NodeJS

NodeJS allows you to run JS at the command line

Programs are "interpreted", not compiled

- Means no compile to check for errors
- But some tools can scan/modify code, with similar results to compiling

Node uses the same JS engine that runs in Chrome

- Does not have DOM and browser-related bits
- Adds file system and networking parts

Require

Because it runs on a system and not a "page", it can easily load additional files

- "modules" will "export" code
- the calling code will get the exported value (obj, string, function, etc)

```
const assert = require('assert');

assert.strictEqual(1, 1);
console.log('it only gets this far if the assert is happy');
```

Unrelated to "RequireJS"

How to export

Write the code you want to export in a separate file

- Name the file meaningfully
- Write the code to be separate and un-coupled
    - It should be useful in more than one place
    - Often newer coders will write it in the same file, then move it out
    - Don't skip the moving and clean it up part

setting module.exports

There are some existing "global" values.

The `module.exports` value defines what someone `require()`ing your module will get

```
// in foo.js
module.exports = {
  one: 1,
  two: 2
};

// in bar.js
const foo = require('./foo');
console.log(foo.one); // 1
```

Export Different Kinds of Values

```
module.exports = {
  one: 1,
  two: 2
};
```

```
module.exports = 'boring';
```

```
module.exports = [ 'a','b','c' ];
```

```
module.exports = function( word ) {
  return word.toLowerCase().replace(/ /g, '-');
};
```

```
module.exports = function() {
  const count = 1;
  return function() {
    return count++;
  };
};
```

Getting part of the export

See how these `require()`s pull in different things.

Understand what they imply about what is exported.

```
const foo = require('./foo').somePart;
```

```
const bar = require('./bar')();
```

```
const { onePart, somePart } = require('./baz');
```

Modules run once

Unless you force it to do otherwise, all modules run once, regardless of how many times they are `require()`ed.

This is good.

Each module is a separate variable scope

```
// In foo.js
const foo = 1;
module.exports = foo;

// In bar.js
const foo = 2;
module.exports = foo;

// In baz.js
const foo = require('./foo');
const bar = require('./bar');
console.log(foo); // 1
console.log(bar); // 2
```

Require() paths

The path passed to `require()` is relative for a local file

The path passed to `require()` can omit the trailing `.js`

The path passed to `require()` needs no path for *external* modules

Names of things

Modules are generally kebab-case

Exported properties are camelCase

- MixedCase for constructors/classes

package vs module vs library

- I can run `npm install` on it - **package**
- I can use `require()` or `import` (more later) - **module**
- I can get the code and use it with my code - **library**