4

OBJECTS

Objects in JavaScript are very similar to arrays, but objects use strings instead of numbers to access the different elements. The strings are called *keys* or *properties*, and the elements they point to are called *values*. Together these pieces of information are called *key-value pairs*. While arrays are mostly used to represent lists of multiple things, objects are often

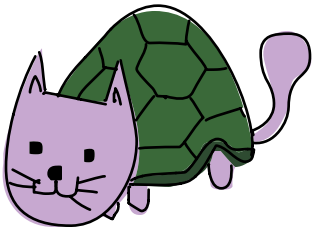
used to represent single things with multiple characteristics, or *attributes*. For example, in Chapter 3 we made several arrays that listed different animal names. But what if we wanted to store different pieces of information about one animal?

CREATING OBJECTS

We could store lots of information about a single animal by creating a JavaScript object. Here's an object that stores information about a three-legged cat named Harmony.

```
var cat = {
  "legs": 3,
  "name": "Harmony",
  "color": "Tortoiseshell"
};
```

Here we create a variable called `cat` and assign an object to it with three key-value pairs. To create an object, we use curly brackets, {}, instead of the straight brackets we used to make arrays. In between the curly brackets, we enter key-value pairs. The curly brackets and everything in between them are called an *object literal*. An object literal is a way of creating an object by writing out the entire object at once.



NOTE

We've also seen array literals (for example, `["a", "b", "c"]`), number literals (for example, `37`), string literals (for example, `"moose"`), and Boolean literals (`true` and `false`). Literal *just means that the whole value is written out at once, not built up in multiple steps.*

For example, if you wanted to make an array with the numbers 1 through 3 in it, you could use the array literal `[1, 2, 3]`. Or you could create an empty array and then use the `push` method to add 1, 2, and 3 to the array. You don't always know at first what's going to be in your array or object, which is why you can't always use literals to build arrays and objects.

Figure 4-1 shows the basic syntax for creating an object.

When you create an object, the key goes before the colon (:), and the value goes after. The colon acts a bit like an equal sign—the values on the right get assigned to the names on the left, just like when you create variables. In between each key-value pair, you have to put a comma. In our example, the commas are at the ends of the lines—but notice that you don't need a comma after the last key-value pair (`color: "Tortoiseshell"`). Because it's the last key-value pair, the closing curly bracket comes next, instead of a comma.

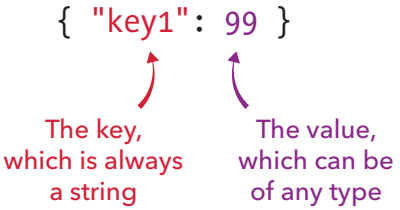


Figure 4-1: The general syntax for creating an object

KEYS WITHOUT QUOTES

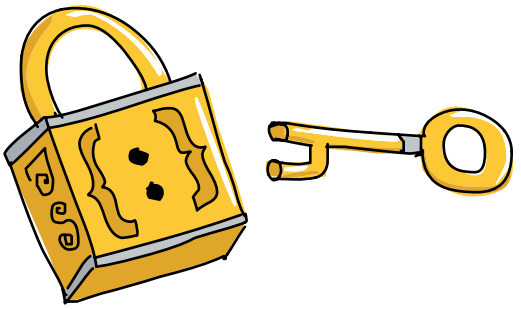
In our first object, we put each key in quotation marks, but you don't necessarily need quotes around the keys—this is a valid `cat` object literal as well:

```
var cat = {
  legs: 3,
  name: "Harmony",
  color: "Tortoiseshell"
};
```

JavaScript knows that the keys will always be strings, which is why you can leave out the quotes. If you don't put quotes around the keys, the unquoted keys have to follow the same rules as variable names: spaces aren't allowed in an unquoted key, for example. If you put the key in quotes, then spaces are allowed:

```
var cat = {
  legs: 3,
  "full name": "Harmony Philomena Snuggly-Pants Morgan",
  color: "Tortoiseshell"
};
```

Note that, while a key is always a string (with or without quotes), the value for that key can be any kind of value, or even a variable containing a value.



You can also put the whole object on one line, but it can be harder to read like that:

```
var cat = { legs: 3, name: "Harmony", color: "Tortoiseshell" };
```

ACCESSING VALUES IN OBJECTS

You can access values in objects using square brackets, just like with arrays. The only difference is that instead of the index (a number), you use the key (a string).

```
cat["name"];
"Harmony"
```

Just as the quotes around keys are optional when you create an object literal, the quotes are also optional when you are accessing keys in objects. If you're not going to use quotes, however, the code looks a bit different:

```
cat.name;
"Harmony"
```

This style is called *dot notation*. Instead of typing the key name in quotes inside square brackets after the object name, we just use a period, followed by the key, without any quotes. As with unquoted keys in object literals, this will work only if the key doesn't contain any special characters, such as spaces.

Instead of looking up a value by typing its key, say you wanted to get a list of all the keys in an object. JavaScript gives you an easy way to do that, using `Object.keys()`:

```
var dog = { name: "Pancake", age: 6, color: "white", bark: "Yip yap ↵
yip!" };
var cat = { name: "Harmony", age: 8, color: "tortoiseshell" };
Object.keys(dog);
["name", "age", "color", "bark"]
Object.keys(cat);
["name", "age", "color"]
```

`Object.keys(`*anyObject*`)` returns an array containing all the keys of *anyObject*.

ADDING VALUES TO OBJECTS

An empty object is just like an empty array, but it uses curly brackets, { }, instead of square brackets:

```
var object = {};
```

You can add items to an object just as you'd add items to an array, but you use strings instead of numbers:

```
var cat = {};
cat["legs"] = 3;
cat["name"] = "Harmony";
cat["color"] = "Tortoiseshell";
cat;
{ color: "Tortoiseshell", legs: 3, name: "Harmony" }
```

Here, we started with an empty object named `cat`. Then we added three key-value pairs, one by one. Then, we type `cat;`, and the browser shows the contents of the object. Different browsers may output objects differently, though. For example, Chrome (at the time I'm writing this) outputs the `cat` object like this:

```
Object {legs: 3, name: "Harmony", color: "Tortoiseshell"}
```

While Chrome prints out the keys in that order (`legs`, `name`, `color`), other browsers may print them out differently. This is

because JavaScript doesn't store objects with their keys in any particular order.

Arrays obviously have a certain order: index 0 is before index 1, and index 3 is after index 2. But with objects, there's no obvious way to order each item. Should `color` go before `legs` or after? There's no "correct" answer to this question, so objects simply store keys without assigning them any particular order, and as a result different browsers will print the keys in different orders. For this reason, you should never write a program that relies on object keys being in a precise order.

ADDING KEYS WITH DOT NOTATION

You can also use dot notation when adding new keys. Let's try the previous example, where we started with an empty object and added keys to it, but this time we'll use dot notation:

```
var cat = {};
cat.legs = 3;
cat.name = "Harmony";
cat.color = "Tortoiseshell";
```

If you ask for a property that JavaScript doesn't know about, it returns the special value `undefined`. `undefined` just means "There's nothing here!" For example:

```
var dog = {
  name: "Pancake",
  legs: 4,
  isAwesome: true
};
dog.isBrown;
undefined
```

Here we define three properties for `dog`: `name`, `legs`, and `isAwesome`. We didn't define `isBrown`, so `dog.isBrown` returns `undefined`.