Lesson 5.3 - Parameter Handling in the Real World

So far in this lesson we have learned some great new ES2015 features that have everything to do with parameter handling. Let's write a little bit of code so that we can see how this might help us in the real world.

In our last lesson we created a receipt calculator that took a list of items and a tax rate and calculated the sub-total and total tax for all the items.

Let's open that back up in the Node REPL and see what happens if we omit the `taxRate` variable from our function:

```
$ babel-node
> var { items, calculate } = require('./calculate')
> calculate(items)
-> Cannot read property 'reduce' of undefined
```

Uh oh! What happened there? Unfortunately, because we are using `positional` arguments here, we can't omit the first `taxRate` variable and also include the `items` variable. When we omit the `taxRate` variable, the `items` variable now becomes the tax rate! Well this doesn't make having default values for parameters very useful now does it? We can get around this by using a JavaScript object in the function, along with some crafty destructuring assignment. We're also going to modify our `calculateTax` function in the math module to use this feature. Let's take a look:

```
math.js

export function calculateTax({ price, taxRate, isTaxable }) {
  ...
}

calculator.js

export function calculate({ taxRate = 8, items = [] }) {
  return items.reduce((prev, curr) => {
    prev.total += curr.price;
    prev.totalTax += calculateTax({ taxRate, price: curr.price, isTaxable: curr.isTaxable });
    return prev;
  }, {
    total: 0,
    totalTax: 0
  });
}
```

So what happened there? The first thing we did was pretty simple. For both functions we added curly braces to the functions arguments. By using a combination of default parameter values and destructuring we are now able to pass in only a list of items and still get the default tax rate. This is because we are no longer using positional arguments and are able to use object keys to reference variables.

You'll notice the second thing we did was change how we were passing arguments into the `calculateTax` function. What we're doing here is using the spread operator to assign all of the values of the `item` object to the object that we are passing into the `calculateTax` function! Now we just need to change the
way we call the function in the REPL:

```
> calculate({ items })
-> { total: 45.45, totalTax: 1.876 }
```

Awesome! We were able to fix our problem using destructuring and enhanced parameter handling! ES2015 is looking really great so far! If you were wondering about the function call:

```
> calculate({ items })
```

Using our new found ES2015 superpowers, we are able to not only destructure objects using that syntax, but we can also structure objects using the same syntax. What you see above is exactly the same as: `{ items: items }`. Because our variable name is the *same* as the name of the key we want in the object
we can assign it this way. It's just a convenient shorthand.