Modules

Prior to ES6, we used libraries such as Browserify to create modules on the client-side, and require in **Node.js**. With ES6, we can now directly use modules of all types (AMD and CommonJS).

Exporting in CommonJS

```
module.exports = 1;
module.exports = { foo: 'bar' };
module.exports = ['foo', 'bar'];
module.exports = function bar () {};
```

Exporting in ES6

With ES6, we have various flavors of exporting. We can perform **Named Exports**:

```
export let name = 'David';
export let age  = 25;ââ
```

As well as **exporting a list** of objects:

```
function sumTwo(a, b) {
    return a + b;
}

function sumThree(a, b, c) {
    return a + b + c;
}

export { sumTwo, sumThree };
```

We can also export functions, objects and values (etc.) simply by using the `export` keyword:

```
export function sumTwo(a, b) {
    return a + b;
}

export function sumThree(a, b, c) {
    return a + b + c;
}
```

And lastly, we can **export default bindings**:

```
function sumTwo(a, b) {
    return a + b;
}

function sumThree(a, b, c) {
    return a + b + c;
}

let api = {
    sumTwo,
    sumThree
};

export default api;

/* Which is the same as
 * export { api as default };
 */
```

Best Practices: Always use the `export default` method at **the end** of the module. It makes it clear what is being exported, and saves time by having to figure out what name a value was exported as. More so, the common practice in CommonJS modules is to export a single value or object. By sticking to this paradigm, we make our code easily readable and allow ourselves to interpolate between CommonJS and ES6 modules.

Importing in ES6

ES6 provides us with various flavors of importing. We can import an entire file:

```
import 'underscore';
```

It is important to note that simply **importing an entire file will execute all code at the top level of that file**.

Similar to Python, we have named imports:

```
import { sumTwo, sumThree } from 'math/addition';
```

We can also rename the named imports:

```
import {
    sumTwo as addTwoNumbers,
    sumThree as sumThreeNumbers
} from 'math/addition';
```

In addition, we can **import all the things** (also called namespace import):

```
import * as util from 'math/addition';
```

Lastly, we can import a list of values from a module:

```
import * as additionUtil from 'math/addition';
const { sumTwo, sumThree } = additionUtil;
```

Importing from the default binding like this:

```
import api from 'math/addition';
// Same as: import { default as api } from 'math/addition';
```

While it is better to keep the exports simple, but we can sometimes mix default import and mixed import if needed. When we are exporting like this:

```
// foos.js
export { foo as default, foo1, foo2 };
```

We can import them like the following:

```
import foo, { foo1, foo2 } from 'foos';
```

When importing a module exported using commonjs syntax (such as React) we can do:

```
import React from 'react';
const { Component, PropTypes } = React;
```

This can also be simplified further, using:

```
import React, { Component, PropTypes } from 'react';
```

Note: Values that are exported are **bindings**, not references. Therefore, changing the binding of a variable in one module will affect the value within the exported module. Avoid changing the public interface of these exported values.