Arrow Function JavaScript Tutorial – How to Declare a JS Function with the New ES6 Syntax

freecodecamp.org/news/arrow-function-javascript-tutorial-how-to-declare-a-js-function-with-the-new-es6-syntax

June 2, 2020

You've probably seen arrow functions written a few different ways.

```
const addTwo = (num) => {return num + 2;};
```

```
const addTwo = (num) => num + 2;
```

```
const addTwo = num => num + 2;
```

```
const addTwo = a => {
 const newValue = a + 2;
 return newValue;
};
```

Some have parentheses around the parameters, while others don't. Some use curly brackets and the `return` keyword, others don't. One even spans multiple lines, while the others consist of a single line.

Interestingly, when we invoke the above arrow functions with the same argument we get the same result.

```
console.log(addTwo(2));
```

How do you know which arrow function syntax to use? That's what this article will uncover: how to declare an arrow function.

A Major Difference

Arrow functions are another—more concise—way to write function expressions. However, they don't have their own binding to the **this** keyword.

```
const addNumbers = function(number1, number2) {
  return number1 + number2;
};
```

```
const addNumbers = (number1, number2) => number1 + number2;
```

When we invoke these functions with the same arguments we get the same result.

```
console.log(addNumbers(1, 2));
```

There's an important syntactical difference to note: arrow functions use the arrow `=>` instead of the `function` keyword. There are other differences to be aware of when you write arrow functions, and that's what we'll explore next.

Parentheses

Some arrow functions have parentheses around the parameters and others don't.

```
const addNums = (num1, num2) => num1 + num2;
```

```
const addTwo = num => num + 2;
```

As it turns out, the number of parameters an arrow function has determines whether or not we need to include parentheses.

An arrow function with **zero parameters** requires parentheses.

```
const hello = () => "hello";
console.log(hello());
```

An arrow function with **one parameter** does *not* require parentheses. In other words, parentheses are optional.

```
const addTwo = num => num + 2;
```

So we can add parentheses to the above example and the arrow function still works.

```
const addTwo = (num) => num + 2;
console.log(addTwo(2));
```

An arrow function with **multiple parameters** requires parentheses.

```
const addNums = (num1, num2) => num1 + num2;
console.log(addNums(1, 2));
```

Arrow functions also support **rest parameters** and **destructuring**. Both features require parentheses.

This is an example of an arrow function with a **rest parameter**.

```
const nums = (first, ...rest) => rest;
console.log(nums(1, 2, 3, 4));
```

And here's one that uses **destructuring**.

```
const location = {
  country: "Greece",
  city: "Athens"
};

const travel = ({city}) => city;

console.log(travel(location));
```

To summarize: if there's only one parameter—and you're not using rest parameters or destructuring—then parentheses are optional. Otherwise, be sure to include them.

The Function Body

Now that we've got the parentheses rules covered, let's turn to the function body of an arrow function.

An arrow function body can either have a "concise body" or "block body". The body type influences the syntax.

First, the "concise body" syntax.

```
const addTwo = a => a + 2;
```

The "concise body" syntax is just that: it's concise! We don't use the `return` keyword or curly brackets.

If you have a one-line arrow function (like the example above), then the value is implicitly returned. So you can omit the `return` keyword and the curly brackets.

Now let's look at "block body" syntax.

```
const addTwo = a => {
   const total = a + 2;
   return total;
}
```

Notice that we use *both* curly brackets and the `return` keyword in the above example.

You normally see this syntax when the body of the function is more than one line. And that's a key point: wrap the body of a multi-line arrow function in curly brackets and use the `return` keyword.

Objects and Arrow Functions

There's one more syntax nuance to know about: wrap the function body in parentheses when you want to return an object literal expression.

```
const f = () => ({
 city:"Boston"
})
console.log(f().city)
```

Without the parentheses, we get an error.

```
const f = () => {
  city:"Boston"
}
```

If you find the arrow function syntax a bit confusing, you're not alone. It takes some time to get familiar with it. But being aware of your options and requirements are steps in that direction.

I write about learning to program and the best ways to go about it (amymhaddad.com).