

Brad Dayley
Brendan Dayley

Sams **Teach Yourself**

AngularJS, JavaScript, and jQuery

All
in **One**

SAMS

FREE SAMPLE CHAPTER



SHARE WITH OTHERS

Brad Dayley
Brendan Dayley

Sams **Teach Yourself**
AngularJS,
JavaScript, and
jQuery All in One

in **24**
Hours

SAMS

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

Sams Teach Yourself AngularJS, JavaScript, and jQuery All in One

Copyright © 2016 by Pearson Education, Inc.

All rights reserved. No part of this book shall be reproduced, stored in a retrieval system, or transmitted by any means, electronic, mechanical, photocopying, recording, or otherwise, without written permission from the publisher. No patent liability is assumed with respect to the use of the information contained herein. Although every precaution has been taken in the preparation of this book, the publisher and author assume no responsibility for errors or omissions. Nor is any liability assumed for damages resulting from the use of the information contained herein.

ISBN-13: 978-0-672-33742-0

ISBN-10: 0-672-33742-8

Library of Congress Control Number: 2015907445

Printed in the United States of America

First Printing August 2015

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 authors 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

Kristy Hart

Project Editor

Andy Beaster

Copy Editor

Barbara Hacha

Indexer

Brad Herriman

Proofreader

Sarah Kearns

Technical Editor

Jesse Smith

Publishing Coordinator

Vanessa Evans

Interior Designer

Gary Adair

Cover Designer

Mark Shirar

Compositor

Nonie Ratcliff

Contents at a Glance

Introduction	1
--------------------	---

Part I: Introduction to AngularJS, jQuery, and JavaScript Development

LESSON 1 Introduction to Dynamic Web Programming	9
LESSON 2 Debugging JavaScript in Web Pages	35
LESSON 3 Understanding Dynamic Web Page Anatomy	69
LESSON 4 Adding CSS/CSS3 Styles to Allow Dynamic Design and Layout	105
LESSON 5 Jumping into jQuery and JavaScript Syntax	145
LESSON 6 Understanding and Using JavaScript Objects	173

Part II: Implementing jQuery and JavaScript in Web Pages

LESSON 7 Accessing DOM Elements Using JavaScript and jQuery Objects	197
LESSON 8 Navigating and Manipulating jQuery Objects and DOM Elements with jQuery	217
LESSON 9 Applying JavaScript and jQuery Events for Richly Interactive Web Pages	235
LESSON 10 Dynamically Accessing and Manipulating Web Pages with JavaScript and jQuery	269
LESSON 11 Working with Window, Browser, and Other Non-Web Page Elements	303

Part III: Building Richly Interactive Web Pages with jQuery

LESSON 12 Enhancing User Interaction Through jQuery Animation and Other Special Effects	321
LESSON 13 Interacting with Web Forms in jQuery and JavaScript	347
LESSON 14 Creating Advanced Web Page Elements in jQuery	391
LESSON 15 Accessing Server-Side Data via JavaScript and jQuery AJAX Requests	423

Part IV: Utilizing jQuery UI

LESSON 16	Introducing jQuery UI	457
LESSON 17	Using jQuery UI Effects	475
LESSON 18	Advanced Interactions Using jQuery UI Interaction Widgets	495
LESSON 19	Using jQuery UI Widgets to Add Rich Interactions to Web Pages	525

Part V: Building Web Applications with AngularJS

LESSON 20	Getting Started with AngularJS	547
LESSON 21	Understanding AngularJS Application Dynamics	567
LESSON 22	Implementing the Scope as a Data Model	583
LESSON 23	Using AngularJS Templates to Create Views	599
LESSON 24	Implementing Directives in AngularJS Views	627
LESSON 25	Creating Your Own Custom Directives to Extend HTML	657
LESSON 26	Using Events to Interact with Data in the Model	685
LESSON 27	Implementing AngularJS Services in Web Applications	699
LESSON 28	Creating Your Own Custom AngularJS Services	731
LESSON 29	Creating Rich Web Application Components the AngularJS Way	751
	Index	777

Table of Contents

Introduction	1
Lesson 1: Introduction to Dynamic Web Programming	9
Understanding the Web Server/Browser Paradigm	9
Setting Up a Web Development Environment	21
Summary	32
Q&A	33
Workshop	33
Lesson 2: Debugging JavaScript in Web Pages	35
Viewing the Developer Tools Console	35
Debugging HTML Elements	41
Debugging CSS	48
Debugging JavaScript	56
Analyzing the Network Traffic	63
Summary	65
Q&A	66
Workshop	66
Lesson 3: Understanding Dynamic Web Page Anatomy	69
Using HTML/HTML5 Elements to Build a Dynamic Web Page	69
Understanding HTML Structure	70
Implementing HTML Head Elements	72
Adding HTML Body Elements	77
Adding Some Advanced HTML5 Elements	93
Summary	103
Q&A	103
Workshop	103

Lesson 4: Adding CSS/CSS3 Styles to Allow Dynamic Design and Layout	105
Adding CSS Styles to the Web Page	105
Adding CSS Styles to HTML Elements	108
Preparing CSS Styles for Dynamic Design	140
Summary	141
Q&A	142
Workshop	142
Lesson 5: Jumping into jQuery and JavaScript Syntax	145
Adding jQuery and JavaScript to a Web Page	145
Accessing the DOM	148
Understanding JavaScript Syntax	151
Summary	169
Q&A	170
Workshop	170
Lesson 6: Understanding and Using JavaScript Objects	173
Using Object Syntax	173
Understanding Built-in Objects	175
Creating Custom-Defined Objects	189
Summary	195
Q&A	195
Workshop	195
Lesson 7: Accessing DOM Elements Using JavaScript and jQuery Objects	197
Understanding DOM Objects Versus jQuery Objects	197
Accessing DOM Objects from JavaScript	201
Using jQuery Selectors	205
Summary	215
Q&A	215
Workshop	216
Lesson 8: Navigating and Manipulating jQuery Objects and DOM Elements with jQuery	217
Chaining jQuery Object Operations	217
Filtering the jQuery Object Results	218

Traversing the DOM Using jQuery Objects	219
Looking at Some Additional jQuery Object Methods	223
Summary	232
Q&A	233
Workshop	233
Lesson 9: Applying JavaScript and jQuery Events for Richly Interactive Web Pages	235
Understanding Events	235
Using the Page Load Events for Initialization	241
Adding and Removing Event Handlers to DOM Elements	242
Triggering Events Manually	253
Creating Custom Events	262
Implementing Callbacks	263
Summary	265
Q&A	266
Workshop	266
Lesson 10: Dynamically Accessing and Manipulating Web Pages with JavaScript and jQuery	269
Accessing Browser and Page Element Values	270
Dynamically Manipulating Page Elements	282
Dynamically Rearranging Elements on the Web Page	292
Summary	299
Q&A	299
Workshop	300
Lesson 11: Working with Window, Browser, and Other Non-Web Page Elements	303
Understanding the Screen Object	303
Using the Window Object	304
Using the Browser Location Object	306
Using the Browser History Object	307
Controlling External Links	308
Adding Pop-up Boxes	313

Setting Timers	314
Summary	318
Q&A	318
Workshop	318
Lesson 12: Enhancing User Interaction Through jQuery Animation and Other Special Effects	321
Understanding jQuery Animation	321
Animating Show and Hide	325
Animating Visibility	329
Sliding Elements	332
Creating Resize Animations	337
Implementing Moving Elements	340
Summary	344
Q&A	344
Workshop	345
Lesson 13: Interacting with Web Forms in jQuery and JavaScript	347
Accessing Form Elements	348
Intelligent Form Flow Control	361
Dynamically Controlling Form Element Appearance and Behavior	368
Validating a Form	375
Summary	387
Q&A	388
Workshop	388
Lesson 14: Creating Advanced Web Page Elements in jQuery	391
Adding an Image Gallery	391
Implementing Tables with Sorting and Filters	397
Creating a Tree View	404
Using Overlay Dialogs	409
Implementing a Graphical Equalizer Display	413
Adding Sparkline Graphics	417
Summary	421
Q&A	421
Workshop	422

Lesson 15: Accessing Server-Side Data via JavaScript and jQuery	
AJAX Requests	423
Making AJAX Easy	423
Implementing AJAX	428
Using Advanced jQuery AJAX	450
Summary	453
Q&A	454
Workshop	454
Lesson 16: Introducing jQuery UI	457
Getting Started with jQuery UI	457
Applying jQuery UI in Your Scripts	463
Summary	472
Q&A	472
Workshop	472
Lesson 17: Using jQuery UI Effects	475
Applying jQuery UI Effects	475
Adding Effects to Class Transitions	482
Adding Effects to Element Visibility Transitions	485
Summary	492
Q&A	492
Workshop	492
Lesson 18: Advanced Interactions Using jQuery UI Interaction Widgets	495
Introducing jQuery UI Interactions	495
Using the Drag-and-Drop Widgets	497
Resizing Elements Using the Resizable Widget	507
Applying the Selectable Widget	511
Sorting Elements with the Sortable Widget	516
Summary	522
Q&A	522
Workshop	523

Lesson 19: Using jQuery UI Widgets to Add Rich Interactions to Web Pages	525
Reviewing Widgets	525
Adding an Expandable Accordion Element	526
Implementing Autocomplete in Form Elements	527
Applying jQuery UI Buttons to Form Controls	528
Creating a Calendar Input	530
Generating Stylized Dialogs with jQuery UI	532
Implementing Stylized Menus	533
Creating Progress Bars	535
Implementing Slider Bars	536
Adding a Value Spinner Element	538
Creating Tabbed Panels	539
Adding Tooltips to Page Elements	542
Creating Custom Widgets	544
Summary	545
Q&A	545
Workshop	545
Lesson 20: Getting Started with AngularJS	547
Why AngularJS?	548
Understanding AngularJS	549
An Overview of the AngularJS Life Cycle	552
Separation of Responsibilities	553
Integrating AngularJS with Existing JavaScript and jQuery	553
Adding AngularJS to Your Environment	554
Bootstrapping AngularJS in an HTML Document	555
Using the Global APIs	555
Using jQuery or jQuery Lite in AngularJS Applications	560
Summary	564
Q&A	564
Workshop	565

Lesson 21: Understanding AngularJS Application Dynamics	567
Looking at Modules and Dependency Injection	567
Defining an AngularJS Module Object	569
Creating Providers in AngularJS Modules	570
Implementing Providers and Dependency Injection	572
Applying Configuration and Run Blocks to Modules	577
Summary	581
Q&A	581
Workshop	582
Lesson 22: Implementing the Scope as a Data Model	583
Understanding Scopes	583
Implementing Scope Hierarchy	593
Summary	597
Q&A	597
Workshop	597
Lesson 23: Using AngularJS Templates to Create Views	599
Understanding Templates	599
Using Expressions	600
Using Filters	611
Creating Custom Filters	620
Summary	623
Q&A	624
Workshop	624
Lesson 24: Implementing Directives in AngularJS Views	627
Understanding Directives	627
Using Built-In Directives	628
Summary	655
Q&A	656
Workshop	656

Lesson 25: Creating Your Own Custom Directives to Extend HTML	657
Understanding Custom Directive Definitions	657
Implementing Custom Directives	668
Summary	683
Q&A	683
Workshop	683
Lesson 26: Using Events to Interact with Data in the Model	685
Browser Events	685
User Interaction Events	686
Adding <code>\$watches</code> to Track Scope Change Events	686
Emitting and Broadcasting Custom Events	691
Summary	696
Q&A	697
Workshop	697
Lesson 27: Implementing AngularJS Services in Web Applications	699
Understanding AngularJS Services	699
Using the Built-In Services	700
Using the <code>\$q</code> Service to Provide Deferred Responses	727
Summary	728
Q&A	729
Workshop	729
Lesson 28: Creating Your Own Custom AngularJS Services	731
Understanding Custom AngularJS Services	731
Integrating Custom Services into Your AngularJS Applications	733
Summary	748
Q&A	748
Workshop	748
Lesson 29: Creating Rich Web Application Components the AngularJS Way	751
Summary	774
Q&A	774
Workshop	774
Index	777

About the Authors

Brad Dayley is a senior software engineer with more than 20 years of experience developing enterprise applications and web interfaces. He has used JavaScript, jQuery, and AngularJS to develop a wide array of feature-rich web applications. He has a passion for new technologies, especially ones that really make a difference in the software industry. He is the author of *Node.js*, *MongoDB*, and *AngularJS Web Development*, *Learning AngularJS*, *jQuery*, and *JavaScript Phrasebook*, and *Sams Teach Yourself jQuery and JavaScript in 24 Hours*.

Brendan Dayley is a university student majoring in computer science. He is an avid web application developer who loves learning and implementing the latest and greatest technologies. He recently attended Dev-Mountain's Immersive Web Development program, specializing in web application development and AngularJS in particular. He has written a number of web applications using JavaScript, jQuery, and AngularJS and is excited about the future of these technologies.

Dedication

For D!

A & F

Jessie

My one and only

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, Jesse Smith for keeping me honest with his technical review, Barbara Hacha for turning the technical ramblings of my brain into a fine text, and Andy Beaster 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@sampublishing.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

Welcome to *AngularJS, JavaScript, and jQuery All in One*. This book is designed to jumpstart you into the world of dynamic web application development using JavaScript, jQuery, and AngularJS. The book covers the basics of the JavaScript language, jQuery library, and AngularJS framework and how to use them to build well-designed, reusable components for web applications.

With billions of people using the Internet today, there is a rapidly growing trend to replace traditional websites, where one page links to another page and so on, with single page applications that have richly interactive elements.

The main reason 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 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 interface and back-end services.

As you complete the lessons in this book, you will gain a practical understanding of how to incorporate JavaScript with the powerful jQuery library as well as the exciting AngularJS framework 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 lesson 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 AngularJS, jQuery, and JavaScript.

Who Should Read This Book

This book is aimed at readers who already have an understanding of the basics of HTML and have done some programming in a modern programming language. Having an understanding of JavaScript will make this book easier to digest, but it is not required because the basics of JavaScript are covered.

Why You Should Read This Book

This book will teach you how to create powerful, interactive web applications that have a well-structured, easy-to-reuse code base that will be easy to maintain. The typical readers of this book want to learn JavaScript, jQuery, and AngularJS for the purpose of building highly interactive web applications. The typical reader will also want to leverage the innovative Model View Controller (MVC) approach of AngularJS to extend HTML and implement well-designed and structured web pages and web applications.

What You Will Learn from This Book

Reading this book will enable you to build rich, dynamic interactions into your web pages and applications. Websites are no longer simple static content that consist of HTML pages with integrated images and formatted text. Instead, websites have become much more dynamic, with a single page providing a wide array of functionality and interactions.

Using AngularJS, jQuery, and JavaScript enables you to build logic directly into your web applications that allows you to interact with the user from your client-side application. These technologies also allow you to interact with back-end web services on the web server to create a comprehensive client-side web application. The following are a few of the things you will learn while reading this book:

- ▶ The basics of the JavaScript language
- ▶ How to implement JavaScript, jQuery, and AngularJS in your web pages
- ▶ How to dynamically modify page elements in the browser
- ▶ How to use browser events to interact with the user directly
- ▶ How to implement client-side services that can interact with the web server
- ▶ How to implement rich user interface (UI) components, such as zoomable images and expandable lists
- ▶ How to quickly build AngularJS templates with built-in directives that enhance the user experience

- ▶ How to bind UI elements to the data model so that when the model changes, the UI changes, and vice versa
- ▶ How to bind mouse and keyboard events directly to the data model and back-end functionality to provide robust user interactions
- ▶ How to define your own custom AngularJS directives that extend the HTML language
- ▶ How to build dynamic browser views that provide rich user interaction
- ▶ How to create custom services that can be easily reused in other AngularJS applications

Why AngularJS, jQuery, and JavaScript in the Same Book?

The reason we decided to put AngularJS, jQuery, and JavaScript in the same book is that they all relate to each other. We've been asked questions like "Should I use AngularJS or jQuery?" or "Should I use JavaScript or jQuery?" many times. We see them as a stack that works together very well.

JavaScript is the base language that is supported by the browser. jQuery extends JavaScript with a syntax that is much more powerful and user friendly. AngularJS is an extension of jQuery (or at least a stripped-down version of jQuery) that provides an extremely powerful MVC framework with robust data binding functionality.

Understanding all three of these technologies and how they work together will make you a better web developer, even if you use another JavaScript framework or library to develop, because they provide the fundamental functionality that all good web applications need. You may decide that simple JavaScript fits the needs in one area, or jQuery/jQueryUI provides the perfect functionality for some web forms, or that you need the robust functionality of AngularJS for your web application. Either way, you will have the skills and understanding to be able to choose and implement the right technology.

What Is JavaScript?

JavaScript is a programming language much like any other. What separates JavaScript the most from other programming languages is that the browser has a built-in interpreter that can parse and execute the language. That means you can write complex applications that have direct access to the browser events and Document Object Model (DOM) objects.

Access to the DOM means that you can add, modify, or remove elements from a web page without reloading it. Access to the browser gives you access to events such as mouse movements and

clicks. This is what gives JavaScript the capability to provide functionality such as dynamic lists and drag and drop.

What Is jQuery?

jQuery is a library that is built on JavaScript. The underlying code is JavaScript; however, jQuery simplifies a lot of the JavaScript code into simple-to-use functionality. The two main advantages to using jQuery are selectors and built-in functions.

Selectors provide quick access to specific elements on the web page, such as a list or table. They also provide access to groups of elements, such as all paragraphs or all paragraphs of a certain class. This allows you to quickly and easily access specific DOM elements.

jQuery also provides a rich set of built-in functionality that makes it easy to do a lot more with a lot less code. For example, tasks such as hiding an element on the screen or animating the resizing of an element take just one line of code.

What Is AngularJS?

AngularJS is a client-side framework developed by Google. It is written in JavaScript with a reduced jQuery library called jQuery lite. The entire ideology behind AngularJS is to provide a framework that makes it easy to implement well-designed and well-structured web pages and applications using an MVC framework.

AngularJS provides all that functionality to handle user input in the browser, manipulate data on the client side, and control how elements are displayed in the browser view. Here are some of the benefits AngularJS provides:

- ▶ **Data Binding**—AngularJS has a very clean method to bind data to HTML elements using its powerful scope mechanism.
- ▶ **Extensibility**—The AngularJS architecture enables you to easily extend almost every aspect of the language to provide your own custom implementations.
- ▶ **Clean**—AngularJS forces you to write clean, logical code.
- ▶ **Reusable Code**—The combination of extensibility and clean code makes it very easy to write reusable code in AngularJS. In fact, the language often forces you to do so when you're creating custom services.
- ▶ **Support**—Google is investing a lot in this project, which gives it an advantage where other similar initiatives have failed.

- ▶ **Compatibility**—AngularJS is based on JavaScript and has a close relationship with jQuery. That makes it easier to begin integrating AngularJS into your environment and reuse pieces of your existing code within the structure of the AngularJS framework.

Beyond AngularJS, 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 29 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 and jQueryUI 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 libraries that provide rich UI components for 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

Many 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 the AngularJS, jQuery, and JavaScript code that provide interactions between 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 titles for the listing blocks include a filename of the file that contains the source. These files can be obtained from the book's website (follow the directions on the back cover of this book).

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 lesson have you expand on the examples.

Development Web Server

I chose Node.js with Express as the web server for the development environment for this book. You will get a chance to set up Node.js as the web server in Lesson 1. There are several reasons I chose Node.js over a more traditional web server like Apache or IIS, including the following:

- ▶ Node.js is extremely easy to install and set up.
- ▶ You can use Node.js to test your JavaScript snippets without having to use a web browser.
- ▶ There is a great Node.js plug-in for Eclipse that allows you to easily debug JavaScript.
- ▶ You do not need to understand a back-end scripting language such as PHP, Python, or Ruby because you can write your server-side script for Node.js in JavaScript.

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 lesson 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 lesson. Finally, one or two optional exercises give you a chance to practice your new skills before you move on.

Finally

We hope you enjoy this book and enjoy learning about JavaScript, jQuery, and AngularJS as much we did. These are great, innovative technologies that are really fun to use. Soon you'll be able to join the many other web developers who use them to build rich, dynamic, and interactive websites and web applications.

This page intentionally left blank

LESSON 1

Introduction to Dynamic Web Programming

What You'll Learn in This Lesson:

- ▶ Getting ready for creating dynamic web pages
- ▶ Creating an AngularJS, jQuery, and JavaScript-friendly development environment
- ▶ Adding JavaScript and jQuery to web pages
- ▶ Constructing web pages to support jQuery and JavaScript
- ▶ Creating your first dynamic web pages with jQuery and JavaScript

JavaScript and its amped-up companions, jQuery and AngularJS, have completely changed the game when it comes to creating rich interactive web pages and web-based applications. JavaScript has long been a critical component for creating dynamic web pages. Now, with the advancements in the jQuery and AngularJS libraries, web development has changed forever.

This lesson quickly takes you through the world of jQuery and JavaScript development. The best place to start is to ensure that you understand the dynamic web development playground that you will be playing in. To be effective in JavaScript and jQuery, you need a fairly decent understanding of web server and web browser interaction, as well as HTML and CSS.

This lesson includes several sections that briefly give a high-level overview of web server and browser interactions and the technologies that are involved. The rest of this lesson is dedicated to setting up and configuring an AngularJS, jQuery, and JavaScript friendly development environment. You end with writing your very first web pages that include JavaScript and jQuery code.

Understanding the Web Server/Browser Paradigm

JavaScript, jQuery, and AngularJS can interact with every major component involved in communication between the web server and the browser. To help you understand that interaction better, this section provides a high-level overview of the concepts and technologies involved in

web server/browser communication. This is not intended to be comprehensive by any means; it's simply a high-level overview that enables you to put things into the correct context as they are discussed later in the book.

Looking at Web Server to Browser Communication Terms

The World Wide Web's basic concept should be very familiar to you: An address is typed into or clicked in a web browser, and information is loaded and displayed in a form ready to be used. The browser sends a request, the server sends a response, and the browser displays it to the user.

Although the concept is simple, several steps must take place for the data to be requested from the server and displayed in the browser. The following sections define the components involved, their interactions with each other, and how JavaScript, jQuery, and AngularJS are involved.

Web Server

The web server is the most critical component of the web. Without it, no data would be available at all. The web server responds to requests from browsers by sending data that the browsers then use or display. A lot of things happen on the web server, though. For example, the web server and its components check the format and validity of requests. They may also check for security to verify that the request is from an allowed user. To build the response, the server may interact with several components and even other remote servers to obtain the data necessary.

Browser

The next most important component is the browser. The browser sends requests to the web server and then displays the results for the user. The browser also has a lot of things happening under the hood. The browser has to parse the response from the server and then determine how to represent that to the user.

Although several browsers are available, the three most popular are Chrome, Internet Explorer, and Firefox. For the most part, each browser behaves the same when displaying web pages; however, occasionally some differences exist, and you will need to carefully test your JavaScript, jQuery, and AngularJS scripts in each of the major browsers that you are required to support.

JavaScript, jQuery, and AngularJS can be very involved in the interactions that occur between the browser receiving the response and the final output rendered for the user. These scripts can change the format, content, look, and behavior of the data returned from the server. The following sections describe important pieces provided by the browser.

DOM

The browser renders an HTML document into a web page by creating a Document Object Model, or DOM. The DOM is a tree structure of objects with the HTML document as the root object. The root can have several children, and those children can have several children. For example, a

web page that contains a list would have a root object, with a child list object that contained several child list element objects. The following shows an example of simple DOM tree for a web page containing a single heading and a list of three cities:

```
document
  + html
    + body
      + h1
        + text = "City List"
      + ul
        + li
          + text = "New York, US"
        + li
          + text = "Paris, FR"
        + li
          + text = "London, EN"
```

The browser knows how to display each node in the DOM and renders the web page by reading each node and drawing the appropriate pixels in the browser window. As you learn later, JavaScript, jQuery, and AngularJS enable you to interact directly with the DOM, reading each of the objects, changing those objects, and even removing and adding objects.

Browser Events

The browser tracks several events that are critical to AngularJS, jQuery, and JavaScript programs—for example, when a page is loaded, when you navigate away from a page, when the keyboard is pressed, mouse movements, and clicks. These events are available to JavaScript, allowing you to execute functionality based on which events occur and where they occur.

Browser Window

The browser also provides limited access to the browser window itself. This allows you to use JavaScript to determine the display size of the browser window and other important information that you can use to determine what your scripts will do.

URL

The browser is able to access files on the web server using a Uniform Resource Locator, or URL. A URL is a fully unique address to access data on the web server, which links the URL to a specific file or resource. The web server knows how to parse the URL to determine which file/resources to use to build the response for the browser. In some instances, you might need to use JavaScript to parse and build URLs, especially when dynamically linking to other web pages.

HTML/HTML5

Hypertext Markup Language, or HTML, provides the basic building blocks of a web page. HTML defines a set of elements representing content that is placed on the web page. These element tags

are used to create objects in the DOM. Each element tag pair is represented as an object in the DOM. Each element is enclosed in a pair of tags denoted by the following syntax:

```
<tag>content</tag>
```

For example:

```
<p>This is an HTML paragraph.</p>.
```

The web browser knows how to render the content of each of the tags in the appropriate manner. For example, the tag `<p>` is used to denote a paragraph. The actual text that is displayed on the screen is the text between the `<p>` start tag and the `</p>` end tag.

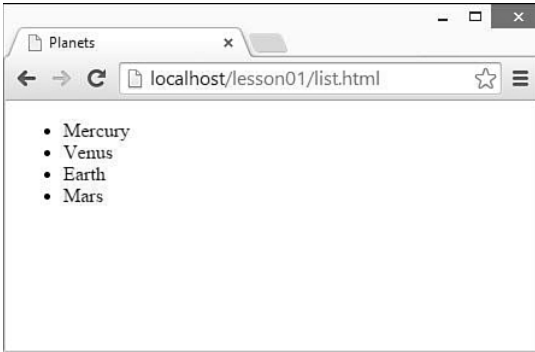
The format, look, and feel of a web page is determined by placement and type of tags that are included in the HTML file. The browser reads the tags and then renders the content to the screen as defined.

HTML5 is the next generation of the HTML language that incorporates more media elements, such as audio and video. It also provides a rich selection of vector graphic tags that allow you to draw sharp, crisp images directly onto the web page using JavaScript.

Listing 1.1 shows an example of the HTML used to build a simple web page with a list of planets. The HTML is rendered by the browser into the output shown in Figure 1.1.

LISTING 1.1 list.html A Simple HTML Document That Illustrates the HTML Code Necessary to Render a List in a Browser

```
01 <!DOCTYPE html>
02 <html>
03   <head>
04     <title>Server Side Script</title>
05     <meta charset="utf-8"/>
06   </head>
07   <body>
08     <ul>
09       <li>Mercury</li>
10       <li>Venus</li>
11       <li>Earth</li>
12       <li>Mars</li>
13     </ul>
14   </body>
15 </html>
```

**FIGURE 1.1**

List of planets rendered in a browser using the code from Listing 1.1.

CSS/CSS3

One of the challenges with web pages is getting them to look sharp and professional. The generic look and feel that browsers provide by default is functional; however, it is a far cry from the sleek and sexy eye candy that users of today's Internet have come to expect.

Cascading Style Sheets, or CSS, provide a way to easily define how the browser renders HTML elements. CSS can be used to define the layout as well as the look and feel of individual elements on a web page.

CSS3, or Cascading Style Sheets level 3, is the next generation of CSS that incorporates more special effects, such as transformations and animations. It also provides rich additions for borders, backgrounds, and text.

To illustrate CSS, we've added some CSS code to our example from Listing 1.1. Listing 1.2 uses CSS to modify several attributes of the list items, including the text alignment, font style, and changing the list bullet from a dot to a check-mark image. Notice how the CSS style changes how the list is rendered in Figure 1.2.

LISTING 1.2 style.htm HTML with Some CSS Code in `<STYLE>` Element to Alter the Appearance of the List

```
01 <!DOCTYPE html>
02 <html>
03   <head>
04     <title>Style</title>
05     <meta charset="utf-8" />
06     <style>
07       li {
08         text-align: center;
```

```
09     font-family: "Times New Roman", Times, serif;
10     font-size: 30px;
11     font-style: italic;
12     font-weight: bold;
13     list-style-image: url('/images/check.png');
14     list-style-position: inside;
15 }
16 </style>
17 </head>
18 <body>
19   <ul>
20     <li>Mercury</li>
21     <li>Venus</li>
22     <li>Earth</li>
23     <li>Mars</li>
24   </ul>
25 </body>
26 </html>
```

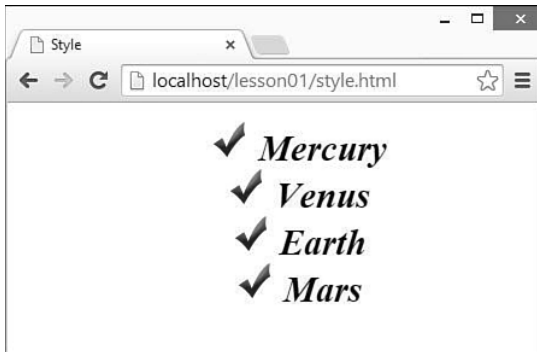


FIGURE 1.2
The CSS code dramatically changes the look of the list in the browser.

HTTP/HTTPS Protocols

Hypertext Transfer Protocol (HTTP) defines communication between the browser and the web server. It defines what types of requests can be made, as well as the format of those requests and the HTTP response.

Hypertext Transfer Protocol with Secure Sockets Layer (HTTPS) adds an additional security layer, SSL/TLS, to ensure secure connections. When a web browser connects to a web server via HTTPS, a certificate is provided to the browser. The user is then able to determine whether to accept the

certificate. Without the certificate, the web server will not respond to the user's requests, thus ensuring that the request is coming from a secured source.

The following sections discuss HTTP headers and the two most common types of HTTP request, GET and PUT.

HTTP Headers

HTTP headers allow the browser to define the behavior and format of requests made to the server and the response back to the web browser. HTTP headers are sent as part of an HTTP request and response. You can send HTTP requests to web servers from JavaScript, so you need to know a little bit about the headers required.

The web server reads the request headers and uses them to determine how to build a response for the browser. As part of the response, the web server includes response headers that tell the browser how to process the data in the response. The browser reads the headers first and uses the header values when handling the response and rendering the page.

Following are a few of the more common ones:

- ▶ **ACCEPT**—Defines content types that are acceptable in the response.
- ▶ **AUTHORIZATION**—Specifies authentication credentials used to authenticate the requesting user.
- ▶ **COOKIE**—Cookie value that was previously set in the browser by a server request. Cookies are key/value pairs that are stored on the client. They can be set via server requests or JavaScript code and are sent back to the server as part of HTTP requests from the browser.
- ▶ **SET-COOKIE**—Cookie value from the server that the browser should store if cookies are enabled.
- ▶ **CONTENT-TYPE**—Type of content contained in the response from the web server. For example, this field may be “text/plain” for text or “image/png” for a .png graphic.
- ▶ **CONTENT-LENGTH**—Amount of data that is included in the body of the request or response.

Many more headers are used in HTTP requests and responses, but the preceding list should give you a good idea of how they are used.

GET Request

The most common type of HTTP request is the GET request. The GET request is generally used to retrieve information from the web server—for example, to load a web page or retrieve images to display on a web page. The file to retrieve is specified in the URL that is typed into the browser, for example:

```
http://www.dayleycreations.com/tutorials.html
```


A GET request is composed entirely of headers with no body data. However, data can be passed to the server in a GET request using a query string. A query string is sent to the web server as part of the URL. The query string is formatted by specifying a ? character after the URL and then including a series of one or more key/value pairs separated by & characters using the following syntax:

```
URL?key=value&key=value&key=value...
```

For example, the following URL includes a query string that specifies a parameter `gallery` with a value of `01` that is sent to the server:

```
http://www.dayleycreations.com/gallery.html?gallery=01
```

POST Request

A POST request is different from a GET request in that there is no query string. Instead, any data that needs to be sent to the web server is encoded into the body of the request. POST request are generally used for requests that change the state of data on the web server. For example, a web form that adds a new user would send the information that was typed into the form to the server as part of the body of a POST.

Web Server and Client-Side Scripting

Originally, web pages were static, meaning that the file that was rendered by the browser was the exact file that was stored on the server. The problem is that when you try to build a modern website with user interactions, rich elements, and large data, the number of web pages needed to support the different static web pages is increased dramatically.

Rather than creating a web server full of static HTML files, it is better to use scripts that use data from the web server and dynamically build the HTML that is rendered in the browser.

Those scripts can run either on the server or in the client browser. The following sections discuss each of those methods. Most modern websites use a combination of server-side and client-side scripting.

Client-Side Scripting

Client-side scripting is the process of sending JavaScript code along with the web page. That code gets executed either during the loading of the web page or after the web page has been loaded.

There are a couple of great advantages of client-side scripting. One is that data processing is done on the client side, which makes it easier to scale applications with large numbers of users. Another is that browser events can often be handled locally without the need to send requests to the server. This enables you to make interfaces respond to user interaction much more quickly.

JavaScript, jQuery, and now AngularJS are by far the most common forms of client-side scripting. Throughout this book, you learn why that is the case.

Figure 1.3 diagrams the flow of data between the web server and the browser for a simple client-side script that uses JavaScript to populate an empty `` element with a list of planets. Notice that the file located on the server is the same one sent to the browser, but in the browser, the JavaScript adds `` elements for each planet. You do not need to fully understand the JavaScript code yet, just that the HTML is dynamically changed on the client and not the server.

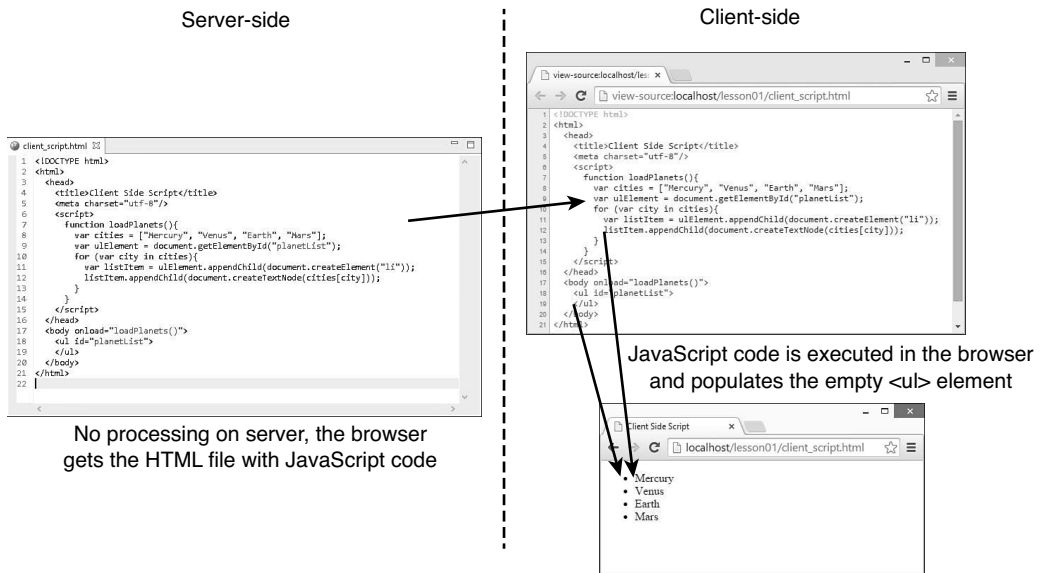


FIGURE 1.3 The JavaScript is executed in the browser, and so the HTML document rendered by the browser is different from the one that was originally sent.

Server-Side Scripting

There are two major types of server-side scripting. These are server-side templates and AJAX request handlers. Each of these methods requires that code be written on the server to either dynamically generate an HTML document before it is sent to the browser or to dynamically generate data that can be consumed by a client-side application.

Server-Side Templates

The first type is to use a PHP, .Net, Java, or other type of application that is run on the server that generates the HTML page, or at least parts of the HTML page, dynamically as they are requested by the client.

The main advantages of this type of server-side scripting is that data processing is done completely on the server side and the raw data is never transferred across the Internet; also, problems and data fix-ups can be done locally within the server processing.

The disadvantage of this type of server-side scripting is that it requires more processing on the server side, which can reduce the scalability of some applications.

Figure 1.4 illustrates using a simple Node.js application on the server that will dynamically create an HTML document that populates a list of planets. In the example in Figure 1.3, PHP code is used, and the web server's PHP engine will replace the code in the `<?php>` tag with the output generated by the PHP script.

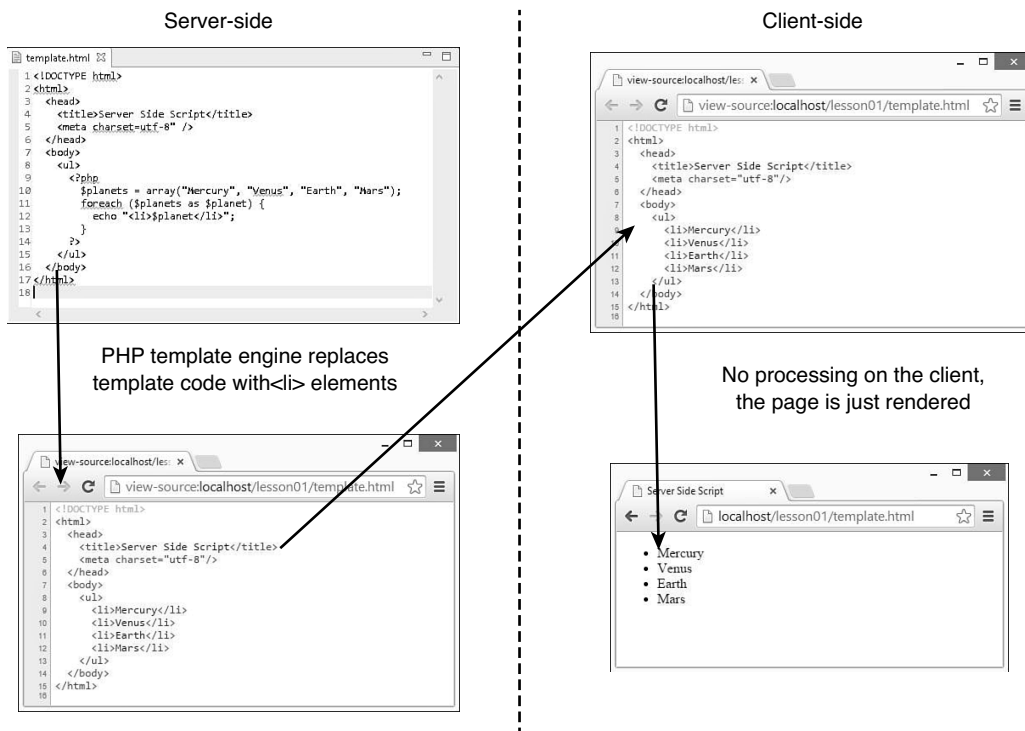


FIGURE 1.4 The PHP script is executed on the web server, and so the HTML document sent to the browser is different from what is actually contained on the server.

You don't necessarily need to understand how the code works at this point; you only need to understand that the HTML document is dynamically generated on the server and not the client.

NOTE

There are numerous methods of using HTML templates on the server-side. We do not cover those here because they are out of the scope of the book. If you would like to learn more about server-side scripting, you might investigate PHP, Ruby on Rails, and Node.js more fully.

AJAX Handlers

The second major type of server-side scripts are applications that return raw data in the form of raw JSON or XML to the browser in response to an Asynchronous JavaScript plus XML or AJAX request. AJAX requests are designed to allow JavaScript running in the browser client to get raw data from the server.

AJAX reduces the need to reload the web page or load other web pages as the user interacts. This reduces the amount of data that needs to be sent with the initial web server response and also allows web pages to be more interactive.

For a simple example of AJAX, we've constructed two scripts—Listing 1.3 and Listing 1.4. Listing 1.3 is an HTML document with JavaScript that runs on the client after the page is loaded. The JavaScript makes an AJAX request back to the server to retrieve the list of planets via a server-side script. Listing 1.4 simulates the JSON data that could be returned by the server-side script. The list of planets returned is then used to populate the HTML list element with items.

LISTING 1.3 ajax.html A Simple JavaScript Client-Side Script Executes an AJAX Request to the Server to Retrieve a List of Planets to Use When Building the HTML List Element

```

01 <!DOCTYPE html>
02 <html>
03   <head>
04     <title>AJAX</title>
05     <meta charset="utf-8" />
06     <script>
07       var xmlhttp = new XMLHttpRequest();
08       function loadPlanets(){
09         xmlhttp.open("GET", "/lesson01/data.html", false);
10         xmlhttp.send();
11         var planets = JSON.parse( xmlhttp.responseText );
12         var ulElement = document.getElementById("planetList");
13         for (var planet in planets){
14           var listItem = ulElement.appendChild(document.createElement("li"));
15           listItem.appendChild(document.createTextNode(planets[planet]));
16         }
17       }
18     </script>
19   </head>
20   <body onload="loadPlanets()">

```

```

21     <ul id="planetList">
22     </ul>
23 </body>
24 </html>
    
```

LISTING 1.4 data.html Dynamic JSON Data Generated by a Server-Side Script

```

01 [
02 "Mercury",
03 "Venus",
04 "Earth",
05 "Mars"
06 ]
    
```

Figure 1.5 illustrates the flow of communication that happens during the AJAX request/response. Notice that a second request is made to the server to retrieve the list of cities.

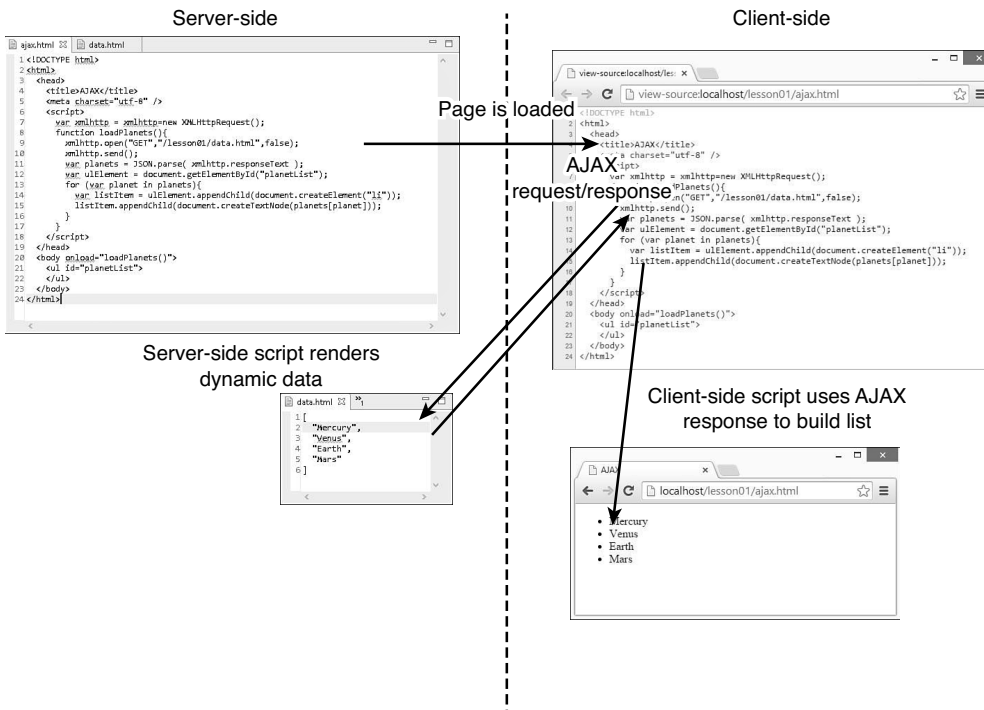


FIGURE 1.5 Using an AJAX request, JavaScript can send an additional request to the server to retrieve additional information that can be used to populate the web page.

Setting Up a Web Development Environment

With the brief introduction to dynamic web programming out of the way, it is time to cut to the chase and get your development environment ready to write jQuery and JavaScript.

The development environment can make all the difference when you are writing jQuery and JavaScript projects. The development environment should have these following components:

- ▶ **Easy to Use IDE**—The IDE provides text editors that allow you to modify your code in the simplest manner possible. Choose an IDE that you feel comfortable with and that is extensible to support HTML, CSS, JavaScript, jQuery, and AngularJS.
- ▶ **Development Web Server**—You should never develop directly on a live web server (although most of us have done it at one point or another). A test development web server is required to test out scripts and interactions.
- ▶ **Development Web Browser(s)**—Again, you should initially develop to the browser that you are most comfortable with or that will be the most commonly used.

For the purposes of this book, we have chosen to use Eclipse for the IDE and Node.js for the development web server. These technologies are very easy to set up, configure, and get going with. They also integrate well with each other and are easily extended. The following sections take you through the process of setting up Node.js and Eclipse for JavaScript development.

Setting Up Node.js

Node.js is a JavaScript platform based on Google Chrome's V8 engine that enables you to run JavaScript applications outside of a web browser. It is an extremely powerful tool, but this book covers only the basics of using it as the web server to support your web application examples.

To install and use Node.js, you need to perform the following steps:

1. Go to the following URL and click **INSTALL**. This will download an installable package to your system. For Windows boxes, you will get an .MSI file; for Macs, you will get a .PKG file; and for Linux boxes, you can get a .tar.gz file.

<http://nodejs.org>

2. Install the package. For Windows and Macs, simply install the package file. For Linux, go to the following location for instructions on installing using a package manager:

<https://github.com/joyent/node/wiki/Installing-Node.js-via-package-manager>

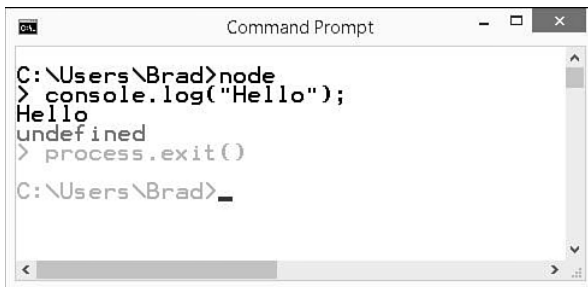
3. Open a terminal or console window.

4. Type `node` to launch the Node.js JavaScript shell, and you should see a `>` prompt. The Node.js shell provides the capability to execute JavaScript commands directly on the underlying JavaScript engine.
5. If the `node` application is not found, you need to add the path to the `node` binary directory to the `PATH` for your development system (this process is different for each different platform). The binary directory is typically `/usr/local/bin/` on Macs and Linux boxes. On Windows, the binary directory will be in the `<install>/bin` folder, where `<install>` is the location you specified during the installation process.
6. Then you get to the `>` prompt. Type the following command and verify that `Hello` is printed on the screen, as shown in Figure 1.6:

```
console.log("Hello");
```

7. Use the following command to exit the Node.js prompt:

```
process.exit();
```

A screenshot of a Windows Command Prompt window titled "Command Prompt". The window shows the following text:

```
C:\Users\Brad>node
> console.log("Hello");
Hello
undefined
> process.exit()

C:\Users\Brad>_
```

The prompt changes from `C:\Users\Brad>` to `>` after typing `node`. The output shows `Hello` on the first line, `undefined` on the second line, and the prompt returns to `C:\Users\Brad>` after typing `process.exit()`. The cursor is shown as a small horizontal line after the final prompt.

FIGURE 1.6

Starting and using the Node.js command prompt.

You have now successfully installed and configured Node.js.

Configuring Eclipse as a Web Development IDE

The IDE is the most important aspect when developing with JavaScript. An IDE integrates the various tasks required to write web applications into a single interface. In reality, you could use any text editor to write HTML, CSS, JavaScript, and jQuery code. However, you will find it much more productive and easy to use a good IDE.

We chose Eclipse for this book because it is a great general IDE that is easy to configure and set up. You can use your own IDE if you would rather; however, this might be a good chance to try a different IDE if you are unfamiliar with Eclipse.

NOTE

You will need to have a Java JRE or JDK installed to be able to install Eclipse.

Use the following steps to download, install, and configure Eclipse:

1. Install a Java JRE or JDK. For this book, we downloaded and installed the Java SE Development Kit 8 from the following location:
<http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html>
2. Download and extract Eclipse. The location you extract the Eclipse files to will be the installation location. For this book, we installed the Luna version Eclipse IDE for Java Developers from the following location:
<http://www.eclipse.org/downloads/>
3. Start Eclipse by double-clicking the Eclipse executable file.
4. After Eclipse has loaded, install the Node.js plug-in for Eclipse by selecting Help, Eclipse Marketplace from the main menu. Then type **nodeclipse** into the Find box and click Install to install the package, as shown in Figure 1.7. You will need to accept the license agreement as part of the install process.

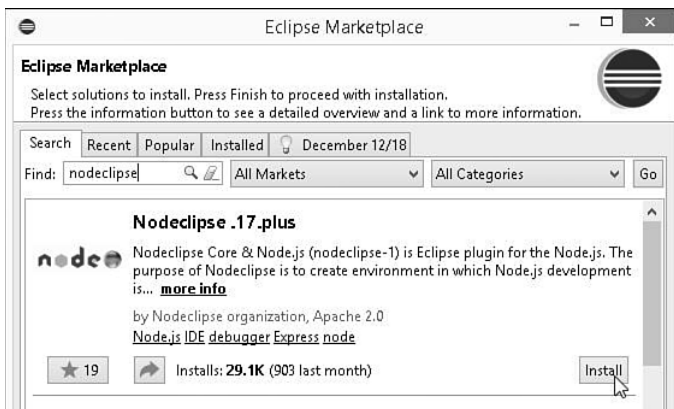


FIGURE 1.7
Installing the Nodeclipse plug-in for Eclipse.

5. After the Nodeclipse plug-in is installed, install the HTML Editor plug-in by selecting Help, Eclipse Marketplace from the main menu. Then type **html editor** into the Find box and click Install to install the package, as shown in Figure 1.8. You will need to accept the license agreement as part of the install process.

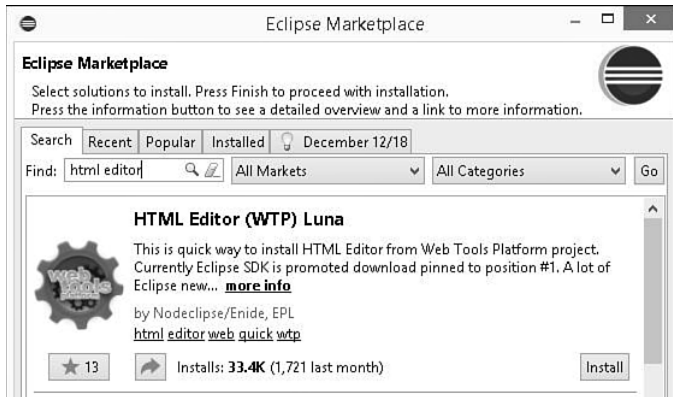


FIGURE 1.8
Installing the HTML Editor plug-in for Eclipse.

6. Restart Eclipse to enable the new plug-ins.
7. Verify that the .js extensions uses Nodeclipse and .html extensions use HTML Editor as their default editors. To do this, select Window, Preferences, and then select General, Editors, File Associations and click the file types to see the associated editors, as shown in Figure 1.9.
8. Set the path to the Node.js executable by selecting Window, Preferences and then selecting Nodeclipse in the navigation pane. The Node.js path option is toward the top of the options.
9. Create a project for this book by selecting File, New, Project to launch the New Project Wizard. Then select Node, Node.js Project. Click Next and type in the name of the project; for example, LearningJavaScript. Then click Finish to create the project.
10. Now we'll validate that things work by creating and running a JavaScript Application from Eclipse. Select the new project and then select File, New JavaScript File from the main menu. Name the file **first.js** and click Finish to create the file.
11. Type the following line of code into the file and save it:


```
console.log("Hello");
```
12. Double-click to the left of line number 2 so that a small circle appears, noting that a breakpoint has been set.
13. Select Run, Run As, Node Application. You should see the word "Hello" printed to the Console window, as shown in Figure 1.10.

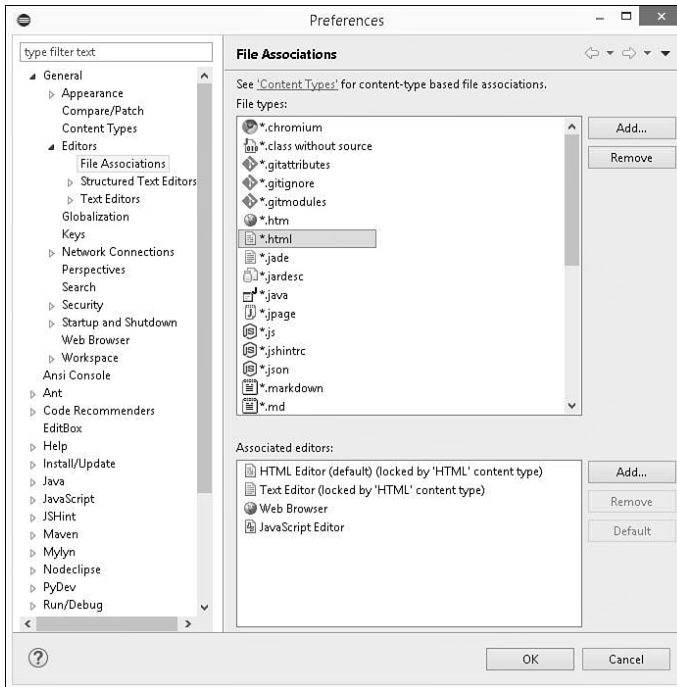


FIGURE 1.9
Setting default editors for file types in Eclipse.

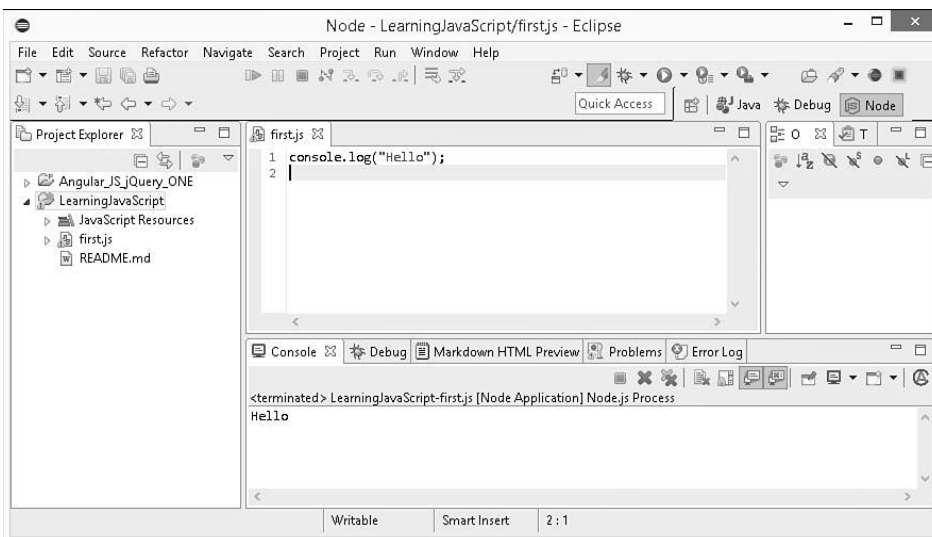


FIGURE 1.10
Running a JavaScript application in Eclipse.

Eclipse is now set up and ready for you to begin developing JavaScript.

NOTE

You can also run JavaScript applications from a console prompt by typing in the command:

```
node <path_to_JavaScript_file>
```

Creating an Express Web Server Using Node.js

Node.js is a very modular platform, meaning that Node.js itself provides a very efficient and extensible framework, and external modules are utilized for much of the needed functionality. Consequently, Node.js provides a very nice interface to add and manage these external modules.

Express is one of these modules. The Express module provides a simple-to-implement web server with a robust feature set, such as static files, routes, cookies, request parsing, and error handling.

The best way to use Node.js as the web server for your web development is to utilize the Express module. In the following exercise, you build a Node.js/Express web server and use it to serve static files.

Use the following steps to build and test a Node.js/Express web server capable of supporting static files and server-side scripting:

1. Open a console prompt and navigate to the location where you created the project folder for this book. If you don't know the path, right-click the project in Eclipse and select Properties from the menu. Then select Resource, and the full path to the project folder is shown in the Location field to the right.
2. From a console prompt in the project folder, execute the following command, as shown in Figure 1.11. This command will install the Express module version 4.6.1 for Node.js into a subfolder named `node_modules`:

```
npm install express@4.6.1
```

NOTE

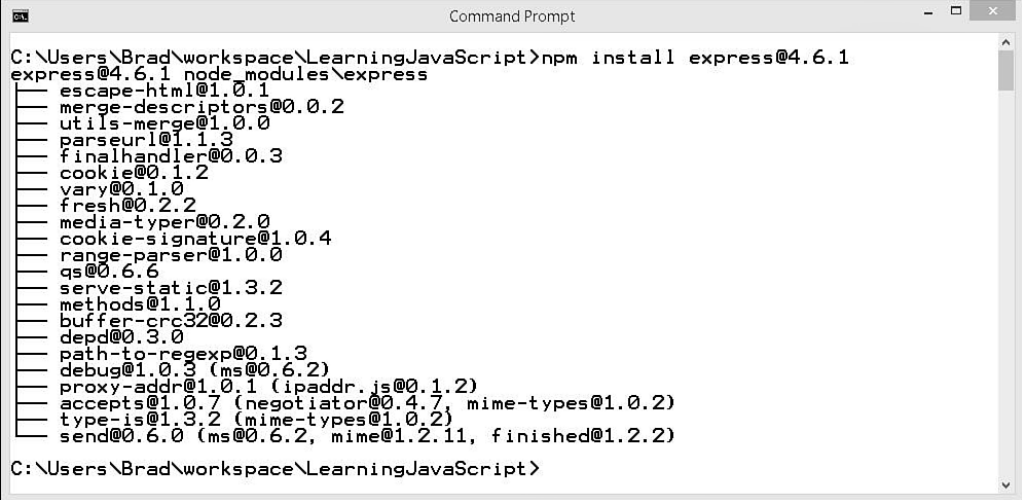
Node occasionally has an issue on some systems where it cannot automatically create a folder to store modules in. You may see an error message similar to the following. If you do, create the `npm` folder in the path specified, and the `npm install` command should work:

```
Error: ENOENT, s#tat 'C:\Users\Brad\AppData\Roaming\npm'  
node <path_to_JavaScript_file>
```

3. Execute the following command to install the `body-parser` module for Node.js. This module makes it possible to parse the query parameters and body from HTTP GET and POST

requests. This command will install the body-parser module version 1.6.5 for Node.js into a subfolder named node_modules:

```
npm install body-parser@1.6.5
```



```
Command Prompt
C:\Users\Brad\workspace\LearningJavaScript>npm install express@4.6.1
express@4.6.1 node_modules\express
├── escape-html@1.0.1
├── merge-descriptors@0.0.2
├── utils-merge@1.0.0
├── parseurl@1.1.3
├── finalhandler@0.0.3
├── cookie@0.1.2
├── vary@0.1.0
├── fresh@0.2.2
├── media-typer@0.2.0
├── cookie-signature@1.0.4
├── range-parser@1.0.0
├── qs@0.6.6
├── serve-static@1.3.2
├── methods@1.1.0
├── buffer-crc32@0.2.3
├── depd@0.3.0
├── path-to-regexp@0.1.3
├── debug@1.0.3 (ms@0.6.2)
├── proxy-addr@1.0.1 (ipaddr.js@0.1.2)
├── accepts@1.0.7 (negotiator@0.4.7, mime-types@1.0.2)
├── type-is@1.3.2 (mime-types@1.0.2)
└── send@0.6.0 (ms@0.6.2, mime@1.2.11, finished@1.2.2)
C:\Users\Brad\workspace\LearningJavaScript>
```

FIGURE 1.11

Installing the Express npm module for Node.js from a console prompt.

4. Go back to Eclipse, right-click the project, and select Refresh. You should see the node_modules folder with body-parser and express subfolders.
5. Create a file named server.js in the root of your project directory, place the contents from Listing 1.5 inside of it, and save it. This is a basic Node.js/Express web server that will service static files using the root of your project directory as the website root location.
6. Verify that your Node.js web server will run correctly. Start the web server by right-clicking the server.js file and selecting Run As, Node Application from the menu. The Console window, shown in Figure 1.12, should show that the server.js file is running and provide a red box to stop the server. If you are running multiple applications in Eclipse, you can click the Console Select button to select a specific console, as shown in Figure 1.12.
7. Hit the server from a web browser at the following address. Because the web server is servicing static files using the ./ path, the actual contents of the server.js file should be displayed in the browser:

```
localhost/server.js
```

8. Stop the web server by clicking the red box in the Console window.

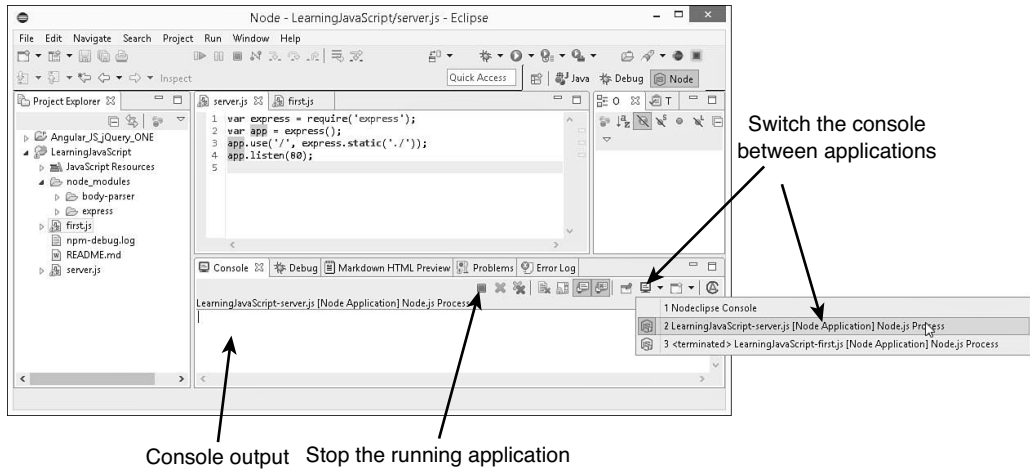


FIGURE 1.12
Running a JavaScript application in Eclipse.

NOTE

If you are not familiar with Eclipse, you should take a minute to practice starting and stopping the server and running the first.js application at the same time to understand starting and stopping applications and how to navigate between them in the console window.

You have now successfully set up a Node.js/Express web server in Eclipse. You will use this web server for most of the examples in the book. A few of the lessons require AJAX interaction, and a separate server will be created for those lessons.

LISTING 1.5 server.js Creating a Basic Node.js/Express Web Server

```
01 var express = require('express');
02 var app = express();
03 app.use('/', express.static('./'));
04 app.listen(80);
```

▼ **TRY IT YOURSELF**

Creating a Dynamic Web Page with jQuery and JavaScript

Now that you have a project created and a working web server, you are ready to create your dynamic web pages. In this section, you follow the steps to create a fairly basic dynamic web page. When you are finished, you will have a dynamic web page based on HTML, stylized with CSS with interaction through jQuery and JavaScript.

NOTE

The images and code files for this and all the examples throughout this book can be downloaded from the code archive on Github.

Adding HTML

The first step is to create a simple web page that has an HTML element that you can stylize and manipulate. Use the following steps in the editor to create the HTML document that you will use as your base:

1. Create a folder named lesson01 in your project.
2. Right-click the lesson01 folder that you created.
3. Select New, File from the pop-up menu.
4. Name the file first.html and click OK. A blank document should be opened up for you.
5. Type in the following HTML code. Don't worry if you are not too familiar with HTML; you'll learn enough to use it a bit later in the book:

```
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="content-type" content="text/html; charset=utf-8" />
  </head>
  <body>
    <span>Click Me</span>
  </body>
</html>
```

6. Save the file.
7. Open the following URL in your web browser and you should see the text "Click Me" appear:

```
http://localhost/lesson01/first.html
```

That's it. All the basic HTML elements are now in place. In the next section, you stylize the `` element so that Click Me looks more like a button.

Adding CSS

The simple text rendered by the browser is pretty plain, but that problem can quickly be solved by adding a CSS style. In this section, you use CSS to make the text appear more like a button.

Use the following steps to add the CSS style to the `` element. For reference, the style changes you make in these steps are shown in the final script in Listing 1.6:

1. Add the following code inside the `<head>` tags of the web page to include a CSS `<style>` element for all `` elements:

```
<style>
  span{
  }
</style>
```

2. Add the following property setting to the span style to change the background of the text to a dark blue color:

```
background-color: #0066AA;
```

3. Add the following property settings to the span style to change the font color to white and the font to bold:

```
color: #FFFFFF;
font-weight: bold;
```

4. Add the following property settings to the span style to add a border around the span text:

```
border-color: #C0C0C0;
border:2px solid;
border-radius:5px;
padding: 3px;
```

5. Add the following property settings to the span style to set an absolute position for the span element:

```
position:absolute;
top:150px;
left:100px;
```

6. Save the file.

7. Open the following URL in your web browser, and you should see the stylized text Click Me appear, as shown in Figure 1.13:

```
http://localhost/lesson01/first.html
```



FIGURE 1.13
`` element stylized to look like a button.

Writing a Dynamic Script

Now that the HTML is stylized the way you want it, you can begin adding dynamic interactions. In this section, you add a link to a hosted jQuery library so that you will be able to use jQuery, and then you link the browser mouse event `mouseover` to a JavaScript function that moves the text.

Follow these steps to add the jQuery and JavaScript interactions to your web page:

1. Change the `` element to include an ID so that you can reference it, and also add a handler for the `mouseover` event, as shown in line 30 of Listing 1.6:

```
<span id="elusiveText" onmouseover="moveIt()">Click Me</span>
```

2. Add the following line of code to the `<head>` tag, as shown in line 6 of Listing 1.6. This loads the jQuery library from a hosted source:

```
<script src="https://code.jquery.com/jquery-2.1.3.min.js"></script>
```

3. Add the following JavaScript function to the `<head>`, as shown in lines 6–13 of Listing 1.6. This function creates an array of coordinate values from 10 to 350, then randomly sets the top and left CSS properties of the span element each time the mouse is moved over it:

```
function moveIt(){
    var coords = new Array(10,50,100,130,175,225,260,300,320,350);
    var x = coords[Math.floor((Math.random()*10))];
    var y = coords[Math.floor((Math.random()*10))];
    $("#elusiveText").css({"top": y + "px", "left": x + "px"})
}
```

4. Save the file.
5. Open the following URL in your web browser, and you should see the stylized text Click Me appear, as shown in Figure 1.13:

```
http://localhost/lesson01/first.html
```

6. Now try to click the Click Me button. The button should move each time the mouse is over it, making it impossible to click it.
7. Find someone who annoys you, and ask them to click the button.

LISTING 1.6 A Simple Interactive jQuery and JavaScript Web Page

```
01 <!DOCTYPE html>
02 <html>
03   <head>
04     <meta charset="utf-8" />
05     <script src="https://code.jquery.com/jquery-2.1.3.min.js"></script>
```



```
06     <script>
07         function moveIt(){
08             var coords = new Array(10,50,100,130,175,225,260,300,320,350);
09             var x = coords[Math.floor(Math.random()*10)];
10             var y = coords[Math.floor(Math.random()*10)];
11             $("#elusiveText").css({"top": x + "px", "left": y + "px"})
12         }
13     </script>
14     <style>
15         span{
16             background-color: #0066AA;
17             color: #FFFFFF;
18             font-weight: bold;
19             border-color: #C0C0C0;
20             border:2px solid;
21             border-radius:5px;
22             padding: 3px;
23             position:absolute;
24             top:150px;
25             left:100px;
26         }
27     </style>
28 </head>
29 <body>
30     <span id="elusiveText" onmouseover="moveIt()">Click Me</span>
31 </body>
32 </html>
```

Summary

In this lesson, you learned the basics of web server and browser communications. You learned differences between GET and POST requests, as well as the purposes of server-side and client-side scripts. You also learned about the DOM and how the browser uses it to render the web page that is displayed to the user.

You have set up a good web development environment and created your first project. As part of creating your first project, you created a dynamic web page that incorporates HTML, CSS, jQuery, and JavaScript.

Q&A

Q. Which is better—a client-side or a server-side script?

- A.** It really depends on what you are trying to accomplish. Some people say that one way or the other is the only way to go. In reality, it is often a combination of the two that provides the best option. A good rule to follow is that if the interaction with the data is heavier based on user interaction such as mouse clicks, use a client-side script. If validation or error handling of the data requires interaction with the server, use a server-side script.

Q. Why don't all browsers handle JavaScript the same way?

- A.** To render HTML and interact with JavaScript, the browsers use an engine that parses the data from the server, builds objects, and then feeds them into a graphical rendering engine that writes them on the screen. Because each browser uses a different engine, each interprets the scripts slightly differently, especially with fringe elements that have not yet become standardized. If you want to support all browsers, you need to test your web pages in each of them to verify that they work correctly.

Workshop

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

Quiz

1. Would you send a GET or a POST request to a web server to open a web page?
2. What type of script has access to browser mouse events: server-side, client-side, or both?
3. True or false: JavaScript consoles are enabled by default on all browsers.
4. What type of script is the best to use when defining the appearance of DOM elements?

Quiz Answers

1. GET
2. Client-side
3. False. You must manually enable JavaScript debugging on all browsers. Pressing F12 in most browsers will launch the Developer Tools that allow you to debug JavaScript.
4. CSS scripts are the simplest to use when defining the appearance of DOM elements.

Exercises

1. Modify your first.html file to change the background color of your button randomly each time it is moved. Add the following two lines to randomly select a color:

```
var colors = new Array("#0066AA", "#0000FF", "#FF0000", "#00FF00");  
var color = colors[Math.floor((Math.random()*4))];
```

Then modify the CSS change in your JavaScript to include background-color, as shown next:

```
$("#elusiveText").css({"top": y + "px", "left": x + "px", "background-color":  
    color});
```

2. Add an additional `` element to your first.html file with the same behavior as the first. To do this, add the following two lines in the appropriate locations. You should be able to figure out where they go:

```
$("#elusiveText2").css({"top": x + "px", "left": y + "px"})  
<span id="elusiveText2" onmouseover="moveIt()">Click Me</span>
```

This page intentionally left blank

Index

Symbols

- \$anchorScroll service (AngularJS), 700**
 - \$animate service (AngularJS), 700, 714-719**
 - \$broadcast() method, 691**
 - \$cacheFactory service (AngularJS), 700, 704-709**
 - \$compile service (AngularJS), 700**
 - \$cookies service (AngularJS), 700**
 - \$cookieStore service (AngularJS), interacting with browser cookies, 709-711**
 - :data() selector, 464**
 - \$destroy event (jQuery Lite objects), 562**
 - \$document service (AngularJS), 700**
 - \$emit() method, 691**
 - \$exceptionHandler service (AngularJS), 700**
 - :focusable selector, 464**
 - \$http service (AngularJS), 700**
 - HTTP servers, accessing, 703-708
 - sending GET and PUT requests, 701-702
 - configuring requests, 702
 - implementing callback functions, 703-708
 - \$ingerpolate service (AngularJS), 700**
 - \$interval service (AngularJS), 700, 714**
 - \$locale service (AngularJS), 700**
 - \$location service (AngularJS), 700**
 - providing wrapper for window.location object, 721-724
 - \$log service (AngularJS), 700**
 - \$on() method, 691-692**
 - \$parse service (AngularJS), 700**
 - \$q service (AngularJS), 700**
 - providing deferred responses, 726-728
 - \$resource service (AngularJS), 700**
 - \$rootElement service (AngularJS), 700**
 - \$rootScope service (AngularJS), 700**
 - \$route service (AngularJS), 700**
 - \$routeParams service (AngularJS), 700**
 - \$sanitize service (AngularJS), 700**
 - \$sce service (AngularJS), 700**
 - \$swipe service (AngularJS), 700**
 - :tabbable selector, 465**
 - \$templateCache service (AngularJS), 700**
 - \$timeout service (AngularJS), 700, 714**
 - \$watch() method, 689**
 - tracking scope variables, 687
 - \$watchCollection() method, 689**
 - tracking changes to object properties in scope, 688-690
 - \$watchGroup() method, 689**
 - track multiple scope variables, 687
 - \$window service (AngularJS), 700**
 - implementing browser alerts, 704-709
- ## A
- a directive (AngularJS templates), 634**
 - abort event, 239**
 - abort() method, 453**
 - absUrl() method, 722**
 - ACCEPT header (HTTP), 15**
 - accept option (jQuery droppable widget), 501**
 - accept rule (validation), 376**
 - active option (tabs widget), 541**
 - activeClass option (jQuery droppable widget), 501**
 - addClass() method, 199, 561**
 - addEventListener() method, 253**
 - after() method, 561**
 - AJAX handlers, 19-20**
 - ajax() method, 450-452**
 - AJAX requests, 423, 453**
 - advanced jQuery, 450-453
 - global event handlers, 451
 - global setup, 450
 - asynchronous communication, 425-426
 - cross-domain, 426
 - GET versus POST, 427

- handling responses, 433-434
 - HTML response data, 439-444
 - JSON response data, 438-441
 - success and failures, 434-437
 - XML response data, 439-444
- implementing, 428-450
 - from JavaScript, 428-429
 - from jQuery, 429-433
 - low-level, 451-453
 - versus page requests, 424-425
 - response data types, 427
 - server-side services, 425
- `ajax_post.css` listing (15.19), 448-449
- `ajax_post.html` listing (15.17), 447
- `ajax_post.js` listing (15.18), 447-448
- `ajax_response.css` listing (15.7), 437
- `ajax_response.html` listing (15.5), 436
- `ajax_response.js` listing (15.6), 436-437
- `ajaxComplete()` method, 451
- `ajaxError()` method, 451
- `ajax.html` listing (1.3), 19-20
- `ajaxSend()` method, 451
- `ajaxSetup()` method, 450
- `ajaxStart()` method, 451
- `ajaxStop()` method, 451
- `ajaxSuccess()` method, 451
- `alert()` method (window object), 305
- `alsoResize` option (jQuery resizable widget), 507
- altKey attribute (event objects), 237**
- always() method, 453**
- analysis, network traffic, 63-65**
- AngularJS, 549, 564**
 - adding to environment, 554-555
 - animating elements, 717-719
 - applications, 774
 - creating basic, 557-560
 - creating rich web components, 751-773
 - dynamics, 567
 - using jQuery and jQuery Lite in, 560-564
 - benefits, 548-549
 - bootstrapping in HTML document, 555
 - compiler, 552
 - controllers, 551
 - data binding, 551
 - data model, 550
 - debugging, 63
 - dependency injection, 551, 567, 568-569
 - implementing, 572-574
 - injector service, 569
 - directives, 550, 627
 - automatically added and removed during animation, 715
 - built-in, 628-653
 - creating custom, 657-668
 - custom, 683
 - elements
 - adding star ratings, 770-773
 - draggable and droppable, 756-761
 - expandable/collapsible, 764-770
- events, 685, 696-697
 - browser, 685
 - emitting and broadcasting custom, 691-693
 - tracking scope change, 686-689
- expressions, 550
- global APIs, 555-560
- images, adding a zoom view field to, 761-764
- integrating with JavaScript and jQuery, 553-554
- life cycle, 552
 - bootstrap phase, 552
 - compilation phase, 552
 - runtime data binding phase, 552-553
- loading jQuery library, 563-564
- modules, 549, 567-568
 - adding configuration blocks, 575-578
 - adding run blocks, 578
 - creating providers, 570-572
 - defining module object, 569-570
 - injecting into another, 575-577
 - providers, 569
- providers
 - implementing, 572
 - injecting into controller, 572
- scope, 550, 583, 597
 - implementing hierarchy, 593-595
 - life cycle, 591-593
 - relationship between backend server data, 591
 - relationship between controllers, 584-586

- relationship between root scope and applications, 584
- relationship between templates, 587-589
- template values, 588-589
- separation of responsibilities, 553
- services, 551, 699-700, 728
 - \$animate, 714-719
 - \$cacheFactory, 704-709
 - \$cookieStore, 709-711
 - \$http, 701-708
 - \$interval, 714
 - \$location, 721-724
 - \$q, 726-728
 - \$timeout, 714
 - \$window, 704-709
- building factory, 732
- built-in, 700-701
- creating custom, 731-748
- database access, 741-747
- defining constant, 732
- defining service, 733
- defining value, 732
- time, 737
- templates, 550, 599-600, 623
 - custom filters, 620-621
 - directives, 599
 - expressions, 599, 600-609
 - filters, 599, 611-618
- views, 550
 - building tabbed, 751-755
- angular.module() method, 569-570**
- animate() method, 322**
- animate.css listing (27.9), 720**
- animated_resize.css listing (12.12), 339**
- animated_resize.html listing (12.10), 338-339**
- animated_resize.js listing (12.11), 339**
- animation, 321-325**
 - adding effects to, 487-491
 - applying promise() to, 325
 - CSS (Cascading Style Sheets)
 - implementing in, 715-716
 - settings, 322
 - delaying, 324-325
 - elements, adding to web forms, 368-370
 - hide() method, 325, 326-327, 328-329
 - interactive, 341-343
 - JavaScript, implementing in, 716-719
 - moving elements, 340-344
 - queues, 323
 - resize, creating, 337-339
 - setting effect easing, 478
 - sliding, 332-337
 - stopping, 323
 - toggle() method, 326-327
 - visibility, 329-332
- animation_effects.css listing (17.12), 491**
- animation_effects.html listing (17.10), 489-491**
- animation_effects.js listing (17.11), 491**
- append() method, 561**
- appendChild() method, 198**
- appendTo option (jQuery selectable widget), 511**
- applications (AngularJS), 567**
 - building tabbed views, 751-755
 - creating basic, 557-560
 - creating rich web components, 751-773
- dependency injection, 568-569
 - implementing, 572-574
 - injector service, 569
- elements
 - adding star ratings, 770-773
 - draggable and droppable, 756-761
 - expandable/collapsible, 764-770
- events, 685, 696-697
 - browser, 685
 - emitting and broadcasting custom, 691-693
 - global APIs, 555-560
 - tracking scope change, 686-689
- images, adding a zoom view field to, 761-764
- integrating custom services into, 733-747
- modules, 567-568
 - adding configuration blocks, 575-578
 - adding run blocks, 578
 - creating providers, 570-572
 - defining module object, 569-570
 - injecting into another, 575-577
 - providers, 569
- providers
 - implementing, 572
 - injecting into controller, 572
- scope, 583, 597
 - life cycle, 591-595
 - relationship between backend server data, 591

- relationship between controllers, 584-586
 - relationship between root scope and applications, 584
 - relationship between templates, 587-589
 - services, 699-700
 - \$animate, 714-719
 - \$cacheFactory, 704-709
 - \$cookieStore, 709-711
 - \$http, 701-708
 - \$interval, 714
 - \$location, 721-724
 - \$q, 726-728
 - \$timeout, 714
 - \$window, 704-709
 - built-in, 700-701
 - using jQuery and jQuery Lite in, 560-564
 - arithmetic operators (JavaScript), 154
 - Array data type (JavaScript), 153
 - Array object (JavaScript), 181-187
 - array_manipulation.html listing (6.2), 186-187
 - arrays
 - adding/removing items, 184-187
 - checking for items, 184
 - combining, 183
 - converting into strings, 183
 - creating, 184-186
 - iterating through, 183
 - manipulating, 184-187
 - article1.html listing (15.3), 433
 - ASP (Active Server Pages), 425
 - aspectRatio option (jQuery resizable widget), 507
 - asrfHeaderName property (config parameter), \$http service requests, 702
 - assign() method (window object), 307
 - assignment operators (JavaScript), 154
 - Associative Array/Objects data type (JavaScript), 153
 - asynchronous communication, AJAX requests, 425-426
 - attr() method, 199, 561
 - attribute selectors (jQuery), 206
 - attributes
 - ajax() method, 450-452
 - DOM objects, 198
 - HTML elements, 71-72
 - web form elements, 348-349
 - XMLHttpRequest object, 428
 - <audio> tag, 102
 - AUTHORIZATION header (HTTP), 15
 - autoHide option (jQuery resizable widget), 507
 - availHeight property (screen object), 304
 - availWidth property (screen object), 304
 - axis option
 - draggable widget, 497
 - sortable widget, 516
- ## B
- backend server data, relation between scopes, 591
 - background-attachment property (CSS), 119
 - background-color property (CSS), 119
 - background-image property (CSS), 119
 - background-position property (CSS), 119
 - background-repeat property (CSS), 120-121
 - backgrounds.html listing (4.3), 124-125
 - background-size property (CSS), 119
 - behavior, directives, restricting, 660-661
 - bind() method, 561
 - bindToController definition property (AngularJS), 658
 - blind effect (jQuery UI), 476
 - block elements, 78-79
 - blur events, 239, 647-648
 - blur() method (window object), 305
 - blurring elements, web forms, 361
 - <body> elements, 77-90
 - Boolean data type (JavaScript), 153
 - bootstrap phase, AngularJS life cycle, 552
 - bootstrapping AngularJS web pages, 555
 - border property (CSS), 123
 - border-color property (CSS), 123
 - border-radius property (CSS), 123
 - borders (CSS), adding to HTML elements, 123-128
 - border-style property (CSS), 123
 - border-width property (CSS), 123
 - bounce effect (jQuery UI), 476
 - box model, HTML elements, 132
 - box-shadow property (CSS), 123
 -
 tag, 79
 - breakpoints (debugger), 58
 - broadcasting custom events, 691-693

- broken_event.css listing (9.3), 248**
- broken_event.html listing (9.1), 247**
- broken_event.js listing (9.2), 248**
- browser events, AngularJS applications, 685**
- browser history object, 307-308**
- browser location object, 306-307**
- browsers, 10**
 - accessing element values, 270-281
 - obtaining and setting values, 271
 - obtaining mouse position, 270
 - alerts, implementing with \$window service, 704-709
 - color information, 274-281
 - cookies, interacting with using \$cookieStore service, 709-711
 - development, 21
 - events, 11
 - lists
 - altering appearance, 13-14
 - rendering in, 12
 - screen size, 274-281
 - URLs (uniform resource locators), 11
 - web server/browser paradigm, 9-20
 - window, 11
- built-in directives (AngularJS), 628-653**
 - binding model to page elements, 639-643
 - binding page events to controllers, 645-653
 - extending form elements, 631-637
 - functionality support, 628-630

- built-in filters, AngularJS templates, 612**
- built-in JavaScript objects, 175-189**
 - Array, 181-187
 - Date, 187-188
 - Math, 188
 - Number, 175-176
 - RegExp, 189
 - String, 176-178
- built-in services (AngularJS), 700-701**
- button attribute (event objects), 237**
- button input elements, web forms, accessing, 352**
- <button> tag, 89**
- buttonImage option (datepicker widget), 530**
- buttonImageOnly option (datepicker widget), 530**
- buttons option (dialog widget), 532**

C

- cache property (config parameter), \$http service requests, 702**
- calendar input, web pages, 530-531**
- Call Stack pane (debugger), 58**
- callbacks, jQuery, 263-265**
- cancelable attribute (event objects), 237**
- <canvas> element, 99-101**
- <caption> tag, 85**
- card_suits.css listing (9.12), 261-262**
- card_suits.html listing (9.10), 261**

- card_suits.js listing (9.11), 261**
- CDN (Content Delivery Network), 555**
- change event, 239**
- change option (slider widget), 537**
- charAt() method, 177**
- charCode attribute (event objects), 237**
- check box elements, web forms, accessing, 350**
- children() method, 561**
- class attributes**
 - body elements, 77
 - DOM objects, 198
- class transitions**
 - adding effects to, 482-485
 - applying easing to, 483-484
- class_transitions.css listing (17.6), 485**
- class_transitions.html listing (17.4), 484**
- class_transitions.js listing (17.5), 485**
- classes**
 - HTML elements, 274
 - transitions, 484-485
- clearInterval() method (window object), 305**
- clearTimeout() method (window object), 305**
- click event, 239**
- click() method, 198-199**
- client-side scripting, 16-17**
 - AJAX request, 19-20
- clientX attribute (event objects), 237, 270**
- clientY attribute (event objects), 237, 270**
- clip effect (jQuery UI), 476**
- clock.cs CSS Code That Styles the Clock listing (11.6), 317**
- clock.cs listing (11.6), 317**

- clock.html HTML listing (11.4), 316-317
- clock.js listing (11.5), 317
- clone() method, 561
- close() method, 305
- closed property (window object), 304
- code. *See also* listings, delaying, \$interval and \$timeout services, 714
- <col> tag, 85
- <colgroup> tag, 85
- collapse element, creating, 326-329
- collapsible elements, AngularJS applications, 764-770
- collapsible option (tabs widget), 541
- collapsing branches, dynamic tree view, 404-408
- color design properties (CSS), 112-130
- color property (CSS), 113
- colorDepth property (screen object), 304
- combining
 - arrays, 183
 - strings, 178
- comparison operators (JavaScript), 155-160
- compilation phase, AngularJS life cycle, 552
- compile definition property (AngularJS), 658
- compile() function, 665-667
- compiler (AngularJS), 552
- complete option (jQuery animation), 322
- complex validation, web forms, 377-384
 - adding messages, 378-380
 - adding rules, 377-378
 - placing messages, 380-384
- concat() method, 177, 182
- config() method, 578
- config_run_blocks.js listing, 578-580
- configuration
 - directive scope, 663-664
 - Node.js, 21-22
- configuration blocks, applying to modules, 575-578
- confirm() method (window object), 305
- connectTo option (sortable widget), 516
- container elements, 80-82
- containment option (draggable widget), 497
- containment option (jQuery resizable widget), 507
- content, HTML elements, 71
- content selectors (jQuery), 207
- CONTENT-LENGTH header (HTTP), 15
- contents() method, 561
- CONTENT-TYPE header (HTTP), 15
- controller definition property (AngularJS), 658
- controller() method, 562
- controllerAs definition property (AngularJS), 658
- controllers
 - AngularJS, 551
 - binding page events to, 645-653
 - directives, adding to, 661-662
 - implementing basic, 585-586
 - implementing watches in, 688-689
 - injecting built-in providers into, 572
 - nested
 - custom events, 692-693
 - relationship between scopes, 584-586
 - scopes, 593-595
- COOKIE header (HTTP), 15
- cookies, obtaining and setting, 308-313
- Cookies tab (network traffic analyzer), 65
- cookies.css listing (11.3), 312-313
- cookies.html HTML listing (11.1), 309
- cookies.js listing (11.2), 312
- copy() method, 556
- create event (jQuery UI widgets), 495
- createPop() method (window object), 305
- creation phase (AngularJS scope), 591-592
- creditcard rule (validation), 376
- cross-domain AJAX requests, 426
- CSS (Cascading Style Sheets), 13, 105, 141. *See also* CSS3
 - adding
 - backgrounds, 119-121
 - code to headers, 107
 - styles, 29-30, 105-108
 - animation, implementing in, 715-716
 - borders, adding to HTML elements, 123-128
 - buttons, styling, 312-313

- creating
 - dynamic graphic equalizer, 413-417
 - sparkline graphics, 417-421
- debugging, 48-51
- defining styles in HTML elements, 108
- design properties, 112-131
 - color, 112-130
 - cursor, 130
 - opacity, 131
 - visibility, 131
- editing properties, 48-49
- elements
 - rearranging, 299
 - styles, 291-292
- images, styling, 312-313
- layout properties, applying, 119-140
- loading styles from files, 106-107
- obtaining and setting properties, 272-273
- properties, preparing to directly adjusting, 141
- selectors, 110-111
- settings, animating, 322
- styles
 - adding to HTML elements, 108-140
 - applying text, 113-119
 - preparing for dynamic design, 140-141
- syntax, 108-110
- using styles in HTML body, 107-108
- z-index property, adjusting, 292-295

css() method, 199, 561

CSS Style editor, 48-49

css_errors.html listing (2.3), 50-52

css_styling.html listing (3.4), 82

CSS3, 13, 105. See also CSS (Cascading Style Sheets)

ctrlKey attribute (event objects), 237

currency[:symbol] filter (AngularJS templates), 612

Currently Executing Line option (debugger), 58

currentTarget attribute (event objects), 237

cursor design properties (CSS), 130

cursor option

- draggable widget, 497
- sortable widget, 516

custom directives (AngularJS), 657, 683

- adding a controller to, 661-662
- configuring scope, 663-664
- defining, directive view template, 659-660
- definitions, 657-668
- implementing, 668-680
- implementing event handlers in, 672-674
- manipulating DOM in, 668-670
- manipulating DOM with link() function, 665-667
- nested, 677-680
- restricting behavior, 660-661
- transcluding elements, 664-665

custom events, 262

- creating, 262
 - JavaScript, 262-263
 - jQuery, 262-263
- emitting and broadcasting, 691-693
- nested controllers, 692-693

custom filters, AngularJS templates, creating, 620-621

custom services (AngularJS), 731, 748

- building factory, 732
- database access, 741-747
- defining
 - constant service, 732
 - service, 733
 - value service, 732
- integrating into applications, 733-747
- time, 737

custom_objects.html listing (6.3), 194

D

data attribute, event objects, 237

data binding, AngularJS, 551

data() method, 561

data model, AngularJS, 550

data property (config parameter), \$http service requests, 702

data rendering, AngularJS template filters, 613-615

data types, JavaScript, 152-153

database access service, AngularJS, implementing, 741-747

data.html listing (1.4), 20

Date object (JavaScript), 187-188

date rule (validation), 376

date[:format] filter, AngularJS templates, 613

date_picker.html listing (16.1), 462

dateFormat option (datepicker widget), 530

dateISO rule (validation), 376

- dbclick event, 239**
- debugger (JavaScript), 51-63**
- debugging, 35, 65-66**
 - AngularJS, 63
 - CSS (Cascading Style Sheets), 48-51
 - HTML elements, 41-48
 - JavaScript, 51-63
 - jQuery, 63
- deferred responses, \$q service, 726-728**
- defining, JavaScript objects, 190**
- definitions, AngularJS custom directives, 657-668**
 - directive view template, 659-660
- delay timers, adding, 314-315**
- delaying animation, 324-325**
- delaying code, \$interval and \$timeout services, 714**
- delegateTarget attribute (event objects), 237**
- dependency injection, AngularJS, 551, 568-569**
 - implementing, 572-574
 - injector service, 569
- design properties (CSS), 112-131**
 - color, 112-130
 - cursor, 130
 - opacity, 131
 - visibility, 131
- destroy() method, 495**
- detach() method, 561**
- developer tools console, 35-40**
- development web servers, 21**
- dialogs**
 - overlay, 409-413
 - simple, 315-316
 - stylized, generating, 532-533
- digits rule (validation), 376**
- directive() method, 657-658, 683**
- directive_angular_include.html listing (24.2), 632**
- directive_angular_include.js listing (24.1), 630**
- directive_bind.html listing (24.8), 644**
- directive_bind.js listing (24.7), 644**
- directive_custom_dom.html listing (25.2), 670-671**
- directive_custom_dom.js listing (25.1), 670**
- directive_custom_photos.html listing (25.6), 680-681**
- directive_custom_photos.js listing (25.5), 680**
- directive_custom_zoom.html listing (25.4), 676**
- directive_custom_zoom.js listing (25.3), 674-676**
- directive_focus_events.html listing (24.10), 648-649**
- directive_focus_events.js listing (24.9), 648**
- directive_form.html listing (24.6), 638**
- directive_form.js listing (24.5), 638**
- directive_keyboard_events.html listing (24.12), 651**
- directive_keyboard_events.js listing (24.11), 649-651**
- directive_mouse_events.html listing (24.14), 654-655**
- directive_mouse_events.js listing (24.13), 654**
- directives (AngularJS), 550, 627**
 - a, 634
 - animation, automatically added and removed during, 715
 - built-in, 628-653
 - binding model to page elements, 639-643
 - binding page events to controllers, 645-653
 - extending form elements, 631-637
 - functionality support, 628-630
 - custom, 657, 683
 - adding a controller to, 661-662
 - configuring scope, 663-664
 - definitions, 657-668
 - directive view template, 659-660
 - DOM (Document Object Model), 670-671
 - implementing, 668-680
 - implementing event handlers in, 672-674
 - manipulating DOM in, 668-670
 - manipulating DOM with link() function, 665-667
 - nested, 677-680
 - restricting behavior, 660-661
 - transcluding elements, 664-665
 - zoom, 674-676
 - focus events, 648-649
 - form/ngForm, 634
 - input, 634
 - input.checkbox, 634
 - input.email, 634
 - input.number, 635
 - input.radio directive, 634
 - input.text directive, 634
 - input.time directive, 578
 - input.url directive, 634
 - input.week directive, 634
 - keyboard events, 649-651

- mouse events, 652-655
- templates, 599
- `disable()` method, 495
- disabling elements, web forms, 362
- `<div>` elements (HTML), 136-139, 279-281
- document structure (HTML), 70-72
- DOM (Document Object Model), 10-11, 32
 - accessing, 148-151
 - determining, 200
 - element properties, viewing and editing, 46-48
 - elements, adding and removing event handlers, 242-253
 - JavaScript objects, 197-198
 - manipulating in custom directives, 668-670
 - manipulating with `compile()` function, 665-667
 - manipulating with `link()` function, 665-667
 - objects
 - accessing, 201-204, 211-213
 - adding event handlers, 245-246
 - changing to jQuery, 201
 - finding by class name, 201
 - finding by ID, 201
 - finding by tag name, 202-203
 - obtaining and setting attributes and properties, 271-272
 - traversing methods, jQuery objects, 219-220, 227-230
- DOM editor, 47-48
- `dom_manipulation.css` listing (8.1), 228
- `dom_manipulation.css` listing (8.3), 229
- `dom_manipulation.js` listing (8.2), 228-229
- `dom_objects.css` listing (7.3), 204
- `dom_objects.html` listing (7.1), 204
- `dom_objects.js` listing (7.2), 204
- `done()` method, 453
- do/while loop (JavaScript), 161
- `drag_n_drop.html` listing (18.4), 503-505
- `drag_n_drop.js` listing (18.5), 506-507
- `drag_n_drop.js` listing (18.6), 506-507
- drag-and-drop widgets (jQuery), 497-507
 - draggable widget, dragging elements, 497-501
 - droppable widget, creating drop targets, 497-507
- `dragdrop.html` listing (29.6), 759-760
- `dragdrop.js` listing (29.5), 756-759
- draggable elements, implementing, 756-761
- draggable widget (jQuery), 497-507
- `draggable_images.css` listing (18.2), 501
- `draggable_images.html` listing (18.1), 500
- `draggable_images.js` listing (18.2), 500-501
- drop effect (jQuery UI), 476
- drop targets, creating with droppable widget, 497-507
- droppable elements, implementing, 756-761
- droppable widget (jQuery), 497-507
- duration option (jQuery animation), 322
- dynamic dialogs, adding using overlays, 409-413
- dynamic flow control, web forms, 362-364
- dynamic scripts, writing, 31-32
- dynamic sparkline graphics, creating, 417-421
- dynamic tree view, adding with expanding and collapsing branches, 404-408
- dynamic web pages, 69
 - creating with jQuery, 28
 - HTML
 - document structure, 70-72
 - elements, 69-70
- `dynamic_dialog.css` listing (14.12), 412-413
- `dynamic_dialog.html` listing (14.10), 410-411
- `dynamic_dialog.js` listing (14.11), 412
- `dynamic_positioning.css` listing (16.7), 472
- `dynamic_positioning.html` listing (16.5), 471
- `dynamic_positioning.js` listing (16.6), 471-472
- `dynamic_spark.css` listing (14.18), 421
- `dynamic_spark.html` listing (14.16), 418-420
- `dynamic_spark.js` listing (14.17), 420-421
- `dynamic_tree.css` listing (14.9), 408
- `dynamic_tree.html` listing (14.7), 407

dynamic_tree.js listing (14.8),
407-408

dynamically manipulating HTML
elements, 282-292

dynamically rearranging elements,
292-299

E

easing

applying to class transitions,
 483-484

setting on animation, 478

easing option (jQuery animation),
322

Eclipse, configuring as web
development IDE, 22-26

effects

animation, adding to, 487-491
 jQuery UI

adding to class transitions,
 482-485

adding to element visibility
 transitions, 485-491

applying, 475-482

Element inspector (JavaScript),
41-46

element() method, 556

elements (DOM), adding and
removing event handlers,
242-253

elements (HTML), 103, 303

accessing with jQuery
 selectors, 149-150

accessing class, 274

adding

classes to dynamically,
 140-141

CSS borders to, 123-128

dynamically, 282-283

margins around, 133

padding around, 133

animated, adding, 368-370

animating, AngularJS,
 717-719

attributes, 71-72

block, 78-79

<body>, 76-90

box model, 132

changing classes, 286

collapse, creating, 326-329

container, 80-82

content, 71

CSS (Cascading Style Sheets)
 styles, adding to, 108-140
 selectors, 110-111

debugging, 41-48

defining CSS styles in, 108

<div>, 279-281

DOM properties, viewing and
 editing, 46-48

drag-and-droppable, 502-507

dragging, jQuery draggable
 widget, 497-507

dynamic web pages, 69-70

dynamically manipulating,
 282-292

dynamically rearranging,
 292-299

expand, creating, 326-329

expandable accordion,
 526-527

form, 88-90

extending with directives,
 631-637

implementing
 autocomplete, 528-530

grouping, 80-81

<head>, 72-77

<link> tag, 76-77

<meta> tag, 73-74

<noscript> tag, 76

<script> tag, 75

<style> tag, 74

<title> tag, 72-73

image, 83-84

inline, 78-79

<input>, 232, 261, 309

inserting, jQuery, 285-286

inspecting, 41-46

<label>, 232

link, 82-83

list, 84

modifying flow, 133-134

moving, implementing,
 340-344

obtaining and setting size,
 273

obtaining information,
 275-277

<p>, 261

page, binding model to
 elements, 639-643

positioning, 273-274
 from CSS, 134-135
 jQuery UI, 468-472

removing, 284

replacing, 285

resizable, 511-516

selectable sets, 512-516

sorting, 516-522

, 232, 257-258,
 261, 309

syntax, 71

table, 85-88

tooggling visibility, 286-287

transcluding, 664-665

value spinner, 538-539

visibility, adding effects to
 transitions, 485-491

web forms, 348-360

automatically focusing and
 blurring, 361

- button input, 352
- check box, 350
- disabling, 362
- dynamically controlling appearance and behavior, 368-374
- file input, 352-353
- hiding and showing, 361
- obtaining and setting values, 348-353
- radio input, 350-351
- select input, 351-352
- text input, 349-350
- elements (HTML5), 93-102**
 - <canvas>, 99-101
 - graphical, 93-101
 - media, 102
 - <path>, 95-99
 - <svg>, 95-99
- email rule (validation), 376**
- emitting custom events, 691-693**
- empty() method, 561**
- enable() method, 495**
- eq() method, 561**
- equals() method, 556**
- equalTo rule (validation), 376**
- error event, 239**
- error handling, JavaScript, 168-169**
- escape codes, JavaScript strings, 176**
- event handlers**
 - DOM elements, 242-253
 - implementing in custom directives, 672-674
 - jQuery, applying, 248-253
- event handlers (HTML), accessing, 147-148**
- event object attributes, mouse position specification, 270**
- event option (tabs widget), 541**
- eventPhase attribute (event objects), 237**
- events, 235-236, 265-266**
 - AngularJS applications, 685, 696-697
 - browser, 685
 - emitting and broadcasting custom, 691-693
 - tracking scope change, 686-689
 - blur, 647-648
 - broken, 247-248
 - browsers, 11
 - creating custom, 262
 - JavaScript, 262-263
 - jQuery, 262-263
 - draggable widget (jQuery), 498
 - droppable widget, focus, 647-648
 - jQuery Lite objects, 562
 - jQuery sortable widget, 517
 - jQuery UI widgets, 495
 - keyboard, handling on AngularJS elements, 649
 - mouse, handling on AngularJS elements, 652-653
 - objects, 237-239
 - attributes, 237
 - onload, 240-242
 - page, binding to controllers, 645-653
 - page load for initialization, 240-242
 - process, 236-237
 - selectable widget events, 511
 - triggering manually, 253
 - JavaScript, 254-258
 - jQuery, 258-262
 - types, 239
- expand element, creating, 326-329**
- expand_item.html listing (29.12), 768**
- expand_list.html listing (29.11), 767**
- expandable accordion element, adding, 526-527**
- expand.html listing (29.13), 768-770**
- expandable elements, AngularJS applications, 764-770**
- expanding branches, dynamic tree view, 404-408**
- expand.js listing (29.10), 766-767**
- explode effect (jQuery UI), 476**
- Express web servers, creating, 26-29**
- expressions**
 - AngularJS templates, 550, 599-609
 - basic, 602-603
 - JavaScript, 608-609
 - scopes, 604-606
- expressions_basic.html listing (23.2), 603-604**
- expressions_basic.js listing (23.1), 603**
- expressions_javascript.html listing (23.6), 610**
- expressions_javascript.js listing (23.5), 610**
- expressions_scope.html listing (23.4), 607**
- expressions_scope.js listing (23.3), 606**
- extend() method, 556**
- external links, 308**
 - forcing to open in new browser windows, 308-311
 - stopping, 308

F

- factory services (AngularJS), building using factory provider, 732
- fade animations, 330
- fade effect (jQuery UI), 476
- fadeOut() method, 329
- fadeTo() method, 330-331
- fadeToggle() method, 329-330
- fail() method, 453
- failure, AJAX requests, handling, 433-444
- <fieldset> tag, 89
- file input elements, web forms, accessing, 352-353
- files, loading CSS styles from, 106-107
- filter() method, 620
- filter option (jQuery selectable widget), 511
- filter_custom.html listing (23.12), 622
- filter_custom.js listing, 621-622
- filter_sort.html listing, 618-619
- filter_sort.js listing (23.9), 618
- filtered selector (jQuery), 211
- filter:exp:compare filter (AngularJS templates), 612
- filtering tables, implementing, 397-404
- filtering object results, jQuery, 218-219
- filters, AngularJS templates, 599, 611-618
 - built-in, 612
 - custom, 620-621
 - data rendering, 613-615
 - ordering and sorting, 616-618
- filters.html listing (23.8), 615
- filters.js listing (23.7), 615
- find() method, 561
- first.html listing (20.1), 560
- first.js listing (20.2), 555-560
- flow control, web forms, 361-368
- flying saucer app, creating, 340-342
- focus event, 239, 647-648
- focus() method (window object), 305
- focusin event, 239
- focusing elements, web forms, 361
- focusout event, 239
- fold effect (jQuery UI), 476
- font property (CSS), 113-114
- for loops (JavaScript), 161-162
- forEach() method, 556
- for/in loops (JavaScript), 162
- form elements, 88-90
 - binding to scope, 636-637
 - extending with directives, 631-637
- form selector (jQuery), 209
- form_effects.html listing (13.7), 371-372
- form_effects.js listing (13.8), 372-373
- form_effects.js listing (13.9), 374
- form_flow.css listing (13.6), 367-368
- form_flow.html listing (13.4), 365-366
- form_flow.js listing (13.5), 366-367
- form_manipulation.css listing (13.3), 360
- form_manipulation.html listing (13.1), 358-359
- form_manipulation.js listing (13.2), 359-360
- form_validation.css listing (13.12), 386-387
- form_validation.html listing (13.10), 382-386
- form_validation.js listing (13.11), 386
- <form> tag, 88
- form/ngform directive (AngularJS templates), 634
- forms (web), 347-348, 387
 - accessing hidden inputs, 353
 - controlling submission and reset, 362-364
 - elements, 348-360
 - animated, 368-370
 - automatically focusing and blurring, 361
 - button input, 352
 - check box, 350
 - disabling, 362
 - dynamically controlling appearance and behavior, 368-374
 - file input, 352-353
 - hiding and showing, 361
 - obtaining and setting values, 348-353
 - radio input, 350-351
 - select input, 351-352
 - text input, 349-350
 - flow control, 361-368
 - dynamic, 362-364
 - serializing data, 354-360
 - validating, 375-387
 - adding messages, 378-380
 - adding rules, 377-378
 - complex validation, 377-384
 - jQuery validation plug-in, 376

- jQuery validation with HTML, 376-377
- manually, 375
- placing messages, 380-384
- forms.html listing (3.6), 90
- fromCharCode() method, 177
- fromJson() method, 556
- functionality, AngularJS built-in directives, 628-630
- functions. *See also* methods
 - JavaScript, creating, 163-167
 - writeln(), 168

G

- GET requests
 - HTTP (Hypertext Transform Protocol), 15-16, 32
 - versus POST, 427
 - sending, \$http service, 701-708
- getAllResponseHeaders() method, 453
- getAttribute() method, 198
- getResponseHeader() method, 453
- ghost option (jQuery resizable widget), 507
- global APIs, AngularJS, 555-560
- global event handlers, AJAX, 451
- graphic_equalizer.css listing (14.15), 417
- graphic_equalizer.html listing (14.13), 415-416
- graphic_equalizer.js listing (14.14), 416
- graphical elements (HTML5), 93-101

- graphical equalizer displays, implementing, 413-417
- graphics, adding sparkline, 417-421
- greedy option (jQuery droppable widget), 501
- grouping elements, 80-81

H

- handlers, AJAX, 19-20
- handles option (jQuery resizable widget), 507
- handling, AJAX responses, 433-444
- hasClass() method, 561
- hash() method, 722
- hash property (window object), 307
- head> elements, 72-77
 - link> tag, 76-77
 - meta> tag, 73-74
 - noscript> tag, 76
 - script> tag, 75
 - style> tag, 74
 - title> tag, 72-73
- headers
 - adding CSS code to, 107
 - HTTP (Hypertext Transform Protocol), 15
- headers property (config parameter), \$http service requests, 702
- Headers tab (network traffic analyzer), 65
- height() method, 199, 273
- height property (screen object), 304
- helper functions (jQuery), assigning event handlers, 253
- helper option (draggable widget), 497
- helper option (jQuery resizable widget), 507
- helper option (sortable widget), 516
- hide() method, 199, 325
 - animating, 325-329
- hiding elements, web forms, 361
- hierarchy selector (jQuery), 208-209
- highlight effect (jQuery UI), 476
- history.back() method (browser history object), 308
- history.forward() method (browser history object), 307
- host() method, 722
- host property (window object), 307
- hostname property (window object), 307
- hoverClass option (jQuery droppable widget), 501
- <hr> tag, 79
- href property (window object), 307
- HTML (HyperText Markup Language), 11-12, 103. *See also* elements (HTML); HTML5
 - adding to web pages, 29
 - assigning event handlers, 243-244
 - creating tables, 86
 - documents
 - bootstrapping AngularJS in, 555
 - structure, 70-72
 - elements, 303
 - accessing class, 274
 - adding classes to dynamically, 140-141

- adding dynamically, 282-283
 - adding margins around, 133
 - adding padding around, 133
 - attributes, 71-72
 - block, 78-79
 - body, 77-90
 - box model, 132
 - changing classes, 286
 - container, 80-82
 - content, 71
 - debugging, 41-48
 - defining CSS styles in, 108
 - dynamic web pages, 69-70
 - dynamically manipulating, 282-292
 - dynamically rearranging, 292-299
 - end tag, 72
 - form, 88-90
 - head, 72-77
 - image, 83-84
 - inline, 78-79
 - inspecting, 41-46
 - link, 82-83
 - list, 84
 - modifying flow, 133-134
 - obtaining and setting size, 273
 - obtaining information, 275-277
 - positioning, 273-274
 - positioning from CSS, 134-135
 - removing, 284
 - replacing, 285
 - syntax, 71
 - table, 85-88
 - tags, 71
 - toggling visibility, 286-287
 - event handlers, accessing, 147-148
 - response data, AJAX requests, 439-444
 - HTML 4.01 Strict, 70**
 - HTML 4.01 Transitional, 70**
 - html() method, 199, 561
 - html_errors.html listing (2.2), 42
 - HTML5, 12, 70, 103. See also HTML (HyperText Markup Language)**
 - elements, 93-102
 - dynamic web pages, 69-70
 - graphical, 93-101
 - HTTP (Hypertext Transform Protocol)**
 - GET requests, 15-16
 - headers, 15
 - POST requests, 16
 - protocols, 14-15
 - HTTPS (Hypertext Transfer Protocol Secure), protocols, 14-15**
- I**
- id attribute**
 - body elements,
 - DOM objects, 198
 - IDEs (integrated development environments), 21**
 - configuring Eclipse as, 22-26
 - if conditional logic, JavaScript, 157-160**
 - if_logic.html listing (5.3), 158-160
 - image elements, 83-84**
 - image gallery**
 - adding, 391-397
 - views, 337-338
 - image_fade.html listing (12.4), 331**
 - image_fade.js listing (12.5), 332**
 - image_fade.js listing (12.6), 332**
 - image_hide.css listing (12.3), 328-329**
 - image_hide.html listing (12.1), 328**
 - image_hide.js listing (12.2), 328**
 - image_slider.css listing (14.3), 397**
 - image_slider.html listing (14.1), 395**
 - image_slider.js listing (14.2), 395-396**
 - images**
 - adding a zoom view field to, 761-764
 - adding draggable to web pages, 498-501
 - fade, 331-332
 - hiding, 328
 - slider, 392-397
 - images.json listing (15.11), 440**
 - tag, 83-84**
 - indexOf() method, 177, 182**
 - inactive_table.css listing (14.6), 403-404**
 - inactive_table.html listing (14.4), 400-401**
 - inactive_table.js listing (14.5), 402-403**
 - inherited scope, 663**
 - inheritedData() method, 562**
 - initialization, page load events, 240-242**
 - Initiator item (network traffic analyzer), 64**

inject_builtin.html listing (21.2), 574
 inject_builtin.js listing (21.1), 572
 inject_custom.html listing (21.4), 576
 inject_custom.js listing (21.3), 576
 injector() method, 562
 injector service (AngularJS), 569
 inline elements, 78-79
 innerHeight() method, 273
 innerHeight property (window object), 304
 innerHTML attribute (DOM objects), 198
 innerWidth property (window object), 304
 input directive (AngularJS templates), 634
 <input> element, 232, 261, 309
 <input> tag, 89
 input.checkbox directive (AngularJS templates), 634
 input.date directive (AngularJS templates), 578
 input.dateTimeLocal directive (AngularJS templates), 578
 input.email directive (AngularJS templates), 634
 input.month directive (AngularJS templates), 578
 input.number directive (AngularJS templates), 634
 input.radio directive (AngularJS templates), 634
 input.text directive (AngularJS templates), 634
 input.time directive (AngularJS templates), 578
 input.url directive (AngularJS templates), 634

input.week directive (AngularJS templates), 634
 inspecting HTML elements, 41-46
 instances, JavaScript objects, creating new, 174
 interactions, jQuery UI
 jQuery.widget factory, 495-496
 mouse interaction widget, 496
 interactive animation, 341-344
 interactive tables, creating with sorting and filtering, 398-404
 interactive_animation.html listing (12.13), 341-343
 interactive_animation.js listing (12.14), 343-344
 interactive_animation.js listing (12.15), 344
 interrupting loops, JavaScript, 163
 isArray() method, 556
 isDate() method, 556
 isDefaultPrevented() method, 239
 isDefined() method, 556
 isElement() method, 556
 isFunction() method, 556
 isImmediatePropagationStopped() method, 239
 isNumber() method, 556
 isObject() method, 556
 isolate scope, 663-664
 isolateScope() method, 562
 isPropagationStopped() Method, 239
 isString() method, 556
 isUndefined() method, 556
 items option (sortable widget), 516
 iterating through arrays, 183

J

JavaScript, 9, 169, 215

adding elements dynamically, 282-283
 AJAX requests from, 428-429
 AngularJS expressions, 608-609
 animation, implementing in, 716-719
 cookies, obtaining and setting, 312
 creating sparkline graphics, 417-421
 data types, 152-153
 debugger, 51-63
 developer tools console, 35-40
 DOM (Document Object Model), 148-151, 201-204
 adding event handlers, 245-246
 error handling, 168-169, 244-248
 events, 235, 265-266
 creating custom, 262-263
 objects, 237-239
 page load for initialization, 240-242
 process, 236-237
 types, 239
 functions, creating, 163-167
 if conditional logic, 157-160
 implementing, 146-148
 integrating with AngularJS, 553-554
 looping, 160-163
 objects, 173, 195, 197
 accessing methods, 174
 accessing properties, 174

- adding methods to, 190-191
- assigning new values and methods, 174-175
- built-in, 175-189
- creating custom-defined, 189-194
- creating new instance, 174
- defining, 190
- prototyping patterns, 191-194
- syntax, 173-175
- operators, 154
 - arithmetic, 154
 - assignment, 154
 - comparison, 155-160
- overlay dialogs, 409-413
- pop-up boxes, 313-314
- strings, escape codes, 176
- syntax, 145, 151-169
 - objects, 173-175
- timers, 314-317
- variables
 - creating, 151-152
 - scope, 167-168
- JavaScript view (debugger), 57**
- join() method, 182**
- jQuery, 169, 215, 217, 232-233, 391**
 - adding
 - elements dynamically, 283
 - image gallery, 391-397
 - sparkline graphics, 417-421
 - advanced AJAX, 450-453
 - global event handlers, 451
 - global setup, 450
 - AJAX requests from, 429-433
 - AngularJS applications, using in, 560-564
 - animation, 321-325
 - applying promise() to, 325
 - CSS settings, 322
 - delaying, 324-325
 - moving elements, 340-344
 - queues, 323
 - resize, 337-339
 - sliding elements, 332-337
 - stopping, 323
 - visibility, 329-332
 - callbacks, 263-265
 - deferred objects, 265
 - cookies, obtaining and setting, 312
 - creating dynamic web pages, 28
 - creating tree view, 404-408
 - debugging, 63
 - DOM (Document Object Model)
 - accessing, 148-151, 211-213
 - traversing DOM, 231
 - obtaining and setting object attributes and properties, 271-272
 - .each() method, 224-229
 - event handlers, applying, 248-253
 - events, 235, 265-266
 - creating custom, 262-263
 - objects, 237-239
 - page load for initialization, 240-242
 - process, 236-237
 - types, 239
 - HTML elements, using selectors to access, 149-150
 - implementing, 146-148
 - graphical equalizer displays, 413-417
 - tables with sorting and filtering, 397-404
 - inserting elements, 285-286
 - integrating with AngularJS, 553-554
 - library, loading, 145-146
 - loading library, 563-564
 - .map() method, 225-229
 - objects, 197, 199
 - adding effects to, 478-482
 - chaining operations, 217-218
 - changing to DOM, 201
 - determining, 200
 - filtering results, 218-219
 - methods, 223
 - traversing DOM, 219-220
 - overlay dialogs, 409-413
 - replacing elements, 285
 - selectors, 205-215
 - applying basic, 205
 - attribute, 206
 - content, 207
 - filtered, 211
 - form, 209
 - hierarchy, 208-209
 - visibility, 210
 - simple interactive web page, 31-32
 - special effects, 321
 - syntax, 145
 - UI (user interface), 457, 472, 475, 492, 522, 525, 545
 - adding and removing unique IDs, 463
 - adding effects to class transitions, 482-485

- adding to element visibility transitions, 485-491
- adding to web pages, 461-462
- applying buttons to form controls, 528-530
- applying effects, 475-482
- applying to scripts, 463-472
- autocomplete widget, 527-528
- creating custom theme, 459-461
- creating custom widgets, 544-545
- CSS, 457-458
- datepicker widget, 530-531
- dialog widget, 532-533
- drag-and-drop, 497-507
- expandable accordion element, 526-527
- functionality, 463-464
- implementing stylized menus, 533-535
- JavaScript, 457-458
- obtaining library, 458-459
- positioning elements, 468-472
- progress bar widget, 535-536
- resizable widget, 507-511
- reviewing, 525
- selectable widget, 511-516
- selectors, 464-466
- slider widget, 536-538
- sortable widget, 516-522
- spinner widget, 538-539
- tabs widget, 539-542
- tooltips widget, 542-544

- UI widgets, 495
 - events, 495
 - interactions, 495-496
 - jQuery.widget factory, 495-496
 - methods, 495
 - mouse interaction widget, 496
- web forms
 - adding messages, 378-380
 - complex validation, 377-384
 - validation plug-in, 376
 - validation with HTML, 376-377

jQuery Lite, 561-562

- AngularJS applications, using in, 560-564
- jQuery methods supported, 561
- loading library, 563-564

jquery_effects.html listing (17.1), 481

jquery_effects.js listing (17.2), 481-482

jquery_effects.js listing (17.3), 482

jquery_image_adder.css listing (16.4), 468

jquery_image_adder.html listing (16.2), 467

jquery_image_adder.js listing (16.3), 467

jquery_selectors.css listing (7.6), 215

jquery_selectors.html listing (7.4), 214

jquery_selectors.js listing (7.5), 214-215

jquery_version.html listing (5.1), 148

jQuery.widget factory, 495-496

jqXHR object, 453

js_errors.html listing (2.1), 37

js_functions.html listing (5.4), 166-167

js_reversed_text.html listing (3.3), 75

JSON (JavaScript Object Notation)

- dynamic data generation, 20
- response data, AJAX requests, 438-441
- responses, AJAX requests, 427

json filter (AngularJS templates), 612

K

keyboard events, handling on AngularJS elements, 649

keydown event, 239

keypress event, 239

keyup event, 239

L

<label> element, 232

<label> tag, 89

large_title.html listing (24.4), 633

lastIndexOf() method, 177, 182

Latency item (network traffic analyzer), 64

Layout editor, 50-55

layout properties (CSS)

- adding padding around HTML content, 133
- applying, 119-140
- setting content size, 132-133

<legend> tag, 89

letter-spacing property (CSS), 114**libraries (jQuery)**

- loading, 145-146
- obtaining, 458-459

life cycle, AngularJS applications, 552-553

- scopes, 591-595

limitTo:limit filter (AngularJS templates), 612**line-height property (CSS), 115****link definition property (AngularJS), 658****link elements, adding, 82-83****link() function, 665-667****<link> tag, 76-77****links, external, 308**

- forcing to open in new browser windows, 308-311
- stopping, 308

list elements, applying, 84**list.html listing (1.1), 12****listings, 309, 382-386**

- ajax_post.css CSS Code That Styles the Page Elements (15.19), 448-449
- ajax_post.html HTML Document That Loads the jQuery and JavaScript (15.17), 447
- ajax_post.js jQuery and JavaScript Code That Implements the AJAX Request That Populates the Page and Updates the Server Data (15.18), 447-448
- ajax_response.css CSS Code That Styles the Page (15.7), 437
- ajax_response.html HTML Document That Creates the Form Dialog (15.5), 436

ajax_response.js jQuery and JavaScript That Sends the Form Request to the Server via an AJAX GET Request and Handles Success and Failure Conditions (15.6), 436-437

ajax.html (1.3), 19-20

animate.css CSS Code That Provides Transition Effects for the Various Class Stages of the AngularJS Animation Code (27.9), 720

animated_resize.css CSS Code That Styles the Images (12.12), 339

animated_resize.html HTML File Basic Web Used to Display the Images (12.10), 338-339

animated_resize.js jQuery and JavaScript Code That Implements the Resize Effect (12.11), 339

animation_effects.css CSS Code That Styles the Page (17.12), 491

animation_effects.html HTML Document That Adds the Web Page (17.10), 489-491

animation_effects.js jQuery and jQuery UI That Implements the Reposition Effects (17.11), 491

array_manipulation.html Example of Creating and Manipulating Array Objects in JavaScript (6.2), 186-187

article1.html Article HTML Code That Is Dynamically Loaded (15.4), 433

backgrounds.html HTML and CSS Code That Add Different Types of Background Styles to Elements (4.3), 124-125

broken_event.html HTML

File That Loads jQuery and JavaScript, Attaches Event Handlers Elements to Provide User Interaction, and Then Defines the <div> and <h1> Elements Used in the Example (9.1), 247

broken_event.js JavaScript Code That Defines an Event Handler and Dynamically Attaches It to the <div> Elements (9.2), 248

card_suits.css CSS Code That Styles the and <p> Elements (9.12), 261-262

card_suits.html HTML File That Loads CSS and JavaScript and Defines the <p>, text <input>, and Elements (9.10), 261

card_suits.js JavaScript Code That Triggers the keypress Event for the <input> Element from the Elements' Event Handler (9.11), 261

class_transitions.css CSS Code That Styles the Page (17.6), 485

class_transitions.html HTML Document That Adds the Web Page (17.4), 484

class_transitions.js jQuery and jQuery UI Code That Implements the Class Transitions with Animation Effects (17.5), 485

clock.html HTML File Basic Web Used to Display a Time Element (11.4), 316-317

clock.js jQuery and JavaScript Code That Implements a Timeout and Interval Timer (11.5), 317

- config_run_blocks.js
Implementing Configuration and Run Blocks in an AngularJS Module (21.5), 578-580
- cookies.css CSS Code That Styles the Buttons and Images (11.3), 312-313
- cookies.html HTML File Basic Web Page Used in the Example That Defines Several `` Elements Used for Buttons and `<input>` Elements to Input Cookie Names and Values (11.1), 309
- cookies.js jQuery and JavaScript Code That Gets, Sets, and Deletes Cookies (11.2), 312
- CSS Rules That Define Several Border Styles (4.5), 129-130
- css_errors.html (2.3), 51-52
- css_styling.html Adding CSS to an HTML Document Using the `<style>` Tag (3.4), 82
- custom_objects.html Example of Creating and Manipulating Custom Objects in JavaScript (6.3), 194
- data.html (1.4), 20
- date_picker.html HTML Document That Adds the jQuery UI Libraries and Renders a Date Picker (16.1), 462
- directive_angular_include.html An AngularJS Template That Uses the `ng-include` Directive to Change the Title Bar of the Page by Swapping Between Two HTML Files (24.2), 632
- directive_angular_include.js Implementing a Controller to Store the HTML Filename for a Title Element in the Scope (24.1), 630
- directive_bind.html An AngularJS Template That Implements Several Data Binding Directives (24.8), 644
- directive_bind.js Implementing a Controller with a Scope Model to Support Data Binding Directives (24.7), 644
- directive_custom_dom.html An AngularJS Template That Utilizes a Custom Directive That Manipulates the DOM (25.2), 670-671
- directive_custom_dom.js Implementing Custom Directives That Manipulate the DOM (25.1), 670
- directive_custom_photos.html An AngularJS Template That Implements Nested Custom Directives (25.6), 680-681
- directive_custom_photos.js Implementing Custom Directives That Interact with Each Other (25.5), 680
- directive_custom_zoom.html An AngularJS Template That Utilizes a Custom Directive to Provide Interactions with Mouse Events (25.4), 676
- directive_custom_zoom.js Implementing Custom Directives That Register with DOM Events (25.3), 674-676
- directive_focus_events.html An AngularJS Template That Implements the `ngFocus` and `ngBlur` Directives (24.10), 648-649
- directive_focus_events.js Implementing a Controller with Scope Data and Event Handlers to Support Blur and Focus Events from the View (24.9), 648
- directive_form.html An AngularJS Template That Implements Several Form Element Directives (24.6), 638
- directive_form.js Implementing a Controller for Form Directives (24.5), 638
- directive_keyboard_events.html An AngularJS Template That Implements the `ngKeydown` and `ngKeypress` Directives (24.12), 651
- directive_keyboard_events.js Implementing a Controller with Scope Data and Event Handlers to Support Key-Down and Key-Up Events from the View (24.11), 649-651
- directive_mouse_events.html An AngularJS Template That Implements the `ngClick` and Other Mouse Click and Move Event Directives (24.14), 654-655
- directive_mouse_events.js Implementing a Controller with Scope Data and Event Handlers to Support Mouse Click and Movement Events from the View (24.13), 654

- dom_manipulation.css CSS Code That Styles the and <p> Elements (8.3), 229
- dom_manipulation.css HTML File That Loads jQuery and JavaScript (8.1), 228
- dom_manipulation.js jQuery and JavaScript Code Gets the <p> Elements and Iterates Through Them Using .map() and .each() to Apply Different Changes for Each Element (8.2), 228-229
- dom_objects.css CSS That Styles <p> Elements (7.3), 204
- dom_objects.html HTML File That Loads JavaScript and Attaches an Event Handler to a Button Element to Update the Page (7.1), 204
- dom_objects.js JavaScript File Contains a Function Showing Examples of Accessing Variables by id, tag, and class Attributes (7.2), 204
- drag_n_drop.html HTML Document That Adds the Web Page (18.4), 503-505
- drag_n_drop.js CSS Code That Styles the Page (18.6), 506-507
- drag_n_drop.js jQuery and jQuery UI Implements Draggable and Droppable Elements (18.5), 506
- dragdrop.html AngularJS Template That Uses the dragit and dropit Directives to Add Draggable and Droppable Elements to the Web Page (29.6), 759-760
- dragdrop.js AngularJS Application That Implements dragit and dropit Custom AngularJS Directives to Provide Drag and Drop Functionality (29.5), 756-759
- draggable_images.css CSS Code That Styles the Page (18.3), 501
- draggable_images.html HTML Document That Adds the Web Page (18.1), 500
- draggable_images.js jQuery and jQuery UI Implements Draggable Images (18.2), 500-501
- dynamic_dialog.css CSS Code That Styles the Page, Overlay, and Dialog Elements (14.12), 412-413
- dynamic_dialog.html HTML Document That Implements the Page, Overlay, and Dialog (14.10), 410-411
- dynamic_dialog.js jQuery and JavaScript Code That Shows and Hides the Dialog and Updates the Web Page (14.11), 412
- dynamic_positioning.css CSS Code That Styles the Page (16.7), 472
- dynamic_positioning.html HTML Document That Adds the Images to the Web Page (16.5), 471
- dynamic_positioning.js jQuery and jQuery UI That Dynamically Positions the Images (16.6), 471-472
- dynamic_spark.css CSS Code That Styles the Page Elements and Sparkline (14.18), 421
- dynamic_spark.html HTML Document That Implements Page Elements (14.16), 418-420
- dynamic_spark.js jQuery and JavaScript Code Dynamically Populates and Updates the Sparklines (14.17), 420-421
- dynamic_tree.css CSS Code That Styles the Form Elements (14.9), 408
- dynamic_tree.html HTML Document That Implements the Root Tree Element (14.7), 407
- dynamic_tree.js jQuery and JavaScript Code Populates and Controls the Expansion and Collapsing of the Tree (14.8), 407-408
- expand_item.html AngularJS Partial Template That Defines the expandItem Element (29.12), 768
- expand_list.html AngularJS Partial Template That Defines the expandList Element (29.11), 767
- expand.html AngularJS Code That Styles and Implements Expandable/Collapsible Elements Using the expandList and expandItem Custom Directives (29.13), 768-770
- expand.js AngularJS Application That Implements the expandList and expandItem Custom Directive to Provide Expandable and Collapsible Elements (29.10), 766-767

- `expressions_basic.html` Applying Basic Strings and Numbers with Simple Math Operations to an AngularJS Template (23.2), 603-604
- `expressions_basic.js` Basic AngularJS Application Code with Empty Controller (23.1), 603
- `expressions_javascript.html` An AngularJS Template That Uses Expressions That Contain Arrays and Math Logic in Various Ways to Interact with Data from the Scope Model (23.5), 610
- `expressions_javascript.js` Building a Scope with Arrays and the Math Object That AngularJS Expressions Can Use (23.5), 610
- `expressions_scope.html` An AngularJS Template That Uses Expressions in Various Ways to Interact with Data from the Scope Model (23.4), 607
- `expressions_scope.js` Building a Scope That AngularJS Expressions Can Use (23.3), 606
- `filter_custom.html` An AngularJS Template That Uses a Custom Filter (23.12), 622
- `filter_custom.js` Implementing a Custom Filter Provider in AngularJS (23.11), 621-622
- `filter_sort.html` An AngularJS Template That Implements filter and orderBy Filters to Order and Filter Items in a Table View (23.10), 618-619
- `filter_sort.js` AngularJS Module that Builds a List of Planes and Provides Functionality to Sort and Order the List (23.9), 618
- `filters.html` An AngularJS Template That Implements Various Types of Filters to Modify Data Displayed in the Rendered View (23.8), 615
- `filters.js` Building a Scope That AngularJS Filters Can Use (23.7), 615
- `first.html` A Simple AngularJS Template That Provides Two Input Elements and a Button to Interact with the Model (20.1), 560
- `first.js` A Simple AngularJS Module That Implements a Controller to Support the Template in Listing 20.1 (20.2), 560
- `form_effects.html` HTML Document That Implements the Registration Form Used in the Example (13.7), 371-372
- `form_effects.js` CSS Code That Styles the Form Elements and Provide Classes for Dynamic Adjustments (13.9), 374
- `form_effects.js` jQuery Code Implements the Animated Visual Elements (13.8), 372-373
- `form_flow.css` CSS Code That Styles the Payment Form Elements (13.6), 367-368
- `form_flow.html` HTML Document That Implements the Payment Form Used in the Example (13.4), 365-366
- `form_flow.js` jQuery Code That Provides the Intelligent Flow Control for the Payment Form (13.5), 366-367
- `form_manipulation.css` CSS Code That Styles the Form Elements (13.3), 360
- `form_manipulation.html` HTML Document That Implements the Form Elements Used in the Example (13.1), 358-359
- `form_manipulation.js` jQuery and JavaScript Code That Implements a Series of Event Handlers That Read Data from an Element in One Form as It Changes and Updates the Second (13.2), 359-360
- `form_validation.css` CSS Code That Styles the Form Elements and Errors (13.12), 386-387
- `form_validation.html` HTML Document That Implements the Registration Form Used in the Example (13.10), 382-386
- `form_validation.js` jQuery Code Implements the Validation of Form Elements (13.11), 386
- `forms.html` HTML Generating a Form with Text, Radio, and Select Inputs (3.6), 90
- `graphic_equalizer.css` CSS code That Styles Elements to Render the Graphical Equalizer (14.15), 417
- `graphic_equalizer.html` HTML Document That Implements a Web Page (14.13), 415-416

- graphic_equalizer.js jQuery and JavaScript Code Dynamically Build and Populate the Graphical Equalizer (14.14), 416
- HTML That Creates a Series of <div> Elements That Are Styled by Listing 4.5 (4.4), 129
- html_errors.html (2.2), 43
- if_logic.html Simple Example of Using Conditional Logic Inside JavaScript (5.3), 158-160
- image_fade.html HTML File Basic Web Used to Display the Images (12.4), 331
- image_fade.js CSS Code That Styles the Collapsible Image Element (12.6), 332
- image_fade.js jQuery and JavaScript Code That Implements Image Selection Fades (12.5), 332
- image_hide.css CSS Code That Styles the Collapsible Image Element (12.3), 328-329
- image_hide.html HTML File Basic Web Used to Display the Collapsible Image Element (12.1), 328
- image_hide.js jQuery and JavaScript Code That Implements the Collapsible Image (12.2), 328
- image_slider.css CSS Code That Styles the Images and Controls (14.3), 397
- image_slider.html HTML Document That Implements the Slider, Control, and Image Elements (14.1), 395
- image_slider.js jQuery and JavaScript Code Implements the Mouse Event Handlers for the Image Slider Controls and Thumbnails (14.2), 395-396
- images.json JSON Data from the Book Website at lesson15/data/images.json Containing Image Filenames and Captions (15.11), 440
- inertive_table.css CSS Code That Styles the Table Elements (14.6), 403-404
- inertive_table.html HTML Document That Implements the Table Elements Used in the Example (14.4), 400-401
- inertive_table.js jQuery and JavaScript Code Define the Interactions of the Table, Including Sorting and Filtering (14.5), 402-403
- inject_builtin.html Using HTML Code to Implement an AngularJS Module That Implements Dependency Injection (21.2), 574
- inject_builtin.js Implementing Dependency Injection of Built-in Services in a Controller (21.1), 572
- inject_custom.html Using HTML Code to Implement an AngularJS Module That Depends on Another Module (21.4), 576
- inject_custom.js Implementing Dependency Injection in Controller and Module Definitions (21.3), 576
- interactive_animation.html HTML File Basic Web Used to Display the Controls, Cones, and Flying Saucer (12.13), 341-343
- interactive_animation.js CSS Code That Styles the Controls, Cones, and Flying Saucer (12.15), 344
- interactive_animation.js jQuery and JavaScript Code That Implements the Flying Saucer Movement Animation Using Click Handlers (12.14), 343-344
- JavaScript and HTML Code That Draws a Cube onto a <canvas> Element (3.8), 100-101
- JavaScript and HTML Code That Uses <svg> Elements to Create a Pie Graph and a Text Border Around a Clock (3.7), 98-99
- jquery_effects.html HTML Document That Adds the Web Page (17.1), 481
- jquery_effects.js CSS Code That Styles the Page (17.3), 482
- jquery_effects.js jQuery and jQuery UI That Apply Several Effects on Images (17.2), 481-482
- jquery_image_adder.css CSS Code That Styles the Page Elements (16.4), 468
- jquery_image_adder.html HTML Document That Adds the Web Page (16.2), 467
- jquery_image_adder.js jQuery and jQuery UI Code That Uses the :data() Selector to Select Elements (16.3), 467
- jquery_selectors.css CSS That Styles Elements and Elements with class="label" (7.6), 215

- jquery_selectors.html HTML File That Loads jQuery and JavaScript and Attaches Event Handlers Elements to Provide User Interaction (7.4), 214
- jquery_selectors.js JavaScript File Containing Event Handler Functions That Use jQuery in Various Ways to Select and Alter Page Elements (7.5), 214-215
- jquery_version.html Very Basic Example of Loading Using jQuery in a Web Page to Print Out Its Own Version (5.1), 148
- js_errors.html (2.1), 37
- js_functions.html Simple Example of JavaScript Functions (5.4), 166-167
- js_reversed_text.html Adding JavaScript and jQuery to an HTML Document Using the `<script>` Tag (3.3), 75
- large_title.html A Partial HTML File That Contains the Large Version of the Title (24.4), 633
- list.html (1.1), 12
- load_content.css CSS Code That Styles the Page (15.3), 433
- load_content.html HTML Document That Adds Menu and Content (15.1), 432
- load_content.js jQuery and JavaScript That Implements the AJAX `.load()` Requests (15.2), 432
- load_json.css CSS Code That Styles the Images (15.12), 441
- load_json.html HTML Document That Loads the jQuery and JavaScript (15.9), 440
- load_json.js jQuery and JavaScript Code That Implements the AJAX Request and Handles the JSON Response (15.10), 440
- load_xml.css CSS Code That Styles the Table (15.16), 444
- load_xml.html HTML Document That Loads the jQuery and JavaScript (15.13), 443
- load_xml.js jQuery and JavaScript Code That Implements the AJAX Request and Handles the XML Response (15.14), 443
- manual_event.css CSS Code That Styles the `` Elements (9.9), 258
- manual_event.html HTML File That Loads CSS and JavaScript and Defines the Check Boxes and `` Elements (9.7), 257
- manual_event.js JavaScript Code Manually Triggers the Click Event for the Check Box Elements from the `` Elements (9.8), 257-258
- my_photos.html A Partial AngularJS Template That Provides the Root Element for the myPhotos Custom Directive (25.7), 681
- page_title.html JavaScript Code Changing the `<title>` Value After the Page Has Loaded (3.1), 73
- pane.html AngularJS Partial Template That Contains the Template Code to Build the Individual Panes of the Tabbed Container (29.3), 754
- parkdata.xml XML Data File with Raw Table Data (15.15), 444
- rating.html AngularJS Template That Utilizes Data from the Scope to Display a List of Images with Descriptions and Ratings (29.15), 772-773
- rating.js AngularJS Application That Provides the Data and Functionality to Support Star Ratings in the View (29.14), 770-772
- rearranging_elements.css CSS Code That Styles the Buttons and Images (10.9), 299
- rearranging_elements.html HTML File Basic Web Page Used in the Example That Defines Several `` Elements Used for Buttons and `` Elements (10.7), 297
- rearranging_elements.js jQuery and JavaScript Code That Dynamically Moves, Resizes, and Adjusts the z-index of Several `` Elements (10.8), 297-298
- resizable_elements.css CSS Code That Styles the Page (18.9), 510-511
- resizable_elements.html HTML Document That Adds the Web Page (18.7), 510

- resizable_elements.js jQuery and jQuery UI Implements Resizing and Moving the Page Elements (18.8), 510
- reversed_text.html Adding CSS to an HTML Document Using the <style> Tag (3.2), 74
- run_blocks.html Using HTML Code to Display the configTime and runTime Values Generated in the Configuration and Run Blocks of the AngularJS Module (21.6), 580
- scope_controller.html HTML Template That Enables You to See the Data in the Scope Change Dynamically Based on Incrementing and Decrementing Values (22.2), 586-587
- scope_controller.js Implementing a Basic Controller That Uses Dependency Injection, Initializes Scope Values, and Implements Business Logic (22.1), 586
- scope_events.html HTML Template Code That Renders the Scope Hierarchy for Listing 26.3 Controllers (26.4), 694-695
- scope_events.js Implementing \$emit() and \$broadcast() Events Within the Scope Hierarchy (26.3), 694-695
- scope_hierarchy.js Implementing a Basic Scope Hierarchy with Access to Properties at Each Level (22.5), 595
- scope_template.html HTML Template Code That Implements a Controller and Various HTML Fields Linked to the Scope (22.4), 590
- scope_watch.html HTML Template Code That Provides the View and Interactions with the Scope and Controller Defined in Listing 26.1 (26.2), 689-690
- scope_watch.js Implementing \$watch(), \$watchGroup(), and \$watchCollection() Handlers to Watch the Value of Scope Variables (26.1), 689
- selectable_sets.html HTML Document That Adds the Web Page (18.10), 514
- selectable_sets.js CSS Code That Styles the Page (18.12), 515-516
- selectable_sets.js jQuery and jQuery UI Implements Item Selection (18.11), 515
- server_lesson15_ajax_handling.js Node.js Server That Will Handle the POST and GET Requests for This Exercise (15.8), 437
- server_lesson15_ajax_post.js Node.js Server That Will Handle the POST and GET Requests for This Exercise (15.20), 449-450
- server.js (1.5), 28
- service_animate.html An AngularJS Template That Implements Buttons That Change the Class on an Image to Animate Fading and Resizing (28.8), 719-720
- service_animate.js Implementing an AngularJS Controller That Implements jQuery Animation Using the \$animate Service (27.7), 719
- service_cache.js Implementing a \$cacheFactory Service in an AngularJS Application (27.4), 704-709
- service_cookie.html An AngularJS Template That Implements Radio Buttons to Set a Cookie Value (27.6), 712
- service_cookie.js Implementing an AngularJS Controller That Interacts with Browser Cookies by Using the \$cookieStore Service (27.5), 711-712
- service_custom_censor.html AngularJS Template That Illustrates the Interaction of Multiple Custom Services in an AngularJS Controller (28.2), 736-737
- service_custom_censor.js Implementing and Consuming Multiple Custom Services in an AngularJS Controller (28.1), 735-736
- service_custom_db_access.js Implementing a Custom AngularJS Service That Utilizes the \$http and \$q Services to Provide Interaction with Data Stored on the Server (28.6), 744
- service_custom_db.html AngularJS Template That Uses a Series of <input> Elements to Display and Update Data Retrieved from the Server (28.8), 746-747

- service_custom_db.js
Implementing an AngularJS Application That Injects the Module and Service from Listing 28.6 to Utilize the Database Access Service (28.7), 745
- service_custom_time.html
AngularJS Template That Illustrates Injecting a Custom AngularJS Service into Multiple Controllers (28.4), 739-740
- service_custom_time.js
Implementing and Consuming a Custom AngularJS Service in Multiple Controllers (28.7), 737-739
- service_db_server.js
Implementing a Node.js Express Server That Supports GET and POST Routes to Simulate a Database Service for the AngularJS Controller (28.5), 742-743
- service_http.html
An AngularJS Template That Implements Directives That Are Linked to Web Server Data (27.3), 707
- service_http.js
Implementing an AngularJS Controller That Interacts with the Web Server Using the \$http Service (27.2), 706
- service_location.html
An AngularJS Template That Displays Information Gathered from the \$location Service and Provides Links to Change the path, search, and hash Values (27.11), 725
- service_location.js
An AngularJS Application That Implements a Controller to Gather Information from the \$location Service and Provides Functions to Change the path, search, and hash Values (27.10), 727
- service_server.js
Implementing an Express Server That Supports GET and POST Routes for an AngularJS Controller (27.1), 705
- Simple Interactive jQuery and JavaScript Web Page (1.6), 31-32
- sliding_images.css
CSS Code That Styles the Sliding Menu (12.9), 336-337
- sliding_images.html
HTML File Basic Web Used to Display the Sliding Menu Element (12.7), 335-336
- sliding_images.js
jQuery and JavaScript Code That Implements the Sliding Image Menu (12.8), 336
- small_title.html
A Partial HTML File That Contains the Small Version of the Title (24.3), 632
- sortable_elements.css
CSS Code That Styles the Page (18.15), 521-522
- sortable_elements.html
HTML Document That Adds the Web Page (18.13), 518-520
- sortable_elements.js
jQuery and jQuery UI Implements Sorting (18.14), 521
- string_manipulation.html
Example of Combining Multiple Lines of Text into a Single String and Using replace() to Fill in Specifically Formatted Sections of the String with Variable Values (6.1), 181
- style.html (1.2), 13-14
- tabbable.html
AngularJS Template That Implements the myTabs and myPane Custom Directives to Create a Tabbed View (29.4), 754-755
- tabbable.js
AngularJS Application That Defines Two Custom Directives That Can Be Nested to Provide a Tabbed Panel View (29.1), 753
- tables.html
HTML Generating a Table with Headers, Rows, and Columns (3.5), 87-88
- tabs.html
AngularJS Partial Template That Contains the Template Code to Build the Tabs Container (29.2), 754
- text_styles.css
CSS Code That Stylizes the Various Textual Elements by Adjusting the Color, Alignments, Adding Lines, and Adjusting the Spacing (4.2), 118-119
- text_styles.html
HTML Code with Several Paragraph Elements to Be Stylized (4.1), 118
- traverse_dom.css
CSS Code That Styles the , <input>, and <label> Element (8.6), 232
- traverse_dom.html
HTML File That Loads jQuery and JavaScript and Attaches Event Handler Elements to Provide User Interaction (8.4), 229-232

- [traverse_dom.js](#) JavaScript Code That Handles the Key Up Event and Uses jQuery to Manipulate the Color of the `` Elements Based on the Input Value (8.5), 232
 Very Basic Example of Using JavaScript and jQuery to Access DOM Elements (5.2), 151
- [visibility_transitions.css](#) CSS Code That Styles the Page (17.9), 488-489
- [visibility_transitions.html](#) HTML Document That Adds the Web Page (17.7), 490-488
- [visibility_transitions.js](#) jQuery and jQuery UI That Implements the Visibility and Effects (17.8), 488
- [web_element_manipulation.css](#) CSS Code That Styles the Banner, Buttons, and Other Elements (10.6), 291-292
- [web_element_manipulation.html](#) HTML File Basic Web Page Used in the Example (10.4), 290
- [web_element_manipulation.js](#) jQuery and JavaScript Code That Dynamically Builds the Left Navigation Items Based on the Button Clicked in the Top Menu (10.5), 290-291
- [web_layout.css](#) CSS Code Used to Apply a Page Layout That Places Elements in Fixed Positions (4.7), 139-140
- [web_layout.html](#) HTML That Creates a Pair of `<div>` Elements That Are Styled by Listing 4.7 to Be Song Info and a Set of Playback Controls (4.6), 136-139
- [web_page_manipulation.css](#) CSS Code That Styles the `<div>` and Other Elements (10.3), 281
- [web_page_manipulation.html](#) HTML File That Provides Several Elements to Play with, as Well as an Independent `<div>` That Displays Info (10.1), 279-280
- [web_page_manipulation.js](#) jQuery and JavaScript Code That Retrieves and Displays Information About the Screen, Browser, Mouse, and HTML Elements (10.2), 280-281
- [widgets_accordian.html](#) jQuery, CSS, and HTML to Implement the Accordion (19.1), 527
- [widgets_autocomplete.html](#) jQuery, CSS, and HTML to Implement the Autocomplete Field (19.2), 528
- [widgets_calendar.html](#) jQuery, CSS, and HTML to Implement the Datepicker Widget (19.4), 531
- [widgets_custom.html](#) jQuery Code Outline to Implement a Custom Widget (19.12), 544-545
- [widgets_dialogs.html](#) jQuery, CSS, and HTML to Implement the Dialog (19.5), 533
- [widgets_menus.html](#) jQuery, CSS, and HTML to Implement the Menus (19.6), 534-535
- [widgets_progress_bars.html](#) jQuery, CSS, and HTML to Implement the Progress Bar (19.7), 536
- [widgets_slider_bars.html](#) jQuery, CSS, and HTML to Implement the Sliders (19.8), 537-538
- [widgets_spinner.html](#) jQuery, CSS, and HTML to Implement the Spinner (19.9), 539
- [widgets_tabs.html](#) jQuery, CSS, and HTML to Implement the Tabbed Panel (19.10), 541-542
- [widgets_tooltips.html](#) jQuery, CSS, and HTML to Implement the Tabbed Panel (19.11), 543-544
- [working_event.css](#) CSS Code That Styles the `<div>` Elements (9.6), 252
- [working_event.html](#) HTML File That Loads jQuery and JavaScript, Attaches Event Handlers Elements to Provide User Interaction, and Defines the `<div>` and `<h1>` Elements Used in the Example (9.4), 251-252
- [working_event.js](#) jQuery and JavaScript Code That Defines an Event Handler and Dynamically Attaches It to the `<div>` Elements (9.5), 252
- [zooming.html](#) AngularJS Template That Styles and Implements the `<zoomit>` Custom AngularJS Directive (29.9), 763-764
- [zooming.js](#) AngularJS Application That Defines a Custom AngularJS Directive Called `zoomit` That Implements an `` Element with a Zoom View Field (29.7), 762-763

zoomit.html AngularJS Partial Template That Implements the and <div> Elements for the Image and Zoom View Field (29.8), 763

lists, browsers

- altering appearance, 13-14
- rendering in, 12

load event, 239

load_content.css listing (15.3), 433

load_content.html listing (15.1), 432

load_content.js listing (15.2), 432

load_json.css listing (15.12), 441

load_json.html listing (15.9), 440

load_json.js listing (15.10), 440

load_xml.css listing (15.16), 444

load_xml.html listing (15.13), 443

load_xml.js listing (15.14), 443

loading library, jQuery, 145-146

looping, JavaScript, 160-163

lowercase filter (AngularJS templates), 612

lowercase() method, 556

low-level AJAX requests, 451-453

M

manual_event.css listing (9.9), 258

manual_event.html listing (9.7), 257

manually triggering events, 253

- JavaScript, 254-258
- jQuery, 258-262

manually validating web forms, 375

margins, adding around HTML elements, 133

match() method, 177

Math object (JavaScript), 188

max option (slider widget), 526

max rule (validation), 376

maxlength rule (validation), 376

media elements, 102

menus

- sliding animations, 333-334
- stylized, implementing, 533-535

messages, web form validation, 378-384

<meta> tag, 73-74

metaKey attribute (event objects), 237

Method item (network traffic analyzer), 64

method property (config parameter), \$http service requests, 702

methods, 177

- \$broadcast() method, 691
- \$emit(), 691
- \$on(), 691-692
- \$watch(), 687, 689
- \$watchCollection(), 688-690, 689
- \$watchGroup(), 687, 689
- .animate(), 322
- abort(), 453
- absUrl(), 722
- addClass(), 199, 561
- addEventListener(), 253
- after(), 561
- ajax(), 450-452
- ajaxComplete(), 451
- ajaxError(), 451
- ajaxSend(), 451
- ajaxSetup(), 450
- ajaxStart(), 451
- ajaxStop(), 451
- ajaxSuccess(), 451
- always(), 453
- angular.module(), 569-570
- append(), 561
- appendChild(), 198
- Array object, 182
- attr(), 199, 561
- bind(), 561
- charAt(), 177
- charCodeAt(), 177
- children(), 561
- click(), 198
- clone(), 561
- compile(), 665-667
- concat(), 177, 182
- config(), 578
- contents(), 561
- controller(), 562
- copy(), 556
- css(), 199, 561
- data(), 561
- destroy(), 495
- detach(), 561
- directive(), 683
- directive(), 657-658
- disable(), 495
- DOM objects, 198
- done(), 453
- element(), 556
- empty(), 561
- enable(), 495
- eq(), 561
- equals(), 556
- events, 239
- extend(), 556
- fadeOut(), 329
- fadeTo(), 330-331
- fadeToggle(), 329-330
- fail(), 453
- filter(), 620

- find(), 561
- forEach(), 556
- fromCharCode(), 177
- fromJson(), 556
- getAllResponseHeaders(), 453
- getAttribute(), 198
- getResponseHeader(), 453
- hasClass(), 561
- hash(), 722
- height(), 199, 273
- hide(), 199, 325
 - animating, 325-329
- history.back(), 308
- history.forward(), 307
- host(), 722
- html(), 199, 561
- indexOf(), 177, 182
- inheritedData(), 562
- injector(), 562
- innerHeight(), 273
- isArray(), 556
- isDate(), 556
- isDefined(), 556
- isElement(), 556
- isFunction(), 556
- isNumber(), 556
- isObject(), 556
- isolateScope(), 562
- isString(), 556
- isUndefined(), 556
- JavaScript objects
 - accessing, 174
 - adding to, 190-191
 - assigning new, 174-175
- join(), 182
- lastIndexOf(), 177, 182
- link(), 665-667
- lowercase(), 556
- match(), 177
- Number object, 175
- off(), 250, 561
- on(), 249, 561
- one(), 561
- onloadHandler(), 240
- open(), 428
- option(), 495
- outerHeight(), 273
- outerWidth(), 273
- parent(), 561
- path(), 722
- pop(), 182
- port(), 722
- position(), 468-472
- prepend(), 561
- promise(), 325
- prop(), 561
- protocol(), 722
- push(), 182
- ready(), 561
- remove(), 561
- removeAttr(), 561
- removeClass(), 561
- removeData(), 561
- replace(), 177, 722
- replaceWith(), 561
- reverse(), 182
- scope(), 562
- scrollParent(), 463
- search(), 177, 722
- send(), 428
- setAttribute(), 198
- setRequestHeader(), 453
- shift(), 182
- show(), 199, 325
 - animating, 326-327
- slice(), 177, 182
- slideDown(), 332-333
- slideToggle(), 332-333
- slideUp(), 332-333
- sort(), 182
- splice(), 182
- split(), 177
- String object, 177
- substr(), 177
- substring(), 177
- text(), 561
- toggle(), animating, 326-327
- toggleClass(), 561
- toLowerCase(), 177
- toString(), 182
- toUpperCase(), 177
- triggerHandler(), 561
- unbind(), 561
- uniqueID(), 463
- unshift(), 182
- uppercase(), 556
- url(), 722
- val(), 199, 561
- validate(), 376-387
- valueOf(), 177, 182
- widget(), 495
- width(), 199, 273
- window object, 305, 307
- wrap(), 562
- x.toExponential(), 175
- x.toFixed(2), 175
- x.toPrecision(5), 175
- x.toString(), 175
- x.valueOf(), 175
- zIndex(), 463
- min rule (validation), 376**
- minlength rule (validation), 376**
- modal mutation phase (AngularJS scope), 592**
- modal option (dialog widget), 532**
- Model View Controller (MVC). See MVC (Model-View Controller)**
- modules, AngularJS, 549, 567-568**
 - adding configuration blocks, 575-578

- adding run blocks, 578
- creating providers, 570-572
- defining module object, 569-570
- injecting into another, 575-577
- providers, 569
- mouse events, handling on
 - AngularJS elements, 652-653
- mouse interaction widget, 496
- mouse position, obtaining, 270
- mousecenter event, 239
- mousedown event, 239
- mouseleave event, 239
- mousemove event, 239
- mouseout event, 239
- mouseover event, 239
- mouseup event, 239
- moveBy() method (window object), 305
- moveTo() method (window object), 305
- multiElement definition property (AngularJS), 658
- MVC (Model-View Controller), 547
 - implementation, 548-549
- my_photos.html listing (25.7), 681
- MySQL, 424

N

- name property (window object), 304
- nested controllers
 - custom events, 692-693
 - scopes, 593-595
- nested directives, implementing, 677-680
- network traffic, analyzing, 63-65
- ngApp directive (AngularJS templates), 628
- ngBind directive (AngularJS templates), 640
- ngBindHtml directive (AngularJS templates), 640
- ngBindTemplate directive (AngularJS templates), 640
- ngBlur directive (AngularJS templates), 646
- ngChange directive (AngularJS templates), 646
- ngChecked directive (AngularJS templates), 646
- ngClass directive (AngularJS templates), 640
- ngClassEven directive (AngularJS templates), 640
- ngClassOdd directive (AngularJS templates), 640
- ngClick directive (AngularJS templates), 646
- ngCloak directive (AngularJS templates), 628
- ngController directive (AngularJS templates), 629
- ngCopy directive (AngularJS templates), 646
- ngCut directive (AngularJS templates), 646
- ngDbclick directive (AngularJS templates), 646
- ngDisabled directive (AngularJS templates), 640
- ngFocus directive (AngularJS templates), 646
- ngHide directive (AngularJS templates), 641
- ngHref directive (AngularJS templates), 629
- ngIf directive (AngularJS templates), 641
- ngInclude directive (AngularJS templates), 629
- ngInit directive (AngularJS templates), 642
- ngKeydown directive (AngularJS templates), 646
- ngKeyPress directive (AngularJS templates), 646
- ngKeyUp directive (AngularJS templates), 646
- ngList directive (AngularJS templates), 629
- ngModel directive (AngularJS templates), 641
- ngMouseDown directive (AngularJS templates), 646
- ngMouseenter directive (AngularJS templates), 646
- ngMouseleave directive (AngularJS templates), 646
- ngMouseMove directive (AngularJS templates), 646
- ngMouseover directive (AngularJS templates), 646
- ngMouseup directive (AngularJS templates), 646
- ngNonBindable directive (AngularJS templates), 629
- ngOpen directive (AngularJS templates), 629
- ngOptions directive (AngularJS templates), 634
- ngPaste directive (AngularJS templates), 646
- ngPluralize directive (AngularJS templates), 629
- ngReadOnly directive (AngularJS templates), 629
- ngRepeat directive (AngularJS templates), 641
- ngRequired directive (AngularJS templates), 629
- ngSelected directive (AngularJS templates), 629
- ngShow directive (AngularJS templates), 641

- ngSrc directive (AngularJS templates), 629
 - ngSrcset directive (AngularJS templates), 629
 - ngStyle directive (AngularJS templates), 642
 - ngSubmit directive (AngularJS templates), 646
 - ngSwipeLeft directive (AngularJS templates), 646
 - ngSwipeRight directive (AngularJS templates), 646
 - ngSwitch directive (AngularJS templates), 642
 - ngTransclude directive (AngularJS templates), 629
 - ngValue directive (AngularJS templates), 642
 - ngView directive (AngularJS templates), 629
 - Node.js, 424
 - configuring, 21-22
 - creating Express web servers, 26-29
 - <noscript> tag, 76
 - Null data type (JavaScript), 153
 - Number data type (JavaScript), 153
 - Number object (JavaScript), 175-176
 - number rule (validation), 376
 - number[:fraction] filter (AngularJS templates), 612
 - numberOfMonths option (datepicker widget), 530
- O**
- objects
 - browser history, 307-308
 - browser location, 306-307
 - chaining operations, jQuery, 217-218
 - determining, 200
 - events, 237-239
 - attributes, 237
 - filtering results, jQuery, 218-219
 - jQuery, adding effects to, 478-482
 - jQuery Lite, events and methods, 562
 - jqXHR, 453
 - screen, 303-304
 - window, 304-305
 - methods, 305
 - properties, 304
 - window.location, providing wrapper for, 721-724
 - XMLHttpRequest, 428
 - objects (DOM), 197-198
 - accessing, 201-204
 - jQuery, 211-213
 - finding by class name, 201
 - finding by ID, 201
 - finding objects by tag name, 202-203
 - objects (JavaScript), 173, 195
 - accessing methods, 174
 - accessing properties, 174
 - adding methods to, 190-191
 - Array, 181-187
 - assigning new values and methods, 174-175
 - built-in, 175-189
 - creating new instance, 174
 - custom-defined, creating, 189-194
 - Date, 187-188
 - defining, 190
 - Math, 188
 - Number, 175-176
 - prototyping patterns, 191-194
 - RegExp, 189
 - String, 176-178
 - syntax, 173-175
 - off() method, 250, 561
 - on() method, 249, 561
 - one() method, 561
 - onload event, 240-242
 - onloadHandler() method, 240
 - onreadystatechange attribute (XMLHttpRequest object), 428
 - onSelect option (datepicker widget), 530
 - opacity design properties (CSS), 131
 - opacity option (draggable widget), 497
 - opacity option (sortable widget), 516
 - open() method, 305, 428
 - opener property (window object), 304
 - operators, JavaScript, 154
 - arithmetic, 154
 - assignment, 154
 - comparison, 155-160
 - option() method, 495
 - <option> tag, 89
 - orderBy:exp:reverse filter (AngularJS templates), 612
 - ordering, AngularJS template filters, 616-618
 - orientation option (slider widget), 536
 - outerHeight() method, 273
 - outerHeight property (window object), 304
 - outerHTML attribute (DOM objects), 198
 - outerWidth() method, 273

outerWidth property (window object), 304
 overflow property (CSS), 135-136
 overlay dialogs, 409-413

P

<p> element, 261
 padding, adding around HTML content, 133
 page events, binding to controllers, 645-653
 page load events, initialization, 240-242
 page_title.html listing (3.1), 73
 pageX attribute (event object), 270
 pageXOffset property (window object), 304
 pageY attribute (event object), 270
 pageYOffset property (window object), 304
 pane.html listing (29.3), 754
 params property (config parameter), \$http service requests, 702
 parent() method, 561
 parent property (window object), 304
 parent scope hierarchy, emitting and broadcasting custom, 691
 parentNode attribute (DOM objects), 198
 parkdata.xml listing (15.15), 444
 path() method, 722
 <path> element (HTML5), 95
 pathname property (window object), 307
 patterns, JavaScript objects, prototyping, 191-194

Pause on Exceptions command (debugger), 57
 PHP (Hypertext Preprocessor), 425
 pixelDepth property (screen object), 304
 placeholder option (sortable widget), 516
 pop() method, 182
 pop-up boxes, adding, 313-314
 port() method, 722
 port property (window object), 307
 position() method, 468-472
 POST requests
 versus GET, 427
 HTTP (Hypertext Transform Protocol), 16, 32
 preventDefault() method, 239
 prepend() method, 561
 Preview tab (network traffic analyzer), 65
 print() method (window object), 305
 priority definition property (AngularJS), 658
 progress bars, web pages, 535-536
 promise() method, 325
 prompt() method (window object), 305
 prop() method, 561
 properties
 CSS (Cascading Style Sheets), obtaining and setting, 272-273
 JavaScript objects, accessing, 174
 screen object, 304
 window object, 304, 307
 protocol() method, 722
 protocol property (window object), 307

protocols
 HTTP (Hypertext Transform Protocol), 14-15
 HTTPS (Hypertext Transfer Protocol Secure), 14-15
 prototyping patterns, JavaScript objects, 191-194
 providers (AngularJS modules), 569
 creating, 570-572
 implementing, 572
 injecting into controller, 572
 puff effect (jQuery UI), 476
 pulsate effect (jQuery UI), 476
 push() method, 182
 PUT requests, sending, \$http service, 701-708

Q-R

queue option (jQuery animation), 322
 queues, animation, 323

 radio input elements, web forms, accessing, 350-351
 range option (slider widget), 537
 range rule (validation), 376
 ranglength rule (validation), 376
 rating.html listing (29.15), 772-773
 rating.js listing (29.14), 770-772
 ratings, elements, adding to, 770-773
 ready() method, 561
 readyState attribute (jqXHR object), 453
 rearranging, elements, 292-299
 rearranging_elements.css listing (10.9), 299

- rearranging_elements.html listing (10.7), 297
 - rearranging_elements.js listing (10.8), 297-298
 - RegExp object (JavaScript), 189
 - relatedTarget attribute (event objects), 237
 - reload() method (window object), 307
 - remote rule (validation), 376
 - remove() method, 561
 - removeAttr() method, 561
 - removeClass() method, 561
 - removeData() method, 561
 - reoccurring timers, adding, 315-316
 - replace() method, 177, 307, 722
 - replaceWith() method, 561
 - requests (AJAX), 423
 - advanced jQuery, 450-453
 - global event handlers, 451
 - global setup, 450
 - cross-domain, 426
 - GET versus POST, 427
 - handling responses, 433-444
 - HTML response data, 439-444
 - JSON response data, 438-441
 - success and failures, 434-437
 - XML response data, 439-444
 - implementing, 428-450
 - from JavaScript, 428-429
 - from jQuery, 429-433
 - low-level, 451-453
 - versus page requests, 424-425
 - response data types, 427
 - server-side services, 425
 - require definition property (AngularJS), 658
 - required rule (validation), 376
 - reset, web forms, controlling, 362-364
 - reset event, 239
 - resizable widget (jQuery), 507-511
 - resizable_elements.css listing (18.9), 510-511
 - resizable_elements.html listing (18.7), 510
 - resizable_elements.js listing (18.8), 510
 - resizable elements, 508-510
 - resize animations, creating, 337-339
 - resize event, 241, 508
 - resizeBy() method (window object), 305
 - resizestart event (jQuery resizable widget), 508
 - resizestop event (jQuery resizable widget), 508
 - resizeTo() method (window object), 305
 - response attribute (XMLHttpRequest object), 428
 - response data types, AJAX requests, 427
 - Response tab (network traffic analyzer), 65
 - responses (AJAX), handling, 433-444
 - HTML response data, 439-444
 - JSON response data, 438-441
 - success and failures, 434-437
 - XML response data, 439-444
 - responseText attribute (XMLHttpRequest object), 428
 - responseType property (config parameter), \$http service requests, 702
 - restrict definition property (AngularJS), 658
 - results attribute (event objects), 237
 - Resume option (debugger), 58
 - reverse() method, 182
 - reversed_text.html listing (3.2), 74
 - revert option (draggable widget), 497
 - root scope, relation between applications, 584-586
 - rules, web form validation, 377-378
 - run blocks, applying to modules, 578
 - run_blocks.html listing (21.6), 580
 - runtime data binding phase, AngularJS life cycle, 552-553
- ## S
- scale effect (jQuery UI), 476
 - scope, JavaScript variables, 167-168
 - scope change events, AngularJS applications, tracking with \$watches, 686-689
 - scope definition property (AngularJS), 658
 - scope destruction phase (AngularJS scope), 592
 - scope() method, 562
 - scope_controller.html listing, 586-587
 - scope_controller.js listing (22.1), 586
 - scope_events.html listing (26.4), 694-695
 - scope_events.js listing (26.3), 694

- scope_hierarchy.html listing (22.6), 596-595
- scope_hierarchy.js listing (22.5), 595
- scope_template.html listing (22.4), 590
- scope_watch.html listing (26.2), 689-690
- scope_watch.js listing (26.1), 689
- scopes (AngularJS), 550, 583, 597
 - directives, configuring, 663-664
 - expressions, 604-606
 - inherited, 663
 - isolate, 663-664
 - life cycle, 591-595
 - nested controllers, 593-595
 - object properties, tracking changes to with \$watchCollection, 688-690
 - parent hierarchy, emitting, 691
 - relationship between backend server data, 591
 - relationship between root scope and applications, 584
 - relationship between templates, 587-589
 - template values, 588-589
 - variables, tracking with \$watch, 686-687
- screen object, 303-304
- screenX attribute (event objects), 237, 270
- screenX property (window object), 304
- screenY attribute (event objects), 237, 270
- screenY property (window object), 304
- script directive (AngularJS templates), 628
- <script> element, 146
- <script> tag, 75
- scripting
 - client-side, 16-17
 - server-side, 17
- scripts. See also listings, applying jQuery UI, 463-472
- scroll event, 241
- scroll option (sortable widget), 516
- scrollBy() method (window object), 305
- scrollParent() method, 463
- scrollTo() method (window object), 305
- search() method, 177, 722
- search property (window object), 307
- select directive (AngularJS templates), 634
- select event, 241
- select input elements, web forms, accessing, 351-352
- <select> tag, 89
- selectable widget (jQuery), 511-516
- selectable_sets.html listing (18.10), 514
- selectable_sets.js listing (18.11), 515
- selectable_sets.js listing (18.12), 515-516
- selectors
 - CSS, styling HTML elements, 110-111
 - jQuery UI
 - applying based on data values, 465-466
 - applying basic, 205
 - attribute, 206
 - content, 207
 - filtered, 211
 - form, 209
 - hierarchy, 208-209
 - new, 464-465
 - visibility, 210
- self property (window object), 304
- send() method, 428
- separation of responsibilities, AngularJS, 553
- serializing data, web forms, 354-360
- server_lesson15_ajax_handling.js listing (15.7), 437
- server_lesson15_ajax_post.js listing (15.20), 449-450
- server.js listing (1.5), 28
- servers
 - updating data from jQuery, AJAX, 442-450
 - web server/browser paradigm, 9-20
- server-side scripting, dynamic JSON data generation, 20
- server-side services, 425
- server-side templates, 17-18
- service providers (AngularJS), 570-571
- service services (AngularJS), defining, 733
- service_animate.html listing (27.8), 719-720
- service_animate.js listing (27.7), 719
- service_cache.js listing (27.4), 704-709
- service_cookie.html listing (27.6), 712
- service_cookie.js listing (27.5), 711-712
- service_custom_censor.html listing (28.2), 736-737
- service_custom_censor.js listing (28.1), 735-736

- service_custom_db_access.js listing (28.6), 744
- service_custom_db.html listing (28.8), 746-747
- service_custom_db.js listing (28.7), 745
- service_custom_time.html listing (28.4), 739-740
- service_custom_time.js listing (28.3), 737-739
- service_db_server.js listing (28.5), 742-743
- service_http.html listing (27.3), 707
- service_http.js listing (27.2), 706
- service_location.html listing (27.11), 725
- service_location.js listing (27.10), 727
- service_server.js listing (27.1), 705
- services (AngularJS), 551, 699-700, 728
 - built-in, 700-701
 - \$animate, 714-719
 - \$cacheFactory, 704-709
 - \$cookieStore, 709-711
 - \$http, 701-708
 - \$interval, 714
 - \$location, 721-724
 - \$q, 726-728
 - \$timeout, 714
 - \$window, 704-709
 - custom, 731, 748
 - cursor, 735-737
 - database access, 741-747
 - defining constant service, 732
 - defining value service, 732
 - integrating into applications, 733-747
 - time, 737
 - defining, 733
 - factory, building using factory provider, 732
 - setAttribute() method, 198
 - SET-COOKIE header (HTTP), 15
 - setInterval() method (window object), 305
 - setRequestHeader attribute (XMLHttpRequest object), 428
 - setRequestHeader() method, 453
 - setTimeout() method, 305
 - shift() method, 182
 - shiftKey attribute (event objects), 237
 - show() method, 199, 325-327
 - animating, 326-327
 - showButtonPanel option (datepicker widget), 530
 - showOn option (datepicker widget), 530
 - size, HTML elements, obtaining and setting, 273
 - size effect (jQuery UI), 476
 - Size item (network traffic analyzer), 64
 - slice() method, 177, 182
 - slide effect (jQuery UI), 476
 - slide option (slider widget), 536
 - slideDown() method, 332-333
 - slider bars, implementing, 536-538
 - slider-based image gallery, adding, 392-397
 - slideToggle() method, 332-333
 - slideUp() method, animating, 332-333
 - sliding animations, 332-337
 - sliding_images.css listing (12.9), 336-337
 - sliding_images.html listing (12.7), 335-336
 - sliding_images.js listing (12.8), 336
 - small_title.html listing (24.3), 632
 - snake effect (jQuery UI), 476
 - sort() method, 182
 - sortable widget (jQuery), 516-522
 - sortable_elements.css listing (18.15), 521-522
 - sortable_elements.html listing (18.13), 518-520
 - sortable_elements.js listing (18.14), 521
 - sorting
 - AngularJS template filters, 616-618
 - tables, implementing, 397-404
 - Source Selection menu (debugger), 57
 - element, 232, 257-258, 261, 309
 - sparkline graphics, adding, 417-421
 - special effects, 321
 - specialEasing option (jQuery animation), 322
 - specialized object providers (AngularJS), 570-571
 - splice() method, 182
 - split() method, 177
 - stack option (draggable widget), 497
 - star ratings, elements, adding to, 770-773
 - status attribute
 - jqXHR object, 453
 - XMLHttpRequest object, 428
 - Status item (network traffic analyzer), 64
 - statusText attribute (jqXHR object), 453

Step Into option (debugger), 58
step option (jQuery animation), 322
Step Out option (debugger), 58
Step Over option (debugger), 58
stopImmediatePropagation() method, 239
stopping animation, 323
stopPropagation() method, 239
String data type (JavaScript), 152
String object (JavaScript), 176-178
string_manipulation.html listing (6.1), 181
strings
 combining, 178
 converting arrays into, 183
 escape codes, 176
 manipulating, 179-180
 replacing words in, 179
 searching for substrings, 178
 splitting into arrays, 179-181
style attribute (body element), 77
style attribute (DOM objects), 198
<STYLE> element (CSS), 13-14
<style> tag, 74, 82
style.html listing (1.2), 13-14
styles (CSS), 13-14
 adding, 105-108
 adding to HTML elements, 108-140
 applying text, 113-119
 defining in HTML elements, 108
 preparing for dynamic design, 140-141
 selectors, using to style HTML elements, 110-111
submission, web forms, controlling, 362-364
submit event, 241

substr() method, 177
substring() method, 177
substrings, searching for, 178
success, AJAX requests, handling, 433-444
<svg> element, 93, 95-99
syntax
 CSS (Cascading Style Sheets), 108-110
 HTML elements, 71
 JavaScript, 145, 151-169
 objects, 173-175
 jQuery, 145

T

tabbable.html listing (29.4), 754-755
tabbable.js listing (29.1), 753
tabbed panels, creating, 539-542
tabbed views, building, 751-755
table elements, 85-88
<table> tag, 85
tables, 87-88
implementing with sorting and filtering, 397-404
tables.html listing (3.5), 87-88
tabs.html listing (29.2), 754
tags
 <audio>, 102
 <button>, 89
 <caption>, 85
 <col>, 85
 <colgroup>, 85
 <fieldset>, 89
 <form>, 88
 , 83-84
 <input>, 89

<label>, 89
 <legend>, 89
 <link>, 76-77
 <meta>, 73-74
 <noscript>, 76
 <option>, 89
 <script>, 75
 <select>, 89
 <style>, 74, 82
 <table>, 85
 <tbody>, 85
 <textarea>, 89
 <tfoot>, 85
 <th>, 85
 <thead>, 85
 <title>, 72-73
 <tr>, 85
 <video>, 102
target attribute (event objects), 237
<tbody> tag, 85
template definition property (AngularJS), 658
templates (Angular JS), 550, 599-600, 623, 650
 directives, 599, 627
 adding a controller to, 661-662
 built-in, 628-653
 configuring scope, 663-664
 creating custom, 657-668
 directive view template, 659-660
 implementing custom, 668-680
 implementing event handlers in, 672-674
 manipulating DOM in, 668-670

- manipulating DOM with
 - compile() function, 634-636
 - manipulating DOM with
 - link() function, 665-667
 - nested, 677-680
 - restricting behavior, 660-661
 - transcluding elements, 664-665
 - expressions, 599-609
 - basic, 602-603
 - JavaScript, 608-609
 - scope, 604-606
 - filters, 599, 611-618
 - built-in, 612
 - custom, 620-621
 - data rendering, 613-615
 - ordering and sorting, 616-618
 - relation between scopes, 587-589
 - server-side, 17-18
 - templateUrl** definition property (AngularJS), 658
 - terminal** definition property (AngularJS), 658
 - text input elements, web forms, accessing, 349-350
 - text() method, 561
 - text responses, AJAX requests, 427
 - text styles (CSS), applying, 113-119
 - text_styles.css listing (4.2), 118-119
 - text_styles.html listing (4.1), 118
 - text-align property (CSS), 114
 - textarea directive (AngularJS templates), 634
 - <textarea> tag, 89
 - text-decoration property (CSS), 115
 - text-indent property (CSS), 115
 - text-overflow property (CSS), 116
 - text-transform property (CSS), 115
 - <tfoot> tag, 85
 - <th> tag, 85
 - <thead> tag, 85
 - ThemeRoller, 459-461
 - themes, jQuery UI, 459-461
 - time service (AngularJS), implementing, 737
 - Timeline item (network traffic analyzer), 64
 - timeout property (config parameter), \$http service requests, 702
 - timers, 314-317
 - delay, 314-315
 - reoccurring, 315-316
 - simple, 315-316
 - timeStamp attribute (event objects), 237
 - Timing tab (network traffic analyzer), 65
 - <title> tag, 72-73
 - toggle() method, animating, 326-327
 - toggleClass() method, 561
 - tolerance option
 - droppable widget, 501
 - selectable widget, 511
 - sortable widget, 516
 - toLowerCase() method, 177
 - tooltips widget (jQuery), 542-544
 - top property (window object), 304
 - toString() method, 182
 - toUpperCase() method, 177
 - <tr> tag, 85
 - traffic, network, analyzing, 63-65
 - transclude definition property (AngularJS), 658
 - transfer effect (jQuery UI), 476
 - transformRequest property (config parameter), \$http service requests, 702
 - transformResponse property (config parameter), \$http service requests, 702
 - traverse_dom.css listing (8.6), 232
 - traverse_dom.html listing (8.4), 229-232
 - traverse_dom.js listing (8.5), 232
 - tree view, creating, 404-408
 - triggerHandler() method, 561
 - triggering events manually, 253
 - JavaScript, 254-258
 - jQuery, 258-262
 - type** attribute
 - ajax() method, 450
 - event objects, 237
 - type** definition property (AngularJS), 658
 - Type** item (network traffic analyzer), 64
- ## U
- UI (user interface) (jQuery), 457, 472, 475, 492
 - adding and removing unique IDs, 463
 - adding effects to class transitions, 482-485
 - adding to element visibility transitions, 485-491
 - adding to web pages, 461-462
 - applying effects, 475-482
 - applying to scripts, 463-472
 - creating custom theme, 459-461

- CSS, 457-458
 - functionality, 463-464
 - JavaScript, 457-458
 - obtaining library, 458-459
 - positioning elements, 468-472
 - selectors, 464-466
 - widgets, 495, 522, 525, 545
 - autocomplete, 527-528-530
 - creating custom, 544-545
 - datepicker, 530-531
 - dialog, 532-533
 - drag-and-drop, 497-507
 - events, 495
 - expandable accordion element, 526-527
 - interactions, 495-496
 - jQuery.widget factory, 495-496
 - methods, 495
 - mouse interaction widget, 496
 - progress bar, 535-536
 - resizable, 507-511
 - reviewing, 525
 - selectable, 511-516
 - slider, 536-538
 - sortable, 516-522
 - spinner, 538-539
 - stylized menus, 533-535
 - tabs, 539-542
 - tooltips, 542-544
 - unbind() method, 561**
 - uniqueID() method, 463**
 - unload event, 241**
 - unshift() method, 182**
 - updating server data from jQuery, AJAX, 442-450**
 - uppercase filter (AngularJS templates), 612**
 - uppercase() method, 556**
 - url attribute, ajax() method, 450**
 - url() method, 722**
 - URL Path item (network traffic analyzer), 64**
 - url property (config parameter), \$http service requests, 702**
 - url rule (validation), 376**
 - URLs (uniform resource locators), 11**
 - users**
 - confirmation pop-up boxes, 314
 - notification pop-up boxes, 313
 - prompting for input, 314
 - utilities, AngularJS global API, 556**
- V**
- val() method, 199, 561**
 - validate() method, 376-387**
 - validation, web forms, 375-387**
 - adding messages, 378-380
 - adding rules, 377-378
 - complex validation, 377-384
 - jQuery validation plug-in, 376
 - jQuery validation with HTML, 376-377
 - manually, 375
 - placing messages, 380-384
 - validator object messages, 378-380**
 - value attribute (DOM objects), 198**
 - value option (slider widget), 526**
 - value services (AngularJS), defining, 732**
 - value spinner element, adding, 538-539**
 - valueOf() method, 177, 182**
 - values**
 - JavaScript objects, assigning new, 174-175
 - web form elements, obtaining and setting, 348-353
 - variables (JavaScript)**
 - creating, 151-152
 - scope, 167-168
 - tracking with \$watch, 687
 - <video> tag, 102**
 - views (AngularJS), 550**
 - building tabbed, 751-755
 - directives, 627
 - built-in, 628-653
 - creating custom, 657-668
 - templates, 599-600
 - custom filters, 620-621
 - directives, 599
 - expressions, 599-609
 - filters, 599, 611-618
 - visibility**
 - animating, 329-332
 - elements, toggling, 286-287
 - visibility design properties (CSS), 131**
 - visibility selector (jQuery), 210**
 - visibility_transitions.css listing (17.9), 488-489**
 - visibility_transitions.html listing (17.7), 490-488**
 - visibility_transitions.js listing (17.8), 488**

W

- Watch pane (debugger), 57-58**
- watcher registration phase (AngularJS scope), 592**
- web development environment, setting up, 21-32**
- web forms, 347-348, 387**
 - accessing hidden inputs, 353
 - controlling submission and reset, 362-364
 - elements, 348-360
 - animated, 368-370
 - automatically focusing and blurring, 361
 - button input, 352
 - check box, 350
 - disabling, 362
 - dynamically controlling appearance and behavior, 368-374
 - file input, 352-353
 - hiding and showing, 361
 - obtaining and setting values, 348-353
 - radio input, 350-351
 - select input, 351-352
 - text input, 349-350
 - flow control, 361-368
 - dynamic, 362-364
 - serializing data, 354-360
 - validating, 375-387
 - adding messages, 378-380
 - adding rules, 377-378
 - complex validation, 377-384
 - jQuery, 381-384
 - jQuery validation plug-in, 376
 - jQuery validation with HTML, 376-377
 - manually, 375
 - placing messages, 380-384
- web pages. See also elements (HTML)**
 - accessing element values, 270-281
 - obtaining and setting values, 271
 - obtaining mouse position, 270
 - adding
 - CSS, 29-30, 105-108
 - draggable images to, 498-501
 - HTML, 29
 - image gallery, 391-397
 - jQuery UI, 461-462
 - sparkline graphics, 417-421
 - animation, 321-325
 - applying buttons to form controls, 528-530
 - bootstrapping AngularJS, 555
 - calendar input, 530-531
 - cookies, obtaining and setting, 308-313
 - creating
 - progress bars, 535-536
 - tree view, 404-408
 - dynamic, 69
 - HTML document structure, 70-72
 - HTML elements, 69-70
 - effects
 - adding to class transitions, 482-485
 - adding to element visibility transitions, 485-491
 - applying with jQuery UI, 475-482
 - elements
 - drag-and-droppable, 502-507
 - expandable accordian, 526-527
 - implementing autocomplete form, 528-530
 - positioning, 273-274
 - resizable, 508-511
 - selectable sets, 512-516
 - sorting, 516-522
 - value spinner, 538-539
 - external links, 308
 - forcing to open in new browser windows, 308-311
 - stopping, 308
 - implementing graphical equalizer displays, 413-417
 - menus, implementing stylized, 533-535
 - overlay dialogs, 409-413
 - pop-up boxes, adding, 313-314
 - positioning images, jQuery UI, 469-472
 - setting timers, 314-317
 - slider bars, 536-538
 - special effects, 321
 - stylized dialogs, 532-533
 - tabbed panels, 539-542
 - tables, implementing with sorting and filtering, 397-404
 - tooltips, 542-544
- web server/browser paradigm, 9-20**

- web servers, 10**
 - client-side scripting, 16-17
 - development, 21
 - Express, creating, 26-29
 - web server/browser paradigm, 9-20
- web_element_manipulation.css listing (10.6), 291-292**
- web_element_manipulation.html listing (10.4), 290**
- web_element_manipulation.js listing (10.5), 290-291**
- web_layout.css listing (4.7), 139-140**
- web_layout.html listing (4.6), 136-139**
- web_page_manipulation.css listing (10.3), 281**
- web_page_manipulation.html listing (10.1), 279-280**
- web_page_manipulation.js listing (10.2), 280-281**
- which attribute (event objects), 237**
- while loop (JavaScript), 160-161**
- widget() method, 495**
- widgets (jQuery), 495, 522, 525, 545**
 - autocomplete, 527-530
 - creating custom, 544-545
 - datepicker, 530-531
 - dialog, 532-533
 - drag-and-drop
 - creating drop targets, 497-507
 - draggable, 497-507
 - dragging elements, 497-507
 - droppable widget, 497-507
 - events, 495
 - expandable accordion element, 526-527
 - interactions, 495-496
 - jQuery.widget factory, 495-496
 - mouse interaction widget, 496
 - methods, 495
 - progress bar, 535-536
 - resizable, 507-511
 - reviewing, 525
 - selectable, 511-516
 - slider, 536-538
 - sortable, 516-522
 - spinner, 538-539
 - tabs, 539-542
 - tooltips, 542-544
- widgets_accordian.html listing (19.1), 527**
- widgets_autocomplete.html listing (19.2), 528**
- widgets_calendar.html listing (19.4), 531**
- widgets_custom.html listing (19.12), 544-545**
- widgets_dialogs.html listing (19.5), 533**
- widgets_menus.html listing (19.6), 534-535**
- widgets_progress_bars.html listing (19.7), 536**
- widgets_slider_bars.htm listing (19.8), 537-538**
- widgets_spinner.html listing (19.9), 539**
- widgets_tabs.html listing (19.10), 541-542**
- widgets_tooltips.html listing (19.11), 543-544**
- width() method, 199, 273**
- width property (screen object), 304**
- window, browser, 11**
- window object, 304-305**
 - methods, 305
 - properties, 304
- window.location object, providing wrapper for, 721-724**
- withCredentials property (config parameter), \$http service requests, 702**
- words, replacing in strings, 179**
- word-spacing property (CSS), 115**
- working_event.css listing (9.6), 252**
- working_event.html listing (9.4), 251-252**
- working_event.js listing (9.5), 252**
- wrap() method, 562**
- writeln() function, 168**
- writing dynamic scripts, 31-32**

X

- XML responses, AJAX requests, 427**
 - handling data, 439-444
- XMLHttpRequest object, 428**
- xsrCookieName property (config parameter), \$http service requests, 702**
- x.toExponential() method, 175**
- x.toFixed() method, 175**
- x.toPrecision() method, 175**
- x.toString() method, 175**
- x.valueOf() method, 175**

Z

zIndex() method, 463

zIndex option, 497

draggable widget, 497

sortable widget, 516

z-index property (CSS), 135

adjusting, 292-295

**zoom view field, images, adding
to, 761-764**

**zooming.html listing (29.9),
763-764**

zooming.js listing (29.7), 762-763

zoomit.html listing (29.8), 763