Lesson 2.2 - How to use modules

In the last lesson we ran through all the different ways to import and export modules. But how and why do we put this to use? Because JavaScript has never had a built in module system, we tend to see a lot of code out in the wild that is just a whole bunch of unrelated code thrown together into a huge file.

Granted this is not true for every case, but it's good practice to keep your code self contained in small pieces. This helps immensely with maintenance, readability, and reusability.

Let's take a quick look at how this can help us out by looking at some of the code that we created in the last lesson. In that lesson, we created some simple math functions for exporting. Let's create something a little more useful this time.

First make sure you're in the directory where you previously installed `babel-node` and the presets module, and let's create a file called `math.js`.

Let's create a function that will calculate the total amount that an item on a receipt will cost with (or without) tax.

```
export function calculateTax(price, taxRate, isTaxable) {
  if (!isTaxable) return 0;
  return price * taxRate / 100;
}
```

Great! This is just a simple function that takes in a price, a tax rate, and a boolean that determines if the item is taxable. Pretty simple!

Now that we've created this module, let's see how we can use it! Fire up the REPL and we'll test it out.

```
$ babel-node
> var { calculateTax } = require('./math')
> calculateTax(10, 8, true)
-> .8
> calculateTax(10, 8, false)
-> 0
```

See how easy that was? Now, any time we would like to use this function, we can simply require it wherever we need it! Unfortunately, we can't use the `import` statement in the REPL, but if we were to import this module using ES2015 syntax in another module it would look like this:

```
import { calculateTax } from './math';
```