

Reading files

As we noted before, read and writing files is also slow.

Reading files

That means that in JavaScript those tasks should normally be asynchronous.

fs.readFile

It will look something like this. Let's break it down.

```
var fs = require('fs');  
  
function print (error, content) {  
  console.log(content);  
}  
  
fs.readFile('file.txt', 'utf8', print);
```

`fs.readFile`
To interact with files, we need to
require the `fs` module.

```
var fs = require('fs');
```

fs.readFile

Now we can use `readFile`.

```
var fs = require('fs');
```

```
fs.readFile();
```

fs.readFile

It needs three arguments, starting with the path of the file.

```
var fs = require('fs');  
  
fs.readFile('file.txt');
```

fs.readFile

Second, we need the file's
encoding

```
var fs = require('fs');  
  
fs.readFile('file.txt', 'utf8');
```

fs.readFile

It's basically always going to be
utf8. Trust me on this one.

```
var fs = require('fs');  
  
fs.readFile('file.txt', 'utf8');
```


fs.readFile

Unlike Ruby's IO.read...

```
var fs = require('fs');
```

```
var content = fs.readFile('file.txt', 'utf8');
```

fs.readFile
readFile won't return the
contents.

```
var fs = require('fs');
```

```
var content = fs.readFile('file.txt', 'utf8');
```

fs.readFile

That is handled in the callback.

```
var fs = require('fs');
```

```
function print () {  
}
```

```
fs.readFile('file.txt', 'utf8', print);
```

fs.readFile

You get it as a parameter.

```
var fs = require('fs');
```

```
function print (content) {  
}
```

```
fs.readFile('file.txt', 'utf8', print);
```

fs.readFile

Note that this parameter is only defined *inside the callback*.

```
var fs = require('fs');  
  
function print (content) {  
  console.log(content);  
}  
  
fs.readFile('file.txt', 'utf8', print);
```

fs.readFile

One minor detail though.

```
var fs = require('fs');  
  
function print (content) {  
  console.log(content);  
}  
  
fs.readFile('file.txt', 'utf8', print);
```

fs.readFile

This particular callback actually receives two parameters.

```
var fs = require('fs');  
  
function print (otherParam, content) {  
  console.log(content);  
}  
  
fs.readFile('file.txt', 'utf8', print);
```

fs.readFile

That first parameter is an error object.

```
var fs = require('fs');  
  
function print (error, content) {  
    console.log(content);  
}  
  
fs.readFile('file.txt', 'utf8', print);
```


fs.readFile

Many callbacks in Node.js use this convention.

```
var fs = require('fs');  
  
function print (error, content) {  
  console.log(content);  
}  
  
fs.readFile('file.txt', 'utf8', print);
```

fs.readFile

They send an error variable in case something went wrong

```
var fs = require('fs');  
  
function print (error, content) {  
  console.log(content);  
}  
  
fs.readFile('file.txt', 'utf8', print);
```

fs.readFile

For example, if we give `readFile` a bad file path.

```
var fs = require('fs');  
  
function print (error, content) {  
  console.log(content);  
}  
  
fs.readFile('asdfghj.txt', 'utf8', print);
```

fs.readFile

How do we make use of this?

```
var fs = require('fs');  
  
function print (error, content) {  
  console.log(content);  
}  
  
fs.readFile('asdfghj.txt', 'utf8', print);
```

fs.readFile

Checking the error with an if.

```
var fs = require('fs');  
  
function print (error, content) {  
  if (error) {  
    console.log('Oh no! Error!', error);  
  }  
  console.log(content);  
}  
  
fs.readFile('asdfghj.txt', 'utf8', print);
```

fs.readFile

Finally, put the other code in the `else` to avoid problems

```
var fs = require('fs');

function print (error, content) {
  if (error) {
    console.log('Oh no! Error!', error);
  } else {
    console.log('Success!', content);
  }
}

fs.readFile('asdfghj.txt', 'utf8', print);
```

fs.readFile

Final work, with a good file path

```
var fs = require('fs');

function print (error, content) {
  if (error) {
    console.log('Oh no! Error!', error);
  } else {
    console.log('Success!', content);
  }
}

fs.readFile('file.txt', 'utf8', print);
```

Example: `readFile`

Let's do a small throwback to
Terminal Keynote.

Example: `readFile`

In particular, reading the slides
from our presentation file.

Example: `readFile`

If you recall the format was:

Coffee is actually very healthy.

It's loaded with antioxidants.

It also smells and tastes great.

Example: `readFile`

Let's write a function that receives a presentation file's path and results in an array of slides.

Example: readFile

As always, we start with our function definition.

```
function slideLoader (file) {  
}  
module.exports = slideLoader;
```

Example: `readFile`

Notice that we are receiving the file's path as a parameter.

```
function slideLoader (file) {  
  }  
module.exports = slideLoader;
```

Example: `readFile`

And also that we are exporting the function for our `main.js`.

```
function slideLoader (file) {  
  }  
module.exports = slideLoader;
```

Example: readFile

Now we add the `fs` stuff.

```
var fs = require('fs');  
  
function slideLoader (file) {  
  fs.readFile(file, 'utf8');  
}  
module.exports = slideLoader;
```

Example: readFile

Notice how we just pass on the file path variable directly.

```
var fs = require('fs');  
  
function slideLoader (file) {  
  fs.readFile(file, 'utf8');  
}  
module.exports = slideLoader;
```


Example: readFile

Of course, we need a callback for `readFile`.

```
var fs = require('fs');

function slideLoader (file) {
  function splitSlides () {
  }
  fs.readFile(file, 'utf8', splitSlides);
}
module.exports = slideLoader;
```

Example: readFile

Let's handle the error first.

```
var fs = require('fs');

function slideLoader (file) {
  function splitSlides (err) {
    if (err) {
      console.log('Oh no! Error!', err);
    }
  }
  fs.readFile(file, 'utf8', splitSlides);
}

module.exports = slideLoader;
```

Example: readFile

Now we can split the slides.

```
var fs = require('fs');

function slideLoader (file) {
  function splitSlides (err, str) {
    if (err) {
      console.log('Oh no! Error!', err);
    } else {
      var slides = str.split('\n---\n');
    }
  }
  fs.readFile(file, 'utf8', splitSlides);
}

module.exports = slideLoader;
```

Example: readFile

But how can we send slides?

```
var fs = require('fs');

function slideLoader (file) {
  function splitSlides (err, str) {
    if (err) {
      console.log('Oh no! Error!', err);
    } else {
      var slides = str.split('\n---\n');
    }
  }
  fs.readFile(file, 'utf8', splitSlides);
}

module.exports = slideLoader;
```

Example: readFile

Only `splitSlides` can see it.

```
var fs = require('fs');

function slideLoader (file) {
  function splitSlides (err, str) {
    if (err) {
      console.log('Oh no! Error!', err);
    } else {
      var slides = str.split('\n---\n');
    }
  }
  fs.readFile(file, 'utf8', splitSlides);
}

module.exports = slideLoader;
```

Example: readFile

And we can't return it

```
var fs = require('fs');

function slideLoader (file) {
  function splitSlides (err, str) {
    if (err) {
      console.log('Oh no! Error!', err);
    } else {
      var slides = str.split('\n---\n');
    }
  }
  fs.readFile(file, 'utf8', splitSlides);
}

module.exports = slideLoader;
```

Example: readFile

Let's think about our `main.js`.

```
// main.js
```

```
var slideLoader = require('./slide-loader');
```

Example: readFile

Imagine this was synchronous.

```
// main.js
```

```
var slideLoader = require('./slide-loader');
```

```
var slides = slideLoader('slides.txt');
```


Example: readFile

It would return the slides array

```
// main.js
```

```
var slideLoader = require('./slide-loader');
```

```
var slides = slideLoader('slides.txt');
```

Example: readFile

And then we could use it

```
// main.js

var slideLoader = require('./slide-loader');

var slides = slideLoader('slides.txt');

slides.forEach(function (slide) {
  console.log('\n\n\n\n' + slide);
});
```

Example: readFile

Here we want to do a `forEach`.

```
// main.js

var slideLoader = require('./slide-loader');

var slides = slideLoader('slides.txt');

slides.forEach(function (slide) {
  console.log('\n\n\n\n      ' + slide);
});
```

Example: readFile

And print slides with spacing.

```
// main.js

var slideLoader = require('./slide-loader');

var slides = slideLoader('slides.txt');

slides.forEach(function (slide) {
  console.log('\n\n\n\n' + slide);
});
```

Example: readFile

In reality, this is asynchronous.

```
// main.js

var slideLoader = require('./slide-loader');

slideLoader('slides.txt', function (slides) {
});

slides.forEach(function (slide) {
  console.log('\n\n\n\n' + slide);
});
```

Example: readFile

We get `slides` in a callback.

```
// main.js
```

```
var slideLoader = require('./slide-loader');
```

```
slideLoader('slides.txt', function (slides) {  
  });
```

```
slides.forEach(function (slide) {  
  console.log('\n\n\n\n      ' + slide);  
});
```

Example: readFile

We need to use them inside.

```
// main.js
```

```
var slideLoader = require('./slide-loader');

slideLoader('slides.txt', function (slides) {
  slides.forEach(function (slide) {
    console.log('\n\n\n\n' + slide);
  });
});
```

Example: readFile

Back to `slide-loader.js`.

```
var fs = require('fs');

function slideLoader (file) {
  function splitSlides (err, str) {
    if (err) {
      console.log('Oh no! Error!', err);
    } else {
      var slides = str.split('\n---\n');
    }
  }
  fs.readFile(file, 'utf8', splitSlides);
}

module.exports = slideLoader;
```


Example: readFile

How can we ask for a callback?

```
var fs = require('fs');

function slideLoader (file) {
  function splitSlides (err, str) {
    if (err) {
      console.log('Oh no! Error!', err);
    } else {
      var slides = str.split('\n---\n');
    }
  }
  fs.readFile(file, 'utf8', splitSlides);
}

module.exports = slideLoader;
```

Example: readFile

Add another parameter!

```
var fs = require('fs');

function slideLoader (file, callback) {
  function splitSlides (err, str) {
    if (err) {
      console.log('Oh no! Error!', err);
    } else {
      var slides = str.split('\n---\n');
    }
  }
  fs.readFile(file, 'utf8', splitSlides);
}

module.exports = slideLoader;
```

Example: readFile

Use it like a named function.

```
var fs = require('fs');

function slideLoader (file, callback) {
  function splitSlides (err, str) {
    if (err) {
      console.log('Oh no! Error!', err);
    } else {
      var slides = str.split('\n---\n');
      callback(slides);
    }
  }
  fs.readFile(file, 'utf8', splitSlides);
}

module.exports = slideLoader;
```

Example: readFile

Our final product:

```
var fs = require('fs');

function slideLoader (file, callback) {
  function splitSlides (err, str) {
    if (err) {
      console.log('Oh no! Error!', err);
    } else {
      var slides = str.split('\n----\n');
      callback(slides);
    }
  }
  fs.readFile(file, 'utf8', splitSlides);
}

module.exports = slideLoader;
```

Example: readFile

Give it a try.

```
// main.js

var slideLoader = require('./slide-loader');

slideLoader('slides.txt', function (slides) {
  slides.forEach(function (slide) {
    console.log('\n\n\n\n      ' + slide);
  });
});
```

Exercise: `readFile`

Your pair programming exercise
today will use `fs.readFile`.
Check it out!