

# Methods

## Lesson Objectives

When you complete this course, you will be able to:

- create your own methods.
- pass arguments to your methods.

## Objects and Methods

You know about objects and properties, and you know that `document.URL` refers to a property named `URL` that is in the object `document`. So what is `document.getElementById()`? We've used this a number of times when we want to **get an element from the DOM** so we can update a web page, like we did in the previous lab:

OBSERVE:

```
window.onload = init;
function init() {
  var div = document.getElementById("result");
  div.innerHTML =
    "The domain where this document lives is: " + document.domain + "<br>" +
    "The title of the document is: " + document.title + "<br>" +
    "The URL of the document is: " + document.URL;
}
```

In this example, we used `document.getElementById("result")` to get the `<div>` element with the id "result" so we could update the page with information from the `document` object's properties.

## A Method is a Function in an Object

We say that `getElementById()` is a *method*; that means it's a *function in an object*. In the case of `getElementById()`, this method is in the `document` object. To illustrate how you can create your own methods, why don't we add a method to the `Pet` constructor? Open `constructor.html` from the last lesson and then use the Save As icon to save it in your `/javascript1` folder as `method.html`. Then, edit it as shown:

## CODE TO TYPE:


```
<!doctype html>
<html lang="en">
<head>
  <title> Object Methods </title>
  <meta charset="utf-8">
  <script>
    window.onload = init;

    function Pet(type, name, weight, likes) {
      this.type = type;
      this.name = name;
      this.weight = weight;
      this.likes = likes;

      this.bark = function() {
        return "Woof!";
      };
    }

    function init() {
      var annie = new Pet("cat", "Annie", 6, ["sleeping", "teasing pickles"]);
      var willie = new Pet("dog", "Willie", 45, ["slobbering", "panting", "eating
"]);

      var div = document.getElementById("pets");
      div.innerHTML = annie.name + " is a " + annie.type + " and " + willie.name
      +
      " is a " + willie.type;
      div.innerHTML = willie.name + " says " + willie.bark();
    }
  </script>
</head>
<body>
  <div id="pets">
  </div>
</body>
</html>
```

Save it and click . You should see "Willie says Woof!" in the web page.

We changed the Pet constructor to include a property whose value is a function. The property name is **bark** and the value of the property is a function. You might be thinking that's an odd way to define a function, and you're right, it's different from what we've seen up to now.

Rather than writing **function bark() { ... }**, we write **this.bark = function() { ... }**, which looks a little odd. But this is just another way of defining a function, and it's how you define a method in an object: you're setting the value of the property **bark** to a *function value*. (Notice that, just like for the other properties we assign in the constructor, we end the assignment statement with a ";", so don't forget that.)

Try adding another method, **meow**, to the constructor and test it using **annie.meow()**:

#### CODE TO TYPE:

```
<!doctype html>
<html lang="en">
<head>
  <title> Object Methods </title>
  <meta charset="utf-8">
  <script>
    window.onload = init;

    function Pet(type, name, weight, likes) {
      this.type = type;
      this.name = name;
      this.weight = weight;
      this.likes = likes;

      this.meow = function() {
        return "Meow!";
      };
      this.bark = function() {
        return "Woof!";
      };
    }

    function init() {
      var annie = new Pet("cat", "Annie", 6, ["sleeping", "teasing pickles"]);
      var willie = new Pet("dog", "Willie", 45, ["slobbering", "panting", "eating
"]);

      var div = document.getElementById("pets");
      div.innerHTML = willie.name + " says " + willie.bark();
      div.innerHTML += "<br>" + annie.name + " says " + annie.meow();
    }
  </script>
</head>
<body>
  <div id="pets">
  </div>
</body>
</html>
```

Notice that when you add methods to objects, you're adding *behavior* to that object. Now the object can *do* something. In this example, now our Pet objects can bark and meow!

## Methods with Parameters

So, your Pet objects have a method **bark()**; what if you want to pass arguments to the method? You can add parameters to the method definition, in the parentheses, just like you would in a normal function:

#### CODE TO TYPE:


```
<!doctype html>
<html lang="en">
<head>
  <title> Object Methods </title>
  <meta charset="utf-8">
  <script>
    window.onload = init;

    function Pet(type, name, weight, likes) {
      this.type = type;
      this.name = name;
      this.weight = weight;
      this.likes = likes;

      this.meow = function() {
        return "Meow!";
      };
      this.bark = function(howMany) {
        var says = "";
        for (var i = 0; i < howMany; i++) {
          says += "Woof! ";
        }
        return says;
      };
    }

    function init() {
      var annie = new Pet("cat", "Annie", 6, ["sleeping", "teasing pickles"]);
      var willie = new Pet("dog", "Willie", 45, ["slobbering", "panting", "eating"]);

      var div = document.getElementById("pets");
      div.innerHTML = willie.name + " says " + willie.bark(3);
      div.innerHTML += "<br>" + annie.name + " says " + annie.meow();
    }
  </script>
</head>
<body>
  <div id="pets">
  </div>
</body>
</html>
```

Save it and click [Preview](#) . You should see your page updated with the text "Willie says Woof! Woof! Woof!".

We pass an argument into the pet object's **bark()** method which now has one parameter, **howMany**. We use this parameter to determine how many times the pet should say "Woof!". In this case, we passed in 3, so the **willie** object says "Woof!" three times. Notice that you can declare local variables, and return values from methods, just like you can in a regular function.

## This

Suppose you don't like having both **meow()** and **bark()** methods in your Pet objects because you could easily have a situation where you have cats barking and dogs meowing, and that would be bad. You want to replace these methods with another one, **speak()**, and pass in the appropriate sound that a given pet will make when you construct the Pet.

## CODE TO TYPE:

```
<!doctype html>
<html lang="en">
<head>
  <title> Object Methods </title>
  <meta charset="utf-8">
  <script>
    window.onload = init;


    function Pet(type, name, weight, likes, sound) {
      this.type = type;
      this.name = name;
      this.weight = weight;
      this.likes = likes;
      this.sound = sound;

      this.meow = function() {
  return "Meow!";
};
this.bark = function(howMany) {
  var says = "";
  for (var i = 0; i < howMany; i++) {
    says += "Woof! ";
  }
  return says;
};

      this.speak = function(howMany) {
        var says = "";
        for (var i = 0; i < howMany; i++) {
          says += this.sound + " ";
        }
        return says;
      };
    }

    function init() {
      var annie = new Pet("cat", "Annie", 6, ["sleeping", "teasing pickles"], "Meow");
      var willie = new Pet("dog", "Willie", 45, ["slobbering", "panting", "eating"], "Woof!");

      var div = document.getElementById("pets");
      div.innerHTML = willie.name + " says " + willie.speak(3);
    }
  </script>
</head>
<body>
  <div id="pets">
  </div>
</body>
</html>
```

Save it and click . You should see the same message, "Willie says Woof! Woof! Woof!" in your web page, but now you're using the **speak()** method instead of the **bark()** method. Try changing the code to use the **speak()** method on the annie object.

Notice that we pass in an additional argument to the parameter **sound**, the sound the pet makes ("Meow" for annie, and "Woof!" for willie). As usual, we use `this.sound = sound;` to set the *value of the property **this.sound** to the value of the parameter*.

We deleted the **meow()** and **bark()** methods and replaced them with one method, **speak()**, which does the same thing, except now it uses (with **says += this.sound + " "**;) the **sound** property to create the **says** string that it returns.

Notice that we use **this.sound** to access the property within the function. *This is important!* If you *don't* use **this.sound**, and use just **sound** instead, you're accessing the *parameter sound*, *not* the property of the object! In this case, it wouldn't matter, but what if you don't name your parameters and properties the same? In that case you'll get an error because **sound** would not be a variable. What if you changed the value of the **sound** property after creating the object? In that case, you won't get the right return value from the **speak()** method. Try this:

CODE TO TYPE:

```
<!doctype html>
<html lang="en">
<head>
  <title> Object Methods </title>
  <meta charset="utf-8">
  <script>
    window.onload = init;


    function Pet(type, name, weight, likes, sound) {
      this.type = type;
      this.name = name;
      this.weight = weight;
      this.likes = likes;
      this.sound = sound;

      this.speak = function(howMany) {
        var says = "";
        for (var i = 0; i < howMany; i++) {
          says += this.sound + " ";
        }
        return says;
      };
    }

    function init() {
      var annie = new Pet("cat", "Annie", 6, ["sleeping", "teasing pickles"], "Meow");
      var willie = new Pet("dog", "Willie", 45, ["slobbering", "panting", "eating"], "Woof!");

      var div = document.getElementById("pets");
      div.innerHTML = willie.name + " says " + willie.speak(3);

      annie.sound = "Purrrr";
      div.innerHTML += "<br>" + annie.name + " says " + annie.speak(2);
    }
  </script>
</head>
<body>
  <div id="pets">
  </div>
</body>
</html>
```

Save it and click . You *should* see the message "Purrrr Purrrr" in the page—but because we didn't use **this.sound**, it didn't work correctly! You'll see "Meow Meow" in the page instead.

### How does JavaScript know which object "this" is?

When you create a "new" object, like **var willie = new Pet(...)**, JavaScript makes sure that **this** points to that new (willie) object. That way, each new object has its own value for **this** that points to itself. So willie has a **this** that points to willie, and annie has a **this** that points to annie.

Copyright © 1998-2014 O'Reilly Media, Inc.



*This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License.  
See <http://creativecommons.org/licenses/by-sa/3.0/legalcode> for more information.*