Before we begin...

- Open up these slides:
  - https://bit.ly/2HFGckf

Modules, Babel & Webpack

Learning Objectives

- **Introduce** modular JavaScript with CommonJS modules
- **Utilise** require statements and exports effectively
- **Understand** compilation and transpilation
- **Effectively** transpile ES2015 code
- **Understand** Webpack and its role in web development
- **Effectively** use Webpack

Agenda

- Modules
- Transpilation & Compilation
- Babel
- Webpack

A quick review

- Prettier
- ES2015
- APIs
- AJAX
- Fetch

Projects Time!

Modules

What are modules?

- An approach to making our code modular
  - It's an approach for organising code
  - It adds structure to our projects!
- Each file is considered a **module:**
  - That can export contents
  - And can require dependencies
- I try and break down my files into single responsibilities

Exporting

```
// Exporting just one thing

const greetings = name => `Hello ${name}`;

module.exports = greetings;
```

Exporting

```
const add = (x, y) => x + y;
const subtract = (x, y) => x - y;

module.exports = {
  add: add,
  subtract: subtract
};
```

Exporting

```
const add = (x, y) => x + y;
const subtract = (x, y) => x - y;

// Or, with Enhanced Object Literals...

module.exports = {
  add,
  subtract
};
```

Requiring

```
const math = require("./math");
const greetings = require("./greetings");

console.log(math, greetings);
```

Requiring (Destructuring)

```
const { add } = require("./math");

console.log(add);
```

Resources

- ES6 Features and CommonJS
- Addy Osmani's "Writing Modular JavaScript" - *The CommonJS section!*
- RisingStack: How the Module System Works
- Auth0: JavaScript Module Systems

NPM

What is NPM?

- The **N**ode **P**ackage **M**anager
  - Makes sharing and reusing code easier
  - "Use npm to install, share, and distribute code; manage dependencies in your projects"
- It is a command line tool that is installed when you install Node
- It is the largest software registry in the world

Common Commands

```
npm init
# Start a new Node project

npm install -g PACKAGE_NAME
# Install a package globally

npm install --save PACKAGE_NAME
# Add a package as a dependency for a project

npm install --save-dev PACKAGE_NAME
# Add a package as a dev dependency for a project

npm run COMMAND_NAME
# Run a script you have defined in `package.json`
```

An NPM Project

```
node_modules/

package.json

package-lock.json
```

An NPM Project

```
node_modules/

app/
    html/
    css/
    js/

build/

package.json

package-lock.json

CONFIG_FILES
```

This works with modules

- If you have installed a NPM package
  - We can then include that code in our project using a require statement

This works with modules

```
// Let's pretend we had p5 installed...
//     npm install p5

const p5 = require("p5");

// Let's pretend we had react installed...
//     npm install react

const React = require("react");

// This also works with destructuring!
```

Resources

- NPM
- A Beginner's Guide to NPM
- SitePoint: What is NPM?

Babel

Dealing with Compatibility

- Three options:
  - Write code that works everywhere
  - **Polyfill** - Adding features that are missing
  - **Compile || Transpile** - Translating code
- See Browser Compatibility for ES2015 here

Compilers vs. Transpilers

- Both translate code from one language to another
  - Transpiling is when the two languages involved are at a similar level of abstraction
  - Compiling is when the translation takes place and the resulting code is at a much lower level of abstraction

What is Babel?

- Babel is a JavaScript compiler/transpiler
- It allows us to use next-generation JS today
  - e.g. All of the new ES2015 features (plus newer stuff like ES2016, ES2017, ES2018...)
  - Plus, all sorts of other things...
- It translates our "new" code into things that work in browsers
- See how it does this here

How do you use Babel?

- We can use Babel through:
  - A Command Line Tool
  - Its API
  - Or through a build system (e.g. Webpack) - this is how we are going to be doing it!

Babel Concepts

- **Presets**
  - A whole heap of translations (e.g. all necessary translations for ES2015, or for React etc.)
- **Plugins**
  - A single translation (e.g. arrow functions -> functions)
- We install both of these things with NPM!

Configuring Babel

- We create a .babelrc file in the root directory of our project (the preferred method)
  - In this file, we configure with JSON
- Or in package.json

Configuring Babel (Basic)

```
{
  "presets": [
    [
      "env",
      {
        "targets": {
          "browsers": [">0.25%"]
        }
      }
    ]
  ],
  "plugins": []
}
```

Resources

- BabelJS.io
- CodeMix
- Getting Started with Babel

Webpack

What is Webpack?

- It is a *Build System* and a *bundler*
- It automates tasks for us
- It takes our code, transforms and bundles it, then returns a new version of our code
- We need to make sure our code is browser compatible:
  - SCSS -> CSS
  - ES2015 -> JavaScript

What is Webpack?

- It doesn't do anything by default
- But can be extended to do lots of other things:
  - Minifying and Optimizing Code
  - Minifying Images
  - etc.
- Before this, we have to add lots of scripts if our code is broken up - Webpack brings all of our code together

Why do you need it?

- It helps structure our code
- It organises and automates the tasks we need to do
  - e.g. using Babel
- It saves us from having to combine files ourselves
- It helps us work with larger applications (e.g. by splitting code)
- It can help create our server, can replicate different environments (e.g. development or production) and can add **Hot Module Replacement**

Any alternatives?

- Parcel
- FuseBox
- RollUp
- Browserify
- Grunt
- Gulp
- Make
- Using NPM as a Task Runner

What it needs to know

- The starting point of your application
- What transformations it needs to perform
- The "mode" (whether it is development or production)
- Where it should save your transformed code

We define all of this in a file called *webpack.config.js*

Some Webpack concepts

- **entry** - Where your application starts
- **output** - Where your resulting code goes
- **loaders** - A single transformation/process (e.g. Babel)
- **rule** - All transformations that need to take place for certain files
- **bundle** - Your transformed code (once it is combined)
- **mode** - The current environment (development or production)

webpack.config.js example

```
const config = {
  entry: ["./app/js/index.js"],
  output: {
    path: __dirname + "/dist",
    publicPath: "/",
    filename: "bundle.js"
  },
  module: {
    rules: [
      {
        test: /\.jsx?$/,
        exclude: /node_modules/,
        loaders: ["babel-loader"]
      }
    ]
  }
};

module.exports = config;
```

ES2015 to ES5

Resources

- TinselCity: whys:packers
- Webpack
- SurviveJS
- Webpack Academy

Homework

- Add Babel and Webpack to previous homework!
- Read up on ES2015
- - Translate some of your previous code into it!
- Finish all exercises from class
- Upload your homework to GitHub
- Prepare for next lesson

Homework (Extra)

- Go through some tasks in Exercism
- Get into JavaScript30
- Go through The Modern JavaScript Tutorial
- Read Exploring ES6
- Read Eloquent JavaScript
- Read Speaking JavaScript

What's next?

- More Webpack
- Classes in JS
- React

Questions?

Thanks!