SAMPLE CHAPTER

Get Programming with JavaScript

John R. Larsen

manning

Get Programming with JavaScript

by John R. Larsen

Chapter 3

brief contents

3 Objects: grouping your data

This chapter covers

- Organizing information with JavaScript objects
- Creating objects
- Adding properties to objects
- Accessing properties using dot notation
- Examples of objects

In chapter 2 you saw how to declare variables and assign them values, and you considered the variables you could use to model a player in *The Crypt*. As your programs grow, so does the number of variables you use; you need ways to organize all this data, to make your programs easier to understand and easier to update and add to in the future.

Sometimes it makes sense to group items and see them as a whole. Consider a first-aid kit; we happily treat it as a single item—"Have you packed the first-aid kit?" "Pass the first-aid kit." "We need the first-aid kit, *now*!"—but will quickly switch focus to its contents when the need arises—"Pass the antiseptic and the bandages from the first-aid kit, please." A number of items is neatly encapsulated by a single object.

This chapter introduces JavaScript objects, a simple and efficient way to collect variables together so that you can pass them around as a group rather than individually.

3.1 A need for organization

Your library of adventure stories is growing, and you decide to write a program to keep track of your precious collection. The following listing shows variables you use to generate this test output on the console:

```
> The Hobbit by J. R. R. Tolkien
```

Listing 3.1 Using variables to represent a book (http://jsbin.com/fucuxah/edit?js,console)

```
var bookTitle;
var bookAuthor;

bookTitle = "The Hobbit";
bookAuthor = "J. R. R. Tolkien";

console.log(bookTitle + " by " + bookAuthor);
```

Declare the variables you'll use in the program

Assign values to the variables

Use the variables to display information about the book

First, you declare two variables, `bookTitle` and `bookAuthor`, using the `var` keyword. You're going to use those two names to store and access values in the program. You then assign strings (text) to your freshly created variables. You wrap the strings in quotation marks so JavaScript doesn't try to interpret them as keywords or variable names. Finally, you log a message to the console. You build the message by using the concatenation operator (the + symbol) to join three strings.

It may be early days but you certainly have more than one book. How can you cope with the variables needed as you buy more? You could have a different prefix for each book. The next listing ups the number of books to three, printing these messages to the console:

```
> There are three books so far ...
> The Hobbit by J. R. R. Tolkien
> Northern Lights by Philip Pullman
> The Adventures of Tom Sawyer by Mark Twain
```

Listing 3.2 Using prefixes to tell book variables apart (http://jsbin.com/qowagi/edit?js,console)

```
var book1Title = "The Hobbit";
var book1Author = "J. R. R. Tolkien";

var book2Title = "Northern Lights";
var book2Author = "Philip Pullman";

var book3Title = "The Adventures of Tom Sawyer";
var book3Author = "Mark Twain";
```

Declare variables and assign them values in one step

```
console.log("There are three books so far...");
console.log(book1Title + " by " + book1Author);
console.log(book2Title + " by " + book2Author);
console.log(book3Title + " by " + book3Author);
```

This works up to a point. But as the number of books and the number of facts about each book increase, the number of variables is harder to manage. It would be helpful to be able to group all of the information about a book together, using a single variable.

3.2 Creating objects

In the same way as it is easier to ask for a first-aid kit rather than for the scissors, antiseptic, bandages, and sticking plasters separately, it can be easier to ask for `book1` rather than `book1Title`, `book1Author`, `book1ISBN`, and so on separately. JavaScript provides us with the ability to create *objects* to group variables. Very specific notation, or *syntax*, is used to define a new object. Let's look at a full example and then break it down into stages.

Listing 3.3 shows how to create a book as an object rather than as separate variables. Figure 3.1 shows the output on JS Bin when you log the `book` object to the console.

Listing 3.3 A book as an object (http://jsbin.com/ruruko/edit?js,console)

```
var book;

book = {
    title : "The Hobbit",
    author : "J. R. R. Tolkien",
    published : 1937
};

console.log(book);
```

Figure 3.1 You log an object to the console on JS Bin.

When you run listing 3.3 on JS Bin, the console displays all the properties of your new `book` object, as well as telling you it's an object. Notice that it displays the properties in alphabetical order. The object itself doesn't order the properties; JS Bin has chosen an ordering purely for display.

Let's break down the object creation to get a clearer idea of what's going on and what all the different bits of notation represent.

3.2.1 Creating an empty object

In chapter 2 you saw that variables can be declared but not assigned a value until later in a program. You might have to wait for some user input or a response from a server or a reading from a sensor before you know the value you want to assign to the variable. In the same way, you can create an object with no properties, knowing that properties will be added at some point in the future.

To create an object, use curly braces, as in the following listing.

Listing 3.4 Creating an empty object (http://jsbin.com/kaqatu/edit?js,console)

```
var book;

book = {};
```

Declare a variable by using the var keyword

Create an object by using curly braces; assign it to the variable

You create an empty object, one with no properties, and assign it to the variable `book`. It's not much use without any properties, and you'll see how to add new properties to an existing object in section 3.4. But how would you create your book object with properties in place?

3.2.2 Properties as key-value pairs

The book in listing 3.3 includes three properties: its title, its author, and its year of publication. The values of those properties are `"The Hobbit"`, `"J. R. R. Tolkien"`, and `1937`. In a JavaScript object, the names of the properties are called *keys*. For `book` the keys are `title`, `author`, and `published`. When creating an object, you add a property by including its key and value, separated by a colon, between the curly braces. Figure 3.2 shows a property definition.

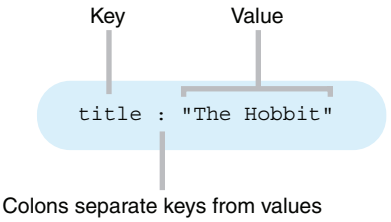

Figure 3.2 Set properties by using key-value pairs.

Another name for a key-value pair is a name-value pair, but we'll stick with keys and values in this book.

In the next listing you create an object with a single property.

Listing 3.5 An object with a single property (http://jsbin.com/robupi/edit?js,console)

```
var book;

book = {
    title : "The Hobbit"
};
```

You declare a variable and then create an object and assign it to the variable. The object has a single property. The key of the property is `title` and its value is `"The Hobbit"`. We usually simply say that the `title` property of `book` is `"The Hobbit"`.

Property values aren't restricted to number and string literals, like `50` or `"The Hobbit"`. You can also use previously declared variables as values. The following listing assigns the name of a book to a variable and then uses that variable as the value of an object's property.

Listing 3.6 Using a variable as a property value (http://jsbin.com/bafige/edit?js,console)

```
var book;
var bookName;

bookName = "The Adventures of Tom Sawyer";

book = {
    title : bookName
};
```

Assign the name of the book to a variable

Use the variable as the value of the title property

Having an object with a single property is a little extravagant; you might as well stick with a variable. Let's see how to create an object with more than one property.

When you need multiple properties, commas separate the key-value pairs. Figure 3.3 shows two properties as part of an object definition, and listing 3.7 creates two objects, each with two properties.

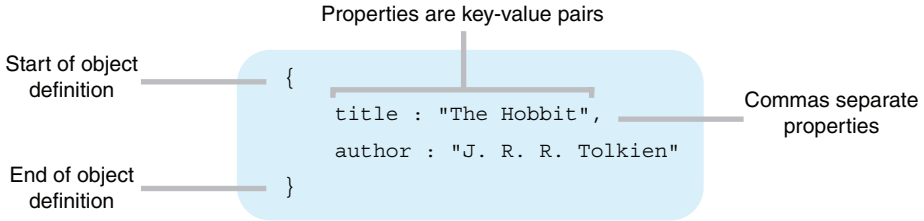

Figure 3.3 An object definition with two properties

Listing 3.7 Objects with multiple properties (http://jsbin.com/bawiqev/edit?js,console)

```
var book1;
var book2;

book1 = {
    title : "The Hobbit",
    author : "J. R. R. Tolkien"
};

book2 = {
    title : "Northern Lights",
    author : "Philip Pullman"
};
```

Separate the properties with a comma

Use key-value pairs to set each property

Now that you've created an object, you need to be able to access its properties.

3.3 Accessing object properties

We're comfortable with the concept of a first-aid kit as a single object we can pass around from person to person and take from place to place. It's only when we need to use the kit that we consider what's inside: antiseptic, scissors, bandages, and so on.

For JavaScript objects, to access the values of an object's properties you can use *dot notation*. Join the name of the variable to the name of the property, its key, with a period or dot. For a first-aid kit as an object you might use `kit.antiseptic` or `kit.scissors` or `kit.bandages`. And for books, to access the `author` property of the object assigned to the variable called `book`, you write `book.author` (figure 3.4).

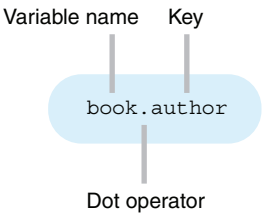

Figure 3.4 Accessing object properties using dot notation

In the next listing, you print the `title` and `author` properties of the `book` object to the console to give the following output:

```
> The Hobbit
> J. R. R. Tolkien
```

Listing 3.8 Using dot notation to access property values (http://jsbin.com/funiyu/edit?js,console)

```
var book;

book = {
    title     : "The Hobbit",
    author    : "J. R. R. Tolkien",
    published : 1937
};

console.log(book.title);
console.log(book.author);
```

Set properties using key-value pairs

Access property values using dot notation

You line up the colons in the object properties in listing 3.8 to aid readability. Although JavaScript will ignore the extra spaces, indenting blocks of code and lining up values can make your programs easier to read and follow, especially as they grow in size. And the easier your code is to read, the easier it is to maintain and update, both for you and for other programmers.

Replacing a bunch of separate variables with a single object helps you manage the complexity of your programs. You can think more clearly about how programs work when details are hidden until you need them. You consider a book a single entity in your program until you need to access the book's title or author or publication date. It may seem that replacing three variables with one variable and three properties isn't an improvement, but when you start to use objects with functions in chapter 7 and arrays in chapter 8, their economy and clarity will be more obvious.

You use property values just like variables. The code in the following listing concatenates each book's title with the string `" by "` and its author to give this output:

```
> The Hobbit by J. R. R. Tolkien
> Northern Lights by Philip Pullman
```

Listing 3.9 Concatenating string properties (http://jsbin.com/yoweti/edit?js,console)

```
var book1;
var book2;

book1 = {
    title: "The Hobbit",
    author: "J. R. R. Tolkien"
};

book2 = {
    title: "Northern Lights",
    author: "Philip Pullman"
};

console.log(book1.title + " by " + book1.author);
console.log(book2.title + " by " + book2.author);
```

3.4 Updating object properties

In a quiz app, players attempt questions one after another. The number of questions attempted, number of questions correct, and score will change over time. You can create a player object with initial values set and then update them whenever a question is attempted. Use dot notation to change a property that already exists or to add a new property to an object, as in the following listing.

Listing 3.10 Using dot notation to update a property (http://jsbin.com/mulimi/edit?js,console)

```
var player1;

player1 = {
    name: "Max",
    attempted: 0,
    correct: 0,
};

player1.attempted = 1;
player1.correct = 1;
player1.score = 50;
```

Set initial properties when creating the object

Update the property using dot notation

Add a new property and assign it a value

Your code in listing 3.10 sets the `attempted` and `correct` properties to an initial value when the object is created but then updates them to a new value. It uses the assignment operator, `=`, to assign the value, `1`, on the right of the operator, to the property, `player1.attempted`, on its left. You set the `attempted` and `correct` properties and then immediately update them; in the actual quiz app, the change would be in response to the player answering a question.

You can add new properties to an object after creating it. In listing 3.10, you assign the value `50` to the `score` property of the `player1` object.

```
player1.score = 50;
```

You didn't set the `score` property when creating the object; assigning a value automatically creates the property if it does not yet exist.

Just like using variables, you can use properties in a calculation and assign the result back to the property. The next listing shows code updating a player's properties:

```
> Max has scored 0
> Max has scored 50
```

Listing 3.11 Using a property in a calculation (http://jsbin.com/cuboko/edit?js,console)

```
var player1;

player1 = {
    name: "Max",
    score: 0
};

console.log(player1.name + " has scored " + player1.score);

player1.score = player1.score + 50;

console.log(player1.name + " has scored " + player1.score);
```

Evaluate the expression on the right and assign the result to the property

When you update the `score` property (in bold in the listing), JavaScript evaluates the right side of the assignment first. Because `player1.score` is `0`, the expression becomes `0 + 50`, which is `50`. JavaScript then assigns that value to the left side, that is, back to the `score` property. So, you update `player1.score` from `0` to `50`.

3.5 Further examples

Although developing *The Crypt* program gives you an ongoing context for introducing and discussing new concepts, a broader range of examples will help to deepen your understanding of the different ideas presented. You can also revisit some of these examples throughout the book to apply fresh techniques as you master them.

The examples all use curly braces to create an object and then assign the object to a variable created with the `var` keyword, in a single step.

3.5.1 Writing a blog

A blog is made up of blog posts. It would be good to have more information about each author, to be able to tag posts with keywords, and to add comments to each post. For now, here's a minimal object to represent a single post.

Listing 3.12 A blog post (http://jsbin.com/jiculu/edit?js,console)

```
var post = {
    id : 1,
    title : "My Crazy Space Adventure",
    author : "Philae",
    created : "2015-06-21",
    body : "You will not believe where I just woke up!! Only on a comet..."
};
```

3.5.2 Creating a calendar

Calendar events clearly involve dates. JavaScript does have a `Date` object, but you won't be using it in the book. The next listing represents dates as strings in a specific format.

Listing 3.13 A calendar event (http://jsbin.com/viroho/edit?js,console)

```
var event = {
    title : "Appraisal Meeting",
    startDate : "2016-10-04 16:00:00",
    endDate : "2016-10-04 17:00:00",
    location : "Martha's office",
    importance: 1,
    notes : 'Don\'t forget the portfolio!'
};
```

Include an apostrophe in the string when the string is delimited by double quotes

Use an escaped apostrophe when the string is delimited by single quotes

Notice how to cope with the apostrophe in the `notes` property. A backslash character before the apostrophe stops JavaScript from seeing it as the end of the string. The backslash is called an escape character and won't be shown.

```
event.notes = 'Don\'t forget the portfolio!';
```

Use the escape character to display double quotes when a string is already wrapped in double quotes.

```
var story = "She looked at me. \"What did you say?\" she asked.";
```

JavaScript also uses the backslash escape character to specify special characters like tabs and new lines. You'll see it in action throughout the book.

A calendar contains lots of event objects. In chapter 9, you'll see how to streamline the process of creating objects that all have a similar structure when you investigate constructor functions.

3.5.3 What's the weather like?

Online weather information services provide weather data you can use in your programs. The data is often formatted using JSON (JavaScript Object Notation—see chapter 20) that's very similar to the objects you've been learning about in this chapter. The data can be quite detailed with many properties. The next listing shows a cut-down version of location data supplied by one of these services.

Listing 3.14 Location for a weather app (http://jsbin.com/diguhe/edit?js,console)

```
var location = {
    "city"      : "San Francisco",
    "state"     : "CA",
    "country"   : "US",
    "zip"       : "94101",
    "latitude"  : 37.775,
    "longitude" : -122.418,
    "elevation" : 47.000
};
```

The property keys are within double quotation marks. JavaScript is happy for you to wrap keys, the property names, in quotation marks, single or double, although you haven't been doing that in the examples so far. In fact, quotation marks are required if the property name doesn't satisfy the rules for valid variable names discussed in chapter 2. You'll take a more detailed look at working with such property names in chapter 10. The JSON specification, which sets out how programs should transmit JavaScript object data as text across the internet, requires all keys to be in double quotation marks. Because it's sometimes required, many programmers recommend that property names always be placed within quotation marks to avoid inconsistency and potential errors.

This example also lines up the colons in the key-value pairs. Do you think that helps? Compare it to the other examples in this section. Are they easy to read and follow? You don't have to rigidly stick to one style or the other, although it's quite common for programmers to adopt particular stylistic habits over time.

3.5.4 The testing effect

A great way to learn is to test yourself often. A quiz app could represent its questions and answers as properties of objects like the one in the next listing.

Listing 3.15 A question and answer for a quiz app (http://jsbin.com/damoto/edit?js,console)

```
var questionAndAnswer = {
    question: "What is the capital of France?",
    answer1: "Bordeaux",
    answer2: "F",
    answer3: "Paris",
    answer4: "Brussels",
    correctAnswer: "Paris",
    marksForQuestion: 2
};
```

A quiz app is likely to include a small set of question types. Listing 3.15 is an example of a multiple-choice question type. Each type of question would have a fixed form of presentation. Templates are a great way of presenting copies of similarly structured data, and we'll look at them in more detail in chapter 19.

3.5.5 Create your own

Think of some programs you'd like to create. What kinds of objects could you design to represent entities in the program? Head to JS Bin and have a go at building objects and displaying properties on the console. Maybe share your creations or ask any questions they inspire over on the *Get Programming with JavaScript* forum at https://forums.manning.com/forums/get-programming-with-javascript.

3.6 The Crypt—a player object

You'll now apply your knowledge of JavaScript objects to *The Crypt*. Figure 3.5 shows where the focus of this section, a player object, fits into the overall structure of our ongoing game example.

In chapter 2, you considered the kind of information you need to store for players in *The Crypt*. For a single player, you start off with variables like these:

```
playerName = "Kandra";
playerHealth = 50;
playerPlace = "The Dungeon of Doom";
playerItems = "a rusty key, The Sword of Destiny, a piece of cheese";
```

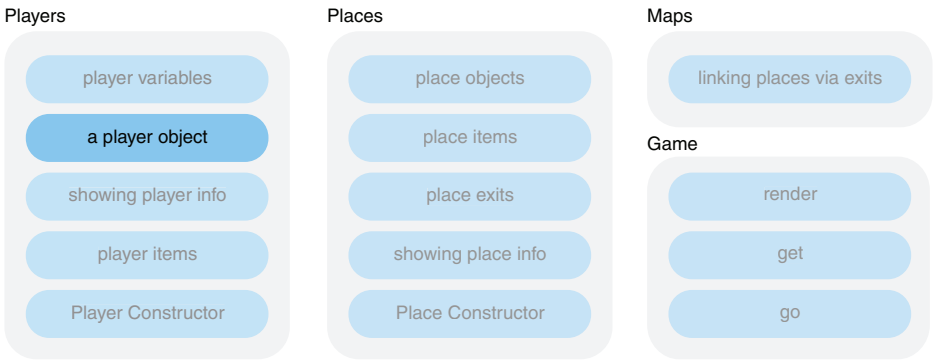

Figure 3.5 Elements in *The Crypt*

You then need to replicate those variables for each player in the game, maybe by using variable prefixes like `player1Name`, `player2Name`, and so on.

Clearly, it's much neater to use JavaScript objects as a way of grouping all of the information about a single player. Listing 3.16 shows how you can represent a player as an object and display some of their properties on the console. The output is as follows:

```
> Kandra
> Kandra is in The Dungeon of Doom
> Kandra has health 50
> Items: a rusty key, The Sword of Destiny, a piece of cheese
```

Listing 3.16 A player object (http://jsbin.com/qelene/edit?js,console)

```
var player;

player = {
    name: "Kandra",
    health: 50,
    place: "The Dungeon of Doom",
    items: "a rusty key, The Sword of Destiny, a piece of cheese"
};

console.log(player.name);
console.log(player.name + " is in " + player.place);
console.log(player.name + " has health " + player.health);
console.log("Items: " + player.items);
```

The last four lines of the listing are just for displaying player information. Having to repeat those lines of code every time you want to display a player's information seems a little tedious. It would be great to be able to write the lines of code once and then call them up on demand.

You're in luck! JavaScript lets you define *functions* to execute blocks of code whenever you need them. Functions are very powerful and will help streamline the display

of player properties and the creation of multiple player objects. You'll take a really detailed look at functions over the next four chapters.

3.7 Summary

- Group related variables as properties of an object.
- Define objects as collections of comma-separated properties between curly braces:

```
var player = { name : "Hadfield", location : "The ISS" };
```

- For each property use a key-value pair, with key and value separated by a colon:

```
name : "Hadfield"
```

- Access property values by using dot notation. If the object is assigned to a variable, join the property name to the variable name with a dot:

```
player.name
```

- Use properties in expressions just as you would variables:

```
console.log(player.name + " is in " + player.location);
```

- Assign values to properties using the assignment operator, `=`:

```
player.location = "On a space walk";
```

- Add new properties to existing objects whenever you want:

```
player.oxygen = 96;
```

JAVASCRIPT / PROGRAMMING

Get Programming with JavaScript

John R. Larsen Foreword by Remy Sharp

Are you ready to start writing your own web apps, games, and programs? You're in the right place! **Get Programming with JavaScript** is a hands-on introduction to programming for readers who have never written a line of code.

Since you're just getting started, this friendly book offers you lots of examples backed by careful explanations. As you go along, you'll find exercises to check your understanding and plenty of opportunities to practice your new skills. You don't need anything special to follow the examples—just the text editor and web browser already installed on your computer. We even give you links to working online code so you can see how everything should look live on your screen.

WHAT'S INSIDE

- All the basics—objects, functions, responding to users, and more
- Think like a coder and design your own programs
- Create a text-based adventure game
- Enhance web pages with JavaScript
- Run your programs in a web browser

No experience required! All you need is a web browser and an internet connection.

John Larsen is a web developer and professional teacher in the UK who has many years of experience working with students of all levels, helping them to successfully write their first lines of code.

"Provides the guidance you need to get started ..., the support to keep practicing, and the encouragement to enjoy the adventure."
—From the Foreword by Remy Sharp
Founder of JS Bin

"A great book for the new programmer who wants to learn JavaScript."
—Alvin Raj, Oracle

"An approachable and interactive way of learning JavaScript."
—Giselle Stidston, Breville Pty Ltd

"Great interactive code examples! Building a computer game was my favorite part of the book."
—Ivan Rubelj, Vipnet

To download their free eBook in PDF, ePub, and Kindle formats, owners of this book should visit manning.com/books/get-programming-with-javascript

ISBN-13: 978-1-61729-310-8
ISBN-10: 1-61729-310-5
53999
9 781617 293108

MANNING US $ 39.99 | Can $45.99