

Objects and Methods

In this section we discuss how methods interact with the objects they belong to, and look into the infamous `this` language construct.

Relevant Links

- [this in MDN](#)¹

Notes

A key characteristic of objects is that all objects “of the same class” tend to have access to a common set of functions, which are typically called the object methods.

In other languages, these would be placed in a “class”. Javascript doesn’t use classes. Instead, objects essentially inherit properties from other objects.

We will see how that happens more extensively soon, but for now let us try a silly example:

```
var a = {  
  name: "Pete",  
  greet: function() { return "Hello! I am " + a.name; }  
};  
a.greet();
```

Now let us suppose we want to create a second object. It will need its own name, but perhaps we could reuse the `greet` function. Here is an attempt at doing so:

```
var b = { name: "Myka" };  
b.greet = a.greet;  
b.greet();
```

Oops, that did not work! `greet` still only knows about Pete. `a.name` still references the `a` object.

What we need is a special way to refer to the “current object”, the object that the function was executed from. This is what the keyword “`this`” is for.

In a call like `o.f(...)`, any references to `this` in the function `f` will refer to the object `o`.

So here is a version of the above that works:

```
var a = {  
  name: "Pete",  
  greet: function() { return "Hello! I am " + this.name; }  
};  
var b = { name: "Myka" };  
b.greet = a.greet;
```

¹<https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/this>

Of course it's not always a great idea to create objects in quite this way, each of them having the methods they need as properties. But it a good example to illustrate some of the mechanics of this.

To understand Javascript's object model further, we will need to study prototypes.