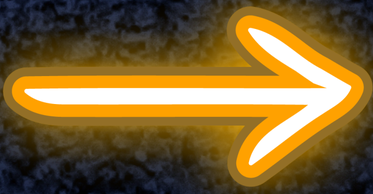
Arrow Function

Introduction

Arrow functions were added to JavaScript for two reasons:

1. To provide a more concise way for creating functions.
2. They work better as real functions inside methods: Methods can refer to the object that received a method call via the special variable this. Arrow functions can access the this of a surrounding method, ordinary functions can't (because they have their own this).

We'll first examine the syntax of arrow functions and then how this works in various functions.



The syntax of arrow functions

Let's review the syntax of an anonymous function expression:

```
const f = function (x, y, z) { return 123 };
```

The (roughly) equivalent arrow function looks as follows. Arrow functions are expressions.

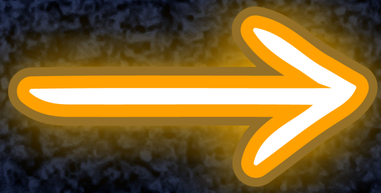
```
const f = (x, y, z) => { return 123 };
```

Here, the body of the arrow function is a block. But it can also be an expression. The following arrow function works exactly like the previous one.

```
const f = (x, y, z) => 123;
```

If an arrow function has only a single parameter and that parameter is an identifier (not a destructuring pattern) then you can omit the parentheses around the parameter:

```
const id = x => x;
```



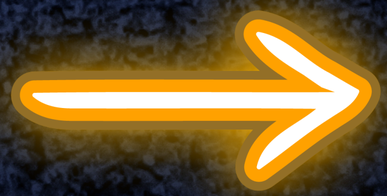
The syntax of arrow functions

That is convenient when passing arrow functions as parameters to other functions or methods:

```
> [1,2,3].map(x => x+1) [ 2, 3, 4 ]
```

This previous example demonstrates one benefit of arrow functions – conciseness. If we perform the same task with a function expression, our code is more verbose:

```
[1,2,3].map(function (x) { return x+1 });
```



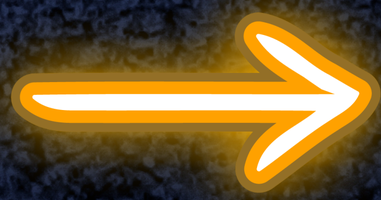
Syntax pitfall: returning an object literal from an arrow function

If you want the expression body of an arrow function to be an object literal, you must put the literal in parentheses:

```
const func1 = () => ({a: 1});
assert.deepEqual(func1(), { a: 1 });
```

If you don’t, JavaScript thinks, the arrow function has a block body (that doesn’t return anything):

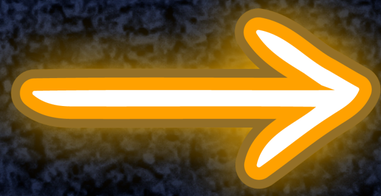
```
const func2 = () => {a: 1};
assert.deepEqual(func2(), undefined);
```



Syntax pitfall: returning an object literal from an arrow function

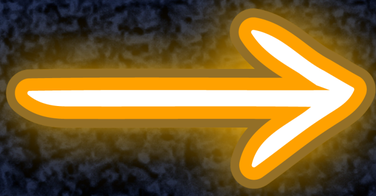
{a: 1} is interpreted as a blockwith the label a: and the expression statement 1. Without an explicit return statement, the block body returns undefined.

This pitfall is caused by syntactic ambiguity: object literals and code blocks have the same syntax. We use the parentheses to tell JavaScript that the body is an expression (an object literal) and not a statement (a block).



Limitations of Arrow Function

- Arrow functions don't have their own bindings to this or super, and should not be used as methods.
- Arrow functions don't have access to the new.target keyword.
- Arrow functions aren't suitable for call, apply and bind methods, which generally rely on establishing a scope.
- Arrow functions cannot be used as constructors.
- Arrow functions cannot use yield, within its body.



AI Arif

FOLLOW

For More