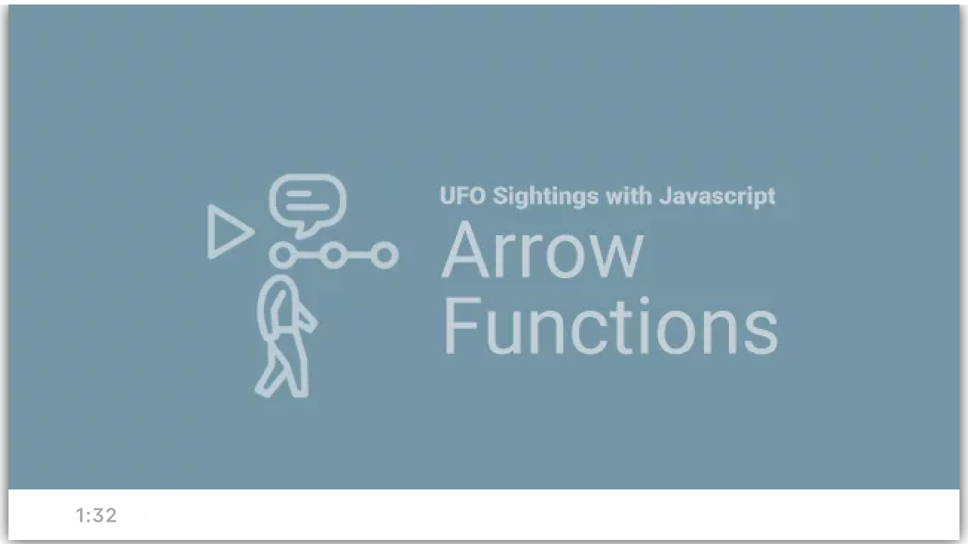
11.3.3 From Simple Functions to Arrow Functions

Having been introduced to JavaScript functions, Dana is now feeling a bit more confident about her JavaScript coding skills. She's excited to explore a shortcut followed by JavaScript insiders: arrow functions, which JavaScript experts use to convert standard functions into a single line of code. That's right: a single line.

Dana is excited about this insider trick because her collection of UFO data is somewhat extensive, and she has a feeling that her code will be complex. She also knows that arrow functions are one of the most popular aspects of the ES6 update, so she's eager to further integrate into the JavaScript community by mastering this function type.

Functions in JavaScript can easily become bulky and difficult to understand. Thankfully, any standard function in JavaScript can be refactored into an arrow function. **Arrow functions** complete the same functions as regular functions, but they use a more compact and concise syntax that makes a code script shorter and easier to read. Watch the following video to learn more about arrow functions.



NOTE

> **Arrow functions** are also known as fat arrow functions because they are introduced with a "fat arrow": =>
>
> This type of function is very similar to how a Python lambda function is written.

Let's take a look at a simple function.

```
// Simple JavaScript log statement
function printHello() {
  return "Hello there!";
}
```

This function, while already fairly short and sweet, can be condensed even further. In the console, type the following code and then press Enter.

```
printHello = () => "Hello there!";
```

When the function is called, our statement will be printed to the console. This is a pretty big change from traditional JavaScript functions.

Let's break down the differences in a bit more detail.

1. The arrow function collapses the function from 3 lines to 1 line, which is a significant reduction in characters.
2. The `function` keyword is not part of the arrow function. This is because the arrow symbol (=>) indicates that this block (or line) of code is a function.
3. The `return` keyword and `console.log()` are removed because with this new syntax, JavaScript inherently knows what will be returned.

Let's convert another function, this time with parameters. Here's the original function:

```
// Original addition function
function addition(a, b) {
  return a + b;
}
```

In your code editor, type the following:

```
// Converted to an arrow function
addition = (a, b) => a + b;
```

Once again, a multi-line function has been reduced to a single line. We have removed the `function` keyword, the curly brackets, and the return statement, and added a fat arrow to indicate that "addition" is a function. It's clear and easy to read—and it performs the same way as the original function!

Now let's step it up one more time and convert the `doubleAddition` function, shown below.

```
// Original doubleAddition function
function doubleAddition(c, d) {
  var total = addition(c, d) * 2;
  return total;
}
```

Even this function can be refactored into a single line. Let's begin the process by following the standard syntax: the name of the function, an equals sign, and then the parameters.

```
doubleAddition = (c, d)
```

The next step in refactoring is to add the fat arrow followed by the argument. In this case, the argument is the second function.

```
=> addition(c, d) * 2;
```

SKILL DRILL

Use the newly refactored `doubleAddition` function to find the total of 33 and 25.

Familiarity with both types—traditional functions and arrow functions—is important. Both are used often in development, and by the time we're done with this project, we'll have used a combination of the two.

Also, keep in mind that while arrow functions are clear and readable, there are still cases in which traditional functions are necessary. For example, when we want to place a function within another function, we would need to use a traditional function.