Arrow Functions

Introduction

We have seen a few examples of arrow functions already, when functions were introduced in the first course, but now we can dive into them a little bit more.

- Minor differences from other ways of creating functions in JavaScript
- Change the scope of the **this** keyword in some circumstances
- More compact and can make code easier to read

Start with a Function Declaration

Make an HTML file and create a function that just takes a string as an input and then makes it uppercase and returns the uppercase version.

```
function makeUpperCase(aString) {
      return aString.toUpperCase();
}

console.log(makeUpperCase("here is a string"));
```

Convert the Function to a Function Expression

Arrow functions are officially arrow function expressions.

```
const makeUpperCase = function(aString) {
      return aString.toUpperCase();
}

console.log(makeUpperCase("here is a string"));
```

Remove the word "function" and add an "arrow" which is really just an equal sign = and and a greater than sign > after the parameter.

```
const makeUpperCase = (aString) => {
      return aString.toUpperCase();
}

console.log(makeUpperCase("here is a string"));
```

Remove the Parentheses

If there is only one parameter, you can further simplify and remove the parentheses around it.

```
const makeUpperCase = (aString) => {
      return aString.toUpperCase();
}

console.log(makeUpperCase("here is a string"));
```

```
const makeUpperCase = aString => {
      return aString.toUpperCase();
}

console.log(makeUpperCase("here is a string"));
```

Removing Curly Braces and Return Keyword

If you have just one statement and it is a return statement, you can also remove the curly braces and the return keyword.

```
const makeUpperCase = aString => {
      return aString.toUpperCase();
}

console.log(makeUpperCase("here is a string"));
```

```
const makeUpperCase = aString => aString.toUpperCase();

console.log(makeUpperCase("here is a string"));
```

If You Have More Than One Parameter

If you have more than one parameter, you need to put the parentheses back in, so that the code is not ambiguous.

```
const fullName = (fname, lname) => `${fname} ${lname}`;

console.log(fullName('Bill', 'Mead'));
```

No Parameters & More than a Return Statement

If you have no parameters, and more than one line in the function, or it's not returning a variable, you need to include the parentheses and the curly braces again:

```
const fullName = () => {
     const fname = "Bill";
     const lname = "Mead";
     return `${fname} ${lname}`;
}

console.log(fullName());
```

When to use Arrow Functions

Arrow functions aren't great in every case, but they can make your code easier to read in some cases.

```
const fruit = ["banana", "apple", "lemon", "kiwi"];
const upperFruit = [];

fruit.forEach( function(thisFruit){
      upperFruit.push(thisFruit.toUpperCase());
});

console.log(upperFruit);
```

The ForEach Example

You should get something like the code below. Because there isn't a one line return statement, you need to keep the curly braces, but still, it is fairly expressive.

```
const fruit = ["banana", "apple", "lemon", "kiwi"];
const upperFruit = [];

fruit.forEach( thisFruit => {
      upperFruit.push(thisFruit.toUpperCase());
});

console.log(upperFruit);
```

Constructor Function Expression

Let’s look at another example.

This is a constructor function.

```
const Employee = function( fname, lname, jobTitle ){
      this.fname = fname;
      this.lname = lname;
      this.position = jobTitle;
      this.fullName = function(){
            return `${fname} ${lname}`;
      }
}

const id1234 = new Employee('Bob', 'Smith', 'Mechanic');

console.log( id1234.fname );
console.log( id1234.fullName() );
```

fullName Converted to Arrow Function

Did you get this?

What happens if you convert the constructor function into an arrow function

```
const Employee = function( fname, lname, jobTitle ){
      this.fname = fname;
      this.lname = lname;
      this.position = jobTitle;
      this.fullName = () => `${fname} ${lname}`;

}

const id1234 = new Employee('Bob', 'Smith', 'Mechanic');

console.log( id1234.fname );
console.log( id1234.fullName() );
```

Error - Constructor Function Converted to Arrow

Arrow functions can not be used as constructor functions because the scope of the keyword `this` is different in arrow functions than it is in the more traditional function expression syntax.

```
const Employee = ( fname, lname, jobTitle ) => {
      this.fname = fname;
      this.lname = lname;
      this.position = jobTitle;
      this.fullName = () => `${fname} ${lname}`;

}

const id1234 = new employee('Bob', 'Smith', 'Mechanic');

console.log( id1234.fname );
console.log( id1234.fullName() );
```

Summary

Arrow functions can be useful, expressive and more compact, but they should not be overused, and there are times when they just won’t work.

For more details check out this excellent article: When 'Not' to Use Arrow Functions

Also check out more details about arrow functions on the MDN.