“The Missing Manual series is simply the most intelligent and usable series of guidebooks…”
—KEVIN KELLY, CO-FOUNDER OF *WIRED*

JavaScript & jQuery

the missing manual®

The book that should have been in the box®

Second Edition

Free Sampler

O’REILLY®

David Sawyer McFarland

Want to read more?

You can buy this book at **oreilly.com**
in print and ebook format.

Buy 2 books, get the 3rd FREE!
Use discount code: OPC10

All orders over $29.95 qualify for **free shipping** within the US.

It's also available at your favorite book retailer,
including the iBookstore, the Android Marketplace,
and Amazon.com.

O'REILLY®

Spreading the knowledge of innovators

oreilly.com

JavaScript & jQuery: The Missing Manual, Second Edition

by David Sawyer McFarland

Copyright © 2012 David Sawyer McFarland. All rights reserved.

Printed in the United States of America.

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly Media books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles: *safari.oreilly.com*. For more information, contact our corporate/institutional sales department: 800-998-9938 or *corporate@oreilly.com*.

Printing History:

July 2008:	First Edition.
October 2011:	Second Edition.

Nutshell Handbook, the Nutshell Handbook logo, the O'Reilly logo, and "The book that should have been in the box" are registered trademarks of O'Reilly Media, Inc. *JavaScript & jQuery: The Missing Manual*, The Missing Manual logo, Pogue Press, and the Pogue Press logo are trademarks of O'Reilly Media, Inc.

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and O'Reilly Media, Inc., was aware of a trademark claim, the designations have been printed in caps or initial caps.

While every precaution has been taken in the preparation of this book, the publisher and authors assume no responsibility for errors or omissions, or for damages resulting from the use of the information contained herein.

ISBN: 978-1-449-3-9902-3

[M]

Table of Contents

Part One: Getting Started with JavaScript

Part Two: Getting Started with jQuery

Part Three: Building Web Page Features

Part Four: Ajax: Communication with the Web Server

Part Five: Tips, Tricks, and Troubleshooting

CHAPTER

1

Writing Your First JavaScript Program

By itself, HTML doesn't have any smarts: It can't do math, it can't figure out if someone has correctly filled out a form, and it can't make decisions based on how a web visitor interacts with it. Basically, HTML lets people read text, look at pictures, and click links to move to other web pages with more text and pictures. In order to add intelligence to your web pages so they can respond to your site's visitors, you need JavaScript.

JavaScript lets a web page react intelligently. With it, you can create *smart* web forms that let visitors know when they've forgotten to include necessary information; you can make elements appear, disappear, or move around a web page (see Figure 1-1); you can even update the contents of a web page with information retrieved from a web server—without having to load a new web page. In short, JavaScript lets you make your websites more engaging and effective.

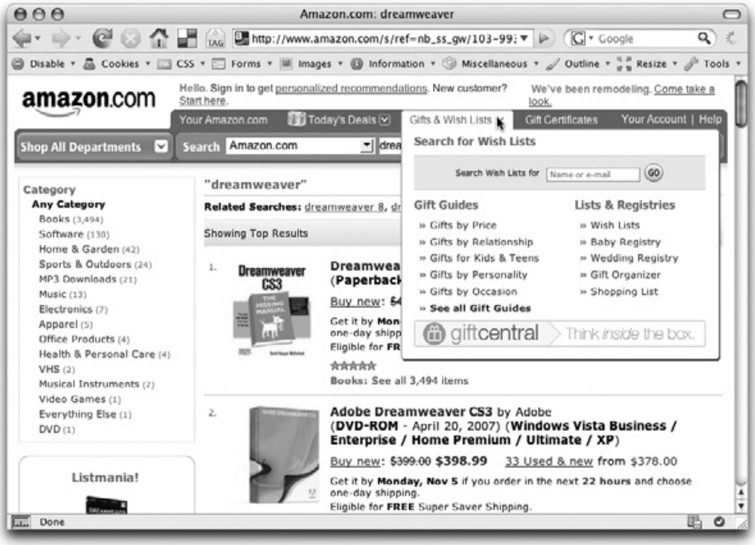


Figure 1-1: JavaScript lets web pages respond to visitors. On Amazon.com, mousing over the "Gifts & Wish Lists" link opens a tab that floats above the other content on the page and offers additional options.

Note: Actually, HTML5 *does* add some smarts to HTML—including basic form validation. But since not all browsers support these nifty additions (and because you can do a whole lot more with forms and JavaScript), you still need JavaScript to build the best, most user-friendly and interactive forms. You can learn more about HTML5 and web forms in Mark Pilgrim's *HTML5: Up and Running* (O'Reilly).

Introducing Programming

For a lot of people, the term "computer programming" conjures up visions of super-intelligent nerds hunched over keyboards, typing nearly unintelligible gibberish for hours on end. And, honestly, some programming is like that. Programming can seem like complex magic that's well beyond the average mortal. But many programming concepts aren't difficult to grasp, and as programming languages go, JavaScript is relatively friendly to nonprogrammers.

Still, JavaScript is more complex than either HTML or CSS, and programming often is a foreign world to web designers; so one goal of this book is to help you think more like a programmer. Throughout this book, you'll learn fundamental programming concepts that apply whether you're writing JavaScript, ActionScript, or even writing a desktop program using C++. More importantly, you'll learn how to approach a programming task so you'll know exactly what you want to do before you start adding JavaScript to a web page.

Many web designers are immediately struck by the strange symbols and words used in JavaScript. An average JavaScript program is sprinkled with symbols ({ } [ ] ; , ()

!=) and full of unfamiliar words (var, null, else if). It's like staring at a foreign language, and in many ways, learning a programming language is a lot like learning another language. You need to learn new words, new punctuation, and understand how to put them together so you can communicate successfully.

In fact, every programming language has its own set of key words and characters, and its own set of rules for putting those words and characters together—the language's *syntax*. Learning JavaScript's syntax is like learning the vocabulary and grammar of another language. You'll need to memorize the words and rules of the language (or at least keep this book handy as a reference). When learning to speak a new language, you quickly realize that placing an accent on the wrong syllable can make a word unintelligible. Likewise, a simple typo or even a missing punctuation mark can prevent a JavaScript program from working, or trigger an error in a web browser. You'll make plenty of mistakes as you start to learn to program—that's just the nature of programming.

UP TO SPEED

The Client Side vs. the Server Side

JavaScript is a *client-side* language, which (in English) means that it works inside a web browser. The alternative type of web programming language is called a *server-side* language, which you'll find in pages built around PHP, .NET, ASP, ColdFusion, Ruby on Rails, and other web server technologies. Server-side programming languages, as the name suggests, run on a web server. They can exhibit a lot of intelligence by accessing databases, processing credit cards, and sending email around the globe. The problem with server-side languages is that they require the web browser to send requests to the web server, forcing visitors to wait until a new page arrives with new information.

Client-side languages, on the other hand, can react immediately and change what a visitor sees in his web browser without the need to download a new page. Content can appear or disappear, move around the screen, or automatically update based on how a visitor interacts with the page. This responsiveness lets you create websites that feel more like desktop programs than static web pages. But JavaScript isn't the only client-side technology in town. You can also use plug-ins to add programming smarts to a web page. Java applets are one example. These are small programs, written in the Java programming language, that run in a web browser. They also tend to start up slowly and have been known to crash the browser.

Flash is another plug-in based technology that offers sophisticated animation, video, sound, and lots of interactive potential. In fact, it's sometimes hard to tell if an interactive web page is using JavaScript or Flash. For example, Google Maps could also be created in Flash (in fact, Yahoo Maps was at one time a Flash application, until Yahoo recreated it using JavaScript). A quick way to tell the difference: Right-click on the part of the page that you think might be Flash (the map itself, in this case); if it is, you'll see a pop-up menu that includes "About the Flash Player."

Ajax, which you'll learn about in Part 4 of this book, brings both client-side and server-side together. Ajax is a method for using JavaScript to talk to a server, retrieve information from the server, and update the web page without the need to load a new web page. Google Maps uses this technique to let you move around a map without forcing you to load a new web page.

In truth, JavaScript can also be a server-side programming language. For example, the node.js web server (*http://nodejs.org/*) uses JavaScript as a server-side programming language to connect to a database, access the web server's file system, and perform many other tasks on a web server. This book doesn't discuss that aspect of JavaScript programming, however.

At first, you'll probably find JavaScript programming frustrating—you'll spend a lot of your time tracking down errors you made when typing the script. Also, you might find some of the concepts related to programming a bit hard to follow at first. But don't worry: If you've tried to learn JavaScript in the past and gave up because you thought it was too hard, this book will help you get past the hurdles that often trip up folks new to programming. (And if you do have programming experience, this book will teach you JavaScript's idiosyncrasies and the unique concepts involved in programming for web browsers.)

In addition, this book isn't just about JavaScript—it's also about jQuery, the world's most popular JavaScript library. jQuery makes complex JavaScript programming easier…much easier. So with a little bit of JavaScript knowledge and the help of jQuery, you'll be creating sophisticated, interactive websites in no time.

What's a Computer Program?

When you add JavaScript to a web page, you're writing a computer program. Granted, most JavaScript programs are much simpler than the programs you use to read email, retouch photographs, and build web pages. But even though JavaScript programs (also called *scripts*) are simpler and shorter, they share many of the same properties of more complicated programs.

In a nutshell, any computer program is a series of steps that are completed in a designated order. Say you want to display a welcome message using the name of the person viewing a web page: "Welcome, Bob!" There are several things you'd need to do to accomplish this task:

1. **Ask the visitor's name.**
2. **Get the visitor's response.**
3. **Print (that is, display) the message on the web page.**

While you may never want to print a welcome message on a web page, this example demonstrates the fundamental process of programming: Determine what you want to do, then break that task down into each step that's necessary to get it done. Every time you want to create a JavaScript program, you must go through the process of determining the steps needed to achieve your goal. Once you know the steps, you're ready to write your program. In other words, you'll translate your ideas into programming *code*—the words and characters that make the web browser behave how you want it to.

FREQUENTLY ASKED QUESTION

Compiled vs. Scripting Languages

JavaScript is called a scripting language. *I've heard this term used for other languages like PHP and ColdFusion as well. What's a scripting language?*

Most of the programs running on your computer are written using languages that are *compiled*. Compiling is the process of creating a file that will run on a computer by translating the code a programmer writes into instructions that a computer can understand. Once a program is compiled, you can run it on your computer, and since a compiled program has been converted directly to instructions a computer understands, it will run faster than a program written with a scripting language. Unfortunately, compiling a program is a time-consuming process: You have to write the program, compile it, and then test it. If the program doesn't work, you have to go through the whole process again.

A scripting language, on the other hand, is only compiled when an *interpreter* (another program that can convert the script into something a computer can understand) reads it. In the case of JavaScript, the interpreter is built into the web browser. So when your web browser reads a web page with a JavaScript program in it, the web browser translates the JavaScript into something the computer understands. As a result, a scripting language operates more slowly than a compiled language, since every time it runs, the program must be translated for the computer. Scripting languages are great for web developers: Scripts are generally much smaller and less complex than desktop programs, so the lack of speed isn't as important. In addition, since they don't require compiling, creating and testing programs that use a scripting language is a much faster process.

How to Add JavaScript to a Page

Web browsers are built to understand HTML and CSS and convert those languages into a visual display on the screen. The part of the web browser that understands HTML and CSS is called the *layout* or *rendering* engine. But most browsers also have something called a *JavaScript interpreter*. That's the part of the browser that understands JavaScript and can execute the steps of a JavaScript program. Since the web browser is usually expecting HTML, you must specifically tell the browser when JavaScript is coming by using the <script> tag.

The <script> tag is regular HTML. It acts like a switch that in effect says "Hey, web browser, here comes some JavaScript code; you don't know what to do with it, so hand it off to the JavaScript interpreter." When the web browser encounters the closing </script> tag, it knows it's reached the end of the JavaScript program and can get back to its normal duties.

Much of the time, you'll add the <script> tag in the <head> portion of the web page like this:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/
html4/strict.dtd">
<html>
<head>
<title>My Web Page</title>
<script type="text/javascript">

</script>
</head>
```

The <script> tag's *type* attribute indicates the format and the type of script that follows. In this case, *type="text/javascript"* means the script is regular text (just like HTML) and that it's written in JavaScript.

If you're using HTML5, life is even simpler. You can skip the type attribute entirely:

```
<!doctype html>
<html>
<head>
<meta charset="UTF-8">
<title>My Web Page</title>
<script>

</script>
</head>
```

In fact, web browsers let you leave out the type attribute in HTML 4.01 and XHTML 1.0 files as well—the script will run the same; however, your page won't validate correctly without the type attribute (see the box on page 7 for more on validation). This book uses HTML5 for the doctype, but the JavaScript code will be the same and work the same for HTML 4.01, and XHTML 1.

You then add your JavaScript code between the opening and closing <script> tags:

```
<!doctype html>
<html>
<head>
<meta charset="UTF-8">
<head>
<title>My Web Page</title>
<script>
alert('hello world!');
</script>
</head>
```

You'll find out what this JavaScript actually does in a moment. For now, turn your attention to the opening and closing <script> tags. To add a script to your page, start by inserting these tags. In many cases, you'll put the <script> tags in the page's <head> in order to keep your JavaScript code neatly organized in one area of the web page.

However, it's perfectly valid to put <script> tags anywhere inside the HTML of the page. In fact, as you'll see later in this chapter, there's a JavaScript command that lets you write information directly into a web page. Using that command, you place the <script> tags in the location on the page (somewhere inside the body) where you want the script to write its message. It's even common to put <script> tags just below the closing </body> tag—this approach makes sure the page is loaded and the visitor sees it before running any JavaScript.

External JavaScript Files

Using the <script> tag as discussed in the previous section lets you add JavaScript to a single page. But many times you'll create scripts that you want to share with all of the pages on your site. For example, you might use a JavaScript program to add animated, drop-down navigation menus to a web page. You'll want that same fancy navigation bar on every page of your site, but copying and pasting the same JavaScript code into each page of your site is a really bad idea for several reasons.

First, it's a lot of work copying and pasting the same code over and over again, especially if you have a site with hundreds of pages. Second, if you ever decide to change or enhance the JavaScript code, you'll need to locate every page using that JavaScript and update the code. Finally, since all of the code for the JavaScript program would be located in every web page, each page will be that much larger and slower to download.

A better approach is to use an external JavaScript file. If you've used external CSS files for your web pages, this technique should feel familiar. An external JavaScript file is simply a text file that ends with the file extension .js—*navigation.js*, for example. The file only includes JavaScript code and is linked to a web page using the <script> tag. For example, to add a JavaScript file named *navigation.js* to your home page, you might write the following:

```
<!doctype html>
<html>
<head>
<meta charset="UTF-8">
<title>My Web Page</title>
<script src="navigation.js"></script>
</head>
```

The src attribute of the <script> tag works just like the src attribute of an <img> tag, or an <a> tag's href attribute. In other words, it points to a file either in your website or on another website (see the box on the next page).

Note: When adding the src attribute to link to an external JavaScript file, don't add any JavaScript code between the opening and closing <script> tags. If you want to link to an external JavaScript file and add custom JavaScript code to a page, use a second set of <script> tags. For example:

```
<script src="navigation.js"></script>
<script >
  alert('Hello world!');
</script>
```

UP TO SPEED

URL Types

When attaching an external JavaScript file to a web page, you need to specify a *URL* for the src attribute of the <script> tag. A URL or *Uniform Resource Locator* is a path to a file located on the web. There are three types of paths: *absolute path, root-relative path*, and *document-relative path*. All three indicate where a web browser can find a particular file (like another web page, a graphic, or a JavaScript file).

An *absolute path* is like a postal address–it contains all the information needed for a web browser located anywhere in the world to find the file. An absolute path includes http://, the hostname, and the folder and name of the file. For example: *http://www.cosmofarmer.com/scripts/site.js*.

A *root-relative* path indicates where a file is located relative to a site's top-level folder–the site's root folder. A root-relative path doesn't include http:// or the domain name. It begins with a / (slash) indicating the site's root folder–the folder the home page is in. For example, */scripts/site.js* indicates that the file site.js is located inside a folder named *scripts*, which is itself located in the site's top-level folder. An easy way to create a root-relative path is to take an absolute path and strip off the http:// and the host name. For example, *http://www.sawmac.com/index.html* written as a root-relative URL is */index.html*.

A *document-relative* path specifies the path from the web page to the JavaScript file. If you have multiple levels of folders on your website, you'll need to use different paths to point to the same JavaScript file. For example, suppose you have a JavaScript file named *site.js* located in a folder named *scripts* in your website's main directory. The document-relative path to that file will look one way for the home page–*scripts/site.js*–but for a page located inside a folder named *about*, the path to the same file would be different; *../scripts/site.js*–the ../ means climb up *out* of the *about* folder, while the */scripts/site.js* means go to the *scripts* folder and get the file *site.js*.

Here are some tips on which URL type to use:

- If you're pointing to a file that's not on the same server as the web page, you *must* use an absolute path. It's the only type that can point to another website.
- Root-relative paths are good for JavaScript files stored on your own site. Since they always start at the root folder, the URL for a JavaScript file will be the same for every page on your website, even when web pages are located in folders and subfolders on your site. However, root-relative paths don't work unless you're viewing your web pages through a web server–either your web server out on the Internet, or a web server you've set up on your own computer for testing purposes. In other words, if you're just opening a web page off your computer using the browser's File→Open command, the web browser won't be able to locate, load, or run JavaScript files that are attached using a root-relative path.
- Document-relative paths are the best when you're designing on your own computer without the aid of a web server. You can create an external JavaScript file, attach it to a web page, and then check the JavaScript in a web browser simply by opening the web page off your hard drive. Document-relative paths work fine when moved to your actual, living, breathing website on the Internet, but you'll have to rewrite the URLs to the JavaScript file if you move the web page to another location on the server. In this book, we'll be using document-relative paths, since they will let you follow along and test the tutorials on your own computer without a web server.

You can (and often will) attach multiple external JavaScript files to a single web page. For example, you might have created one external JavaScript file that controls a drop-down navigation bar, and another that lets you add a nifty slideshow to a page of photos (you'll learn how to do that on page 222). On your photo gallery page, you'd want to have both JavaScript programs, so you'd attach both files.

In addition, you can attach external JavaScript files and add a JavaScript program to the same page like this:

```
<!doctype html>
<html>
<head>
<meta charset="UTF-8">
<title>My Web Page</title>
<script src="navigation.js"></script>
<script src="slideshow.js"></script>
<script>
  alert('hello world!');
</script>
</head>
```

Just remember that you must use one set of opening and closing <script> tags for each external JavaScript file. You'll create an external JavaScript file in the tutorial that starts on page 33.

You can keep external JavaScript files anywhere inside your website's root folder (or any subfolder inside the root). Many web developers create a special directory for external JavaScript files in the site's root folder: common names are *js* (meaning JavaScript) or *libs* (meaning libraries).

Note: Sometimes the order in which you attach external JavaScript files matters. As you'll see later in this book, sometimes scripts you write depend upon code that comes from an external file. That's often the case when using JavaScript libraries (JavaScript code that simplifies complex programming tasks). You'll see an example of a JavaScript library in action in the tutorial on page 33.

Your First JavaScript Program

The best way to learn JavaScript programming is by actually programming. Throughout this book, you'll find hands-on tutorials that take you step-by-step through the process of creating JavaScript programs. To get started, you'll need a text editor (see page 10 for recommendations), a web browser, and the exercise files located at *www.sawmac.com/js2e* (see the following Note for complete instructions).

Note: The tutorials in this chapter require the example files from this book's website, www.*sawmac.com/js2e/*. Click the "Download tutorials" link to download them. (The tutorial files are stored as a single Zip file.)

Windows users should download the Zip file and double-click it to open the archive. Click the Extract All Files option, and then follow the instructions of the Extraction Wizard to unzip the files and place them on your computer. Mac users, just double-click the file to decompress it. After you've downloaded and decompressed the files, you should have a folder named MM_JAVASCRIPT2E on your computer, containing all of the tutorial files for this book.

To get your feet wet and provide a gentle introduction to JavaScript, your first program will be very simple:

1. **In your favorite text editor, open the file *hello.html*.**

   This file is located in the *chapter01* folder in the MM_JAVASCRIPT2E folder you downloaded from *www.sawmac.com/js2e*. It's a very simple HTML page, with an external cascading style sheet to add a little visual excitement.

2. **Click in the empty line just *before* the closing </head> tag and type:**

   ```
   <script>
   ```

   This code is actually HTML, not JavaScript. It informs the web browser that the stuff following this tag is JavaScript.

3. **Press the Return key to create a new blank line, and type:**

   ```
   alert('hello world');
   ```

   You've just typed your first line of JavaScript code. The JavaScript *alert()* function is a command that pops open an Alert box and displays the message that appears inside the parentheses—in this case, *hello world*. Don't worry about all of the punctuation (the parentheses, quotes, and semicolon) just yet. You'll learn what they do in the next chapter.

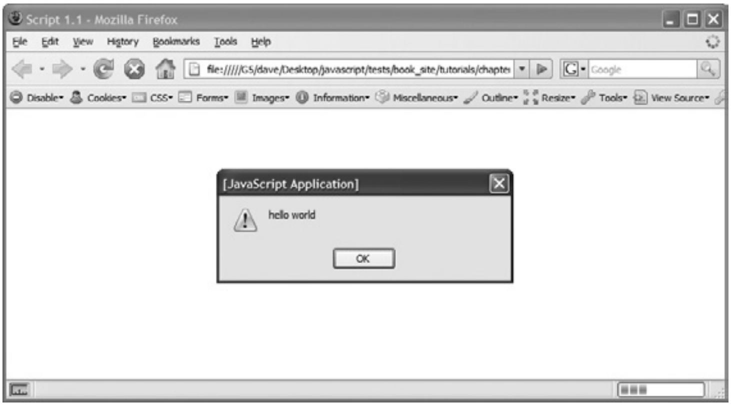
4. **Press the Return key once more, and type *</script>*. The code should now look like this:**

   ```
   <link href="../_css/site.css" rel="stylesheet">
   <script>
   alert('hello world');
   </script>
   </head>
   ```

   In this example, the stuff you just typed is shown in boldface. The two HTML tags are already in the file; make sure you type the code exactly where shown.

5. **Launch a web browser and open the *hello.html* file to preview it.**

   A JavaScript Alert box appears (see Figure 1-2). Notice that the page is blank when the alert appears. (If you don't see the Alert box pictured in Figure 1-2, you probably mistyped the code listed in the previous steps. Double-check your typing and read the following Note.)



Script 1.1 - Mozilla Firefox
File Edit View History Bookmarks Tools Help
file://///G5/dave/Desktop/javascript/tests/book_site/tutorials/chapte
Google
Disable▾ Cookies▾ CSS▾ Forms▾ Images▾ Information▾ Miscellaneous▾ Outline▾ Resize▾ Tools▾ View Source▾
[JavaScript Application]
OK

hello world

Figure 1-2: *The JavaScript Alert box is a quick way to grab someone's attention. It's one of the simplest JavaScript commands to learn and use.*

6. **Click the Alert box's OK button to close it.**

   When the Alert box disappears, the web page appears in the browser window.

Tip: When you first start programming, you'll be shocked at how often your JavaScript programs don't seem to work…at all. For new programmers, the most common cause of nonfunctioning programs is simple typing mistakes. Always double-check to make sure you spelled commands (like *alert* in the first script) correctly. Also, notice that punctuation frequently comes in pairs (the opening and closing parentheses, and single-quote marks from your first script, for example). Make sure you include both opening and closing punctuation marks when they're required.

Although this first program isn't earth-shatteringly complex (or even that interesting), it does demonstrate an important concept: A web browser will run a JavaScript program the moment it reads in the JavaScript code. In this example, the *alert()* command appeared *before* the web browser displayed the web page, because the JavaScript code appeared *before* the HTML in the <body> tag. This concept comes into play when you start writing programs that manipulate the HTML of the web page—as you'll learn in Chapter 3.

Note: Internet Explorer (IE) doesn't like to run JavaScript programs contained in web pages that you open directly off your hard drive, for fear that the JavaScript program might do something harmful. So when you try to preview the tutorial files for this book in Internet Explorer, you'll see a message saying that IE has blocked the script. Click "Allow blocked content" to see the program run. This annoying behavior only applies to web pages you preview from your computer, not from files you put up on a web server. When following along with the tutorials in this book, it's better to preview pages in a different web browser like Firefox, Chrome, or Safari, so you can avoid having to hit the "Allow blocked content" button each time you view your pages.

Writing Text on a Web Page

The last script popped up a dialog box in the middle of your monitor. What if you want to print a message directly onto a web page using JavaScript? There are many ways to do so, and you'll learn some sophisticated techniques later in this book. However, you can achieve this simple goal with a built-in JavaScript command, and that's what you'll do in your second script:

1. **In your text editor, open the file *hello2.html*.**

   While <script> tags usually appear in the <head> of a web page, you can put them and JavaScript programs directly in the body of the web page.

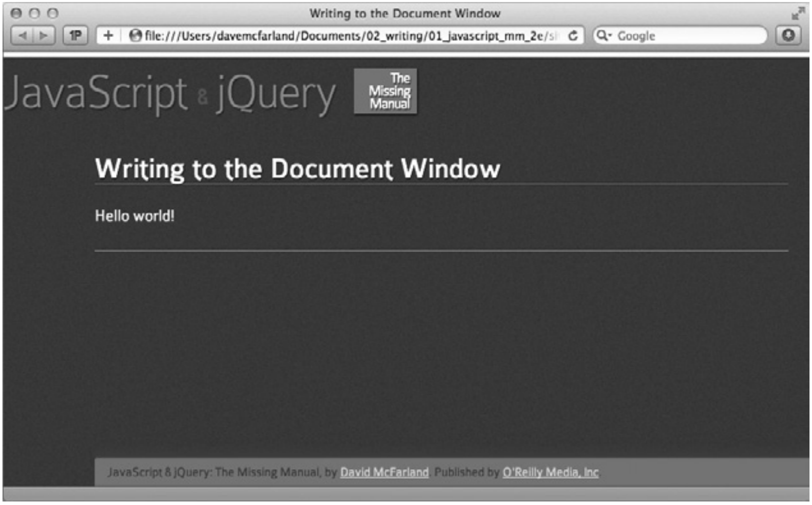
2. **Directly below <h1>Writing to the document window</h1>, type the following code:**

```
<script>
document.write('<p>Hello world!</p>');
</script>
```

Like the *alert()* function, *document.write()* is a JavaScript command that literally writes out whatever you place between the opening and closing parentheses. In this case, the HTML *<p>Hello world!</p>* is added to the page: a paragraph tag and two words.

3. **Save the page, and open it in a web browser.**

   The page opens and the words "Hello world!" appear below the red headline (see Figure 1-3).


Writing to the Document Window
file:///Users/davemcfarland/Documents/02_writing/01_javascript_mm_2e/s
Google
JavaScript & jQuery
The Missing Manual

Writing to the Document Window

Hello world!

JavaScript & jQuery: The Missing Manual, by David McFarland. Published by O'Reilly Media, Inc.

Figure 1-3: *Wow. This script may not be something to "document.write" home about–ha, ha, JavaScript humor–but it does demonstrate that you can use JavaScript to add content to a web page, a trick that comes in handy when you want to display messages (like "Welcome back to the site, Dave") after a web page has downloaded.*

Note: The tutorial files you downloaded also include the completed version of each tutorial. If you can't seem to get your JavaScript working, compare your work with the file that begins with *complete_* in the same folder as the tutorial file. For example, the file *complete_hello2.html* contains a working version of the script you added to file *hello2.html.*

The two scripts you just created may leave you feeling a little underwhelmed with JavaScript…or this book. Don't worry. It's important to start out with a full understanding of the basics. You'll be doing some very useful and complicated things using JavaScript in just a few chapters. In fact, in the remainder of this chapter you'll get a taste of some of the advanced features you'll be able to add to your web pages after you've worked your way through the first two parts of this book.

Attaching an External JavaScript File

As discussed on page 27, you'll usually put JavaScript code in a separate file if you want to use the same scripts on more than one web page. You can then instruct a web page to load that file and use the JavaScript inside it. External JavaScript files also come in handy when you're using someone else's JavaScript code. In particular, there are collections of JavaScript code called *libraries*, which provide useful JavaScript programming: Usually, these libraries make it easy for you to do something that's normally quite difficult to do. You'll learn more about JavaScript libraries on page 117, and, in particular, the JavaScript library this book uses—jQuery.

But for now, you'll get experience attaching an external JavaScript file to a page, and writing a short program that does some amazing things:

1. **In your text editor, open the file *fadeIn.html*.**

   This page contains just some simple HTML—a few <div> tags, a headline, and a couple of paragraphs. You'll be adding a simple visual effect to the page, which causes all of the content to slowly fade into view.

2. **Click in the blank line between the <link> and closing </head> tags near the top of the page, and type:**

   ```
   <script src="../_js/jquery-1.6.3.min.js"></script>
   ```

   This code links a file named *jquery-1.6.3.min.js*, which is contained in a folder named *_js*, to this web page. When a web browser loads this web page, it also downloads the *jquery-1.6.3.min.js* JavaScript file and runs the code inside it.

   Next, you'll add your own JavaScript programming to this page.

Note: It's common to include a version number in the name of a JavaScript file. In this example, the filename is *jquery-1.6.3.min.js*. The 1.6.3 indicates the version 1.6.3 of jQuery. The min part means that the file is *minimized*—which makes the file smaller so that it downloads faster.

---

3. **Press Return to create a new blank line, and then type:**

   ```
   <script>
   ```

   HTML tags usually travel in pairs—an opening and closing tag. To make sure you don't forget to close a tag, it helps to close the tag immediately after typing the opening tag, and then fill in the stuff that goes between the tags.

4. **Press Return twice to create two blank lines, and then type:**

   ```
   </script>
   ```

   This ends the block of JavaScript code. Now you'll add some programming.

5. **Click the empty line between the opening and closing script tags and type:**

   ```
   $(document).ready(function() {
   ```

   You're probably wondering what the heck that is. You'll find out all the details of this code on page 169, but in a nutshell, this line takes advantage of the programming that's inside the *jquery-1.6.3.min.js* file to make sure that the browser executes the next line of code at the right time.

6. **Hit return to create a new line, and then type:**

   ```
   $('body').hide().fadeIn(3000);
   ```

   This line does something magical: It makes the page's content first disappear and then slowly fade into view over the course of 3 seconds (or 3000 milliseconds). How does it do that? Well, that's part of the magic of jQuery, which makes complex effects possible with just a single line of code.

7. **Hit Return one last time, and then type:**

   ```
   });
   ```

   This code closes up the JavaScript code, much as a closing </script> tag indicates the end of a JavaScript program. Don't worry too much about all those weird punctuation marks—you'll learn how they work in detail later in the book. The main thing you need to make sure of is to type the code exactly as it's listed here. One typo, and the program may not work.

   The final code you added to the page should look like the bolded text below:

   ```
   <link href="../_css/site.css" rel="stylesheet">
   <script src="../_js/jquery-1.6.3.min.js"></script>
   <script>
   $(function() {
   $('body').hide().fadeIn(3000);
   });
   </script>
   </head>
   ```

8. **Save the HTML file, and open it in a web browser.**

   You should now see the page slowly fade into view. Change the number 3000 to different values (like 250 and 10000) to see how that changes the way the page works.

Note: If you try to preview this page in Internet Explorer and it doesn't seem to do anything, you'll need to click the "Enable blocked content" box that appears at the bottom of the page (see the Note on page 31).

As you can see, it doesn't take a whole lot of JavaScript to do some amazing things to your web pages. Thanks to JavaScript libraries like jQuery, you'll be able to create sophisticated, interactive websites without being a programming wizard yourself. However, it does help to know the basics of JavaScript and programming. In the next three chapters, we'll cover the very basics of JavaScript so that you're comfortable with the fundamental concepts and syntax that make up the language.

Tracking Down Errors

The most frustrating moment in JavaScript programming comes when you try to view your JavaScript-powered page in a web browser…and nothing happens. It's one of the most common experiences for programmers. Even very experienced programmers don't always get it right the first time they write a program, so figuring out what went wrong is just part of the game.

Most web browsers are set up to silently ignore JavaScript errors, so you usually won't even see a "Hey this program doesn't work!" dialog box. (Generally, that's a good thing, since you don't want a JavaScript error to interrupt the experience of viewing your web pages.)

So how do you figure out what's gone wrong? There are many ways to track errors in a JavaScript program. You'll learn some advanced *debugging* techniques in Chapter 15, but the most basic method is to consult the web browser. Most web browsers keep track of JavaScript errors and record them in a separate window called an *error console*. When you load a web page that contains an error, you can then view the console to get helpful information about the error, like which line of the web page it occurred in and a description of the error.

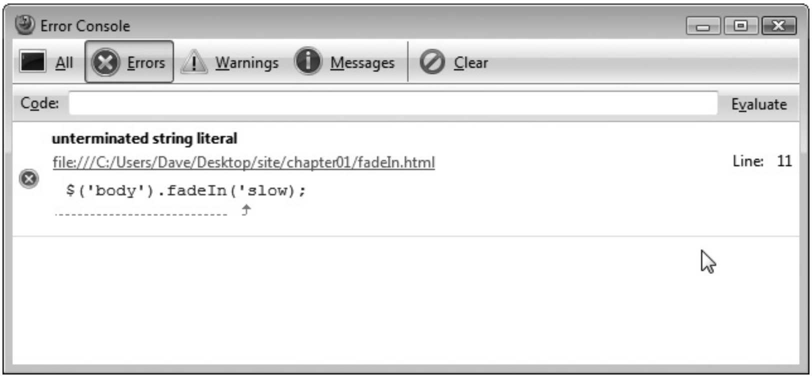
Often, you can find the answer to the problem in the error console, fix the JavaScript, and then the page will work. The console helps you weed out the basic typos you make when you first start programming, like forgetting closing punctuation, or mistyping the name of a JavaScript command. You can use the error console in your favorite browser, but since scripts sometimes work in one browser and not another, this section shows you how to turn on the JavaScript console in all major browsers, so you can track down problems in each.

The Firefox JavaScript Console

Firefox's JavaScript console is a great place to begin tracking down errors in your code. Not only does the console provide fairly clear descriptions of errors (no descriptions are ever *that* clear when it comes to programming), it also identifies the line in your code where the error occurred.

For example, in Figure 1-4, the console identifies the error as an "unterminated string literal," meaning that there's an opening single quote mark before "slow" but no final quote mark. The console also identifies the name of the file the error is in (*fadeIn .html* in this case) and the line number the error occurs (line 11). Best of all, it even indicates the source of the error with an arrow—in this case, highlighting the opening quote mark.

Warning: Although the error console draws an arrow pointing to the location where Firefox encountered the error, that's not always where you made the mistake. Sometimes you need to fix your code before or after that arrow.



unterminated string literal

file:///C:/Users/Dave/Desktop/site/chapter01/fadeIn.html Line: 11

```
$('body').fadeIn('slow);
--------------------------- ↑
```

Figure 1-4: *Firefox's JavaScript console identifies errors in your programs. The console keeps a list of errors for previous pages as well, so pretty soon the list can get very long. Just click the Clear button to erase all the errors listed in the console.*

To show the JavaScript console, click the Firefox menu and choose Web Developer→Error Console (on Windows) or Tools→Error Console (on Macs). The console is a free-floating window that you can move around. It not only displays JavaScript errors, but CSS errors as well, so if you've made any mistakes in your Cascading Styles Sheets, you'll find out about those as well. (Make sure you select the Errors button at the top of the console; otherwise, you might see warnings and messages that aren't related to your JavaScript error.)

Tip: Since the error console displays the line number where the error occurred, you may want to use a text editor that can show line numbers. That way, you can easily jump from the error console to your text editor and identify the line of code you need to fix.

Unfortunately, there's a long list of things that can go wrong in a script, from simple typos to complex errors in logic. When you're just starting out with JavaScript programming, many of your errors will be the simple typographic sort. For example, you might forget a semicolon, quote mark, or parenthesis, or misspell a JavaScript command. You're especially prone to typos when following examples from a book (like this one). Here are a few errors you may see a lot of when you first start typing the code from this book:

- **Missing) after argument list**. You forgot to type a closing parenthesis at the end of a command. For example, in this code—*alert('hello';*—the parenthesis is missing after *'hello'*.

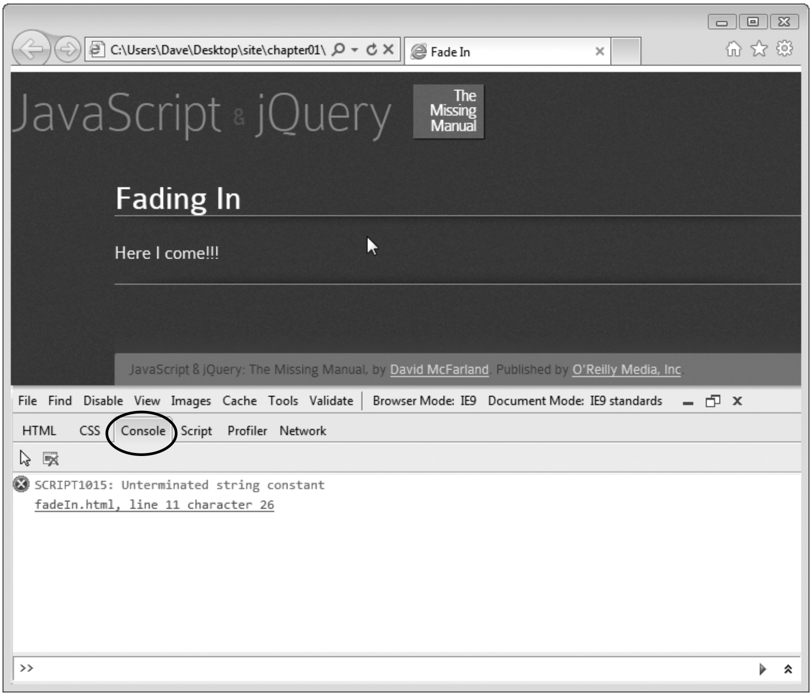
- **Unterminated string literal**. A *string* is a series of characters enclosed by quote marks (you'll learn about these in greater detail on page 43). For example, *'hello'* is a string in the code *alert('hello');*. It's easy to forget either the opening or closing quote mark.
- **XXX is not defined**. If you misspell a JavaScript command—*aler('hello');*—you'll get an error saying that the (misspelled) command isn't defined: for example, "aler is not defined."
- **Syntax error**. Occasionally, Firefox has no idea what you were trying to do and provides this generic error message. A syntax error represents some mistake in your code. It may not be a typo, but you may have put together one or more statements of JavaScript in a way that isn't allowed. In this case, you need to look closely at the line where the error was found and try to figure out what mistake you made. Unfortunately, these types of errors often require experience with and understanding of the JavaScript language to fix.

As you can see from the list above, many errors you'll make simply involve forgetting to type one of a pair of punctuation marks—like quote marks or parentheses. Fortunately, these are easy to fix, and as you get more experience programming, you'll eventually stop making them almost completely (no programmer is perfect).

Note: The Firebug plug-in for Firefox (*http://getfirebug.com/*) greatly expands on Firefox's Error Console. In fact, it provided the model for the other developer tools you'll find in IE9, Chrome, and Safari (discussed next). You'll learn about Firebug in Chapter 15.

Displaying the Internet Explorer 9 Console

Internet Explorer 9 provides a sophisticated set of developer tools for viewing not only JavaScript errors, but also analyzing CSS, HTML, and transfers of information over the network. When open, the developer tool window appears in the bottom half of the browser window (see Figure 1-5). Press the F12 key to open the developer tools, and press it again to close them. You'll find JavaScript errors listed under the Console tab (circled in Figure 1-5). Unlike the Firefox Error Console, which keeps a running total of JavaScript errors for all of the pages you visit, you need to open the IE 9 Console, then reload the page to see an error.



C:\Users\Dave\Desktop\site\chapter01\
Fade In
JavaScript & jQuery
The Missing Manual
Fading In
Here I come!!!
JavaScript & jQuery: The Missing Manual, by David McFarland. Published by O'Reilly Media, Inc.
File Find Disable View Images Cache Tools Validate
Browser Mode: IE9 Document Mode: IE9 standards
HTML CSS Console Script Profiler Network
>>

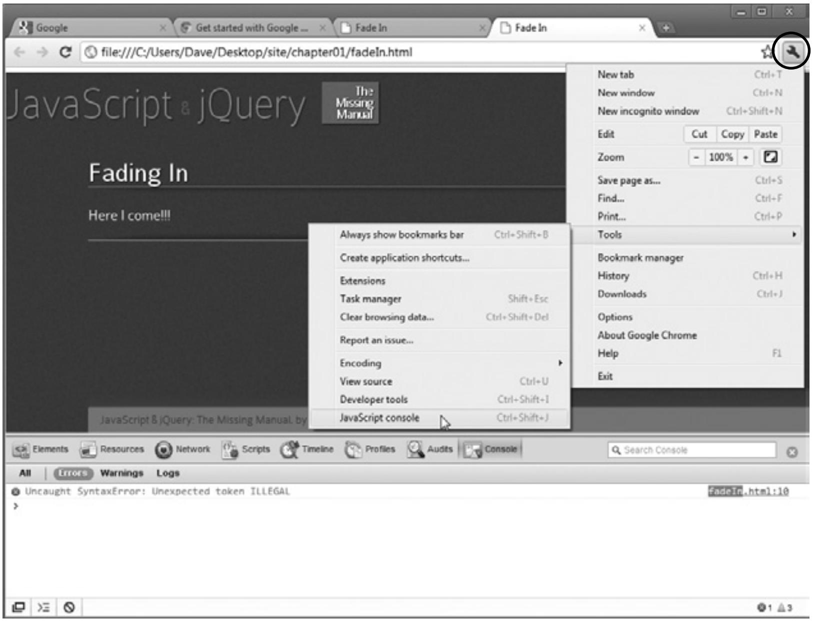
```
SCRIPT1015: Unterminated string constant
fadeIn.html, line 11 character 26
```

Figure 1-5: *The Internet Explorer Developer Tools provide access to JavaScript errors that occur on a page, as well as a whole lot of other information.*

IE9's Console displays error messages similar to those described for Firefox above. For example, "Unterminated string constant" is the same as Firefox's "Unterminated string literal" message—meaning there's a missing quote mark. In addition, Internet Explorer identifies the line of code in the HTML file where the error occurred.

Opening the Chrome JavaScript Console

Google's Chrome browser also lets you view JavaScript errors from its JavaScript console. To open the console, click the tools icon (circled in Figure 1-6), select the Tools menu, and choose JavaScript console from the pop-out menu. Once the console opens, click the Errors button to see just the JavaScript errors you're after. Unfortunately, Chrome's error messages tend to be a little more cryptic. For example, the error message for leaving out a closing quote mark is "Uncaught SyntaxError: Unexpected token ILLEGAL." Not exactly clear or friendly. In fact, it kind of makes you feel as if you make one more mistake like that Chrome will scream, "Release the robotic hounds!"



```
Uncaught SyntaxError: Unexpected token ILLEGAL                    fadeIn.html:10
>
```

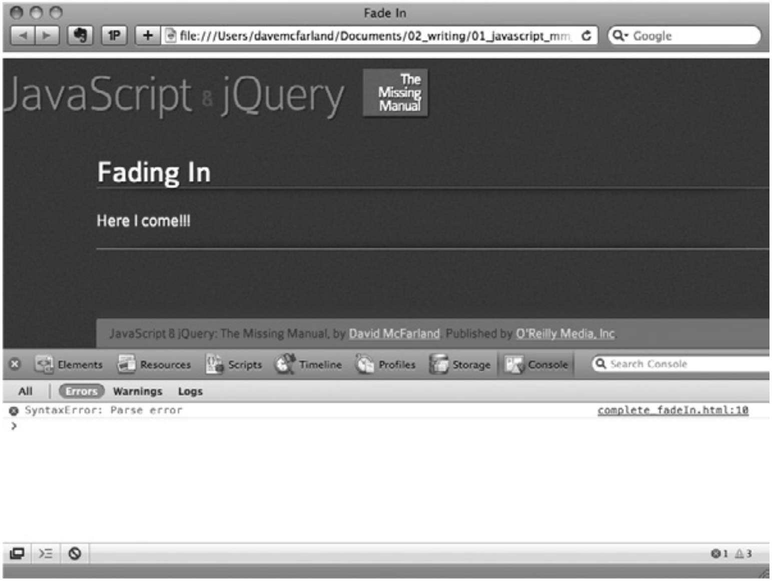
Figure 1-6: *Chrome sports a powerful set of developer tools that looks a lot like the ones you'll find in Internet Explorer 9 and Safari.*

Accessing the Safari Error Console

Safari's error console is available from the Develop menu: Develop→Show Error Console (on the Mac, you can use the keyboard shortcut Option-⌘-C, and on Windows, the shortcut is Ctrl+Alt+C). However, the Develop menu isn't normally turned on when Safari is installed, so there are a couple of steps to get to the JavaScript console.

To turn on the Develop menu, you need to first access the Preferences window. On a Mac, choose Safari→Preferences. On Windows, click the gear icon in the top right of the browser, and choose Preferences. Once the Preferences window opens, click the Advanced button. Check the "Show Develop menu in menu bar" box and close the Preferences window.

When you restart Safari, the Develop menu will appear between the Bookmarks and Window menus in the menu bar at the top of the screen on Macs; and on Windows, you'll find it under the page icon in the top right of the browser. Select Develop→Show Error Console to open the console (see Figure 1-7).



```
SyntaxError: Parse error                                complete_fadeIn.html:10
>
```

Figure 1-7:
The Safari Error Console displays the name of the JavaScript error, the file name (and location), and the line on which Safari encountered the error. Each tab or browser window has its own error console, so if you've already opened the console for one tab, you need to choose Develop→Error Console if you wish to see an error for another tab or window.

O’Reilly Ebooks—Your bookshelf on your devices!

When you buy an ebook through oreilly.com you get lifetime access to the book, and whenever possible we provide it to you in five, DRM-free file formats—PDF, .epub, Kindle-compatible .mobi, Android .apk, and DAISY—that you can use on the devices of your choice. Our ebook files are fully searchable, and you can cut-and-paste and print them. We also alert you when we’ve updated the files with corrections and additions.

Learn more at ebooks.oreilly.com

You can also purchase O’Reilly ebooks through the iBookstore, the Android Marketplace, and Amazon.com.

O’REILLY®

Spreading the knowledge of innovators

oreilly.com