Lesson 4.1 - Arrow Functions

When working in JavaScript, the lexical scoping of `this` often causes a lot of problems with newer developers (and seasoned veterans at times). Lexical this can be described in this way. Every new function defines it's own `this` value, so when you want to reference a `this` that is outside of the scope of
the function you have to use some tricks like assigning another variable to the value of `this` that you want to reference in a function.

This can be overcome, however, with the use of `Function.prototype.bind`. This will change code that looks like this:

```
let self = this;

fetch('www.google.com').then(function(response) {
  self.name = response.body.name;
});
```

to this:

```
fetch('www.sitepoint.com').then(function(response) {
  this.name = response.body.name;
}.bind(this));
```

As you can see, we have avoided assigning an otherwise useless variable by simply binding the function to `this`. This works fine, but it can get a little tedious, and it can be easy to forget.

Have no fear! ES2015 has your back with a new feature called `arrow functions`. Arrow functions automagically bind to the lexical `this`, so no need for `bind(this)`!

That function from above can now be written as:

```
fetch('www.sitepoint.com').then((response) => {
  this.name = response.body.name;
})
```

Wow, that's awesome! It also allows you to use some shorthand syntaxes that come in really handy, and often times make your code easier to read. For instance, you can force an implicit return statement by enclosing the function call in parentheses instead of curly braces:

```
fetch('www.sitepoint.com').then((response) => (
  response.body.name;
));
```

The above function is equivalent to the following:

```
fetch('www.sitepoint.com').then((response) => {
  return response.body.name;
});
```

You can even leave out the parentheses altogether!

```
fetch('www.sitepoint.com').then(response => response.body.name);
```

or with destructuring

```
fetch('www.sitepoint.com').then(({ body: { name } }) => name);
```

Wow! Now that's something. No more endless indentations for one line callback functions.