Brad Dayley

Sams Teach Yourself

jQuery and JavaScript in 24 Hours

SAMS

FREE SAMPLE CHAPTER

SHARE WITH OTHERS

Brad Dayley

Sams **Teach Yourself**

jQuery and JavaScript

in 24 Hours

SAMS 800 East 96th Street, Indianapolis, Indiana, 46240 USA

Sams Teach Yourself jQuery and JavaScript in 24 Hours

ISBN-13: 978-0-672-33734-5

ISBN-10: 0-672-33734-7

Library of Congress Control Number: 2013954604

Printed in the United States of America

First Printing December 2013

Trademarks

All terms mentioned in this book that are known to be trademarks or service marks have been appropriately capitalized. Sams Publishing cannot attest to the accuracy of this information. Use of a term in this book should not be regarded as affecting the validity of any trademark or service mark.

Warning and Disclaimer

Every effort has been made to make this book as complete and as accurate as possible, but no warranty or fitness is implied. The information provided is on an "as is" basis. The author and the publisher shall have neither liability nor responsibility to any person or entity with respect to any loss or damages arising from the information contained in this book.

Special Sales

For information about buying this title in bulk quantities, or for special sales opportunities (which may include electronic versions; custom cover designs; and content particular to your business, training goals, marketing focus, or branding interests), please contact our corporate sales department at corpsales@pearsoned.com or (800) 382-3419.

For government sales inquiries, please contact governmentsales@pearsoned.com.

For questions about sales outside the U.S., please contact international@pearsoned.com.

Acquisitions Editor
Mark Taber

Managing Editor
Sandra Schroeder

Senior Project Editor
Tonya Simpson

Copy Editor
Barbara Hacha

Indexer
Erika Millen

Proofreader
Anne Goebel

Technical Editor
Russell Kloepfer

Publishing Coordinator
Vanessa Evans

Book Designer
Gary Adair

Cover Designer
Mark Shirar

Compositor
Jake McFarland

For D!

—A & F

Contents at a Glance

Introduction 1

Part I: Introduction to jQuery and JavaScript Development

1 Intro to Dynamic Web Programming 5
2 Debugging jQuery and JavaScript Web Pages 35
3 Understanding Dynamic Web Page Anatomy 65
4 Adding CSS/CSS3 Styles to Allow Dynamic Design and Layout 97
5 Jumping into jQuery and JavaScript Syntax 135
6 Understanding and Using JavaScript Objects 161

Part II: Implementing jQuery and JavaScript in Web Pages

7 Accessing DOM Elements Using JavaScript and jQuery Objects 185
8 Navigating and Manipulating jQuery Objects and DOM Elements with jQuery 205
9 Applying Events for Richly Interactive Web Pages 223
10 Dynamically Accessing and Manipulating Web Pages 255
11 Accessing Data Outside the Web Page 285

Part III: Building Richly Interactive Web Pages

12 Enhancing User Interaction Through Animation and Other Special Effects 301
13 Interacting with Web Forms 325
14 Creating Advanced Web Page Elements 365

Part IV: Advanced Concepts

15 Accessing Server-Side Data via AJAX 395
16 Interacting with External Services, Facebook, Google, Twitter, and Flickr 425

Part V: jQuery UI

17 Introducing jQuery UI 459
18 Using jQuery UI Effects 475
19 Advanced Interactions Using jQuery UI Interaction Widgets 493
20 Using jQuery UI Widgets to Add Rich Interactions to Web Pages 521

Part VI: jQuery Mobile

21 Introducing Mobile Website Development 541
22 Implementing Mobile Web Pages 553
23 Formatting Content in Mobile Pages 579
24 Implementing Mobile Form Elements and Controls 599
Index 615

Table of Contents

Introduction 1

Beyond jQuery and JavaScript 1
Code Examples 2
Special Elements 3
Q&A, Quizzes, and Exercises 3

Part I: Introduction to jQuery and JavaScript Development

HOUR 1: Intro to Dynamic Web Programming 5

Understanding the Web Server/Browser Paradigm 5
Preparing to Write jQuery and JavaScript 17
Summary 32
Q&A 32
Workshop 33

HOUR 2: Debugging jQuery and JavaScript Web Pages 35

Viewing the JavaScript Console 35
Debugging HTML Elements 40
Debugging CSS 46
Debugging jQuery and JavaScript 53
Analyzing the Network Traffic 59
Summary 62
Q&A 62
Workshop 63

HOUR 3: Understanding Dynamic Web Page Anatomy 65

Using HTML/HTML5 Elements to Build a Dynamic Web Page 65
Understanding HTML Structure 66
Implementing HTML Head Elements 68
Adding HTML Body Elements 72
Adding Some Advanced HTML5 Elements 87
Summary 95

Q&A ... 95
Workshop ... 95

HOUR 4: Adding CSS/CSS3 Styles to Allow Dynamic Design and Layout **97**
Adding CSS Styles to the Web Page ... 97
Adding CSS Styles to HTML Elements ... 100
Preparing CSS Styles for Dynamic Design ... 130
Summary ... 131
Q&A ... 132
Workshop ... 132

HOUR 5: Jumping into jQuery and JavaScript Syntax **135**
Adding jQuery and JavaScript to a Web Page ... 135
Accessing the DOM ... 138
Understanding JavaScript Syntax ... 141
Summary ... 159
Q&A ... 159
Workshop ... 159

HOUR 6: Understanding and Using JavaScript Objects **161**
Using Object Syntax ... 161
Understanding Built-In Objects ... 163
Creating Custom-Defined Objects ... 177
Summary ... 182
Q&A ... 182
Workshop ... 182

Part II: Implementing jQuery and JavaScript in Web Pages

HOUR 7: Accessing DOM Elements Using JavaScript and jQuery Objects **185**
Understanding DOM Objects Versus jQuery Objects ... 185
Accessing DOM Objects from JavaScript ... 189
Using jQuery Selectors ... 193
Summary ... 203
Q&A ... 203
Workshop ... 203

HOUR 8: Navigating and Manipulating jQuery Objects and DOM Elements with jQuery 205
Chaining jQuery Object Operations 205
Filtering the jQuery Object Results 206
Traversing the DOM Using jQuery Objects 207
Looking at Some Additional jQuery Object Methods 211
Summary 220
Q&A 220
Workshop 221

HOUR 9: Applying Events for Richly Interactive Web Pages 223
Understanding Events 223
Using the Page Load Events for Initialization 229
Adding and Removing Event Handlers to DOM Elements 230
Triggering Events Manually 241
Creating Custom Events 249
Implementing Callbacks 251
Summary 253
Q&A 253
Workshop 253

HOUR 10: Dynamically Accessing and Manipulating Web Pages 255
Accessing Browser and Page Element Values 255
Dynamically Manipulating Page Elements 266
Dynamically Rearranging Elements on the Web Page 277
Summary 283
Q&A 283
Workshop 283

HOUR 11: Accessing Data Outside the Web Page 285
Understanding the Screen Object 285
Using the Window Object 286
Using the Browser Location Object 288
Using the Browser History Object 289
Controlling External Links 290
Adding Pop-up Boxes 294
Setting Timers 296

Summary ... 299
Q&A ... 299
Workshop ... 300

Part III: Building Richly Interactive Web Pages

HOUR 12: Enhancing User Interaction Through Animation and Other Special Effects 301

Understanding jQuery Animation ... 301
Animating Show and Hide ... 305
Animating Visibility ... 309
Sliding Elements ... 312
Creating Resize Animations ... 316
Implementing Moving Elements ... 318
Summary ... 323
Q&A ... 323
Workshop ... 323

HOUR 13: Interacting with Web Forms 325

Accessing Form Elements ... 326
Intelligent Form Flow Control ... 338
Dynamically Controlling Form Element Appearance and Behavior ... 346
Validating a Form ... 351
Summary ... 363
Q&A ... 363
Workshop ... 363

HOUR 14: Creating Advanced Web Page Elements 365

Adding an Image Gallery ... 365
Implementing Tables with Sorting and Filters ... 371
Creating a Tree View ... 377
Using Overlay Dialogs ... 381
Implementing a Graphical Equalizer Display ... 385
Adding Sparkline Graphics ... 389
Summary ... 392
Q&A ... 392
Workshop ... 393

Part IV: Advanced Concepts

HOUR 15: Accessing Server-Side Data via AJAX 395

Making AJAX Easy ... 395
Implementing AJAX ... 399
Using Advanced jQuery AJAX ... 419
Summary ... 422
Q&A ... 422
Workshop ... 423

HOUR 16: Interacting with External Services, Facebook, Google, Twitter, and Flickr 425

Using jQuery and JavaScript to Add Facebook Social Elements to Your Web Pages ... 425
Adding Google Maps to Your Web Pages ... 432
Adding a Custom Google Search ... 439
Adding Twitter Elements to Your Web Pages ... 443
Adding Flickr Images to Your Website ... 451
Summary ... 456
Q&A ... 456
Workshop ... 456

Part V: jQuery UI

HOUR 17: Introducing jQuery UI 459

Getting Started with jQuery UI ... 459
Applying jQuery UI in Your Scripts ... 463
Summary ... 472
Q&A ... 472
Workshop ... 472

HOUR 18: Using jQuery UI Effects 475

Applying jQuery UI Effects ... 475
Adding Effects to Class Transitions ... 482
Adding Effects to Element Visibility Transitions ... 485
Summary ... 490
Q&A ... 491
Workshop ... 491

HOUR 19: Advanced Interactions Using jQuery UI Interaction Widgets 493
Introducing jQuery UI Interactions 493
Using the Drag-and-Drop Widgets 495
Resizing Elements Using the Resizable Widget 503
Applying the Selectable Widget 508
Sorting Elements with the Sortable Widget 512
Summary 518
Q&A 518
Workshop 519

HOUR 20: Using jQuery UI Widgets to Add Rich Interactions to Web Pages 521
Reviewing Widgets 521
Adding an Expandable Accordion Element 522
Implementing Autocomplete in Form Elements 523
Applying jQuery UI Buttons to Form Controls 524
Creating a Calendar Input 525
Generating Stylized Dialogs with jQuery UI 527
Implementing Stylized Menus 528
Creating Progress Bars 529
Implementing Slider Bars 530
Adding a Value Spinner Element 532
Creating Tabbed Panels 533
Adding Tooltips to Page Elements 535
Creating Custom Widgets 537
Summary 538
Q&A 538
Workshop 538

Part VI: jQuery Mobile

HOUR 21: Introducing Mobile Website Development 541
Jumping into the Mobile World 541
Getting Started with jQuery Mobile 543
Understanding jQuery Mobile 545
Summary 551

Q&A 551
Workshop 552

HOUR 22: Implementing Mobile Web Pages 553
Building Mobile Pages 553
Implementing Mobile Sites with Multiple Pages 556
Creating a Navbar 567
Implementing Dialogs 571
Summary 576
Q&A 576
Workshop 576

HOUR 23: Formatting Content in Mobile Pages 579
Adding Basic HTML 579
Creating a Grid Layout 581
Implementing Listviews 585
Using Collapsible Blocks and Sets 590
Adding Auxiliary Content to Panels 592
Working with Pop-ups 594
Building Mobile-Friendly Tables 595
Summary 597
Q&A 597
Workshop 597

HOUR 24: Implementing Mobile Form Elements and Controls 599
Understanding Mobile Forms 599
Using Text Elements 601
Defining Buttons 603
Adding Sliders and Toggle Switches 604
Defining Radios and Check Boxes 608
Implementing Select Menus 610
Summary 612
Q&A 612
Workshop 613

Index 615

About the Author

Brad Dayley is a senior software engineer with more than 20 years of experience developing enterprise applications. He has used HTML/CSS, JavaScript, and jQuery extensively to develop a wide array of web pages, ranging from enterprise application interfaces to sophisticated, rich Internet applications, to smart interfaces for mobile web services. He is the author of *Python Phrasebook* and *jQuery and JavaScript Phrasebook*.

Acknowledgments

I'd like to take this opportunity to thank all those who made this title possible. First, thanks to my wonderful wife and boys for giving me the inspiration and support I need. I'd never make it far without you.

Thanks to Mark Taber for getting this title rolling in the right direction, Russell Kloepfer, for keeping me honest with his technical review, Barbara Hacha, for turning the technical ramblings of my brain into a fine text, and Tonya Simpson, for managing everything on the production end and making sure the book is the finest quality.

We Want to Hear from You!

As the reader of this book, *you* are our most important critic and commentator. We value your opinion and want to know what we're doing right, what we could do better, what areas you'd like to see us publish in, and any other words of wisdom you're willing to pass our way.

We welcome your comments. You can email or write to let us know what you did or didn't like about this book—as well as what we can do to make our books better.

Please note that we cannot help you with technical problems related to the topic of this book.

When you write, please be sure to include this book's title and author as well as your name and email address. We will carefully review your comments and share them with the author and editors who worked on the book.

Email:	feedback@samspublishing.com
Mail:	Sams Publishing ATTN: Reader Feedback 800 East 96th Street Indianapolis, IN 46240 USA

Reader Services

Visit our website and register this book at informit.com/register for convenient access to any updates, downloads, or errata that might be available for this book.

This page intentionally left blank

Introduction

With billions of people using the Internet today, there is a rapidly growing trend to replace traditional websites, where pages link to other pages with a single page, with applications that have richly interactive elements. The main reason for this is that users have become less patient with clicking, waiting, and then having to navigate back and forth between web pages. Instead, they want websites to behave more like the applications they are used to on their computers and mobile devices.

In fact, in just the next 24 hours, millions of new web pages will be added to the Internet. The majority of these pages will be written in HTML, with CSS to style elements and with JavaScript to provide interaction between the user and back-end services.

As you complete the 24 one-hour lessons in this book, you will gain a practical understanding of how to incorporate JavaScript with the powerful jQuery library to provide rich user interactions in your web pages. You will gain the valuable skills of adding dynamic code that allows web pages to instantly react to mouse clicks and finger swipes, interact with back-end services to store and retrieve data from the web server, and create robust Internet applications.

Each hour in the book provides fundamentals that are necessary to create professional web applications. The book includes some basics on using HTML and CSS to get you started, even if you've never used them before. You are provided with code examples that you can implement and expand as your understanding increases. In fact, in just the first lesson in the book, you create a dynamic web page using jQuery and JavaScript.

So pull up a chair, sit back, and enjoy the ride of programming rich Internet applications with jQuery and JavaScript.

Beyond jQuery and JavaScript

This book covers more than jQuery and JavaScript because you need to know more than the language structure to create truly useful web applications. The goal of this book is to give you the fundamental skills needed to create fully functional and interactive web applications in just 24 short, easy lessons. This book covers the following key skills and technologies:

- HTML is the most current recommendation for web page creation. Every example in this book is validated HTML5, the most recent recommended version.
- CSS is the standard method for formatting web elements. You not only learn how to write CSS and CSS3, but also how to dynamically modify it on the fly using jQuery and JavaScript.
- JavaScript is the best method to provide interactions in web pages without the need to load a new page from the server. This is the standard language on which most decent web applications are built.
- jQuery, jQueryUI, and jQueryMobile are some of the most popular and robust libraries for JavaScript. jQuery provides very quick access to web page elements and a robust set of features for web application interaction. jQuery provides additional UI and mobile libraries that provide rich UI components for traditional web applications as well as mobile web applications.
- AJAX is the standard method that web applications use to interact with web servers and other services. The book includes several examples of using AJAX to interact with web servers, Google, Facebook, and other popular web services.

Code Examples

Most of the examples in the book provide the following elements:

- **HTML code**—Code necessary to provide the web page framework in the browser.
- **CSS code**—Code necessary to style the web page elements correctly.
- **JavaScript code**—This includes both the jQuery and JavaScript code that provide interactions among the user, web page elements, and web services.
- **Figures**—Most of the examples include one or more figures that illustrate the behavior of the code in the browser.

The examples in the book are basic to make it easier for you to learn and implement. Many of them can be expanded and used in your own web pages. In fact, some of the exercises at the end of each hour have you expand on the examples.

All the examples in the book have been tested for compatibility with the latest version of the major web browsers, including Google's Chrome, Microsoft's Internet Explorer, and Mozilla's Firefox.

Special Elements

As you complete each lesson, margin notes help you immediately apply what you just learned to your own web pages.

Whenever a new term is used, it is clearly explained. No flipping back and forth to a glossary!

TIP

Tips and tricks to save you precious time are set aside in Tips so that you can spot them quickly.

NOTE

Notes highlight interesting information you should be sure not to miss.

CAUTION

When there's something you need to watch out for, you'll be warned about it in a Caution.

Q&A, Quizzes, and Exercises

Every hour ends with a short question-and-answer session that addresses the kind of "dumb questions" everyone wants to ask. A brief but complete quiz lets you test yourself to be sure you understand everything presented in the hour. Finally, one or two optional exercises give you a chance to practice your new skills before you move on.

This page intentionally left blank

This page intentionally left blank

HOUR 5

Jumping into jQuery and JavaScript Syntax

What You'll Learn in This Hour:

- Ways to add jQuery and JavaScript to your web pages
- Creating and manipulating arrays of objects
- Adding code logic to JavaScript
- Implementing JavaScript functions for cleaner code

Throughout the book, you'll see several examples of using jQuery and JavaScript to perform various dynamic tasks. jQuery doesn't replace JavaScript, it enhances it by providing an abstract layer to perform certain common tasks, such as finding elements or values, changing attributes and properties of elements, and interacting with browser events.

In this hour, you learn the basic structure and syntax of JavaScript and how to use jQuery to ease some of the development tasks. The purpose of this hour is to help you become familiar with the JavaScript language syntax, which is also the jQuery language syntax.

Adding jQuery and JavaScript to a Web Page

Browsers come with JavaScript support already built in to them. That means all you need to do is add your own JavaScript code to the web page to implement dynamic web pages. jQuery, on the other hand, is an additional library, and you will need to add the jQuery library to your web page before adding jQuery scripts.

Loading the jQuery Library

Because the jQuery library is a JavaScript script, you use the `<script>` tag to load the jQuery into your web page. jQuery can either be downloaded to your code directory and then hosted on your web server, or you can use the hosted versions that are available at jQuery.com. The following statement shows an example of each; the only difference is where jQuery is being loaded from:

```
<script src="http://code.jquery.com/jquery-latest.min.js"></script>
<script src="includes/js/jquery-latest.min.js"></script>
```

CAUTION

Remember that you need to place the `<script>` element to load the jQuery library before any script elements that are using it. Otherwise, those libraries will not be able to link up to the jQuery code.

The jQuery library downloads and hosted links can be found at the following location: http://jquery.com/download/

Implementing Your Own jQuery and JavaScript

jQuery code is implemented as part of JavaScript scripts. To add jQuery and JavaScript to your web pages, first add a `<script>` tag that loads the jQuery library, and then add your own `<script>` tags with your custom code.

The JavaScript code can be added inside the `<script>` element, or the `src` attribute of the `<script>` element can point to the location of a separate JavaScript document. Either way, the JavaScript will be loaded in the same manner.

The following is an example of a pair of `<script>` statements that load jQuery and then use it. The `document.write()` function just writes text directly to the browser to be rendered:

```
<script src="http://code.jquery.com/jquery-latest.min.js"></script>
<script>
  function writeIt(){
    document.write("jQuery Version " + $().jquery + " loaded.");
  }
</script>
```

NOTE

The `<script>` tags do not need to be added to the `<head>` section of the HTML document; they can also be added in the body. It's useful to add simple scripts directly inline with the HTML elements that are consuming them.

Accessing HTML Event Handlers

So after you add your JavaScript to the web page, how do you get it to execute? The answer is that you tie it to the browser events. Each time a page or element is loaded, the user moves or clicks the mouse or types a character, an HTML event is triggered.

Each supported event is an attribute of the object that is receiving the event. If you set the attribute value to a JavaScript function, the browser will execute your function when the event is triggered.

For example, the following will execute the `writeIt()` function when the body of the HTML page is loaded:

```
<body onload="writeIt()">
```

TRY IT YOURSELF ▼

Implementing JavaScript and jQuery

Those are the basic steps. Now it is time to try it yourself. Use the following steps to add jQuery to your project and use it dynamically in a web page:

1. In Aptana, create a source folder named hour05.
2. In the same folder as the hour05 folder, add an additional directory called js.
3. Go to jQuery.com/download and download the latest jQuery library to that folder and name the file jquery.min.js. The file may come up as clear text in the browser. If so, just press Ctrl+s (Command-s on Macs) and save the file that way.
4. Now create a source file named hour0501.html in the hour05 folder.
5. Add the usual basic elements (html, head, body).
6. Inside the `<head>` element, add the following line to load the library you just downloaded:

```
06      <script src="../js/jquery.min.js"></script>
```

7. Now you can add your own `<script>` tag with the following code to print out the jQuery version to the browser windows:

```
07      <script>
08        function writeIt(){
09          document.write("jQuery Version " + $().jquery + " loaded.");
10        }
11      </script>
```

8. To have your script execute when the document is loaded, tie the `writeIt()` function to the `<body> onload` event using the following line:

```
13    <body onload="writeIt()">
```

9. Save the file, shown in Listing 5.1, and view it in a web browser. The output should be similar to Figure 5.1.

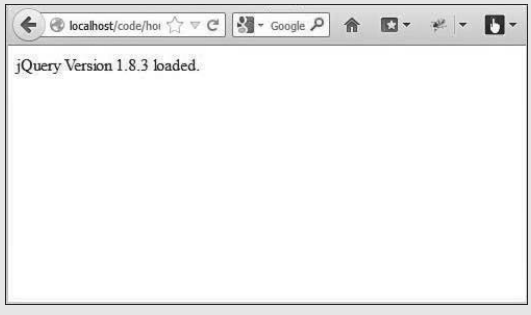

FIGURE 5.1
The function `writeIt()` is executed when the body loads and writes the jQuery version to the browser.

LISTING 5.1 Very Basic Example of Loading Using jQuery in a Web Page to Print Out Its Own Version

```
01 <!DOCTYPE html>
02 <html>
03   <head>
04     <title>Hour 5-1</title>
05     <meta charset="utf-8" />
06     <script src="../js/jquery.min.js"></script>
07     <script>
08       function writeIt(){
09         document.write("jQuery Version " + $().jquery + " loaded.");
10       }
11     </script>
12   </head>
13   <body onload="writeIt()">
14   </body>
15 </html>
```

Accessing the DOM

One of the most important aspects of JavaScript, and especially jQuery, is the capability to access and manipulate the DOM. Accessing the DOM is how you make the web page dynamic by changing styles, size, position, and values of elements.

In the following sections, you learn about accessing the DOM through traditional methods via JavaScript and the much improved methods using jQuery selectors. These sections are a brief introduction. You will get plenty of practice as the hours roll on.

Using Traditional JavaScript to Access the DOM

Traditionally, JavaScript uses the global `document` object to access elements in the web page. The simplest method of accessing an element is to directly refer to it by `id`. For example, if you have a paragraph with the `id="question"` you can access it via the following JavaScript `getElementById()` function:

```
var q = document.getElementById("question");
...
<p id="question">Which method do you prefer?</p>
```

Another helpful JavaScript function that you can use to access the DOM elements is `getElementsByTagName()`. This returns a JavaScript array of DOM elements that match the tag name. For example, to get a list of all the `<p>` elements, use the following function call:

```
var paragraphs = document.getElementsByTagName("p");
```

Using jQuery Selectors to Access HTML Elements

Accessing HTML elements is one of jQuery's biggest strengths. jQuery uses selectors that are very similar to CSS selectors to access one or more elements in the DOM, hence, the name jQuery. jQuery returns back either a single element or an array of jQuerified objects. jQuerified means that additional jQuery functionality has been added to the DOM object, allowing for much easier manipulation.

The syntax for using jQuery selectors is `$(`*`selector`*`).`*`action`*`()`, where *`selector`* is replaced by a valid selector and *`action`* is replaced by a jQuerified action attached to the DOM element(s).

For example, the following command finds all paragraph elements in the HTML document and sets the CSS `font-weight` property to `bold`:

```
$("p").css('font-weight', 'bold');
```

TRY IT YOURSELF ▼

Using jQuery and JavaScript to Access DOM Elements

Now to solidify the concepts, you'll run through a quick example of accessing and modifying DOM elements using both jQuery and JavaScript. Use the following steps to build the HTML document shown in Listing 5.2:

1. Create a source file named hour0502.html in the hour05 folder.
2. Add the usual basic elements (html, head, body).

3. Inside the `<head>` element, add the following line to load the library you just downloaded.

```
06     <script src="../js/jquery.min.js"></script>
```

4. Add the following `<script>` element that accesses the DOM using both the JavaScript and jQuery methods. Notice that with jQuery two actions are chained together. The first sets the CSS `font-weight` property and the second changes text contained in element. With JavaScript, you use the `getElementById()` method, and then you set the `innerHTML` property directly in the DOM to change the text displayed in the browser.

```
07     <script>
08       function writeIt(){
09         $("#heading").css('font-weight', 'bold').html("jQuery");
10         var q = document.getElementById("question");
11         q.innerHTML = "I Prefer jQuery!";
12       }
13     </script>
```

5. To have your script execute when the document is loaded, tie the `writeIt()` function to the `<body> onload` event using the following line:

```
15   <body onload="writeIt()">
```

6. Add the following `<p>` elements to the `<body>` to provide containers for the JavaScript code to access:

```
16     <p id="heading">jQuery or JavaScript</p>
17     <p id="question">Which method do you prefer?</p>
```

7. Save the file and view it in a web browser. The output should be similar to Figure 5.2.

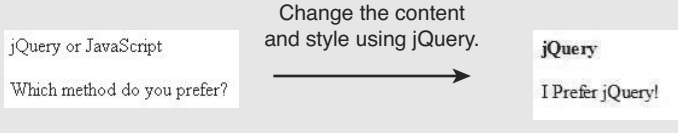

FIGURE 5.2
The function `writeIt()` is executed when the body loads and changes the content and appearance of the text.

LISTING 5.2 Very Basic Example of Using JavaScript and jQuery to Access DOM Elements

```
01 <!DOCTYPE html>
02 <html>
03   <head>
04     <title>Hour 5-2</title>
```

```
05     <meta charset="utf-8" />
06     <script src="../js/jquery.min.js"></script>
07     <script>
08       function writeIt(){
09         $("#heading").css('font-weight', 'bold').html("jQuery");
10         var q = document.getElementById("question");
11         q.innerHTML = "I Prefer jQuery!";
12       }
13     </script>
14   </head>
15   <body onload="writeIt()">
16     <p id="heading">jQuery or JavaScript</p>
17     <p id="question">Which method do you prefer?</p>
18   </body>
19 </html>
```

Understanding JavaScript Syntax

Like any other computer language, JavaScript is based on a rigid syntax where specific words mean different things to the browser as it interprets the script. This section is designed to walk you through the basics of creating variables, working with data types, and using looping and functions in JavaScript to manipulate your web pages.

Creating Variables

The first place to begin with in JavaScript is variables. Variables are a means to name data so that you can use that name to temporarily store and access data from your JavaScript files. Variables can point to simple data types, such as numbers or strings, or they can point to more complex data types, such as objects.

To define a variable in JavaScript, you must use the `var` keyword and then give the variable a name; for example:

```
var myData;
```

You can also assign a value to the variable in the same line. For example, the following line of code creates a variable `myString` and assigns it the value of `"Some Text"`:

```
var myString = "Some Text";
```

This works as well as

```
var myString;
myString = "Some Text";
```

After you have declared the variable, you can use the name to assign the variable a value and access the value of the variable. For example, the following code stores a string into the `myString` variable and then uses it when assigning the value to the `newString` variable:

```
var myString = "Some Text";
var newString = myString + "Some More Text";
```

Your variable names should describe the data that is stored in them so that it is easy to use them later in your program. The only rule for creating variable names is that they must begin with a letter, `$`, or `_`, and they cannot contain spaces. Also remember that variable names are case sensitive, so using `myString` is different from `MyString`.

Understanding JavaScript Data Types

JavaScript uses data types to determine how to handle data that is assigned to a variable. The variable type will determine what operations you can perform on the variable, such as looping or executing. The following list describes the most common types of variables that we will be working with through the book:

- **String**—Stores character data as a string. The character data is specified by either single or double quotes. All the data contained in the quotes will be assigned to the string variable. For example:

  ```
  var myString = 'Some Text';
  var anotherString = "Some Other Text";
  ```

- **Number**—Stores the data as a numerical value. Numbers are useful in counting, calculations, and comparisons. Some examples are

  ```
  var myInteger = 1;
  var cost = 1.33;
  ```

- **Boolean**—Stores a single bit that is either true or false. Booleans are often used for flags. For example, you might set a variable to false at the beginning of some code and then check it on completion to see whether the code execution hit a certain spot. The following shows an example of defining a true and a false variable:

  ```
  var yes = true;
  var no = false;
  ```

- **Array**—An indexed array is a series of separate distinct data items all stored under a single variable name. Items in the array can be accessed by their zero-based index using the `[index]`. The following is an example of creating a simple array and then accessing the first element, which is at index 0:

  ```
  var arr = ["one", "two", "three"]
  var first = arr[0];
  ```

- **Associative Array/Objects**—JavaScript does support the concept of an associative array, meaning accessing the items in the array by a name instead of an index value. However, a better method is to use an object literal. When you use an object literal, you can access items in the object using `object.property` syntax. The following example shows how to create and access an object literal:

```
var obj = {"name":"Brad", "occupation":"Hacker", "age", "Unknown"};
var name = obj.name;
```

- **Null**—At times you do not have a value to store in a variable, either because it hasn't been created or you are no longer using it. At this time you can set a variable to `null`. That way you can check the value of the variable in your code and use it only if it is not `null`.

```
var newVar = null;
```

NOTE

JavaScript is a typeless language, meaning you do not need to tell the browser what data type the variable is; the interpreter will automatically figure out the correct data type for the variable.

Using Operators

JavaScript operators provide the capability to alter the value of a variable. You are already familiar with the = operator because you used it several times in the book already. JavaScript provides several operators that can be grouped into two types—arithmetic and assignment.

Arithmetic Operators

Arithmetic operators are used to perform operations between variable and direct values. Table 5.1 shows a list of the arithmetic operations along with the results that get applied.

TABLE 5.1 Table Showing JavaScript's Arithmetic Operators as Well as Results Based on y=4 to Begin With

Operator	Description	Example	Resulting x	Resulting y
+	Addition	x=y+5	9"49"	444
		x=y+"5"	"Four44"	
		x="Four"+y+"4"		
-	Subtraction	x=y-2	2	4
++	Increment	x=y++	4	5
		x=++y	5	5

Operator	Description	Example	Resulting x	Resulting y
--	Decrement	x=y--	4	3
		x=--y	3	3
*	Multiplication	x=y*4	16	4
/	Division	x=10/y	2.5	4
%	Modulous (remainder of Division)	x=y%3	1	4

TIP

The + operator can also be used to add strings or strings and numbers together. This allows you to quickly concatenate strings and add numerical data to output strings. Table 5.1 shows that when adding a numerical value and a string value, the numerical value is converted to a string, and then the two strings are concatenated.

Assignment Operators

Assignment operators are used to assign a value to a variable. You are probably used to the = operator, but there are several forms that allow you to manipulate the data as you assign the value. Table 5.2 shows a list of the assignment operations along with the results that get applied.

TABLE 5.2 JavaScript's Assignment Operators as Well as Results Based on x=10 to Begin With

Operator	Example	Equivalent Arithmetic Operators	Resulting x
=	x=5	x=5	5
+=	x+=5	x=x+5	15
-=	x-=5	x=x-5	5
*=	x*=5	x=x*5	25
/=	x/=5	x=x/5	2
%=	x%=5	x=x%5	0

Applying Comparison and Conditional Operators

Conditionals are a way to apply logic to your applications so that certain code will be executed only under the correct conditions. This is done by applying comparison logic to variable values. The following sections describe the comparisons available in JavaScript and how to apply them in conditional statements.

Comparison Operators

A comparison operator evaluates two pieces of data and returns true if the evaluation is correct or false if the evaluation is not correct. Comparison operators compare the value on the left of the operator against the value on the right.

The simplest way to help you understand comparisons is to provide a list with some examples. Table 5.3 shows a list of the comparison operators along with some examples.

TABLE 5.3 JavaScript's Comparison Operators as Well as Results Based on x=10 to Begin With

Operator	Example	Example	Result
==	Is equal to (value only)	x==8	false
		x==10	true
===	Both value and type are equal	x===10	true
		x==="10"	false
!=	Is not equal	x!=5	true
!==	Both value and type are not equal	x!=="10"	true
		x!==10	false
>	Is greater than	x>5	true
>=	Is greater than or equal to	x>=10	true
<	Is less than	x<5	false
<=	Is less than or equal to	x<=10	true

You can chain multiple comparisons together using logical operators. Table 5.4 shows a list of the logical operators and how to use them to chain comparisons together.

TABLE 5.4 JavaScript's Comparison Operators as Well as Results Based on x=10 and y=5 to Begin With

Operator	Description	Example	Result
&&	and	(x==10 && y==5)	true
		(x==10 && y>x)	false
||	or	(x>=10 || y>x)	true
		(x<10 && y>x)	false
!	not	!(x==y)	true
		!(x>y)	false
	mix	(x>=10 && y<x || x==y)	true
		((x<y || x>=10) && y>=5)	true
		(!(x==y) && y>=10)	false

If

An `if` statement enables you to separate code execution based on the evaluation of a comparison. The syntax is shown in the following lines of code where the conditional operators are in `()` parentheses and the code to execute if the conditional evaluates to true is in `{}` brackets:

```
if(x==5){
  do_something();
}
```

In addition to executing code only within the `if` statement block, you can specify an `else` block that will get executed only if the condition is false. For example:

```
if(x==5){
  do_something();
} else {
  do_something_else();
}
```

You can also chain `if` statements together. To do this, add a conditional statement along with an `else` statement. For example:

```
if(x<5){
  do_something();
} else if(x<10) {
  do_something_else();
} else {
  do_nothing();
}
```

switch

Another type of conditional logic is the `switch` statement. The `switch` statement allows you to evaluate an expression once and then, based on the value, execute one of many sections of code.

The syntax for the `switch` statement is the following:

```
switch(expression){
  case value:
    code to execute
    break;
  case value2:
    code to execute
    break;
  default:
    code to execute if not value or value2.
}
```

This is what is happening. The `switch` statement will evaluate the expression entirely and get a value. The value may be a string, a number, a Boolean, or even an object. The `switch` value is then compared to each value specified by the `case` statement. If the value matches, the code in the `case` statement is executed. If no values match, the `default` code is executed.

NOTE

Typically, each `case` statement will include a break command at the end to signal a break out of the `switch` statement. If no break is found, code execution will continue with the next `case` statement.

TRY IT YOURSELF ▼

Applying `if` Conditional Logic in JavaScript

To help you solidify using JavaScript conditional logic, use the following steps to build conditional logic into the JavaScript for a dynamic web page. The final version of the HTML document is shown in Listing 5.3:

1. Create a source file named hour0503.html in the hour05 folder.
2. Create a folder under hour05 named images.
3. Copy the day.png and night.png images from the website under code/hour05/images, or substitute your own into the images folder.
4. Add the usual basic elements (html, head, body).

5. Add the following `<script>` element that gets the hour value using the `Date().getHours()` JavaScript code. The code uses `if` statements to determine the time of day and does two things: It writes a greeting onto the screen and sets the value of the `timeOfDay` variable.

```
06     <script>
07       function writeIt(){
08         var hour = new Date().getHours();
09         var timeOfDay;
10         if(hour>=7 && hour<12){
11           document.write("Good Morning!");
12           timeOfDay="morning";
13         } else if(hour>=12 && hour<18) {
14           document.write("Good Day!");
15           timeOfDay="day";
16         } else {
17           document.write("Good Night!");
18           timeOfDay="night";
19         }
32       }
33     </script>
```

6. Now add the following `switch` statement that uses the value of `timeOfDay` to determine which image to display in the web page:

```
20         switch(timeOfDay){
21           case "morning":
22           case "day":
23             document.write("<img src='images/day.png' />")
24             break;
25           case "night":
26             document.write("<img src='images/night.png' />")
27             break;
28           default:
29             document.write("<img src='images/day.png' />")
30             break;
31         }
```

7. Save the file and view it in a web browser. The output should be similar to Figure 5.3, depending on what time of day it is.

Between 7 a.m. and Noon

Between Noon and 6 p.m.

Between 6 p.m. and 7 a.m.

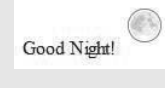

FIGURE 5.3
The function `writeIt()` is executed when the body loads and changes the greeting and image displayed on the web page.

LISTING 5.3 Simple Example of Using Conditional Logic Inside JavaScript

```
<!DOCTYPE html>
<html>
  <head>
    <title>Hour 5-3</title>
    <meta charset="utf-8" />
    <script>
      function writeIt(){
        var hour = new Date().getHours();
        var timeOfDay;
        if(hour>=7 && hour<12){
          document.write("Good Morning!");
          timeOfDay="morning";
        } else if(hour>=12 && hour<18) {
          document.write("Good Day!");
          timeOfDay="day";
        } else {
          document.write("Good Night!");
          timeOfDay="night";
        }
        switch(timeOfDay){
          case "morning":
          case "day":
            document.write("<img src='images/day.png' />")
            break;
          case "night":
            document.write("<img src='images/night.png' />")
            break;
          default:
            document.write("<img src='images/day.png' />")
            break;
        }
      }
    </script>
  </head>
  <body onload="writeIt()">
  </body>
</html>
```

Implementing Looping

Looping is a means to execute the same segment of code multiple times. This is extremely useful when you need to perform the same tasks on a set of DOM objects, or if you are dynamically creating a list of items.

JavaScript provides functionality to perform `for` and `while` loops. The following sections describe how to implement loops in your JavaScript.

`while` Loops

The most basic type of looping in JavaScript is the `while` loop. A `while` loop tests an expression and continues to execute the code contained in its { } brackets until the expression evaluates to false.

For example, the following `while` loop executes until the value of `i` is equal to 5:

```
var i = 1;
while (i<5){
  document.write("Iteration " + i + "<br>");
  i++;
}
```

The resulting output to the browser is as follows:

```
Iteration 1
Iteration 2
Iteration 3
Iteration 4
```

`do/while` Loops

Another type of `while` loop is the `do/while` loop. This is useful if you always want to execute the code in the loop at least once and the expression cannot be tested until the code has executed at least once.

For example, the following `do/while` loop executes until the value of day is equal to Wednesday:

```
var days = ["Monday", "Tuesday", "Wednesday", "Thursday", "Friday"];
var i=0;
do{
  var day=days[i++];
  document.write("It's " + day + "<br>");
} while (day != "Wednesday");
```

The resulting output to the browser is

```
It's Monday
It's Tuesday
It's Wednesday
```

`for` Loops

The JavaScript `for` loop allows you to execute code a specific number of times by using a `for` statement that combines three statements into one using the following syntax:

```
for (statement 1; statement 2; statement 3;){
  code to be executed;
}
```

The `for` statement uses those three statements as follows when executing the loop:

- **statement 1**—Executed before the loop begins and not again. This is used to initialize variables that will be used in the loop as conditionals.
- **statement 2**—Expression that is evaluated before each iteration of the loop. If the expression evaluates to true, the loop is executed; otherwise, the `for` loop execution ends.
- **statement 3**—Executed each iteration after the code in the loop has executed. This is typically used to increment a counter that is used in statement 2.

To illustrate a `for` loop, check out the following example. The example not only illustrates a basic `for` loop, it also illustrates the capability to nest one loop inside another:

```
for (var x=1; x<=3; x++){
  for (var y=1; y<=3; y++){
    document.write(x + " X " + y + " = " + (x*y) + "<br>");
  }
}
```

The resulting output to the web browser is as follows:

```
1 X 1 = 1
1 X 2 = 2
1 X 3 = 3
2 X 1 = 2
2 X 2 = 4
2 X 3 = 6
3 X 1 = 3
3 X 2 = 6
3 X 3 = 9
```

`for/in` Loops

Another type of `for` loop is the `for/in` loop. The `for/in` loop executes on any data type that can be iterated on. For the most part, you will use the `for/in` loop on arrays and objects. The following example illustrates the syntax and behavior of the `for/in` loop in a simple array:

```
var days = ["Monday", "Tuesday", "Wednesday", "Thursday", "Friday"];
for (var idx in days){
```

```
  document.write("It's " + days[idx] + "<br>");
}
```

Notice that the variable `idx` is adjusted each iteration through the loop from the beginning array index to the last. The resulting output is

```
It's Monday
It's Tuesday
It's Wednesday
It's Thursday
It's Friday
```

Interrupting Loops

When working with loops, at times you need to interrupt the execution of code inside the code itself without waiting for the next iteration. There are two ways to do this using the `break` and `continue` keywords.

The `break` keyword stops execution of the `for` or `while` loop completely. The `continue` keyword, on the other hand, stops execution of the code inside the loop and continues on with the next iteration. Consider the following examples:

Using a `break` if the `day` is `Wednesday`:

```
var days = ["Monday", "Tuesday", "Wednesday", "Thursday", "Friday"];
for (var idx in days){
  if (days[idx] == "Wednesday")
    break;
  document.write("It's " + days[idx] + "<br>");
}
```

When the value is `Wednesday`, loop execution stops completely:

```
It's Monday
It's Tuesday
```

Using a `continue` if the `day` is `Wednesday`:

```
var days = ["Monday", "Tuesday", "Wednesday", "Thursday", "Friday"];
for (var idx in days){
  if (days[idx] == "Wednesday")
    continue;
  document.write("It's " + days[idx] + "<br>");
}
```

Notice that the `write` is not executed for `Wednesday` because of the `continue`; however, the loop execution did complete:

```
It's Monday
```

```
It's Tuesday
It's Thursday
It's Friday
```

Creating Functions

One of the most important parts of JavaScript is making code that is reusable by other code. To do this, you combine your code into functions that perform specific tasks. A function is a series of code statements combined in a single block and given a name. The code in the block can then be executed by referencing that name.

Defining Functions

Functions are defined using the keyword `function` followed by a function name that describes the use of the function, list of zero or more arguments in `()` parentheses, and a block of one or more code statements in {} brackets. For example, the following is a function definition that writes "Hello World" to the browser.

```
function myFunction(){
  document.write("Hello World");
}
```

To execute the code in `myFunction()`, all you need to do is add the following line to the main JavaScript or inside another function:

```
myFunction();
```

Passing Variables to Functions

Frequently, you will need to pass specific values to functions that they will use when executing their code. Values are passed in comma-delimited form to the function. The function definition will need a list of variable names in the `()` parentheses that match the number being passed in. For example, the following function accepts two arguments, a `name` and `city`, and uses them to build the output string:

```
function greeting(name, city){
  document.write("Hello " + name);
  document.write(". How is the weather in " + city);
}
```

To call the `greeting()` function, we need to pass in a `name` value and a `city` value. The value can be a direct value or a previously defined variable. To illustrate this, the following code will execute the `greeting()` function with a `name` variable and a direct string for the `city`:

```
var name = "Brad";
greeting(name, "Florence");
```

Returning Values from Functions

Often, functions will need to return a value to the calling code. Adding a `return` keyword followed by a variable or value will return that value from the function. For example, the following code calls a function to format a string, assigns the value returned from the function to a variable, and then writes the value to the browser:

```
function formatGreeting(name, city){
  var retStr = "";
  retStr += "Hello <b>" + name + "</b><br>";
  retStr += "Welcome to " + city + "!";
 return retStr;
}
var greeting = formatGreeting("Brad", "Rome");
document.write(greeting);
```

You can include more than one `return` statement in the function. When the function encounters a `return` statement, code execution of the function is stopped immediately. If the `return` statement contains a value to return, that value is returned. The following example shows a function that tests the input and returns immediately if it is zero:

```
function myFunc(value){
  if (value == 0)
    return;
  code_to_execute_if_value_nonzero;
}
```

▼ TRY IT YOURSELF

Creating JavaScript Functions

To help solidify functions, use the following steps to integrate some functions into a JavaScript application. The following steps take you through the process of creating a function, calling it to execute code, and then handling the results returned:

1. Create a source file named hour0504.html in the hour05 folder.
2. Add the usual basic elements (html, head, body).
3. Add a `<script>` tag to the `<head>` element to house the JavaScript.
4. Insert the following object literal definition at the beginning of the script. The object will have planet names for attributes, and each planet name is a reference to an array of moons.

```
07       var moonData = {"Earth":["Luna"],
08                       "Jupiter":["Io", "Europa"],
09                       "Saturn":["Titan", "Rhea"],
```

```
10                          "Mars":["Phobos"]};
```

5. Add the following function that will be called by the `onload` event. In this function you use a nested `for/in` loop to iterate through the `moonData` object attributes. The outer loop gets the planet `name` and the inner loop loops through the index of the `moon` array.

```
11        function writeIt(){
12          for (planet in moonData){
13            var moons = moonData[planet];
14            for (moonIdx in moons){
15              var moon = moons[moonIdx];
16              var listItem = makeListItem(planet, moon);
17              document.write(listItem);
18            }
19          }
20        }
```

6. Notice that on line 16 of the `writeIt()` function is a call to `makeListItem()`. That function needs to return a value that can be used in line 17 to write to the document. Add the following code to create the function. The function takes two arguments: a `name` and a `value`, then generates an HTML string to create a `<li>` element and returns the string.

```
21        function makeListItem(name, value){
22          var itemStr = "<li>" + name + ": " + value + "</li>";
23          return itemStr;
24        }
```

7. Save the file, shown in Listing 5.4, and open it in a web browser. You should see the results shown in Figure 5.4. You have just created two JavaScript functions: one that takes no arguments and does not return a value and the other that takes two arguments and returns a formatted HTML string containing the argument strings.

- Earth: Luna
- Jupiter: Io
- Jupiter: Europa
- Saturn: Titan
- Saturn: Rhea
- Mars: Phobos

FIGURE 5.4
The function `writeIt()` is executed, which iterates through the `moonData` object and makes calls to the `makeListItem()` function to format the planet and moon names as an HTML `<li>` element.

LISTING 5.4 Simple Example of Using Conditional Logic Inside JavaScript

```
01 <!DOCTYPE html>
02 <html>
```

```
<head>
  <title>Hour 5-4</title>
  <meta charset="utf-8" />
  <script>
    var moonData = {"Earth":["Luna"],
                    "Jupiter":["Io", "Europa"],
                    "Saturn":["Titan", "Rhea"],
                    "Mars":["Phobos"]};
    function writeIt(){
      for (planet in moonData){
        var moons = moonData[planet];
        for (moonIdx in moons){
          var moon = moons[moonIdx];
          var listItem = makeListItem(planet, moon);
          document.write(listItem);
        }
      }
    function makeListItem(name, value){
      var itemStr = "<li>" + name + ": " + value + "</li>";
      return itemStr;
    }
  </script>
</head>
<body onload="writeIt()">
</body>
</html>
```

Understanding Variable Scope

After you start adding conditions, functions, and loops to your JavaScript applications, you need to understand variable scoping. Variable scope is simply this: “what is the value of a specific variable name at the current line of code being executed."

JavaScript enables you to define both a global and a local version of the variable. The global version is defined in the main JavaScript, and local versions are defined inside functions. When you define a local version in a function, a new variable is created in memory. Within that function, you will be referencing the local version. Outside that function, you will be referencing the global version.

To understand variable scoping a bit better, consider the following code:

```
<script>
  var myVar = 1;
  function writeIt(){
```

```
04      var myVar = 2;
05      document.write(myVar);
06      writeMore();
07    }
08    function writeMore(){
09      document.write(myVar);
10    }
11 </script>
```

The global variable `myVar` is defined on line 2. Then on line 4, a local version is defined within the `writeIt()` function. So, line 5 will write 2 to the document. Then in line 6, `writeMore()` is called. Because there is no local version of `myVar` defined in `writeMore()`, the value of the global `myVar` is written in line 9.

Adding Error Handling

An important part of JavaScript coding is adding error handling for instances where there may be problems. By default, if a code exception occurs because of a problem in your JavaScript, the script fails and does not finish loading. This is not usually the desired behavior.

`Try/Catch` Blocks

To prevent your code from totally bombing out, use `try/catch` blocks that can handle problems inside your code. If JavaScript encounters an error when executing code in a `try/catch` block, it will jump down and execute the `catch` portion instead of stopping the entire script. If no error occurs, all of the `try` will be executed and none of the `catch`.

For example, the following `try/catch` block will execute any code that replaces `your_code_here`. If an error occurs executing that code, the error message followed by the string `": happened when loading the script"` will be written to the document:

```
try {
    your_code_here
} catch (err) {
  document.write(err.message + ": happened when loading the script");
}
```

Throw Your Own Errors

You can also throw your own errors using a `throw` statement. The following code illustrates how to add throws to a function to `throw` an error, even if a script error does not occur:

```
01 <script>
02   function sqrRoot(x) {
03     try {
04       if(x=="")    throw "Can't Square Root Nothing";
05       if(isNaN(x)) throw "Can't Square Root Strings";
```

```
06        if(x<0)      throw "Sorry No Imagination";
07        return "sqrt("+x+") = " + Math.sqrt(x);
08      } catch(err){
09        return err;
10      }
11    }
12    function writeIt(){
13      document.write(sqrRoot("four") + "<br>");
14      document.write(sqrRoot("") + "<br>");
15      document.write(sqrRoot("4") + "<br>");
16      document.write(sqrRoot("-4") + "<br>");
17    }
18 </script>
```

The function `sqrRoot()` accepts a single argument `x`. It then tests `x` to verify that it is a positive number and returns a string with the square root of `x`. If `x` is not a positive number, the appropriate error is thrown and returned to `writeIt()`.

Using `finally`

Another valuable tool in exception handling is the `finally` keyword. A `finally` keyword can be added to the end of a `try/catch` block. After the `try/catch` blocks are executed, the `finally` block is always executed. It doesn't matter if an error occurs and is caught or if the `try` block is fully executed.

Following is an example of using a `finally` block inside a web page:

```
function testTryCatch(value){
  try {
    if (value < 0){
      throw "too small";
    } else if (value > 10){
      throw "too big";
    }
    your_code_here
  } catch (err) {
    document.write("The number was " + err.message");
  } finally {
    document.write("This is always written.");
  }
}
```

Summary

In this hour, you learned the basics of adding jQuery and JavaScript to web pages. The basic data types that are used in JavaScript and, consequently, jQuery, were described. You learned some of the basic syntax of applying conditional logic to JavaScript applications. You also learned how to compartmentalize your JavaScript applications into functions that can be reused in other locations. Finally, you learned some ways to handle JavaScript errors in your script before the browser receives an exception.

Q&A

Q. **When should you use a regular expression in string operations?**

A. That depends on your understanding of regular expressions. Those who use regular expressions frequently and understand the syntax well would almost always rather use a regular expression because they are so versatile. If you are not very familiar with regular expressions, it takes time to figure out the syntax, and so you will want to use them only when you need to. The bottom line is that if you need to manipulate strings frequently, it is absolutely worth it to learn regular expressions.

Q. **Can I load more than one version of jQuery at a time?**

A. Sure, but there really isn't a valid reason to do that. The one that gets loaded last will overwrite the functionality of the previous one. Any functions from the first one that were not overwritten may be completely unpredictable because of the mismatch in libraries. The best bet is to develop and test against a specific version and update to a newer version only when there is added functionality that you want to add to your web page.

Workshop

The workshop consists of a set of questions and answers designed to solidify your understanding of the material covered in this hour. Try to answer the questions before looking at the answers.

Quiz

1. What is the difference between `==` and `===` in JavaScript?
2. What is the difference between the `break` and `continue` keywords?
3. When should you use a `finally` block?
4. What is the resulting value when you add a string "1" to a number 1, ("1"+1)?

Quiz Answers

1. `==` compares only the relative value; `===` compares the value and the type.
2. `break` will stop executing the loop entirely, whereas `continue` will only stop executing the current iteration and then move on to the next.
3. When you have code that needs to be executed even if a problem occurs in the `try` block.
4. The string "11" because the number is converted to a string and then concatenated.

Exercises

1. Open hour0504.html and modify it to create a table instead of a list. You will need to add code to the `writeIt()` function that writes the `<table>` open tag before iterating through the planets and then the closing tag after iterating through the planets. Then modify the `makeListItem()` function to return a string in the form of

   ```
   <tr><td>planent</td><td>moon</td></tr>
   ```

2. Modify hour0503.html to include some additional times with different messages and images. For example, between 8 and 9 you could add the message "go to work" with a car icon, between 5 and 6 you could add the message "time to go home" with a home icon. You will need to add some additional cases to the `switch` statement and set the `timeOfDay` value accordingly.

This page intentionally left blank

Index

A

<a> element, 78

abort event, 228

abort() method, 421

abs() method, 176

ACCEPT header, 10

accept option (droppable widget), 499

accept rule (form validation), 354

accessing

- browser values, 260-266
- data outside the web page, 285, 295
  - browser history object, 289
  - browser location object, 285
  - cookies, 291-294
  - external links, controlling, 290
  - pop-up boxes, 294-296
  - screen object, 285-286
  - timers, 296-299
  - window object, 285-288

DOM (Document Object Model), 137-141

- example, 140-141
- finding objects by class name, 189
- finding objects by ID, 189
- finding objects by tag name, 189-190
- with jQuery selectors, 139, 201-203
- sample project, 191-192
- with traditional JavaScript, 139

form elements, 326

- attributes, 326-327
- button inputs, 330
- check box inputs, 328
- file inputs, 330-331
- hidden inputs, 331
- radio inputs, 328-329
- select inputs, 329-330
- text input elements, 327-328

server-side data with AJAX, 406-408

- asynchronous communication, 397

compared to page requests, 395-396
cross-domain requests, 397-398
GET requests, 398-399
global event handlers, 419
global setup, 396-419
implementing from JavaScript, 399-401
implementing from jQuery, 401-404
low-level AJAX requests, handling, 420-422
overview, 395
POST requests, 398-399
request handling, 397
response data, handling, 405, 408-414
response data types, 399
server data, updating, 415-419

accordian widget, 522
accounts (Google), creating, 439
active option (tabs widget), 534
activeClass option (droppable widget), 499
.add() method, 211
.addClass() method, 188, 271, 482
addEventListener() method, 232-236
addFBsdk() function, 427, 431-432
addItem() function, 378
adjValues() function, 390
.after() method, 270
AJAX (Asynchronous JavaScript and XML)
asynchronous communication, 397
bypassing, 562
compared to page requests, 395-396
cross-domain requests, 397-398
DOM insertion, 560-562
GET requests, 398-399
global event handlers, 419
global setup, 396-419
implementing from JavaScript, 399-401
implementing from jQuery, 401-404
login requests, handling, 405-408
low-level AJAX requests, handling, 420-422
overview, 16-17, 395
POST requests, 398-399
request handling, 397
response data, handling, 405, 408-414
JSON response data, 408-411
XML/HTML response data, handling, 412-414
response data types, 399
simple example, 16-17
updating server data with, 415-419
.ajax() method, 397, 420-421
.ajaxComplete() method, 419
.ajaxError() method, 419
.ajaxSend() method, 419
.ajaxSetup() method, 419
.ajaxStart() method, 419
.ajaxStop() method, 419
.ajaxSuccess() method, 419
alert() method, 287, 295
alsoResize option (resizable widget), 504
altKey property (events), 225
.always() method, 405, 422
analyzing network traffic, 59-62
.andSelf() method, 211
animations
adding to form elements, 346-351
animation queues, 302
CSS settings, animating, 301-303
delaying, 304
effects, 488-490
.hide() method, 305
moving elements, 318-322
element position changes on nonstatic elements, 319
element position changes on static elements, 319
paper airplane app, 319-322
overview, 301
.promise() method, 305
resize animations, 316-318
.show() method, 305-306
sliding animation, 312
dynamic menu sample project, 314-316
.slideDown() method, 312
.slideToggle() method, 312

.slideUp() method, 312
width and height, 312
stopping, 302-304
visibility
fade animation to implement image selection effect, 311
.fadeIn() method, 309
.fadeOut() method, 309
.fadeTo() method, 310
.fadeToggle() method, 309
.append() method, 268
appendChild() method, 187, 267
.appendTo() method, 268
appendTo option (selectable widget), 508
Aptana Studio
configuring, 19-20
installing, 18-19
arithmetic operators, 143
Array object, 166-174
adding/removing items, 171
checking whether array contains an item, 171
combining, 169
converting to strings, 171
iterating through, 169
methods, 169-170
overview, 166-169
sample project, 173-174
arrays, 142, 166
aspectRatio option (resizable widget), 504
assign() method, 285
assignment operators, 143
associative arrays, 143
asynchronous communication, 397
Asynchronous JavaScript and XML. *See* AJAX (Asynchronous JavaScript and XML)
at option (.position() method), 467
.attr() method, 188, 327
attribute jQuery selectors, 193-195
<audio> element, 94
AUTHORIZATION header, 11
autocomplete widget, 523-524
autoHide option (resizable widget), 504
availHeight property (screen object), 285
availWidth property (screen object), 285
axis option
draggable widget, 495
sortable widget, 513

B

back button, 557
background-attachment property (CSS), 111
background-color property (CSS), 111
background-image property (CSS), 111
background-position property (CSS), 111
background-repeat property (CSS), 111
background-size property (CSS), 111
backgrounds, applying with CSS, 111
basic jQuery selectors, 193-194
.before() method, 270
bind() method, 236
blind effect, 476
block elements, 73-75
blur event, 228
.blur() method, 287, 339
blurring form elements, 339
<body> element, 66
Boolean data type, 142
border attribute (table elements), 80
border-color property (CSS), 117
border-radius property (CSS), 117
border-style property (CSS), 117
border-width property (CSS), 117
borders, applying with CSS, 117-121
bounce effect, 476
box model, 122-123
box-shadow property (CSS), 117
break keyword, 152
browser development tools, 21-22
Firebug on Firefox, 21-22
Internet Explorer developer tools, 22-23
JavaScript console in Chrome, 22
browser values, accessing, 260-266
browsers. *See also* browser development tools
browser window, 7
events, 7
history object, 289

Internet Explorer developer tools, 22-23
location object, 285
overview, 6
buildData() function, 372
<button> element, 83
button inputs, 330
.button() method, 524
button property (events), 225
buttonImage option (datepicker widget), 525
buttonImageOnly option (datepicker widget), 525
buttons
Follow button, 445
jQuery UI buttons, 524-525
Like button, 427
in mobile forms, 603
navbars, 567-571
navigation buttons
back button, 557
creating, 556
positioning, 557
Send button, 428
Tweet button, 444
buttons option (dialog widget), 527
.buttonset() method, 524
bypassing AJAX, 562

c

calendar input, creating, 525-526
callbacks, 251
callback mechanism, 251-252
deferred objects, 252
cancel option (mouse interaction widget), 494
cancelable property (events), 225
<canvas> element, 91-93
<caption> element, 80-83
Cascading Style Sheets. *See* CSS (Cascading Style Sheets)
ceil() method, 176
center attribute (mapOptions), 433
center_changes event, 434
chaining jQuery object operations, 205-206
change event, 228
change option (slider widget), 531
changeCheckbox() function, 347, 349
.changePage() method, 557-558
changeRadio() function, 347, 349
changing classes, 270
charAt() method, 165
charCode property (events), 225
charCodeAt() method, 165
check box inputs, 328, 608
checkStatus() method, 296
childNodes attribute (DOM objects), 187
.children() method, 208
Chrome JavaScript console, 22
class attribute, 73, 187
class name, finding DOM objects by, 189
class transitions, 482-484
classes. *See* specific classes
className attribute, 260
clearInterval() method, 287, 296
clearTimeout() method, 287, 296
click event, 229
click() method, 187-188
client-side scripts, 14
clientX property (events), 225
clientY property (events), 225
clip effect, 476
close() method, 287
closed property (window object), 287
.closest() method, 209
<col> element, 80-83
<colgroup> element, 80-83
collapsible blocks and sets, 590-592
collapsible option
accordian widget, 522
tabs widget, 534
collision option (.position() method), 467
color property (CSS), 103-105
colorDepth property (screen object), 285
colspan attribute (table elements), 80
columntogglemode (tables), 596
combining
arrays, 169
strings, 164-166
comment fields, 428
compare() function, 373
comparison operators
if, 146
switch, 147
table of, 145
complete option (.animate() method), 302

concat() method, 164-166, 169-170
conditional logic, 148-149
configuring
Aptana Studio, 19-20
browser development tools, 21-22
Firebug on Firefox, 21-22
Internet Explorer developer tools, 22-23
JavaScript console in Chrome, 22
jQuery Mobile default settings, 548
confirm() method, 287, 295
confirmation pop-ups, 295
connectTo option (sortable widget), 513
container elements, 75-78
containment option
draggable widget, 495
resizable widget, 503
content jQuery selectors, 193-196
content option (tooltips widget), 535
content size, setting with CSS, 123
CONTENT-LENGTH header, 11
CONTENT-TYPE header, 11
.contents() method, 208
continue keyword, 152
continueNotify() function, 298
controls attribute (Map), 434
converting
arrays to strings, 171
objects, 188-189
COOKIE header, 11
cookies, 291-294
cos() method, 176
create event, 494
createElement() method, 267
createEvent() method, 241
createPopup() method, 287
createTextNode() method, 267
creditcard rule (form validation), 354
cross-domain requests, 397-398
CSS (Cascading Style Sheets), 9
adding, 27-30
to headers, 99
to HTML body, 99
animating CSS settings, 301-303
applying, 97-98
debugging, 46
with Firebug CSS inspector, 46
with Firebug Layout inspector, 47-52
with Firebug Style inspector, 47
defining in HTML elements, 99-100
design properties
applying, 111-116
backgrounds, 111
borders, 117-121
color, 103-105
cursor, 121
getting and setting, 257-258
opacity, 121-122
text styles, 106-110
visibility, 122
graphical equalizer display, 385-389
layout properties, 122
box model, 122-123
content size, 123
element flow, 124-125
getting and setting, 257-258
laying out web page components with, 127-130
margins, 124
overflow, 126
padding, 123
positioning, 125-126
z-index, 126, 277-282
loading from file, 98
overview, 97
preparing for dynamic design, 130
selectors, 102-104
sparkline graphics, 389-392
syntax, 100-102
css files, 544
CSS inspector (Firebug), 46
.css() method, 188, 258, 277
ctrlKey property (events), 225
culture option (spinner widget), 533
currentTarget property (events), 225
cursor option
CSS (Cascading Style Sheets), 121
draggable widget, 495
sortable widget, 513

custom events, 249
in JavaScript, 249-250
in jQuery, 250
custom Google searches, 439-443
custom widgets, 537-538
custom-defined objects
adding methods to, 177-178
defining objects, 177
prototyping object patterns, 178-179
sample project, 180-181

D

data attributes
mobile web pages, 545
Twitter controls, 444-445
.data() method, 331
data property (events), 225
:data() selector, 465
data types
AJAX response data types, 399
JavaScript, 142-143
data-add-back-btn attribute, 546
data-align attribute, 445
data-close-btn attribute, 546
data-collapsed attribute, 546
data-collapsed-icon attribute, 546
data-corners attribute, 600
data-count attribute, 444
data-counturl attribute, 444
data-direction attribute, 546
data-hashtags attribute, 444
data-icon attribute, 546, 600
data-iconpos attribute, 546, 600
data-lang attribute, 444-445
data-mini attribute, 546, 600
data-position attribute, 555-556
data-rel attribute, 546
data-related attribute, 444
data-role attribute, 546, 600
data-show-count attribute, 445
data-show-screen-name attribute, 445
data-size attribute, 444-445
data-text attribute, 444
data-theme attribute, 546, 600
data-title attribute, 546
data-transition attribute, 546
data-type attribute, 600
data-url attribute, 444
data-via attribute, 444
data-width attribute, 445
Date object, 174-175
date rule (form validation), 354
dateFormat option (datepicker widget), 526
dateISO rule (form validation), 354
datepicker widget, 525-526
dblclick event, 228
debugging
CSS (Cascading Style Sheets), 46
with Firebug CSS inspector, 46
with Firebug Layout inspector, 47-52
with Firebug Style inspector, 47
HTML elements, 40
Firebug DOM inspector, 44-45
with Firebug HTML inspector, 40-44
JavaScript, 35-39, 53-59
jQuery, 59
network traffic analysis, 59-62
overview, 35
deferred objects, 252
.delay() method, 304
delay option (mouse interaction widget), 494
delay timers, 296
delaying animations, 304
delegateTarget property (events), 225
design properties (CSS)
applying, 111-116
backgrounds, 111
borders, 117-121
color, 103-105
cursor, 121
opacity, 121-122
text styles, 106-110
visibility, 122
destroy() method, 494
.detach() method, 269
development environment
characteristics, 17-18
development web server, 24-25
IDEs, 18-20
dialogs
adding to mobile web pages, 571-576
dialog widget, 527-528
overlay dialogs, 381-385

digits rule (form validation), 354
directory structure, 26-27
disable() method, 494
disabled attribute (form elements), 84
disabling
form elements, 339
mobile forms, 600
dispatchEvent() method, 241
displayCookies() function, 292-293
displayTime() function, 298
distance option (mouse interaction widget), 494
dividers, 586-587
do/while loops, 150
<!DOCTYPE> element, 66-67
Document Object Model. *See* DOM (Document Object Model)
DOM (Document Object Model), 6-7
accessing, 137-141
example, 140-141
with jQuery selectors, 139
with traditional JavaScript, 139
editing with Firebug DOM inspector, 44-45
event handlers, 236-241
events
adding in JavaScript, 232-236
assigning in HTML, 231-232
filtering DOM elements in jQuery objects, 206-207
objects
accessing, 189-192, 201-203
attributes, 187
converting to/from jQuery, 188-189
determining whether an object is DOM or jQuery, 188
finding by ID, 189
methods, 187
overview, 185-187
traversing with jQuery objects, 207-209
DOM inspector (Firebug), 44-45
.done() method, 405, 422
downloading
jQuery Mobile library, 543-544
jQuery UI, 459
drag event, 496
drag-and-drop widgets
apply to web pages, 501-503
draggable widget, 495-498
droppable widget, 499-503
draggable widget, 495-498
dragstart event, 496
dragstop event, 496
drop effect, 476
drop event, 500
dropactivate event, 500
dropout event, 500
dropover event, 500
droppable widget, 499-503
dynamic menus, 314-316
dynamic scripts, writing, 30-32

E

E method, 176
.each() method, 211-217
easing
applying to class transitions, 482-484
easing functions, 477-478
easing option (.animate() method), 303
editing DOM (Document Object Model), 44-45
.effect() method, 478-482
effects, 475
adding
to animations, 488-490
to class transitions, 482-484
to element visibility transitions, 485-487
to jQuery objects, 478-482
easing functions, 477-478
table of, 476-477
element() method, 356
elements. *See* specific elements
<ellipse> element, 87
email rule (form validation), 354
embedded timelines, 446-447
embedded tweets, 445-446
.empty() method, 269
.end() method, 212
.eq() method, 207
equalizer display, 385-389
equalTo rule (form validation), 354
error event, 228

error handling
finally keyword, 158
throwing errors, 157-158
try/catch blocks, 157
errorPlacement attribute (Validation object), 358
escape codes, 164
event handlers, 136-137, 230-231
adding in JavaScript, 232-236
applying in jQuery, 236-241
assigning in HTML, 231-232
event option (tabs widget), 534
eventPhase property (events), 225
events
browser events, 7
callbacks, 251
callback mechanism, 251-252
deferred objects, 252
custom events, 249
in JavaScript, 249-250
in jQuery, 250
draggable widget, 496
droppable widget, 499
event handlers, 230-231
adding in JavaScript, 232-236
applying in jQuery, 236-241
assigning in HTML, 231-232
event objects, 225-227
event process, 223-224
event types, 227-229
global event handlers, 419
for initialization
JavaScript onload event, 229-230
jQuery initialization code, 230
Map object, 434
overview, 223
resizable widget, 505
selectable widget, 509
sortable widget, 506-512
triggering manually
in JavaScript, 241-245
with jQuery, 246-249
exp() method, 176
explode effect, 476
external links, controlling, 290

F

Facebook social elements, adding, 425-426
comment fields, 428
Facebook API, 426-427
Like button, 427
sample project, 430-432
Send button, 428
fade animation, 311
fade effect, 476
fade() function, 274
.fadeIn() method, 309
.fadeOut() method, 309
.fadeTo() method, 310
.fadeToggle() method, 309
.fail() method, 405, 422
fb-like class, 427
fb-send class, 428
<fieldset> element, 83
file inputs, 330-331
files
css files, 544
js files, 544
loading CSS styles from, 98
naming, 27
.filter() method, 207
filter option (selectable widget), 508
filterColumn() function, 372
filtered jQuery selectors, 193-199
filters in tables, 371-377
finally keyword, 158
.find() method, 209
finding DOM objects
by class name, 189
by ID, 189
sample project, 191-192
by tag name, 189-190
Firebug
CSS inspector, 46
DOM inspector, 44-45
HTML inspector, 40-44
installing, 21-22
JavaScript console, 35-39
JavaScript debugger, 53-59
Layout inspector, 47-52
Style inspector, 47
traffic analyzer, 59-62
fireEvent() method, 243
.first() method, 207
fixed headers/footers, 555-556
Flickr images, adding, 451-456
flip() function, 279, 281
floor() method, 176

flow control, 338
controlling submit and reset, 340
disabling elements, 339
focusing and blurring form elements, 339
hiding and showing elements, 339
sample project, 341-345
focus event, 228
focus() method, 288, 339, 464
:focusable() selector, 465
focusin event, 228
focusing and blurring form elements, 339
focusout event, 228
fold effect, 476
folders, Images, 544
Follow button, 445
font property (CSS), 106
footers, fixed, 555-556
for loops, 151, 169
for/in loops, 151-152, 169
form elements, 83-84
form jQuery selectors, 193-198
form() method, 356
<form> element, 83
forms
adding to web pages, 83-86
autocomplete widget, 523-524
flow control, 338
controlling submit and reset, 340
disabling elements, 339
focusing and blurring form elements, 339
hiding and showing elements, 339
sample project, 341-345
form elements, 83-84, 326
animations, 346-351
attributes, 326-327
button inputs, 330
check box inputs, 328
disabling, 339
file inputs, 330-331
focusing and blurring, 339
hidden inputs, 331
hiding and showing, 339
radio inputs, 328-329
sample project, 334-338
select inputs, 329-330
text input elements, 327-328
mobile forms, 599
buttons, 603
data attributes, 599
disabling, 600
labels, 600
radio and check box groups, 608
refreshing, 601
select menus, 610-612
sliders, 604-608
submitting, 601
text elements, 601
toggle switches, 604-608
overview, 325
serializing form data, 332-333
validating, 351
jQuery validation plug-in, 352
manually, 351-352
sample project, 359-363
simple jQuery validation with HTML, 352-354
validation messages, 356-358
validation rules, 354
fromCharCode() method, 165
functions. *See also* specific functions
defining, 153, 155-156
passing variables to, 153
returning values from, 154

G

generating tables, 84
geometric shapes, 87
.get() method, 401
GET requests, 11, 398-399
getAllResponseHeaders() method, 422
getAttribute() method, 187
getCenter() method, 434
getCookie() function, 291, 293
getElementById() function, 139, 189
getElementsByClass() method, 189
getElementsByTagName() method, 189-190
.getJSON() method, 401, 451
getMapTypeId() method, 434
getRandomArray() function, 390

getResponseHeader() method, 422
.getScript() method, 401
getTilt() method, 434
getTrip() method, 418
getZoom() method, 434
ghost option (resizable widget), 504
global event handlers, 419
Globalize jQuery plug-in, 533
Google accounts, creating, 439
Google Maps, adding, 432-439
Google search, adding, 439-443
graphical equalizer display, 385-389
graphics. *See also* animations
adding with <img> element, 78-79
canvas, 91-93
fade animation to implement image selection effect, 311
Flickr images, adding, 451-456
geometric shapes, 87
image gallery, 365-370
paths, 88-91
scalable vector graphics
adding to web pages, 87
<svg> element, 87
sparkline graphics, 389-392
greedy option (droppable widget), 499
grid layout, 581-585
groups_pool feed, 451

H

handles option (resizable widget), 504
.has() method, 207
hash property (location object), 285
<head> element, 66, 68, 99
header option (accordian widget), 522
headers
fixed headers, 555-556
HTTP headers, 10-11
headers attribute (table elements), 81
height() method, 188, 259
height property (screen object), 285
helper option
draggable widget, 495
resizable widget, 503
sortable widget, 513
hidden inputs, 331
hide and show animations, 306
.hide() method, 188, 271-272, 305, 339, 485
hiding
form elements, 339
labels, 600
hierarchy jQuery selectors, 193
highlight effect, 476
history object, 289
host property (location object), 285
hostname property (location object), 285
hoverClass option (droppable widget), 499
href property (location object), 285
HTML (Hypertext Markup Language). *See also* CSS (Cascading Style Sheets); HTML5
adding, 29, 579-580
debugging
with Firebug DOM inspector, 44-45
with Firebug HTML inspector, 40-44
elements
<a>, 78
attributes, 73
<audio>, 94
block versus inline elements, 73-75
<body>, 66
<button>, 83
<caption>, 80-83
<col>, 80-83
<colgroup>, 80-83
components of, 67
container elements, 75-78
defining CSS styles in, 99-100
<!DOCTYPE>, 66-67
<fieldset>, 83
<form>, 83
<head>, 66, 68, 99
<img>, 78-79
<input>, 83
<label>, 83
<legend>, 83

, 79
<link>, 72
<meta>, 69-70
<noscript>, 71-72
, 79
<option>, 83
<script>, 70-71, 136
<select>, 83
<style>, 70
<table>, 80
<tbody>, 80-83
<td>, 80-83
<textarea>, 83
<tfoot>, 80-83
<th>, 80-83
<thead>, 80-83
<title>, 68-69
<tr>, 80-83
, 79
event handers, assigning, 231-232
overview, 8, 65-66
structure, 66-68
XML/HTML response data, handling, 412-414

.html() method, 188, 267, 269

HTML5, 8
<canvas>, 91-93
<ellipse>, 87
<path>, 88-91
<polygon>, 87
<svg>, 87
<video>, 94

HTTP (Hypertext Transfer Protocol)
GET requests, 11
headers, 10-11
overview, 10
POST requests, 11

Hypertext Markup Language. *See* HTML (Hypertext Markup Language)

Hypertext Transfer Protocol. *See* HTTP (Hypertext Transfer Protocol)

I

id attribute, 73, 187

IDEs, installing, 18-20

IDs
finding DOM objects by, 189
unique IDs, 463-464

if operator, 146

image elements, 78-79

image gallery, 365-370

images. *See* graphics

images folder, 544

<img> element, 78-79

inc() function, 529

indexOf() method, 165-166, 170-171

initialization
jQuery initialization code, 230
page load events, 229-230

inline elements, 73-75

innerHeight() method, 259

innerHeight property (window object), 286

innerHTML attribute (DOM objects), 187

innerWidth() method, 259

innerWidth property (window object), 286

<input> element, 83

installing
Aptana Studio, 18-19
development web server, 24-25
Firebug, 21-22
IDEs, 18-20
XAAMP stack, 24-25

instances of objects, creating, 161

interactive tables with sorting and filters, 371-377

interation widgets. *See* widgets

Internet Explorer developer tools, 22-23

interrupting loops, 152-153

.is() method, 212, 328

isDefaultPrevented() method, 227

isImmediatePropagationStopped() method, 227

isNaN() function, 163

isPropagationStopped() method, 227

items option
sortable widget, 513
tooltips widget, 535

iterating through arrays, 169

J

JavaScript
accessing DOM with, 137-141
adding to web pages, 136-138
browser values, accessing, 260-266

cookies, 291-294
data types, 142-143
debugging
with JavaScript console, 35-39
with JavaScript debugger, 53-59
overview, 35
development environment
browser development tools, 21-22
characteristics, 17-18
development web server, 24-25
IDEs, 18-20
error handling
error handlers, 232-236
finally keyword, 158
throwing errors, 157-158
try/catch blocks, 157
events
custom events, 249-250
event objects, 225-227
event process, 223-224
event types, 227-229
triggering manually, 241-245
executing with event handlers, 136-137
external links, controlling, 290
forms. *See* forms
functions
defining, 153, 155-156
passing variables to, 153
returning values from, 154
image gallery, adding, 365-370
implementing AJAX from, 399-401
interactive tables with sorting and filters, 371-377
loops, 149-150
do/while loops, 150
for loops, 151, 169
for/in loops, 151-152, 169
interrupting, 152-153
while loops, 150
methods. *See* individual methods
objects. *See* objects
operators, 143
arithmetic operators, 143
assignment operators, 144
comparison operators, 145-149
conditional logic, 148-149
overlay dialogs, 381-385
overview, 14
page elements
accessing, 262-266
adding, 267
class name, 260
classes, changing, 270
CSS properties, 257-258
manipulating dynamically, 273-276
mouse position, 255-256
position, 259-260
rearranging dynamically, 277-282
removing, 268-269
size, 258
values, 256-257
visibility, 271-272
page load events for initialization, 229-230
pop-ups, 294-296
confirmation pop-ups, 295
notification pop-ups, 295
prompts, 295-296
simple example, 14
sparkline graphics, 389-392
timers, 296
delay timers, 296
reoccurring timers, 296
sample project, 298-299
tree views, 377-381
Twitter JavaScript API library, 443-444
variables
creating, 141-142
passing to functions, 153
scope, 156-157
JavaScript console (Chrome), 22
JavaScript debugger (Firebug), 53-59
join() method, 170, 171
jQuery. *See also* jQuery Mobile; jQuery UI
accessing DOM with, 137-141
adding to web pages, 136-138
animations
animation queues, 302
CSS settings, animating, 301-303
delaying, 304
moving elements, 318-322
overview, 301
.promise() method, 305
resize animations, 316-318

show and hide animations, 305-308
sliding animation, 312-316
stopping, 302-304
browser values, accessing, 260-266
debugging, 35, 55
development environment
browser development tools, 21-22
characteristics, 17-18
development web server, 24-25
IDEs, 18-20
event handlers, applying, 236-241
events
custom events, 250
event objects, 225-227
event process, 223-224
event types, 227-229
triggering manually, 246-249
forms. *See* forms
graphical equalizer display, 385-389
image gallery, 365-370
implementing AJAX from, 401-404
initialization code, 230
interactive tables with sorting and filters, 371-377
loading, 135-136
methods. *See* individual methods
objects, 213-217
adding effects to, 478-482
chaining jQuery object operations, 205-206
converting to/from DOM, 188-189
deferred objects, 252
determining whether an object is DOM or jQuery, 188
filtering jQuery object results, 206-207
manipulating DOM elements with, 211-217
overview, 186-188
traversing DOM with, 207-209, 217-220
overlay dialogs, 381-385
overview, 14
page elements
accessing, 262-266
adding, 267-268
class name, 260
classes, changing, 270
CSS properties, 257-258
inserting, 270
manipulating dynamically, 273-276
mouse position, 255-256
position, 259-260
rearranging dynamically, 277-282
removing, 268-269
replacing, 269-270
size, 258
values, 257
visibility, 271-272
selectors
attribute selectors, 194-195
basic selectors, 193-194
content selectors, 195-196
filtered selectors, 198-199
form selectors, 197-198
hierarchy selectors, 196
overview, 193
sample project, 201-203
visibility selectors, 198
sparkline graphics, 389-392
tree views, 377-381
jQuery Mobile
advantages of, 543
data attributes, 545
default settings, 548
downloading jQuery Mobile library, 543-544
events, 547
methods. *See* individual methods
mobile forms, 599
buttons, 603
data attributes, 599
disabling, 600
labels, 600
radio and check box groups, 608
refreshing, 601
select menus, 610-612
sliders, 604-608
submitting, 601
text elements, 601
toggle switches, 604-608
mobile web pages, 553
basic HTML, adding, 579-580
building, 549-551, 564-567
bypassing AJAX, 562

challenges of, 541-542
changing with jQuery code, 557-559
collapsible blocks and sets, 590-592
dialogs, 571-576
fixed headers/footers, 555-556
grid layout, 581-585
linking, 559-562
listviews, 585-590
navbars, 567-571
navigation buttons, 556-557
page anatomy, 553-554
page transitions, 562
panels, 592
pop-ups, 594
size, 542
tables, 595-597
ThemeRoller, 544
viewport meta tag, 548

jQuery UI
adding to projects, 461-463
buttons, 524-525
downloading, 459
effects, 475
adding to animations, 488-490
adding to class transitions, 482-484
adding to element visibility transitions, 485-487
adding to jQuery objects, 478-482
easing functions, 477-478
table of, 476-477
methods. *See* individual methods
overview, 459
positioning elements with, 468-472
selectors
:data(), 465
:focusable(), 465
:tabbable(), 465
applying, 467-468
ThemeRoller, 460-461
unique IDs, 463-464
widgets, 493, 521
accordian widget, 522
attribute values, 522
autocomplete widget, 523-524
custom widgets, 537-538
datepicker widget, 525-526
dialog widget, 527-528
draggable widget, 495-498
droppable widget, 499-503
jQuery.widget factory, 493-494
menu widget, 528-529
methods and events, 493-494
mouse interaction widget, 494
options, 521-522
progress bar widget, 529-530
resizable widget, 503-507
selectable widget, 508-512
slider widget, 530-532
sortable widget, 512-518
spinner widget, 532-533
tabs widget, 533-535
tooltips widget, 535-537

jQuery.widget factory, 493-494
jqXHR object, 421-422
js files, 544
JSON response data, handling, 408-411
JSONP (JSON with Padding), 398

K

keydown event, 228
keypress event, 228, 352
keyup event, 228
keywords. *See* specific keywords

L

<label> element, 83
labels, 600
.last() method, 207
lastIndexOf() method, 165, 170
Layout inspector (Firebug), 47-52
layout properties, 122
box model, 122-123
content size, 123
element flow, 124-125
laying out web page components with, 127-130
margins, 124
overflow, 126
padding, 123

positioning, 125-126
z-index, 126, 277-282
layouts, grid, 581-585
<legend> element, 83
letter-spacing property (CSS), 106
** element, 79**
Like button, adding, 427
line-height property (CSS), 107
lineTo() function, 92
<link> element, 72
linking mobile web pages, 559-562
links
adding with <a> element, 78
external links, controlling, 290
link elements, 78
lists, 585-590
adding to mobile web pages, 588-590
adding to web pages, 79
basic lists, 585
dividers, 586-587
list elements, 79
nested lists, 586
searchable lists, 587
split-button lists, 586
LN10() method, 176
load event, 228
.load() method, 230, 401
loading
CSS styles, 98
jQuery library, 135-136
.loadPage() method, 557
local hash, 559-556
location object, 285
login requests, handling with AJAX, 405-408
loops, 149-150
do/while loops, 150
for loops, 151, 169
for/in loops, 151-152, 169
interrupting, 152-153
while loops, 150
low-level AJAX requests, handling, 420-422

M

manually validating forms, 351-352
Map() function, 433-434
.map() method, 212-217
Map object, 434
mapOptions object, 433-434
Maps (Google), adding, 432-439
mapTypeId attribute (mapOptions), 433
maptypeid_changed event, 434
margins, adding with CSS, 124
match() method, 165
Math object, 175-176
max() method, 176
max option (slider widget), 531
max rule (form validation), 355
maxlength rule (form validation), 355
media elements, 94
memory flag (callbacks), 251
menu widget, 528-529
menus
dynamic menus, 314-316
menu widget, 528-529
in mobile forms, 610-612
messages, validation, 356-358
<meta> element, 69-70
metaKey property (events), 225
methods. *See also* individual methods
accessing, 162
adding to JavaScript objects, 177-178
assigning to objects, 162
min() method, 176
min rule (form validation), 354
minlength rule (form validation), 355
mobile forms, 599. *See also* mobile web pages
buttons, 603
data attributes, 599
disabling, 600
labels, 600
radio and check box groups, 608
refreshing, 601
select menus, 610-612
sliders, 604-608
submitting, 601
text elements, 601
toggle switches, 604-608
mobile web pages. *See also* jQuery Mobile; mobile forms
basic HTML, adding, 579-580
building, 549-551, 564-567
bypassing AJAX, 562
challenges of, 541-542

changing with jQuery code, 557-559
collapsible blocks and sets, 590-592
dialogs, 571-576
fixed headers/footers, 555-556
grid layout, 581-585
linking, 559-562
listviews, 585-590
adding, 588-590
basic lists, 585
dividers, 586-587
nested lists, 586
searchable lists, 587
split-button lists, 586
navbars, 567-571
navigation buttons, 556-557
back button, 557
creating, 556
positioning, 557
overview, 541, 553
page anatomy, 553-554
page transitions, 562
panels, 592
pop-ups, 594
size, 542
tables, 595-597
mobileinit handler, 548, 611
modal option (dialog widget), 527
mouse cursor types, 121
mouse interaction widget, 494
mouse position, getting, 255-256
mousedown event, 228
mouseenter event, 228
mouseleave event, 228
mousemove event, 228
mouseout event, 228
mouseover event, 228
mouseup event, 228
move() function, 278, 281
moveBy() method, 288
moveTo() method, 92, 288
moving elements, 318-322
element position changes on nonstatic elements, 319
element position changes on static elements, 319
paper airplane app, 319-322
my option (.position() method), 469

N

name attribute
form elements, 84
window object, 287
naming files, 27
navbars, 567-571
navigation buttons, 556-557
back button, 557
creating, 556
positioning, 557
nested lists, 586
network traffic analysis, 59-62
.next() method, 209
.nextAll() method, 209
.nextUntil() method, 209
<noscript> element, 71-72
.not() method, 207
notification pop-ups, 295
null data type, 143
number data type, 142
Number object, 163
number rule (form validation), 355
numberFormat option (spinner widget), 533
numberOfInvalids() method, 356
numberofMonths option (datepicker widget), 526

O

objects, 143
Array, 166-174
adding/removing items, 171
checking whether array contains an item, 171
combining, 169
converting to strings, 171
iterating through, 169
methods, 169-170
overview, 166-169
sample project, 173-174
browser
history object, 289
location object, 285
converting type of, 188-189
creating
instances, 161
JavaScript objects, 177, 180-181
Date, 174-175
deferred objects, 252
determining type of, 188

DOM objects
accessing, 189-192, 201-203
attributes, 187
finding by ID, 189
methods, 187
overview, 185-187
event objects, 225-227
jQuery objects
adding effects to, 478-482
chaining jQuery object operations, 205-206
manipulating DOM elements with, 213-217
methods, 188, 207, 209, 213-217
overview, 186-188
traversing DOM with, 206-209, 217-220
jqXHR, 421-422
Map, 434
mapOptions, 433-434
Math, 175-176
methods, 162
adding, 177-178
assigning, 162
Number, 163
overview, 161
properties, 162
prototyping object patterns, 178-179
RegExp, 175
screen, 285-286
String, 164-166
combining, 164-166
converting arrays to, 171
escape codes, 164
methods, 164-165
replacing words in, 166
sample project, 167-166
searching for substrings, 166
splitting into arrays, 166
Validator, 356-357
window, 285
methods, 285-288
properties, 285-287
XMLHttpRequest, 399
of option (.position() method), 469
off() method, 236-237
.offsetParent() method, 209
** element, 79**
on() method, 236-237
once flag (callbacks), 251
onload event, 229-230
onreadystatechange event handler, 400
onSelect option (datepicker widget), 525
opacity property
CSS (Cascading Style Sheets), 121-122
draggable widget, 495
sortable widget, 513
open() method, 288, 400
opener property (window object), 286
operators, 143
arithmetic operators, 143
assignment operators, 144
comparison operators
if, 146
switch, 147
table of, 145
conditional logic, 148-149
option() method, 494
<option> element, 83
orientation option (slider widget), 531
outerHeight() method, 259
outerHeight property (window object), 286
outerHTML attribute (DOM objects), 187
outerWidth() method, 259
outerWidth property (window object), 287
overflow property (CSS), 126
overlay dialogs, 381-385

P

padding, applying with CSS, 123
page elements. *See also* HTML (Hypertext Markup Language)
accessing, 262-266
adding
in JavaScript, 267
in jQuery, 267-268
classes, changing, 271
className attribute, 260
collapsible elements, 590-592
CSS properties, setting, 257-258
element flow, 124-125
element visibility transitions, 485-487
graphical equalizer display, 385-389

image gallery, 365-370
inserting, 270
manipulating dynamically, 273-276
mouse position, getting, 255-256
overlay dialogs, 381-385
position
getting and setting, 258-260
positioning with jQuery UI, 468-472
rearranging dynamically, 277-282
removing, 268-269
replacing, 269-270
resizing, 258, 503-507
sparkline graphics, 389-392
tables
adding to mobile web pages, 595-597
adding to web pages, 79-83
generating, 84
interactive tables with sorting and filters, 371-377
tree views, 377-381
values
getting/setting in JavaScript, 256-257
getting/setting in jQuery, 257
visibility, 271-272
page requests, 395-396
page transitions in mobile web pages, 562
pageXOffset property (window object), 287
pageYOffset property (window object), 287
panBy() method, 434
panels, 592
panTo() method, 434
paper airplane app, 319-322
.parent() method, 209
parent property (window object), 287
parentNode attribute (DOM objects), 187
.parents() method, 210
.parentsUntil() method, 210
passing variables to functions, 153
<path> element, 88-91
pathname property (location object), 285
paths, 88-91
photos_public feed, 451
PI() method, 176
pixelDepth property (screen object), 286
placeholder option (sortable widget), 513
placing validation messages, 357-358
plug-ins, jQuery validation, 352
<polygon> element, 87
pop() method, 170
.popup() method, 594
pop-ups, 294-296, 594
confirmation pop-ups, 295
notification pop-ups, 295
prompts, 295-296
port property (location object), 285
.position() method, 258, 468
position option (tooltips widget), 536
positioning
HTML elements from CSS, 125-126
navigation buttons, 557
page elements, 258-260, 468-472
.post() method, 401
POST requests, 11, 398-399
pow() method, 176
.prev() method, 210
.prevAll() method, 210
.preventDefault() method, 227, 340
.prevUntil() method, 210
print() method, 288
progress bar widget, 529-530
projects
creating, 28
CSS (Cascading Style Sheets), adding, 27-30
directory structure, 26-27
dynamic scripts, writing, 30-32
file naming, 27
.promise() method, 305
prompt() method, 288, 295-296
prompts, 295-296
.prop() method, 327
properties (CSS)
design properties
applying, 111-116
backgrounds, 111

borders, 117-121
color, 103-105
cursor, 121
opacity, 121-122
text styles, 106-110
visibility, 122
getting and setting, 257-258
layout properties, 122
box model, 122-123
content size, 123
element flow, 124-125
margins, 124
overflow, 126
padding, 123
positioning, 125-126
z-index, 126
z-index, 277-282
properties (object)
accessing, 162
location object, 285
screen object, 285-286
window object, 285-287
protocol property (location object), 285
prototyping object patterns, 178-179
puff effect, 476
pulsate effect, 476
push() method, 170

Q

querySelectorAll() method, 207
queue option (.animate() method), 303
queues, animation, 302

R

radio buttons in mobile forms, 608
randInt() function, 372
range option (slider widget), 531
range rule (form validation), 355
rangelength rule (form validation), 355
.ready() method, 230
readyState attribute (jqXHR object), 422
rearranging page elements, 277-282
reflow mode (tables), 596
refreshing forms, 601
RegExp object, 175
relatedTarget property (events), 225
reload() method, 285
remote rule (form validation), 355
.remove() method, 269
removeAttr() method, 328
.removeClass() method, 271, 482
removeElement() method, 268
removeEventListener() function, 233
.removeUniqueId() method, 464
removing
items from arrays, 171
page elements, 268-269
unique IDs, 463-464
renderSpark() function, 390
reoccurring timers, 296
replace() method, 165-166, 285
.replaceAll() method, 270
.replaceWith() method, 270
replacing
page elements, 269-270
words in strings, 166
requests
cross-domain requests, 397-398
GET requests, 11, 398-399
login requests, 405-408
low-level AJAX requests, 420-422
POST requests, 11, 398-399
request handling (AJAX), 397
sending from jQuery, 402-404
required rule (form validation), 355
reset event, 228, 340
.reset() method, 340
resetForm() method, 356
resizable elements, 506-507
resizable widget, 503-507
resize animations, 316-318
resize event, 229, 504
resize() function, 278, 281
resizeBy() method, 288
resizestart event, 505
resizestop event, 505
resizeTo() method, 288
resizing elements, 503-507
.resolve() method, 252
response attribute (XMLHttpRequest object), 400
response data, handling
AJAX, 405, 408-414

JSON response data, 408-411
XML/HTML response data, 412-414
response data types, 399
responseText attribute (XMLHttpRequest object), 400
results property (events), 225
return keyword, 154
reverse() method, 170
revert option (draggable widget), 495
round() method, 176
rowspan attribute (table elements), 81
rules, validation, 354

S

scalable vector graphics
adding to web pages, 87
canvas, 91-93
geometric shapes, 87
paths, 88-91
scale effect, 477
scope of variables, 156-157
screen object, 285-286
screenX property
events, 225
window object, 287
screenY property
events, 225
window object, 287
<script> element, 70-71, 136
scripts
client-side scripts, 14
debugging
CSS (Cascading Style Sheets), 46-52
HTML elements, 40-45
JavaScript, 53-59
with JavaScript console, 35-39
jQuery, 59
network traffic analysis, 59-62
overview, 35
dynamic scripts, writing, 30-32
overview, 12
server-side scripts, 12-13
scroll event, 229
scroll option (sortable widget), 513
scrollBy() method, 288
.scrollParent() method, 464
scrollTo() method, 288
search (Google), adding to web pages, 439-443
search() method, 165
search property (location object), 285
searchable lists, 587
searching strings, 166
select event, 229
select inputs, 329-330
select menus in mobile forms, 610-612
<select> element, 83
selectable sets, 510-512
selectable widget, 508-512
selectableselected event, 508
selectableselecting event, 508
selectablestart event, 508
selectablestop event, 509
selectableunselect event, 509
selectableunselected event, 509
selectors
CSS (Cascading Style Sheets), 102-104
jQuery
:data(), 465
:focusable(), 465
:tabbable(), 465
accessing DOM with, 139
applying, 467-468
attribute selectors, 194-195
basic selectors, 193-194
content selectors, 195-196
filtered selectors, 198-199
form selectors, 197-198
hierarchy selectors, 196
overview, 193
sample project, 201-203
visibility selectors, 198
self property (window object), 287
Send button, 428
send() method, 400
sending AJAX requests from jQuery, 402-404
sendRating() method, 418
.serialize() method, 332
.serializeArray() method, 333
serializing form data, 332-333
servers. *See* web servers

server-side data, accessing with AJAX
asynchronous communication, 397
compared to page requests, 395-396
cross-domain requests, 397-398
GET requests, 398-399
global event handlers, 419
global setup, 396-419
from JavaScript, 399-401
from jQuery, 401-404
login requests, handling, 405-408
low-level AJAX requests, handling, 420-422
overview, 395
POST requests, 398-399
request handling, 397
response data, handling, 405, 408-414
JSON response data, 408-411
XML/HTML response data, handling, 412-414
response data types, 399
server data, updating, 416-419
server-side scripts, 12-13
SET-COOKIE header, 11
setAttribute() method, 187
setCenter() method, 434
setCookie() function, 291, 293
setDoc() function, 274
setDocNav() function, 273
setImages() function, 454
setInterval() method, 288, 296
setList() method, 418
setMapTypeId() method, 434
setRequestHandler attribute (XMLHttpRequest object), 400
setRequestHeader() method, 422
setTimeout() method, 288, 296, 529
setTitle() method, 434
setTrip() method, 418
setZoom() method, 434
shake effect, 477
shift() method, 170
shiftKey property (events), 225
show and hide animations, 306
.show() method, 188, 271-272, 305-306, 339, 485
showButtonPanel option (datepicker widget), 526
showErrors() method, 356
showOn option (datepicker widget), 525
.siblings() method, 210
sin() method, 176
size
of mobile web pages, 542
of page elements, 258
size effect, 477
.slice() method, 165, 170, 207
slide effect, 477
slide option (slider widget), 531
.slideDown() method, 312
sliders
in mobile forms, 604-608
slider bars, 530-532
slider widget, 530-532
slider-based image gallery, 365-370
.slideToggle() method, 312
.slideUp() method, 312
sliding animation, 312-316
dynamic menu sample project, 314-316
.slideDown() method, 312
.slideToggle() method, 312
.slideUp() method, 312
width and height, 312
social media
Facebook social elements, 425-426
comment fields, 428
Facebook API, 426-427
Like button, 427
sample project, 430-432
Send button, 428
Flickr images, 451-456
Google Maps, 432-439
Google search, 439-443
Twitter controls, 443
embedded timelines, 446-447
embedded tweets, 445-446
Follow button, 445
sample project, 449-451
Tweet button, 444
sort event, 514
sort() method, 170
sortable elements, 515-518
sortable widget, 512-518
events, 506-512
options, 513

sortable elements, 515-518
Twitter controls, adding, 443-444
sortactivate event, 514
sortbeforeStop event, 514
sortchange event, 514
sortColumn() function, 373
sorting in tables, 371-377
sortout event, 514
sortover event, 514
sortreceive event, 514
sortremove event, 514
sortstart event, 514
sortstop event, 514
sortupdate event, 514
sparkline graphics, 389-392
specialEasing option (.animate() method), 303
spinner widget, 532-533
spinners, 532-533
splice() method, 170
split() method, 165
split-button lists, 586
splitting strings into arrays, 166
sqrRoot() function, 158
sqrt() method, 176
stack() function, 279, 281
stack option (draggable widget), 495
statements. *See* specific statements
status attribute
jqXHR object, 422
XMLHttpRequest object, 399
statusText attribute (jqXHR object), 422
step option
.animate() method, 302
spinner widget, 533
.stop() method, 302-304
stopImmediatePropagation() method, 227
stopOnFalse flag (callbacks), 251
stopping animations, 302-304
stopPropagation() method, 227
string data type, 142
String object, 164-166
combining, 164-166
converting arrays to, 171
escape codes, 164
methods, 164-165
replacing words in, 166
sample project, 166-167
searching for substrings, 166
splitting into arrays, 166
style attribute, 73, 187
Style inspector (Firebug), 47
<style> element, 70
styles. *See* CSS (Cascading Style Sheets)
stylized dialogs, 527-528
stylized menus, 528-529
submission (forms), 340
submit event, 229
submitting forms, 601
substr() method, 165
substring() method, 165
substrings, searching for, 166
SVC graphics. *See* scalable vector graphics
switch operator, 147
.switchClass() method, 482

T

:tabbable() selector, 465
tabbed panels, 533-535
table elements, 79-83
<table> element, 80
tables
adding to mobile web pages, 595-597
adding to web pages, 79-83
generating, 84
interactive tables with sorting and filters, 371-377
<table> element, 80
tabs widget, 533-535
tag name, finding DOM objects by, 189-190
target property (events), 225
<tbody> element, 80-83
<td> element, 80-83
text elements in mobile forms, 601
text input elements, 327-328
text styles, applying with CSS, 106-110
text-align property (CSS), 106
<textarea> element, 83
text-decoration property (CSS), 107
text-indent property (CSS), 107
text-overflow property (CSS), 107
text-transform property (CSS), 107
<tfoot> element, 80-83
<th> element, 80-83
<thead>, 80-83
ThemeRoller, 460-461, 544
throw statement, 157-158

throwing errors, 157-158
tile() function, 279, 281
tilt_changed event, 434
timelines, embedded, 446-447
timers, 296
delay timers, 296
reoccurring timers, 296
sample project, 298-299
timeStamp property (events), 225
<title> element, 68-69
toExponential() method, 163
toFixed() method, 163
.toggle() method, 306, 485
toggle switches in mobile forms, 604-608
.toggleClass() method, 271, 482
toggleItem() function, 378-379
tolerance option
droppable widget, 499
selectable widget, 508
sortable widget, 513
toLowerCase() method, 165
.tooltip() method, 535
tooltips widget, 535-537
top property (window object), 287
toPrecision() method, 163
toString() method, 163, 170
toUpperCase() method, 165
<tr> elements, 80-83
traffic analyzer (Firebug), 59-62
transfer effect, 477
transitions
class transitions, 482-484
mobile web page transitions, 562
traversing DOM with jQuery objects, 207-209, 217-220
tree views, 377-381
trigger() method, 246
triggering events manually
in JavaScript, 241-245
with jQuery, 246-249
try/catch blocks, 157
Tweet button, 444
tweets. *See* Twitter controls, adding
Twitter controls, adding, 443
embedded timelines, 446-447
embedded tweets, 445-446
Follow button, 445
sample project, 449-451
Tweet button, 444
Twitter JavaScript API library, 443-444
Twitter JavaScript API library, 443-444
twitter-timeline class, 447
twitter-tweet class, 446
type property (events), 225

U

.ui-dialog-contain class, 572
.ui-progressbar-value class, 529
.ui-selecting class, 508
.ui-sortable-helper class, 513
** element, 79**
Uniform Resource Locators. *See* URLs (Uniform Resource Locators)
unique flag (callbacks), 251
unique IDs, 463-464
.uniqueId() method, 463-464
unload event, 229
unshift() method, 170
updateAddr() function, 341
updateEqualizer() function, 386-388
updateImages() function, 454-455
updating server data with AJAX, 415-419
url rule (form validation), 355
URLs (Uniform Resource Locators), 7
using option (.position() method), 467

V

.val() method, 185, 327
.validate() method, 352
validating forms, 351
jQuery validation plug-in, 352
manually, 351-352
sample project, 359-363
simple jQuery validation with HTML, 352-354
validation messages, 356-358
validation rules, 354
Validator object, 356-357
value attribute
DOM objects, 187
form elements, 84
value option (slider widget), 531
valueOf() method, 163, 165, 170

values
assigning to objects, 162
getting and setting
in JavaScript, 256-257
in jQuery, 257
returning from functions, 154
var keyword, 141
variables
creating, 141-142
passing to functions, 153
scope, 156-157
vector graphics. *See* scalable vector graphics
<video> element, 94
viewport meta tag, 548
views, tree, 377-381
visibility
animations
fade animation to implement image selection effect, 311
.fadeIn() method, 309
.fadeOut() method, 309
.fadeTo() method, 310
.fadeToggle() method, 309
toggling, 271-272
visibility jQuery selectors, 193
visibility property (CSS), 122
visibility transitions, 485-487

W

web browsers. *See* browsers
web development projects. *See* projects
web forms. *See* forms
web servers
development web server, installing, 24-25
overview, 6
server data, updating with AJAX, 415-419
XAAMP stack, installing, 24-25
which property (events), 225
while loops, 150
widget() method, 494
widgets, 493, 521
accordian widget, 522
attribute values, 522
autocomplete widget, 523-524
custom widgets, 537-538
datepicker widget, 525-526
dialog widget, 527-528
draggable widget, 495-498
droppable widget, 499-503
jQuery.widget factory, 493-494
menu widget, 528-529
methods and events, 493-494
mouse interaction widget, 494
options, 521
progress bar widget, 529-530
resizable widget, 503-507
selectable widget, 508-512
slider widget, 530-532
sortable widget, 512-518
events, 506-512
options, 513
sortable elements, implementing, 515-518
spinner widget, 532-533
tabs widget, 533-535
tooltips widget, 535-537
width() method, 188, 259
width property (screen object), 286
window object, 285
methods, 285-288
properties, 285-287
windows (browser), 7
within option (.position() method), 467
word-spacing property (CSS), 107
writing dynamic scripts, 30-32

X-Y-Z

XAAMP stack, installing, 24-25
XML/HTML response data, handling, 412-414
XMLHttpRequest object, 399

.zIndex() method, 464
z-index property, 277
CSS (Cascading Style Sheets), 126
draggable widget, 495
sortable widget, 513
zoom attribute (mapObject), 433
zoom_changed event, 434