NodeJS

NodeJS allows you to run JS at the command line

Programs are "interpreted", not compiled

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

Node uses the core 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"

- Node can easily load additional files
- "modules" will "export" code
- Requiring code gets exported value
    - Objects, String, Function, etc

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

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

- Not ES6 import/export (ES Modules/ESM)
    - more on that later

How to export

Write the code you want to export in a separate file

- Name the file meaningfully
    - Usually **kebab-case** filenames
    - Spaces awful for command line
    - lowercase means no guessing about case
- Write the code to be separate and **un-coupled**
    - Should be useful in more than one place
    - Shouldn't "know" much about outside code

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() { // a "closure"
  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').cat; // ./foo.js exports object
```

```
const bar = require('./bar')(); // ./bar.js exports function
```

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

Modules run once

All modules run once

- Regardless of how many times `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

- `const someValue = require('./my-file.js');`

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

- `const someValue = require('./my-file');`

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

- `const express = require('express');`

Names of things

Modules (filenames) are generally **kebab-case**

Exported properties are **camelCase**

- **MixedCase** for constructors/classes
- **CONSTANT_CASE** for actual constants

package vs module vs library vs framework

- I can run `npm install` on it - **package**
- I can use `require()` or `import` (later) - **module**
- I can get the code and use it with my code - **library**
- Creates type of app IF code follows rules - **framework**