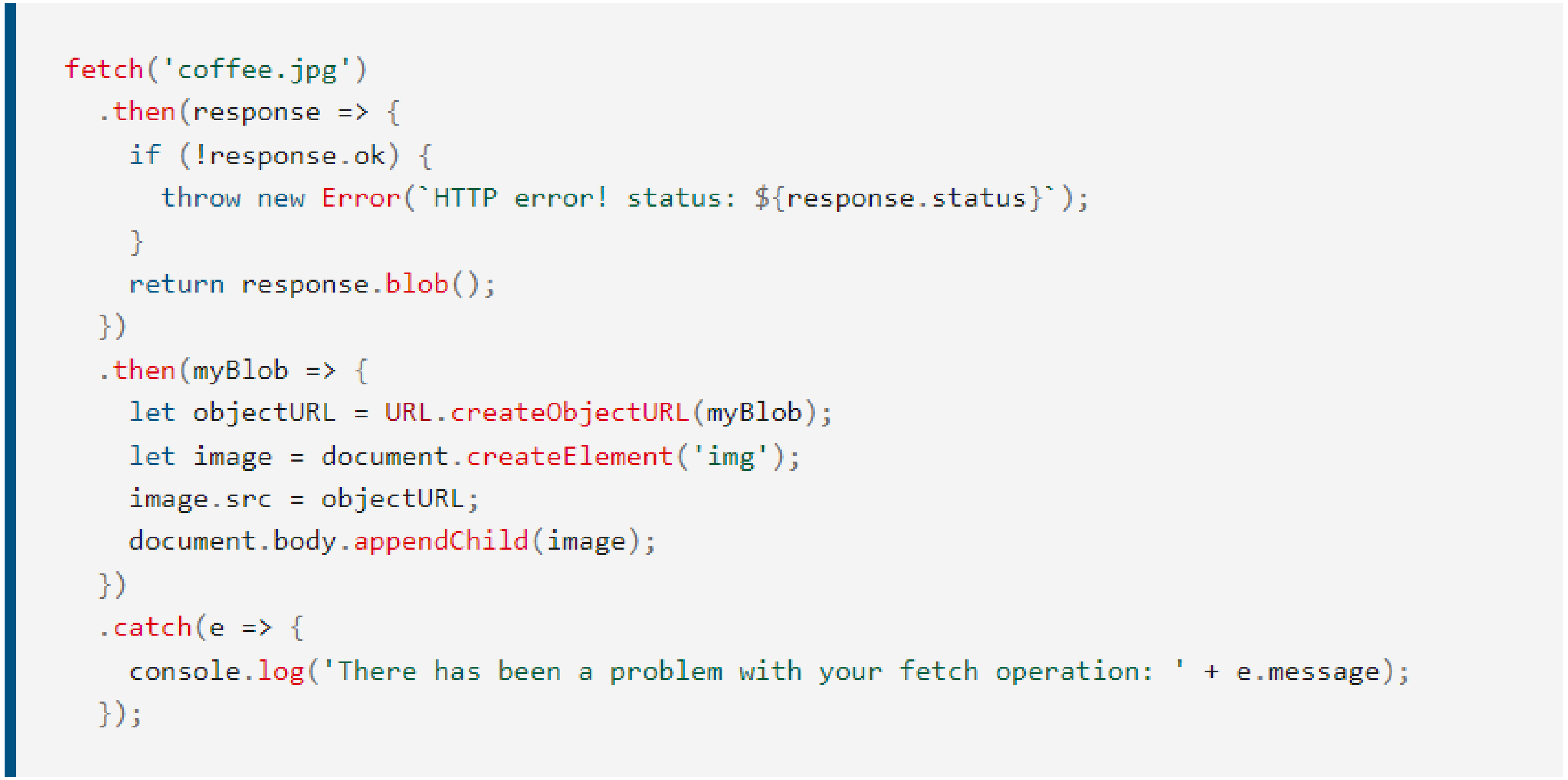
Async / Await



```
async function hello() {
  return await Promise.resolve("Hello");
};

hello().then(alert);
```

Async / Await



```
fetch('coffee.jpg')
  .then(response => {
    if (!response.ok) {
      throw new Error(`HTTP error! status: ${response.status}`);
    }
    return response.blob();
  })
  .then(myBlob => {
    let objectURL = URL.createObjectURL(myBlob);
    let image = document.createElement('img');
    image.src = objectURL;
    document.body.appendChild(image);
  })
  .catch(e => {
    console.log('There has been a problem with your fetch operation: ' + e.message);
  });
```

Async / Await

```
async function myFetch() {
  let response = await fetch('coffee.jpg');

  if (!response.ok) {
    throw new Error(`HTTP error! status: ${response.status}`);
  }

  let myBlob = await response.blob();

  let objectURL = URL.createObjectURL(myBlob);
  let image = document.createElement('img');
  image.src = objectURL;
  document.body.appendChild(image);
}

myFetch()
  .catch(e => {
    console.log('There has been a problem with your fetch operation: ' + e.message);
  });
```

Async / Await

```
async function myFetch() {
  let response = await fetch('coffee.jpg');
  if (!response.ok) {
    throw new Error(`HTTP error! status: ${response.status}`);
  }
  return await response.blob();
}

myFetch()
  .then(blob => {
    let objectURL = URL.createObjectURL(blob);
    let image = document.createElement('img');
    image.src = objectURL;
    document.body.appendChild(image);
  })
  .catch(e => console.log(e));
```