

Theory: Object methods and keyword "this"

🕒 13 minutes

0 / 5 problems solved

Skip this topic

Start practicing

790 users solved this topic. Latest completion was about 16 hours ago.

When writing scripts, we usually need to create objects to present something from the real world: users, products, maps, and more. In real life, it is possible to consider the products in detail or put something in a shopping cart, so in programming it is also an option. Such actions in JavaScript are available thanks to **object methods**, meaning the functions you can use as object properties.

§1. Object method creation

Let's try to understand how object methods are created. To do this, consider the following example:

```
1 let product = {
2   name: "Microwave",
3   description: "With oven mode",
4   price: 398
5 };
6
7 product.giveDiscount = function() {
8   console.log("You have a 10% discount!");
9 };
10
11 product.giveDiscount(); // You have a 10% discount!
```

In this code, we have allowed the object to report a discount on the product by writing the method `giveDiscount`.

§2. Shortened recording

Setting functions as properties can be written shorter. For example:

```
1 let person = {
2   greetings: function() {
3     console.log("Hello");
4   }
5 };
```

We can also skip the keyword `function`:

```
1 let person = {
2   greetings() {
3     console.log("Hello");
4   }
5 };
```

Shortened syntax not only saves you time, but also makes the code more readable.

§3. "this" keyword

In natural language, we can point to a specific person, animal, or object using pronouns. In JavaScript you may just as well refer to a specific object using the keyword `this`:

Current topic:

Object methods and keyword "this" ...

Topic depends on:

✗ Arrow functions ...

✗ Type conversion ...

Topic is required for:

ForEach method ...

Table of contents:

[↑ Object methods and keyword "this"](#)

[§1. Object method creation](#)

[§2. Shortened recording](#)

[§3. "this" keyword](#)

[§4. "this" and arrow functions](#)

[Feedback & Comments](#)

```
1  let user = {
2    firstName: "Elliot",
3    lastName: "Alderson",
4    fullName() {
5      return this.firstName + " " + this.lastName;
6    }
7  };
8
9  console.log(user.fullName()); // Elliot Alderson
```

In the above example, with the help of `this`, the `fullName` function has accessed the `firstName` and `lastName` information stored in the `user` object because it is declared inside it.

Applying `this` to nested objects can create some confusion. In such situations, you should keep it in mind that the keyword `this` refers to the object in whose method it is used.

§4. *"this"* and arrow functions

Arrow functions *cannot* bind `this`. In other words, the arrow functions do not have their own `this`.

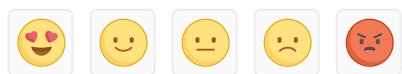
If we use `this` inside an arrow function, its value is taken from an external function declared the usual way:

```
1  let movie = {
2    name: "The Thirteenth Floor",
3    age: 1999,
4    getInfo() {
5
6      let arrow = () => console.log("The movie " + this.name + " was shot in " + thi
7      s.age);
8      arrow();
9    }
10 };
11
12 movie.getInfo(); // The movie The Thirteenth Floor was shot in 1999
```

In fact, the specifics of using `this` in JavaScript are not limited to the examples described above. Here we covered only the basics of working with this keyword, so there's a lot more to learn.

 Report a typo

77 users liked this theory. 1 didn't like it. What about you?



Start practicing

[Comments \(0\)](#)

[Hints \(0\)](#)

[Useful links \(0\)](#)

[Show discussion](#)