O'REILLY®

5TH EDITION

"Unlike all the other books that start at the beginning, this one will get you to the good stuff, fast. Jennifer will explain every step you need, including some very advanced concepts."

—JEN SIMMONS, MOZILLA AND W3C CSS WORKING GROUP

Learning Web Design

A BEGINNER'S GUIDE TO HTML, CSS, JAVASCRIPT, AND WEB GRAPHICS

Jennifer Niederst Robbins

Whether you're a beginner or bringing your skills up to date, this book gives you a solid footing in modern web production. I teach each topic visually at a pleasant pace, with frequent exercises to let you try out new skills. Reading it feels like sitting in my classroom! —Jennifer Robbins

Fifth Edition

LEARNING WEB DESIGN

A BEGINNER'S GUIDE TO HTML, CSS, JAVASCRIPT, AND WEB GRAPHICS

Jennifer Niederst Robbins

Beijing • Boston • Farnham • Sebastopol • Tokyo

O'REILLY®

Learning Web Design, Fifth Edition

A Beginner's Guide to HTML, CSS, JavaScript, and Web Graphics

by Jennifer Niederst Robbins

Printed in Canada.

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

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

EDITORS: Meg Foley and Jeff Bleiel

PRODUCTION EDITOR: Kristen Brown

COVER DESIGNER: Edie Freedman

INTERIOR DESIGNER: Jennifer Robbins

PRINT HISTORY:

March 2001:	First edition.
June 2003:	Second edition.
June 2007:	Third edition.
August 2012:	Fourth edition.
May 2018:	Fifth edition.

ISBN: 978-1-491-96020-2
[TI] [2018-04-16]

CONTENTS

PART II. HTML FOR STRUCTURE

FOREWORD

BY JEN SIMMONS

If you travel to Silicon Valley and navigate between the global headquarters of some of the world's most famous internet companies, you can head to the Computer History Museum. Wander through the museum, past the ancient mainframes and the story of the punch card, and you'll eventually find yourself at the beginning of the Wide World Web. There's a copy of the Mosaic browser on a floppy disk tucked in a book of the same name, a copy of Netscape Navigator that was sold in a box, and something called "Internet in a Box," the #1 best-selling internet solution for Windows. Then there are the websites. Some of the earliest, most notable, and most important websites are on permanent display, including something called the "Global Network Navigator," from 1993. It was designed by none other than the author of this book, Jennifer Robbins. Long before most of us had any idea the web existed, or even before many of you were born, Jen was busy designing the first commercial website. She's been there from the very beginning, and has watched, taught, and written about every stage of evolution of the web.

Learning Web Design is now in its 5th edition, with a gazillion new pages and updates from those early days.

I am constantly asked, "What are the best resources for learning web technology?" I learned by reading books. Blog posts are great, but you also need an in-depth comprehensive look at the subject. In the beginning, all books were beginner books, teaching HTML, URLs, and how to use a browser. When CSS came along, the books assumed you'd already been using HTML, and taught you how to change to the new techniques. Then CSS3 came along, and all the books taught us how to add new CSS properties to our preexisting understanding of CSS2. Of course there were always books for beginners, but they were super basic. They never touched on professional techniques for aspiring professionals. Each new generation of books assumed that you had prior knowledge. Great for those of us in the industry. Tough for anyone new.

But how in the world are you supposed to read about two decades of techniques, discarding what is outdated, and remembering what is still correct? How are you supposed to build a career from knowledge that's so basic that you have no idea what real pros code in their everyday jobs?

You can't. That's why today when people ask me for a book recommendation, I have only one answer. This book.

This book you are reading now doesn't require any prior knowledge. You don't need to have made a web page before, or to have any idea where to get a code editor. It starts at the very beginning. And yet, unlike all the other books that start at the beginning, this one will get you to the good stuff, fast. Jen will explain every step you need, including some very advanced concepts. She's packed this book full of cutting edge, insider knowledge from top experts.

I honestly don't know how she does it. How can someone teach the basics and the advanced stuff at the same time? Usually you'll learn those things years apart, with lots of struggling in the dark in the meantime. Here, Jen will lift you up from wherever you are in your journey, and take you farther. Every one of us—myself included, and I'm on the CSS Working Group (the group of people who invent new CSS)—can learn a lot from this book. I do every time I pick it up.

Pay attention to the notes in the margins. Read the websites she recommends, watch the videos. Jen is giving you a shortcut to a professional network. Follow the people she mentions. Read the links they suggest. These might be your future colleagues. Dare to dream that you will meet them. They are, after all, only a tweet away. It is a small world, full of real people, and you can become part of it all. This book will get you started.

—Jen Simmons
Designer and Developer Advocate at Mozilla
Member of the CSS Working Group
April 2018

PREFACE

Hello and welcome to the *fifth* edition of *Learning Web Design*!

I've been documenting web design and development in books like this one for decades, and it continues to fascinate me how the web landscape changes from edition to edition. This fifth edition is no exception! Not only is this version nearly 200 pages longer than the last one, but there are also some significant updates and additions worth noting.

First, some technologies and techniques that were brand new or even experimental in the last edition have become nicely settled in. HTML5 is the new normal, and CSS is moving ahead with its modular approach, allowing new technologies to emerge and be adopted one at a time. We've largely gotten our heads around designing for a seemingly infinite range of devices. Responsive Web Design is now the de facto approach to building sites. As a result, RWD has earned its own chapter in this edition (**Chapter 17, Responsive Web Design**). Where in the last edition we pondered and argued how to handle responsive image markup, in this edition, the new responsive image elements are standardized and well supported (**Chapter 7, Adding Images**). I think we're getting the hang of this mobile thing!

I've seen a lot of seismic shifts in web design over the years, and this time, Flexbox and Grid are fundamentally changing the way we approach design. Just as we saw CSS put table-based layouts and 1-pixel spacer GIFs out of their misery, Flexbox and Grid are finally poised to kick our old float-based layout hacks to the curb. It is nothing short of a revolution, and after 25 years, it's refreshing to have an honest-to-goodness solution for layout. This edition sports a new (and hefty!) chapter on proper page layout with Flexbox and Grid (**Chapter 1, CSS Layout with Flexbox and Grid**).

Although knowledge of HTML, CSS, and JavaScript is at the heart of web development, the discipline has been evolving, and frankly, becoming more

■ ONLINE RESOURCE

The Companion Website

Be sure to visit the companion website for this book at *learningwebdesign.com*.

It features materials for the exercises, downloadable articles, lists of links from the book, contact information, and more.

complicated. I would be shirking my duty if I didn't at least introduce you to some of the new tools of the trade—CSS processors, feature detection, the command line, task runners, and Git—in a new chapter on the modern web developer toolkit (**Chapter 20, Modern Web Development Tools**). Sure, it's more stuff to learn, but the benefit is a streamlined and more efficient workflow.

The biggest surprise to me personally was how much web *image* production has changed since the fourth edition. Other than the introduction of the PNG format, my graphics chapters have remained essentially unchanged for 20 years. Not so this time around! Our old standby, GIF, is on the brink of retirement, and PNG is the default thanks to its performance advantages and new tools that let even smaller 8-bit PNGs include multiple levels of transparency. But PNG will have to keep its eye on WebP, mentioned in this edition for the first time, which may give it a run for its money in terms of file size and capabilities. The biggest web graphics story, however, is the emergence of SVG (Scalable Vector Graphics). Thanks to widespread browser support (finally!), SVG went from a small "some day" section in the previous edition to an entire "go for it!" chapter in this one (**Chapter 25, SVG**).

As in the first four editions, this book addresses the specific needs and concerns of beginners of all backgrounds, including seasoned graphic designers, programmers looking to expand their skills, and anyone else wanting to learn how to make websites. I've done my best to put the experience of sitting in my beginner web design class into a book, with exercises and tests along the way, so you get hands-on experience and can check your progress.

Whether you are reading this book on your own or using it as a companion to a web design course, I hope it gives you a good head start and that you have fun in the process.

HOW THIS BOOK IS ORGANIZED

Learning Web Design, Fifth Edition, is divided into five parts, each dealing with an important aspect of web development.

Part I: Getting Started

Part I lays a foundation for everything that follows in the book. I start off with some important general information about the web design environment, including the various roles you might play, the technologies you might learn, and tools that are available to you. You'll get your feet wet right away with HTML and CSS and learn how the web and web pages generally work. I'll also introduce you to some Big Concepts that get you thinking in the same way that modern web designers think about their craft.

Part II: HTML for Structure

The chapters in **Part II** cover the nitty-gritty of every element and attribute available to give content semantic structure. We'll cover the markup for text, links, images, tables, forms, and embedded media.

Part III: CSS for Presentation

In the course of **Part III**, you'll go from learning the basics of Cascading Style Sheets for changing the presentation of text to creating multicolumn layouts and even adding time-based animation and interactivity to the page. It provides an introduction to Responsive Web Design, as well as the tools and techniques that are part of the modern developer's workflow.

Part IV: JavaScript for Behavior

Mat Marquis starts **Part IV** out with a rundown of JavaScript syntax so that you can tell a variable from a function. You'll get to know some ways that JavaScript is used (including DOM scripting) and existing JavaScript tools such as polyfills and libraries that let you put JavaScript to use quickly, even if you aren't quite ready to write your own code from scratch.

Part V: Web Images

Part V introduces the various image file formats that are appropriate for the web, provides strategies for choosing them as part of a responsive workflow, and describes how to optimize them to make their file size as small as possible. It also includes a chapter on SVG graphics, which offer great advantages for responsive and interaction design.

Part VI: Appendices

Part VI holds reference material such as test answers, lists of HTML global attributes and CSS Selectors, and a look at HTML5 and its history.

TYPOGRAPHICAL CONVENTIONS

Italic

Used to indicate filenames and directory names, as well as for emphasis.

Colored italic

Used to indicate URLs and email addresses.

Colored roman text

Used for special terms that are being defined.

`Constant width`

Used to indicate code examples and keyboard commands.

`Colored constant width`

Used for emphasis in code examples.

`Constant width italic`

Used to indicate placeholders for attribute and style sheet property values.

→

Indicates that a line of code was broken in the text but should remain together on one line in use.

ACKNOWLEDGMENTS

Once again, many smart and lovely people had my back on this edition.

I want to say a special thanks to my two *amazing* tech reviewers. I am quite indebted to Elika J. Etemad (*fantasai*), who, as a member of the W3C CSS Working Group, helped me make this edition more accurate and up-to-date with standards than ever before. She was *tough*, but the results are worth it. Petter Dessne brought his computer science expertise as well as valuable perspective as a professor and a reader for whom English is a second language. His good humor and photos of his home in Sweden were appreciated as well!

I am also grateful for this roster of web design superstars who reviewed particular chapters and passages in their areas of expertise (in alphabetical order): Amelia Bellamy-Royds (SVG), Brent Beer (developer tools), Chris Coyier (SVG), Terence Eden (audio/video), Brad Frost (Responsive Web Design), Lyza Danger Gardner (developer tools), Jason Grigsby (images), Val Head (animation), Daniel Hengeveld (developer tools), Mat Marquis (responsive images), Eric Meyer (CSS layout), Jason Pamental (web fonts), Dan Rose (images), Arsenio Santos (embedded media), Jen Simmons (CSS layout), Adam Simpson (developer tools), and James Williamson (structured data).

Thanks also to Mat Marquis for his contribution of two lively JavaScript chapters that I could never have written myself, and to Jen Simmons for writing the Foreword and for her ongoing support of *Learning Web Design*.

I want to thank my terrific team of folks at O'Reilly Media: Meg Foley (Acquisitions Editor), Jeff Bleiel (Developmental Editor), Kristen Brown (Production Editor), Rachel Monaghan (Copyeditor), Sharon Wilkey (Proofreader), and Lucie Haskins (Indexer). Special thanks go to InDesign and book production expert Ron Bilodeau, who turned my design into a template and a set of tools that made book production an absolute joy. Special thanks also go to Edie Freedman for the beautiful cover design and half a lifetime of friendship and guidance.

Finally, no Acknowledgments would be complete without profound appreciation for the love and support of my dearest ones, Jeff and Arlo.

ABOUT THE AUTHOR

Jennifer Robbins began designing for the web in 1993 as the graphic designer for Global Network Navigator, the first commercial website. In addition to this book, she has written multiple editions of *Web Design in a Nutshell* and *HTML5 Pocket Reference*, published by O'Reilly. She is a founder and organizer of the Artifact Conference, which addresses issues related to mobile web design. Jennifer has spoken at many conferences and has taught beginning web design at Johnson and Wales University in Providence, Rhode Island. When not on the clock, Jennifer enjoys making things, indie rock, cooking, travel, and raising a cool kid.

HOW TO CONTACT US

Please address comments and questions concerning this book to the publisher:

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

800-998-9938 (in the United States or Canada)
707-829-0515 (international or local)
707-829-0104 (fax)

We have a web page for this book, where we list errata, examples, and any additional information. You can access this page at *bit.ly/learningWebDesign_5e*.

To comment or ask technical questions about this book, send email to *bookquestions@oreilly.com*.

For more information about our books, courses, conferences, and news, see our website at *www.oreilly.com*.

Find us on Facebook: *facebook.com/oreilly*

Follow us on Twitter: *twitter.com/oreillymedia*

Watch us on YouTube: *www.youtube.com/oreillymedia*

1

GETTING STARTED

CHAPTER
1

GETTING STARTED IN WEB DESIGN

The web has been around for more than 25 years now, experiencing euphoric early expansion, an economic-driven bust, an innovation-driven rebirth, and constant evolution along the way. One thing is certain: the web as a communication and commercial medium is here to stay. Not only that, it has found its way onto devices such as smartphones, tablets, TVs, and more. There have never been more opportunities to put web design know-how to use.

Through my experience teaching web design courses and workshops, I've had the opportunity to meet people of all backgrounds who are interested in learning how to build web pages. Allow me to introduce you to just a few:

> *"I've been a print designer for 17 years, and now I am feeling pressure to provide web design services."*
>
> *"I've been a programmer for years, but I want shift my skills to web development because there are good job opportunities in my area."*
>
> *"I tinkered with web pages in high school and I think it might be something I'd like to do for a living."*
>
> *"I've made a few sites using themes in WordPress, but I'd like to expand my skills and create custom sites for small businesses."*

Whatever the motivation, the first question is always the same: "Where do I start?" It may seem like there is a mountain of stuff to learn, and it's not easy to know where to jump in. But you have to start somewhere.

This chapter provides an overview of the profession before we leap into building sites. It begins with an introduction to the roles and responsibilities associated with creating websites, so you can consider which role is right for you. I will also give you a heads-up on the equipment and software you will be likely to use—in other words, the tools of the trade.

IN THIS CHAPTER

WHERE DO I START?

Maybe you are reading this book as part of a full course on web design and development. Maybe you bought it to expand your current skill set on your own. Maybe you just picked it up out of curiosity. Whatever the case, this book is a good place to start learning what makes the web tick.

There are many levels of involvement in web design, from building a small site for yourself to making it a full-blown career. You may enjoy being a "full-stack" web developer or just specializing in one skill. There are a lot of ways you can go.

If you are interested in pursuing web design or production as a career, you'll need to bring your skills up to a professional level. Employers may not require a web design degree, but they will expect to see working sample sites that demonstrate your skills and experience. These sites can be the result of class assignments, personal projects, or a site for a small business or organization. What's important is that they look professional and have well-written, clean HTML; style sheets; and scripts behind the scenes.

If your involvement is at a smaller scale—say you just have a site or two you'd like to publish—you may find using a template on an online website service is a great head start (see the sidebar **"I Just Want My Own Site"**). Most allow you to tweak the underlying code, so what you learn in this book will help you customize the template to your liking.

I Just Want My Own Site

You don't necessarily need to become a web designer or developer to start publishing on the web. There are many website hosting services that provide templates and drag-and-drop interfaces that make it easy to build a site without any code know-how. They can be used for anything from full-service ecommerce solutions to small, personal sites (although some services are better suited to one more than the other).

Here are a few of the most popular site building services as of this writing:

- WordPress (*www.wordpress.com*)
- Squarespace (*squarespace.com*)
- Wix (*wix.com*)
- SiteBuilder (*sitebuilder.com*)
- Weebly (*weebly.com*)

There are many similar services available, so it's worth searching the web to find one that's right for you.

IT TAKES A VILLAGE (WEBSITE CREATION ROLES)

When I look at a site, I see the multitude of decisions and areas of expertise that went into building it. Sites are more than just code and pictures. They often begin with a business plan or other defined mission. Before they launch, the content must be created and organized, research is performed, design from the broadest goals to finest details must happen, code gets written, and everything must be coordinated with what's happening on the server to bring it to fruition.

Big, well-known sites are created by teams of dozens, hundreds, or even thousands of contributors. There are also sites that are created and maintained by a team with only a handful of members. It is also absolutely possible to create a respectable site with a team of only yourself. That's the beauty of the web.

In this section, I'll introduce you to the various disciplines that contribute to the creation of a site, including roles related to content, design, and code. You may end up specializing in just one area of expertise, working as part of a team of specialists. If you are designing sites on your own, you will need to wear many hats. Consider that the day-to-day upkeep of your household

requires you to be part-time chef, housecleaner, accountant, diplomat, gardener, and construction worker—but to you it's just the stuff you do around the house. As a solo designer, you'll handle many web-related disciplines, but it will just feel like the stuff you do to make a website.

Content Wrangling

Anyone who uses the title "web designer" needs to be aware that everything we do supports the process of getting the content, message, or functionality to our users. Furthermore, good writing can help the user interfaces we create be more effective, from button labels to error messages.

Of course, someone needs to create all that content and maintain it—don't underestimate the resources required to do this successfully. Good writers and editors are an important part of the team. In addition, I want to call your attention to two content-related specialists in modern web development: the Information Architect (IA) and the Content Strategist.

Information architecture

An Information Architect (also called an Information Designer) organizes the content logically and for ease of findability. They may be responsible for search functionality, site diagrams, and how the content and data are organized on the server. Information architecture is inevitably entwined with UX and UI design (defined shortly) as well as content management. If you like organizing or are *gaga* for taxonomies, information architecture may be the job for you. The definitive text for this field as it relates to the web is *Information Architecture: For the Web and Beyond*, by Louis Rosenfeld and Peter Morville (O'Reilly).

Content strategy

When the content isn't right, the site can't be fully effective. A Content Strategist makes sure that every bit of text on a site, from long explanatory text down to the labels on buttons, supports the brand identity and marketing goals of the organization. Content strategy may also extend to data modeling and content management on a large and ongoing scale, such as planning for content reuse and update schedules. Their responsibilities may also include how the organization's voice is represented on social media. A good place to learn more is the book *Content Strategy for the Web, 2nd Edition,* by Kristina Halvorson and Melissa Rich (New Riders).

All Manner of Design

Ah, *design*! It sounds fairly straightforward, but even this simple requirement has been divided into a number of specializations when it comes to creating sites. Here are a few of the job descriptions related to designing a site, but

bear in mind that the disciplines often overlap and that the person calling herself the "designer" often is responsible for more than one (if not all) of these responsibilities.

User Experience, Interaction, and User Interface design

Often, when we think of design, we think about how something looks. On the web, the first matter of business is designing how the site *works*. Before you pick colors and fonts, it is important to identify the site's goals, how it will be used, and how visitors move through it. These tasks fall under the disciplines of User Experience (UX) design, Interaction Design (IxD), and User Interface (UI) design. There is a lot of overlap between these responsibilities, and it is not uncommon for one person or team to handle all three.

The User Experience designer takes a holistic view of the design process—ensuring the entire experience with the site is favorable. UX design is based on a solid understanding of users and their needs based on observations and interviews. According to Donald Norman (who coined the term), UX design includes "all aspects of the user's interaction with the product: how it is perceived, learned, and used." For a website or application, that includes the visual design, the user interface, the quality and message of the content, and even the overall site performance. The experience must be in line with the organization's brand and business goals in order to be successful.

The goal of the Interaction Designer is to make the site as easy, efficient, and delightful to use as possible. Closely related to interaction design is User Interface design, which tends to be more narrowly focused on the functional organization of the page as well as the specific tools (buttons, links, menus, and so on) that users use to navigate content or accomplish tasks.

The following are deliverables that UX, UI, or interaction designers produce:

User research and testing reports

Understanding the needs, desires, and limitations of users is central to the success of the design of the site or web application. The approach of designing around the user's needs is referred to as User-Centered Design (UCD), and it is central to contemporary web design. Site designs often begin with user research, including interviews and observations, in order to gain a better understanding of how the site can solve problems or how it will be used. It is typical for designers to do a round of user testing at each phase of the design process to ensure the usability of their designs. If users are having a hard time figuring out where to find content or how to move to the next step in a process, then it's back to the drawing board.

Wireframe diagrams

A wireframe diagram shows the structure of a web page using only outlines for each content type and widget (FIGURE 1-1). The purpose of a wireframe diagram is to indicate how the screen real estate is divided and where functionality and content such as navigation, search boxes, form

elements, and so on, are placed. Colors, fonts, and other visual identity elements are deliberately omitted so as not to distract from the structure of the page. These diagrams are usually annotated with instructions for how things should work so the development team knows what to build.

Site diagram

A site diagram indicates the structure of the site as a whole and how individual pages relate to one another. FIGURE 1-2 shows a very simple site diagram. Some site diagrams fill entire walls!

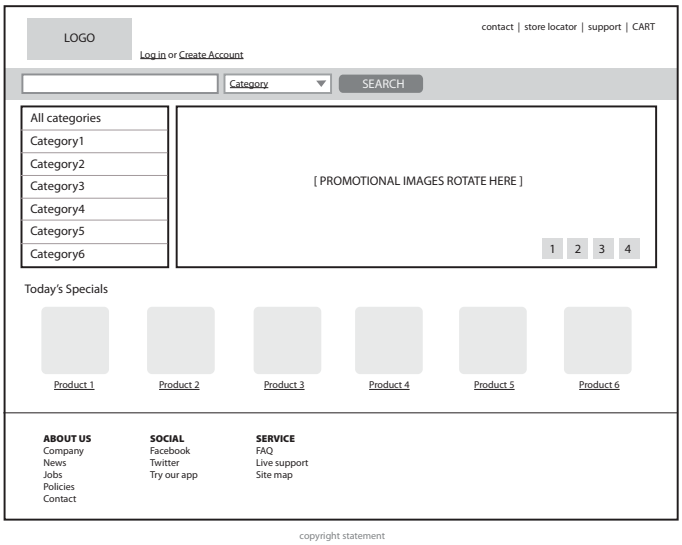


FIGURE 1-1. Wireframe diagram.

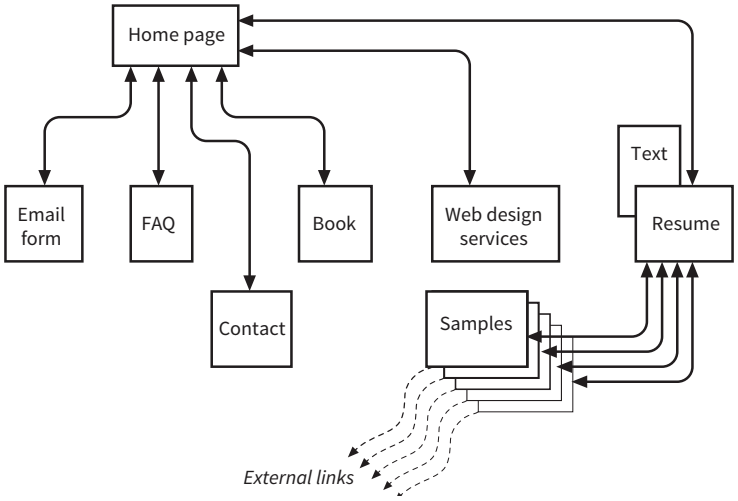


FIGURE 1-2. A simple site diagram.

Storyboards and user flow charts

A storyboard traces the path through a site or application from the point of view of a typical user (a persona in UX lingo). It usually includes a script and "scenes" consisting of screen views or the user interacting with the screen. The storyboard aims to demonstrate the steps it takes to accomplish tasks, outlines possible options, and also introduces some standard page types. FIGURE 1-3 shows a simple storyboard. A user flow chart is another method for showing how the parts of a site or application are connected, but it tends to focus on technical details rather than telling a story. For example, "when the user does *this*, it triggers *that* function on the server." It is common for designers to create a user flow chart for the steps in a process such as member registration or online payments.

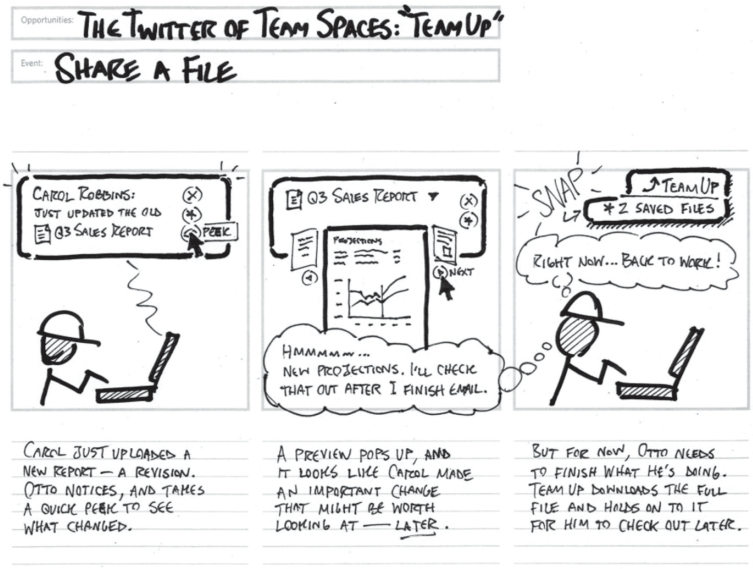


FIGURE 1-3. A typical storyboard (courtesy of Adaptive Path and Brandon Schauer).

There are many books on UX, interaction, and UI design, but these are a few of the classics to get you started:

- *The Elements of User Experience: User-Centered Design for the Web and Beyond* by Jesse James Garrett (New Riders)
- *Don't Make Me Think, Revisited: A Common Sense Approach to Web Usability* by Steve Krug (New Riders)
- *The Design of Everyday Things* by Don Norman (Basic Books)
- *About Face: The Essentials of Interaction Design, 4th Edition* by Alan Cooper, Robert Reimann, David Cronin, and Christopher Noessel (Wiley)
- *Designing Interfaces, 2nd Edition* by Jenifer Tidwell (O'Reilly)

- *100 Things Every Designer Needs to Know about People* by Susan Weinschenk (New Riders)
- *Designing User Experience: A Guide to HCI, UX and Interaction Design* by David Benyon (Pearson)

Visual (graphic) design

Because the web is a visual medium, web pages require attention to their visual presentation. First impressions are everything. A graphic designer creates the "look and feel" of the site—logos, graphics, type, colors, layout, and so on—to ensure that the site makes a good first impression and is consistent with the brand and message of the organization it represents.

There are many methods and deliverables that can be used to present a visual design to clients and stakeholders. The most traditional are sketches or mockups (created in Photoshop or a similar tool) of the way the site might look, such as the home page mockups shown in FIGURE 1-4.

FIGURE 1-4. Look-and-feel sketches (mockups) for a simple site.

Now that sites appear on screens of all sizes, many designers prefer to discuss the visual identity (colors, fonts, image style, etc.) in a way that isn't tied to a specific layout like the typical desktop view shown in FIGURE 1-4. The idea is to agree upon a visual language for the site before production begins.

One option for separating style from screen size is to use style tiles, a technique introduced by Samantha Warren (see **Note**). Style tiles include examples of color schemes, branding elements, UI treatments, text treatment, and mood (FIGURE 1-5). Once the details are decided upon, they can be implemented into working prototypes and the final site. For more on this technique, visit Samantha's excellent site, *styletil.es*, where you can download a template.

Graphic designers may also be responsible for producing the image assets for the site. They will need to know how to optimize images for the fastest delivery and how to address the requirements of varying screen sizes. It is also common for the development team to handle image optimization, but I think it is a skill every visual designer should have. We'll discuss image optimization in **Chapter 24, Image Asset Production**.

NOTE

Designer Dan Mall uses a similar approach that he calls "element collages." An element collage *is a collection of design elements that give the site its unique look and feel, but like style tiles, is not tied to a particular screen layout. Read his article at v3.danielmall.com/articles/rif-element-collages/.*

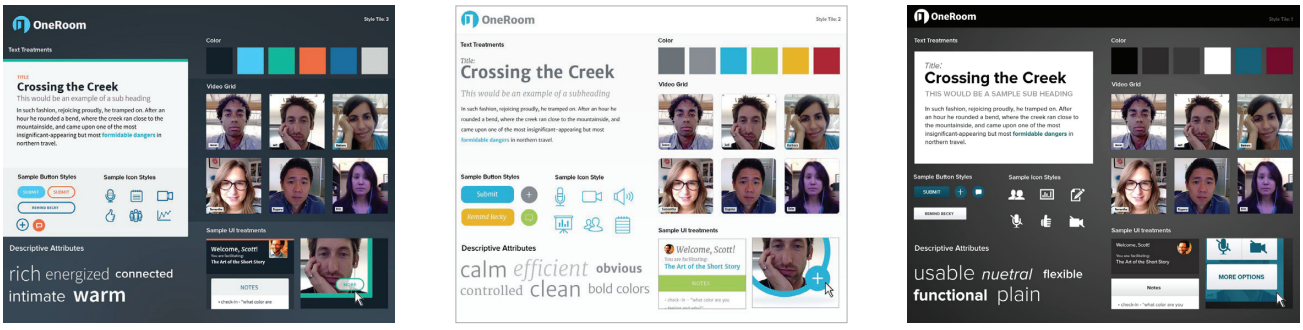


FIGURE 1-5. Style tile technique introduced by Samantha Warren.

Designers may also be responsible for creating a style guide that documents style choices, such as fonts, colors, and other style embellishments, in order to keep the site consistent over time. For a list of examples, articles, books, and podcasts about web style guides, visit the "Website Style Guide Resources" page at *styleguides.io*.

Do Designers Need to Learn to Code?

In short, yes. A basic familiarity with HTML and CSS is now a requirement of anybody joining a web design team. You may not be responsible for creating the final production code for the site, but as HTML and CSS are the native languages of your medium, you need to know your way around them. Some designers also learn JavaScript, but others draw the line there and let a developer handle the programming.

Code is becoming more central to the visual designer's workflow. Where once Photoshop was all you needed to mock up web page designs to send them to production, mockups fixed to a particular size fall short of describing a page that needs to flex to a wide range of screen sizes. For that reason, designers are building their own working prototypes as deliverables that communicate how the design will look and behave in users' hands.

Code Slinging

A large share of the website building process involves creating and troubleshooting the documents, style sheets, scripts, and images that make up a site. At web design firms, the team that handles the creation of the files that make up the site (or templates for pages that get assembled dynamically) is usually called the development or production department.

Development falls under two broad categories: frontend development and backend development. Once again, these tasks may fall to specialists, but it is just as common for one person or team to handle both responsibilities.

■ AT A GLANCE

Frontend Development

Frontend development includes the following web technologies:

- HyperText Markup Language (HTML)
- Cascading Style Sheets (CSS)
- JavaScript and DOM scripting, including AJAX and JavaScript-based frameworks

Frontend development

Frontend refers to any aspect of the design process that appears in or relates directly to the browser. That includes HTML, CSS, and JavaScript, all of which you will need to have *intricate* knowledge of if you want a job as a web developer. Let's take a quick look at each.

Authoring/markup (HTML)

Authoring is the process of preparing content for delivery on the web, or more specifically, marking up the content with HTML tags that describe its content and function.

HTML (HyperText Markup Language) is the authoring language used to create web page documents. The current version (and the version documented

in this book) is HTML 5.2. **Appendix D, From HTML+ to HTML5**, tells the history of HTML and lists what makes HTML5 unique.

HTML is not a programming language; it is a markup language, which means it is a system for identifying and describing the various components of a document such as headings, paragraphs, and lists. The markup indicates the document's underlying structure (you can think of it as a detailed, machine-readable outline). You don't need programming skills—only patience and common sense—to write HTML.

The best way to learn HTML is to write out some pages by hand, as we will be doing in the exercises in **Part II** of this book.

Styling (CSS)

While HTML is used to describe the content in a web page, Cascading Style Sheets (CSS) describe how that content should *look* (see **Note**). The way the page looks is referred to as its presentation. Fonts, colors, background images, line spacing, page layout, and so on, are all controlled with CSS. You can even add special effects and basic animation to your page.

The CSS specification also provides methods for controlling how documents will be presented in contexts other than a browser, such as in print or read aloud by a screen reader; however, we won't be covering them much here.

Although it is possible to publish web pages using HTML alone, you'll probably want to take on style sheets so you're not stuck with the browser's default styles. If you're looking into designing websites professionally, either as a designer or as a developer, proficiency at style sheets is mandatory.

JavaScript and DOM scripting

JavaScript is a scripting language that adds interactivity and behaviors to web pages, including these (to name just a few):

- Checking form entries for valid entries
- Swapping out styles for an element or an entire site
- Loading scrolling feeds with more content automatically
- Making the browser remember information about users
- Building interface widgets, such as embedded video players or special form inputs

You may also hear the term DOM scripting used in relation to JavaScript. DOM stands for Document Object Model, and it refers to the standardized list of web page elements that can be accessed and manipulated using JavaScript (or another scripting language).

Frontend developers may also be required to be familiar with JavaScript frameworks (such as React, Bootstrap, Angular, and others) that automate a lot of the production process. They'll likely also need to be handy with AJAX

NOTE

When this book uses the term "style sheets," it always refers to Cascading Style Sheets, the standard style sheet language for the World Wide Web. Style sheets (including what "cascading" means!) are discussed further in **Part III**.

The World Wide Web Consortium

The World Wide Web Consortium (called the W3C for short) is the organization that oversees the development of web technologies such as HTML, CSS, and JavaScript. The group was founded in 1994 by Tim Berners-Lee, the inventor of the web, at the Massachusetts Institute of Technology (MIT).

In the beginning, the W3C concerned itself mainly with the HTTP protocol and the development of HTML. Now, the W3C is laying a foundation for the future of the web by developing dozens of technologies and protocols that must work together in a solid infrastructure.

For the definitive answer to any web technology question, the W3C site is the place to go: *www.w3.org*.

For more information on the W3C and what it does, see this useful page: *www.w3.org/Consortium/*.

(which stands for "Asynchronous JavaScript And XML"), a technique used to load content in the background, allowing the page to update smoothly without reloading (like those automatically refreshing feeds).

Web scripting definitely requires some traditional computer programming prowess. While many web developers have degrees in computer science, it is also common for developers to be self-taught. A few developers I know started by copying and adapting existing scripts, then gradually added to their programming skills with each new project. Still, if you have no experience with programming languages, the initial learning curve may be a bit steep.

If you want to be a web developer for a living, JavaScript is a basic requirement. Designers will benefit from understanding what JavaScript can do, but may not need to learn to write it if they are working with a development team. **Chapter 21, Introduction to JavaScript**, will get you started understanding how it works, and I recommend *Learning JavaScript* by Ethan Brown (O'Reilly) to learn more.

Backend development

Backend developers focus on the server, including the applications and databases that run on it. They may be responsible for installing and configuring the server software (we'll be looking more at servers in **Chapter 2, How the Web Works**). They will certainly be required to know at least one, and probably more, server-side programming languages, such as PHP, Ruby, .NET (or ASP.NET), Python, or JSP, in order to create applications that provide the functionality required by the site. Applications handle tasks and features like forms processing, content management systems (CMSs), and online shopping, just to name a few.

Additionally, backend developers need to be familiar with configuring and maintaining databases that store all of the data for a site, such as the content that gets poured into templates, user accounts, product inventories, and more. Some common database languages include MySQL, Oracle, and SQL Server.

Backend development is well beyond the scope of this book, but it is important to know the sorts of tasks that get taken care of at the server level. You should be aware that it is possible to get functionality like shopping carts, mailing lists, and so on as prepackaged solutions from your hosting company without having to program it from scratch.

■ AT A GLANCE

Backend Development

The following technologies are typically in the domain of the backend developer:

- Server software (Apache, Microsoft IIS)
- Web application languages (PHP, Ruby, Python, JSP, ASP.NET)
- Database software (MySQL, Oracle, SQL Server)

Full-Stack Developers and Unicorns

When looking for a job in web development, you will frequently see posts looking for "full-stack" developers. That means a person who is fluent in both frontend (HTML, CSS, JavaScript) and backend (server applications, databases) languages.

There is a rare breed of web designer who can handle *all* of the tasks mentioned earlier—from content strategy to UX to frontend development to what happens on the server. These folks are known in the biz as "unicorns." I've met a few!

Other Roles

Not surprisingly, there are a myriad of other roles that contribute to the creation and maintenance of a site. Here are a few common roles that fall just outside the moniker "web design."

Product manager

The product manager of a website or application guides its design and development in a way that meets business goals. This member of the team must have a thorough understanding of the target market as well as the processes involved in the creation of the site itself. Product managers develop the overall strategy for the site from a marketing perspective, including how and when it gets released.

Project manager

The project manager coordinates the designers, developers, and everyone else who is working on the site. They manage things like timelines, development approaches, deliverables, and so on. The project manager works with the product manager and other product owners to make sure that the project gets done on time and on budget.

SEO specialist

A website or application isn't much good if nobody knows it exists, so it is crucial that a site be easily found by search engines. Search Engine Optimization (SEO) is a discipline focused on tweaking the site structure and code in a way that increases the chances it will be highly ranked in search results. There may be an SEO specialist on the in-house team, or a company may choose to hire an outside SEO firm. SEO is sometimes perceived as a dark art, but there are many ways to improve findability that are not underhanded. In fact, the number one technique for improving SEO is simply having good content with savvy HTML markup.

Multimedia producers

One of the cool things about the web is that you can add multimedia elements to a site, including sound, video, animation, and even interactive games. Creating multimedia elements is generally best left to artists and technicians in those fields, although they may be part of the web team if video, animation, or interactivity are core to the site's mission.

That concludes our stroll through the virtual village of workers involved in the creation of a website. The larger the site, the more likely each team member will have a narrow specialization and job titles like "UX Lead for Error Messages." More likely, everybody on the team will possess a spectrum of skills, and the lines between disciplines will blur. For example, I do Interaction and User Interface design, graphic design, HTML, and CSS, but I do not write JavaScript, work on the server, or get involved with content organization. In this book, I aim to give you a foundation in the frontend technologies that will prepare you for a number of roles.

(Soft) Skills Every Web Designer Needs

We've focused on quite a few technical skills that will be helpful in building websites. I would like to mention a few more—often overlooked—skills that are just as critical to your success.

Excellent communication skills

In your work, you will need to communicate in person, on the phone, in email, and in text messaging tools with clients, team members, and superiors. Be clear, proactive, and straightforward with what you have to say. Good communication requires not only that you express yourself clearly, but also that you be a good listener. Make sure that you understand issues being discussed, and don't be afraid to ask for clarification if you don't.

Flexibility

Be able to change direction quickly because not only does web technology change quickly, but you will no doubt be thrown curveballs in your day-to-day work as well. For example, you may arrive at work one day to find that the client has changed your priorities completely. You might find that they've cancelled your project entirely. You might be asked to learn new skills and shift positions in the team. Staying adaptable is the key to survival.

Critical thinking and good judgment

Problem-solving is central to all of the disciplines related to web design, so you need to be able to use critical thinking skills to come up with solutions and always employ basic common sense.

A good attitude

Creating sites means being part of a team, even if you work at home as a freelancer. Be mindful that the attitude with which you approach your work is contagious, so strive to be a positive and friendly team member.

GEARING UP FOR WEB DESIGN

It should come as no surprise that professional web designers require a fair amount of gear, both hardware and software. One question I'm frequently asked is, "What do I need to buy?" I can't tell you specifically what to buy, but I will provide an overview of the typical tools of the trade.

Equipment

For a comfortable web development environment, I recommend the following equipment:

A solid, up-to-date computer

Macintosh, Windows, or Linux is fine, so use whatever you have and are comfortable with. Creative departments in professional web development companies tend to be Mac-based. For backend work, Linux and Windows are popular. Although it is nice to have a super-fast machine, the files that make up web pages are very small and tend not to be too taxing on computers. Unless you're getting into sound and video editing, don't worry if your current setup is not the very latest and greatest.

A large monitor

Although not a requirement, a large monitor makes life easier. The more monitor real estate you have, the more windows and control panels you can have open at the same time. You can also see more of your page to make design decisions. If you're using a large monitor, just make sure you design for users with smaller monitors and devices in mind.

A second computer for testing

Many designers and developers find it useful to have a test computer running a different platform than the computer they use for development (i.e., if you design on a Mac, test on a PC). Because browsers work differently on Macs than on Windows machines, it's critical to test your pages in as many environments as possible, and particularly on the current Windows operating system. If you are a hobbyist web designer working at home, you could check your pages on a friend's machine. Mac users should check out the **"Run Windows on Your Mac"** sidebar.

Mobile devices for testing

The web has gone mobile! That means it is absolutely critical that you test the appearance and performance of your site on browsers on smartphones and tablet devices. Device testing is discussed in **Chapter 17, Responsive Web Design**.

A scanner and/or camera

If you anticipate making your own images and textures, you'll need some tools for creating them.

Web Production Software

There's no shortage of software available for creating web pages. In the early days, we just made do with tools originally designed for print. Today, there are wonderful tools created specifically with web design in mind that make the process more efficient. It is a delicate business listing software in a book such as this because a) there are *so many* programs, b) everyone has their personal favorite, and c) new tools come along so rapidly that there are surely newer, cooler options that you have access to that didn't exist as I wrote this.

Run Windows on Your Mac

If you have a Macintosh computer with an Intel chip running macOS (Leopard or later), you don't need a separate computer to test in a Windows environment. It is now possible to run Windows right on your Mac using the free Boot Camp application, which allows you to switch to Windows on reboot.

There are several other VM (Virtual Machine) products for macOS that allow you to toggle between Mac and Windows, including these:

- VMFusion (*www.vmware.com/fusion*) is a commercial product with a free trial you can download.
- Parallels Desktop for Mac (*www.parallels.com*) is also a commercial product with a free trial.
- Oracle VirtualBox (*virtualbox.org*) is a free program that allows you to run a number of guest operating systems, including Windows and several flavors of Unix.

All VM products require that you purchase a copy of Microsoft Windows, but it sure beats buying a whole machine.

NOTE

To do the exercises in this book, all you'll need is the text editor that came with your operating system and free image creation software. There is no need to purchase anything to follow along.

That said, here is a general overview of the types of software that comprise the tools of our trade, along with a few specific mentions of the most popular in each class.

Coding tools

Although you can get by with the simple text editors that come with your computer, a dedicated code editor makes the task of writing HTML, CSS, and JavaScript much easier. Code editors understand the syntax of the code you write, so they can do things for you like color coding, error detection, and automatically finishing simple tasks like closing HTML tags. Some provide page previews so you can view the results of your code as you work.

FIGURE 1-6 shows how an HTML document looks in the Sublime Text editor. Here are just a few of the better-known code editors for web production that are worth exploring:

- Sublime Text (*sublimetext.com*)
- Atom (free from GitHub; *atom.io*)
- Brackets (free from Adobe; *brackets.io*)
- CodeKit (*codekitapp.com*; *Mac only*)
- Adobe Dreamweaver (*www.adobe.com/products/dreamweaver.html*)
- Coda (*panic.com/coda/*)
- Microsoft Visual Studio (*visualstudio.com*)

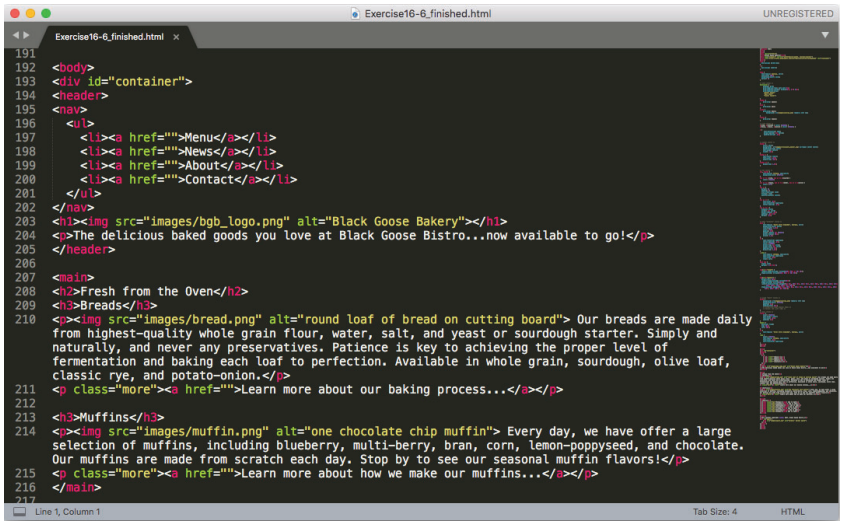


FIGURE 1-6. Sublime Text is one example of a dedicated code editor.

User interface and layout tools

There is a new breed of interface design tools made specifically for websites and other applications. Because they have been designed from scratch with interface design in mind, they seem to anticipate a web designer's every need. Interface design tools make it easy to design multiple layouts (such as layouts at various screen sizes) as well as export images and code for use in production. Some allow basic interactivity such as clicks and swipes, so your mock-ups can be shared online and used for basic interface testing.

Sketch (*sketchapp.com*, Mac only), shown in FIGURE 1-7, is extremely popular at the time of this writing. Other options include the following:

- Affinity Designer (*affinity.serif.com/en-us/designer/*)
- Adobe XD (*www.adobe.com/products/xd.html*)
- Figma (*figma.com*)
- UXPin (*uxpin.com*)

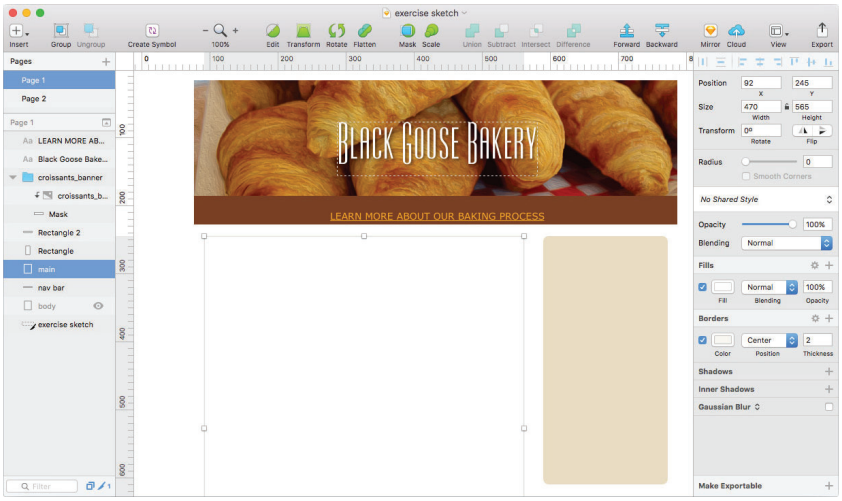


FIGURE 1-7. Sketch (Mac only) is an example of an interface design tool.

Web graphic creation tools

It is certainly possible to create all of the images you need for a site by using one of the interface design tools just listed. There are also programs that focus solely on image creation that can export files in web-appropriate formats. For professional designers, the Adobe Creative Cloud (*adobe.com*) suite of tools, which includes Photoshop (FIGURE 1-8), Illustrator, and other high-end design tools, is worth the investment.

If the Adobe monthly subscription fee is out of reach, you can try lower-cost alternatives that provide many of the same features. The number of graphics tools out there is dizzying, so I'm gathering just a few here:

- GIMP (free, open source; *gimp.org*)
- Corel PaintShop Pro (for photo editing; *paintshoppro.com*; *Windows only*)
- Corel Draw (for vector drawing; *coreldraw.com*; *Windows only*)
- Pixelmator (*pixelmator.com*; *Mac only*)

The following image editors work right in your browser, without the need to download a program, although you do need to pay for an account:

- SumoPaint (*sumopaint.com*)
- Pixlr (*pixlr.com*)

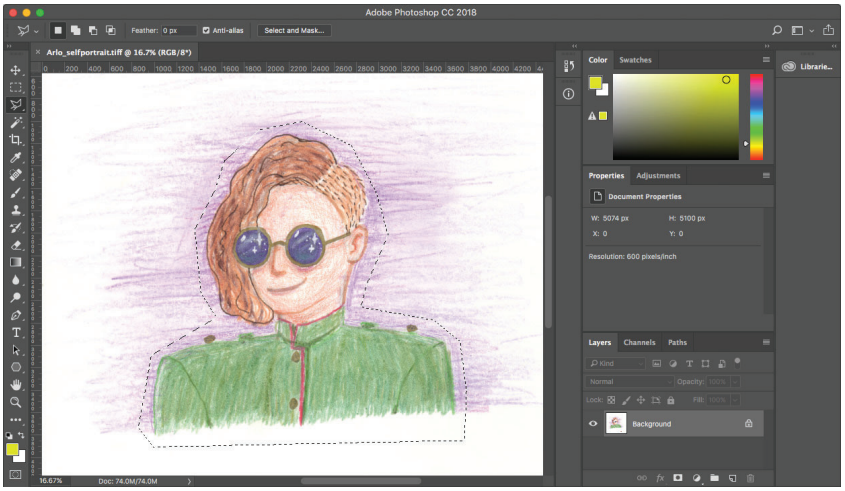


FIGURE 1–8. Adobe Photoshop is the professional standard for image editing.

A variety of browsers

One of the biggest challenges for web designers is that our sites may look and behave differently from browser to browser. For this reason, it is critical that we test our designs early and often on the widest range of browsers possible. These are the browsers designers and developers keep around for testing:

- Chrome (*google.com/chrome*)
- Firefox (*www.mozilla.org*)
- MS Edge (*www.microsoft.com/en-us/windows/microsoft-edge*; *Windows only*)
- Internet Explorer 9–11 (*www.microsoft.com*; search "Internet Explorer"; *Windows only*)
- Safari (*support.apple.com/downloads/#safari*; *Mac only*)
- Opera (*opera.com*)

You will also need to test on a variety of smartphone browsers including iOS Safari, Android browsers, and third-party mobile browsers. We will discuss mobile testing further in **Chapter 17**.

File management and transfer tools

Web design and development involves a lot of moving files around, particularly from the computer where you do your work to the server computer that hosts the site. To move files across the internet, you use an FTP (short for File Transfer Protocol) program. You will find that many hosting services offer their own FTP tools for uploading your files to their servers. Many of the code editors listed earlier also include built-in FTP functionality. Or, you can use a standalone FTP program, such as one of these:

- Filezilla (*filezilla-project.org*; *free, all platforms*)
- Cyberduck (*cyberduck.io*; *Mac and Windows*)
- WinSCP (*winscp.net/eng/index.php*; *free, Windows only*)
- Transmit (*panic.com/transmit/*; *Mac only*)

You may also find it useful to have a terminal application (command-line tool) that allows you to type Unix commands for setting file permissions, moving or copying files and directories, or managing the server software. Command-line tools, which have a number of uses in web design and development workflow, are discussed in more detail in **Chapter 20, Modern Web Development Tools**:

- Terminal (installed with macOS; shown in FIGURE 1-9)
- Cygwin (*cygwin.com*; Linux emulator for Windows that includes a command-line tool)

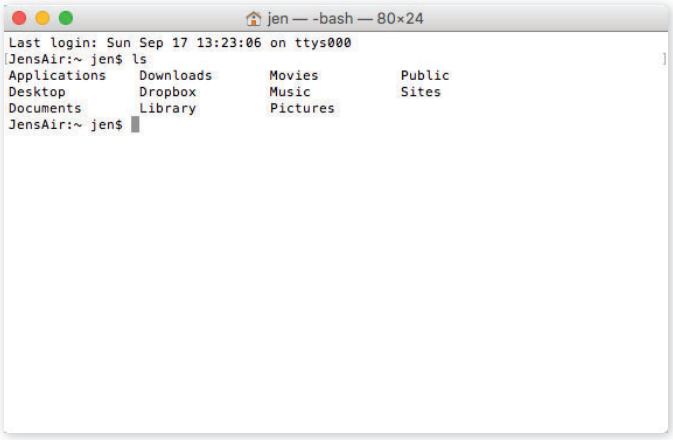


FIGURE 1-9. The Terminal command-line tool for macOS.

EXERCISE 1-1.
Taking stock

Now that you're taking that first step in learning web design, it might be a good time to take stock of your assets and goals. Using the lists in this chapter as a general guide, try jotting down answers to the following questions:

- What are your web design goals? To become a professional web designer? To make personal websites only?
- Which aspects of web design interest you the most?
- What current skills do you have that will be useful in creating web pages?
- Which skills will you need to brush up on?
- Which hardware and software tools do you already have for web design?
- Which tools do you need to buy? Which tools would you like to buy eventually?

WHAT YOU'VE LEARNED

I hope that this chapter has given you an overview of the many roles and responsibilities that fall under the umbrella of "web design." I also hope that you come away realizing that you don't need to learn everything. And even if you want to learn everything eventually, you don't need to learn it all at once. So relax, and don't worry. The other good news is that, while many professional tools exist, it is possible to create a basic website and get it up and running without spending much money by using freely available or inexpensive tools and your existing computer setup.

As you'll soon see, it's easy to get started making web pages—you will be able to create simple pages by the time you're done reading this book. From there, you can continue adding to your bag of tricks and find your particular niche in web design. In the meantime, try answering the questions in EXERCISE 1-1.

TEST YOURSELF

Each chapter in this book ends with a few questions that you can answer to see if you picked up the important bits of information. Answers appear in **Appendix A**.

1. Match these web professionals with the final product they might be responsible for producing:

a. Graphic designer	_____ HTML and CSS documents
b. Production department	_____ PHP scripts
c. User experience designer	_____ "Look and feel" deliverables
d. Backend programmer	_____ Storyboards

2. What does the W3C do?

3. Match the web technology with its appropriate task:

a. HTML	_____ Checks a form field for a valid entry
b. CSS	_____ Creates a custom server-side web application
c. JavaScript	_____ Identifies text as a second-level heading
d. Ruby	_____ Makes all second-level headings blue

4. What is the difference between frontend and backend web development?

5. What does an FTP tool do and how do you get one?

CHAPTER 2

HOW THE WEB WORKS

I got started in web design in early 1993—pretty close to the start of the web itself. That's a quarter of a century ago (gasp!), but I still distinctly remember the first time I looked at a web page. It was difficult to tell where the information was coming from and how it all worked.

This chapter sorts out the pieces and introduces some basic terminology. We'll start with the big picture and work down to specifics.

IN THIS CHAPTER

An explanation of the web as it relates to the internet

The role of the server

The role of the browser

URLs and their components

The anatomy of a web page

THE INTERNET VERSUS THE WEB

No, it's not a battle to the death, just an opportunity to point out the distinction between two words that are increasingly being used interchangeably.

The internet is an international network of connected computers. No company owns the internet; it is a cooperative effort governed by a system of standards and rules. The purpose of connecting computers together, of course, is to share information. There are many ways information can be passed between computers, including email (POP3/IMAP/SMTP), file transfer (FTP), secure shell (SSH), and many more specialized modes upon which the internet is built. These standardized methods for transferring data or documents over a network are known as protocols.

The web (originally called the World Wide Web, thus the "www" in site addresses) is just one of the ways information can be shared over the internet. It is unique in that it allows documents to be linked to one another via hypertext links—thus forming a huge "web" of connected information. The web uses a protocol called HTTP (HyperText Transfer Protocol). That acronym should look familiar because it is the first four letters of nearly all website addresses, as we'll discuss in an upcoming section.

The web is a subset of the internet. It is just one of many ways information can be transferred over networked computers.

A Brief History of the Web

The web was born in a particle physics laboratory (CERN) in Geneva, Switzerland, in 1989. There a computer specialist named Tim Berners-Lee first proposed a system of information management that used a "hypertext" process to link related documents over a network. He and his partner, Robert Cailliau, created a prototype and released it for review. For the first several years, web pages were text-only. It's difficult to believe that in 1992, the world had only about 50 web servers, total.

The real boost to the web's popularity came in 1992 when the first graphical browser (NCSA Mosaic) was introduced, and the web broke out of the realm of scientific research into mass media. The ongoing development of web technologies is overseen by the World Wide Web Consortium (W3C).

If you want to dig deeper into the web's history, check out the W3C's History Archives at *www.w3.org/History.html*.

FUN FACT: If you look at that page, you'll see a July 1993 entry for the first "WWW Wizards Workshop." Although I did not attend that meeting, I *did* design the commemorative t-shirt!

SERVING UP YOUR INFORMATION

Let's talk more about the computers that make up the internet. Because they "serve up" documents upon request, these computers are known as servers. More accurately, the server is the software (not the computer itself) that allows the computer to communicate with other computers; however, it is common to use the word "server" to refer to the computer as well. The role of server software is to wait for a request for information, and then retrieve and send that information back as quickly as possible.

There's nothing special about the computers themselves...picture anything from a high-powered Unix machine to a humble personal computer. It's the server software that makes it all happen. In order for a computer to be part of the web, it must be running special web server software that allows it to handle HyperText Transfer Protocol transactions. Web servers are also called HTTP servers.

There are many server software options out there, but the two most popular are Apache (open source software) and Microsoft Internet Information Services (IIS). Apache is freely available for Unix-based computers and comes installed on Macs running macOS. There is a Windows version as well. Microsoft IIS is part of Microsoft's family of server solutions.

Every computer and device (router, smartphone, car, etc.) connected to the internet is assigned a unique numeric IP address ("IP" stands for "Internet Protocol"). For example, as I write this, the computer that hosts *oreilly.com* has the IP address 199.27.145.64. All those numbers can be dizzying, so fortunately, the Domain Name System (DNS) was developed to allow us to refer to

■ TERMINOLOGY

Open Source

Open source software is developed as a collaborative effort with the intent to make its source code available to other programmers for use and modification. Open source programs are usually free to use.

that server by its domain name, "oreilly.com", as well. The numeric IP address is useful for computer software, while the domain name is more accessible to humans. Matching the text domain names to their respective numeric IP addresses is the job of a separate DNS server. If you think of an IP address as a telephone number, the DNS server would be the phonebook.

It is possible to configure your web server so that more than one domain name is mapped to a single IP address, allowing several sites to share a single server.

A WORD ABOUT BROWSERS

We now know that the server does the servin', but what about the other half of the equation? The software that does the requesting is called the client. People use desktop browsers, mobile browsers, and other assistive technologies (such as screen readers) as clients to access documents on the web. The server returns the documents for the browser (also referred to as the user agent in technical circles) to display.

The requests and responses are handled via the HTTP protocol, mentioned earlier. Although we've been talking about "documents," HTTP can be used to transfer images, movies, audio files, data, scripts, and all the other web resources that commonly make up websites and applications.

It is common to think of a browser as a window on a computer monitor with a web page displayed in it. These are known as graphical browsers or desktop browsers and for a long time, they were the only web-viewing game in town. The most popular desktop browsers as of this writing include Edge and Internet Explorer for Windows, Chrome, Firefox, and Safari, with Opera and Vivaldi bringing up the rear.

These days, however, more than half of web traffic comes from mobile browsers on smartphones and tablets such as Safari on iOS, Android and Chrome browsers on Android devices, Opera Mini, and a myriad of other default and installable mobile browsers (see *en.wikipedia.org/wiki/Mobile_browser* for a complete list). Navigating the web on a touch screen is the new normal.

It is also important to keep alternative web experiences in mind. Users with impaired sight may be listening to a web page read by a screen reader (or simply make their text extremely large). Users with limited mobility may use assistive technology such as joysticks or voice commands to access links and enter content. The sites we build must be accessible and usable for all users, regardless of their browsing experiences.

The web is also finding its way onto smart TVs and gaming systems, where users access our pages with TV remotes or Xbox controllers. You never know where the web will pop up next!

TERMINOLOGY

Intranets and Extranets

When you think of a website, you generally assume that it is accessible to anyone surfing the web. However, many organizations take advantage of the awesome information sharing and gathering power of websites to exchange information just within their own network. These special web-based networks are called intranets. They are created and function like ordinary websites, but they use special security devices (called firewalls) that prevent the outside world from seeing them. Intranets have lots of uses, such as sharing human resource information or providing access to inventory databases.

An extranet is like an intranet, but it allows access to select users outside of the organization. For example, a manufacturing company may provide its customers with passwords that allow them to check the status of their orders in the company's orders database. Passwords determine which slice of the company's information is accessible.

TERMINOLOGY

Server-Side and Client-Side

Often in web design, you'll hear references to "client-side" or "server-side" applications. These terms are used to indicate which machine is doing the processing. Client-side applications run on the user's machine (also referred to as the frontend), while server-side applications and functions use the processing power of the server computer (the backend).

Browser Rendering Engines

The program that is responsible for converting HTML and CSS into what you see rendered on the screen is called a rendering engine (also browser engine or layout engine). Browsers that you use on desktop computers and mobile devices are made up of rendering engines as well as other code used for their own user interfaces and functionality. Although I talk a lot about which browsers support particular functions in this book, I'm technically referring to the browser's rendering engine. Various browsers often share a rendering engine; for example, the Blink engine powers Chrome, Opera, and a variety of Android browsers. TABLE 2-1 lists the rendering engines used by the most popular web browsers today. For more information, search Wikipedia.com for "Comparison of web browser engines" and "Comparison of web browsers."

TABLE 2-1. Current browsers and their rendering engines

Browser	Rendering engine
Chrome 28+	Blink (forked from WebKit)
Firefox (all)	Gecko (except Firefox for iOS, which uses WebKit)
Safari and Safari iOS (all)	WebKit
Internet Explorer 4–11	Trident
MS Edge (all)	EdgeHTML (forked from Trident)
Opera 15+	Blink (forked from WebKit)

The reality is that pages may look and perform differently from browser to browser. This is due to varying support for web technologies, varying device capabilities, and the users' ability to set their own browsing preferences. It is the most challenging aspect of designing and developing for our medium.

WEB PAGE ADDRESSES (URLS)

Every page and resource on the web has its own special address called a URL, which stands for Uniform Resource Locator. It's nearly impossible to get through a day without seeing a URL (pronounced "U-R-L," not "erl") plastered on the side of a bus, printed on a business card, or broadcast on a television commercial. Web addresses are fully integrated into modern vernacular.

Some URLs are short and sweet. Others may look like crazy strings of characters separated by dots (periods) and slashes, but each part has a specific purpose. Let's pick one apart.

The Parts of a URL

A complete URL is generally made up of three components: the protocol, the site name, and the absolute path to the document or resource, as shown in FIGURE 2-1.

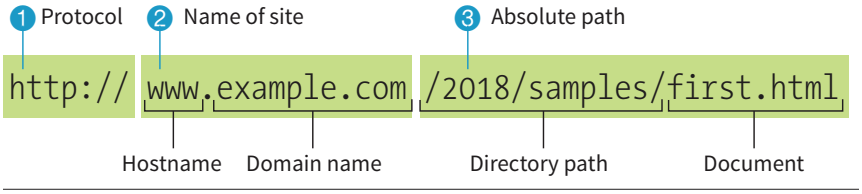


FIGURE 2-1. The parts of a URL

❶ `http://`

The first thing the URL does is to define the protocol that will be used for that particular transaction. The letters "HTTP" let the server know to use HyperText Transfer Protocol, or get into "web mode." You may also see a URL begin with `https://`, which I explain in the **"HTTPS, The Secure Web Protocol"** sidebar.

❷ `www.example.com`

The next portion of the URL identifies the website by its domain name. In this example, the domain name is "example.com." The "www." part at the beginning is the particular hostname at that domain. The hostname "www" has become a convention, but is not a rule. In fact, sometimes the hostname may be omitted. There can be more than one website at a domain (called subdomains). For example, there might also be "development.example.com," "clients.example.com," and so on.

❸ `/2018/samples/first.html`

This is the absolute path through directories on the server to the requested HTML document, *first.html*. The words separated by slashes are the directory names, starting with the root directory of the host (as indicated by the initial `/`). Because the internet originally comprised computers running the Unix operating system, our current way of doing things still follows Unix rules and conventions, hence the `/` separating directory names.

To sum it up, the URL in FIGURE 2-1 says it would like to use the HTTP protocol to connect to a web server on the internet called "www.example.com" and to request the document *first.html*, located in the *samples* directory, which is in the *2018* directory.

URL Versus URI

The W3C and the development community are moving away from the term URL (Uniform Resource Locator) and toward the more generic and technically accurate URI (Uniform Resource Identifier). On the street and even on the job, however, you're still likely to hear URL.

Here's the skinny on URL versus URI: a URL is one type of a URI that identifies the resource by its location (the L in URL) on the network. The other type of URI is a URN that identifies the resource by name or namespace (the N in URN).

Because it is more familiar, I will be sticking with URL throughout this book. Just know that URLs are a subset of URIs, and the terms are often used interchangeably.

If you like to geek out on this kind of thing, I refer you to the URI Wikipedia entry: *en.wikipedia.org/wiki/Uniform_Resource_Identifier*.

Simplified URLs

Obviously, not every URL you see is so lengthy. To get to O'Reilly's site, you'd expect to type `oreilly.com` instead of `http://www.oreilly.com/index.html`. Here's why that works.

Skipping the protocol

Because nearly all web pages use the HyperText Transfer Protocol, the **`http://`** part is often just implied. This is the case when site names are advertised in print or on TV, as a way to keep the URL easy to remember.

Additionally, browsers are programmed to add **`http://`** automatically as a convenience to save you some keystrokes. It may seem like you're leaving it out, but it is being sent to the server behind the scenes.

When we begin using URLs to create hyperlinks in HTML documents in **Chapter 6, Adding Links**, you'll learn that it is necessary to include the protocol when making a link to a web page on another server.

Pointing to default files

Many addresses do not include a filename, but simply point to a directory, like these:

```
http://www.oreilly.com
http://www.jendesign.com/resume/
```

When a server receives a request for a directory name rather than a specific file, it looks in that directory for a default document, typically named *index.html*. So when someone types the previous URLs into his browser, what he'll actually see is this:

```
http://www.oreilly.com/index.html
http://www.jendesign.com/resume/index.html
```

The name of the default file (also referred to as the index file) may vary, and depends on how the server is configured. In these examples, it is named *index.html*, but some servers use the filename *default.htm*. If your site uses server-side programming to generate pages, the index file might be named *index.php* or *Default.aspx*. Just check with your server administrator or the tech support department at your hosting service to make sure you give your default file the proper name.

Another thing to notice is that in the first example, the original URL did not have a trailing slash to indicate it was a directory. If the slash is omitted, the server checks to see if the request is a file or a directory. If it is a directory, the server asks the browser to send the request again with a slash. In the end, the slash is included for directories, even if it isn't included the first time it is entered (see **Performance Tip**).

The index file is also useful for security. Some servers (depending on their configuration) display the contents of the directory if the default file is not found. FIGURE 2-2 shows how the documents in the *housepics* directory are exposed as the result of a missing default file. One way to prevent people from snooping around in your files is to be sure there is an index file in every directory. Your server administrator may also add other protections to prevent your directories from displaying in the browser.

HTTPS, the Secure Web Protocol

If you look at the address bar while shopping online or using a banking site, you'll notice that they use the HTTPS protocol. HTTPS, where "S" stands for "secure," is a modification of HTTP that encrypts form information when it is sent between the user's client and the server. Any web page that has form fields that accept text (such as a search bar or a login) should use HTTPS.

As of this writing, around 60% of pages (and growing!) use HTTPS, and for good reason. Not only is it a good idea to keep your user's data secure in transit, but Google is pushing along the transition to HTTPS with some serious incentives as well. If you have a site that accepts text input and you don't use HTTPS, your site won't rise as high in the Google search results. In addition, in Chrome, these sites are marked with "Not Secure" in the top bar of the browser.

HTTPS works in tandem with another protocol, SSL (for Secure Socket Layer), which needs to be enabled on the server for secure transactions to work. Hosting companies have options for enabling SSL, often for free.

Keep in mind that HTTPS protects form data as it is sent to the server, but doesn't do anything to make your site "secure" and safe from hackers.

■ PERFORMANCE TIP

If you want to minimize round-trips to the server, include slashes at the end of directory names in URLs in your links.

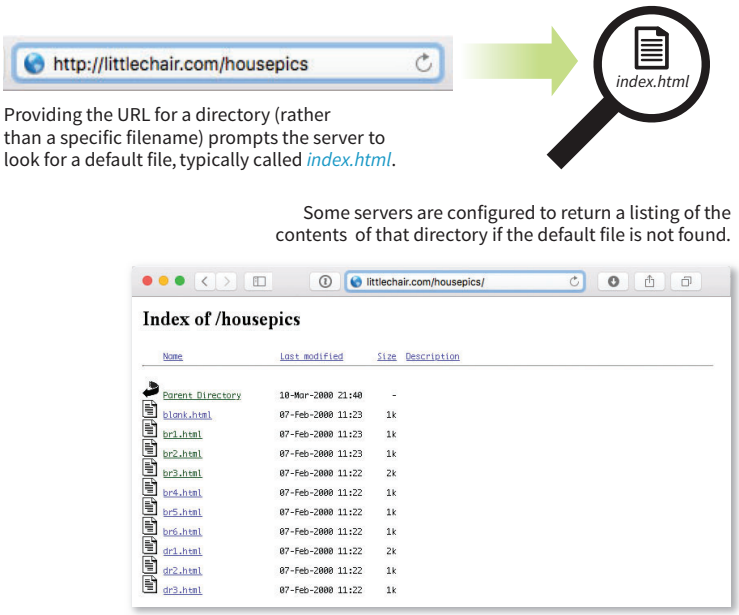


FIGURE 2-2. Some servers display the contents of the directory if an index file is not found.

THE ANATOMY OF A WEB PAGE

We're all familiar with what web pages look like in the browser window, but what's happening "under the hood"?

At the top of FIGURE 2-3, you see a minimal web page as it appears in a graphical browser. Although you see it as one coherent page, it is actually assembled from four separate files: an HTML document (*index.html*), a style sheet (*kitchen.css*), and two graphics (*foods.png* and *spoon.png*). The HTML document is running the show.

HTML Documents

You may be as surprised as I was to learn that the graphically rich and interactive pages we see on the web are generated by simple, text-only documents. The text file behind the scenes is referred to as the source document.

Take a look at *index.html*, the source document for the Jen's Kitchen web page. You can see that it contains the text content of the page plus special tags (indicated with angle brackets, < and >) that describe each element on the page.

The web page shown in this browser window consists of four separate files:

- An HTML text document
- A style sheet
- Two images

Tags in the HTML source document give the browsers instructions for how the text is structured and where the images should be placed.

index.html

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>Jen's Kitchen</title>
  <link rel="stylesheet" href="kitchen.css" type="text/css">
</head>

<body>
<h1><img src="foods.png" alt="food illustration"> Jen's Kitchen</h1>

<p>If you love to read about <strong>cooking and eating</strong>, would like to learn about some of the best
restaurants in the world, or just want a few choice recipes to add to your collection, <em>this is the site
for you!</em></p>

<p><img src="spoon.png" alt="spoon illustration"> Your pal, Jen at Jen's Kitchen</p>

<hr>
<small>Copyright 2018, Jennifer Robbins</small>
</body>
</html>
```

kitchen.css

```
body { font: normal 1em Verdana; width: 80%; margin: 1em auto; }

h1 { font: italic 3em Georgia; color: rgb(23, 109, 109);
     margin: 1em 0 1em; }

img { margin: 0 20px 0 0; }

h1 img { margin-bottom: -20px; }

small { color: #666666; }
```

foods.png

spoon.png

FIGURE 2-3. The source file, style sheet, and images that make up a simple web page.

Adding descriptive tags to a text document is known as "marking up" the document. Web pages use a markup language called HyperText Markup Language, or HTML for short, which was created especially for documents with hypertext links. HTML defines dozens of text elements that make up documents such as headings, paragraphs, emphasized text, and of course, links. There are also elements that add information about the document (such as its title), media such as images and videos, and widgets for form inputs, just to name a few.

You can view the source for any web page. EXERCISE 2-1 gives you some prompts and pointers.

The version of HTML we use today is HTML5. There have been several versions of HTML since its birth in 1989, and a few that are still in use today. There is a complete history of HTML, all its versions, and an overview of what makes HTML5 unique in **Appendix D, From HTML+ to HTML5**.

EXERCISE 2-1. View source

You can see the HTML file for any web page by viewing its source in a desktop browser. Most modern browsers keep the View Source function with the developer tools and typically open the source document in a separate window or in a developer's panel at the bottom of the current window.

Here's where to find the View Source function on the major desktop browsers:

Safari: **Develop → Show → Page Source**

Chrome: **View → Developer → View Source**

Firefox: **Tools → Web Developer → Page Source**

MS Edge: Right-click on the page and select **View Source**. If you do not see that option in the context menu, you may need to turn it on in the Developer Settings. Open a new browser window and type **`about:flags`** in the address bar. Under "Developer settings," check "Show View source" and "Inspect element" in the context menu. Now when you go to a web page, you can right-click on the page and access the View Source function. You may also use the Ctrl+U keyboard shortcut or F12 key.

1. With the browser of your choice, enter this URL into your browser:

   ```
   www.learningwebdesign.com/5e/kitchen.html
   ```

   You should see the Jen's Kitchen web page from FIGURE 2-3.
2. Follow the directions for your browser listed above to view the source HTML document for the Jen's Kitchen page. It should be the same as shown in the figure.
3. To view a page that is a little more complicated, take a look at the source for the *learningwebdesign.com* home page.
4. The source for most sites is considerably more complicated. View the source of *oreilly.com*. It's got style sheets, scripts, inline SVG graphics…the works! Don't worry if you don't understand what's going on. Much of it will look more familiar by the time you are done with this book.

WARNING

Keep in mind that while learning from others' work is fine, stealing other people's code is poor form (or even illegal). If you want to use code as you see it, ask for permission and always give credit to those who did the work.

A Quick Introduction to HTML Markup

You'll be learning the nitty-gritty of markup in **Part II**, so I don't want to bog you down with too much detail right now, but there are a few things I'd like to point out about how HTML works and how browsers interpret it.

Read through the HTML document in FIGURE 2-3 and compare it to the browser results. It's easy to see how the elements marked up with HTML tags in the source document correspond to what displays in the browser window.

First, you'll notice that the text within brackets (for example, `<body>` and `<strong>`) does not display in the final page. The browser displays only what's between the tags—the content of the element. The markup is hidden. The tag provides the name of the HTML element—usually an abbreviation such as "h1" for "heading level 1," or "em" for "emphasized text."

Second, you'll see that most of the HTML tags appear in pairs surrounding the content of the element. In our HTML document, `<h1>` indicates that the following text should be a first-level heading; `</h1>` indicates the end of the heading. Some elements, called empty elements, do not have content. In our sample, the `<hr>` tag indicates an empty element that tells the browser to "insert a horizontal rule here" as a thematic divider.

Because I was unfamiliar with computer programming when I first began writing HTML, it helped me to think of the tags and text as "beads on a string" that the browser interprets one by one, in sequence. For example, when the browser encounters an open bracket (`<`), it assumes all of the following characters are part of the markup until it finds the closing bracket (`>`). Similarly, it assumes all of the content following an opening `<h1>` tag is a heading until it encounters the closing `</h1>` tag. This is the manner in which the browser parses the HTML document. Understanding the browser's method can be helpful when troubleshooting a misbehaving HTML document.

Where Are the Pictures?

Obviously, there are no pictures in the HTML file itself, so how do they get there when you view the final page? You can see in FIGURE 2-3 that each image is a separate file. The images are placed in the flow of the text with the HTML image element (`img`), which tells the browser where to find the graphic (its URL). When the browser sees the `img` element, it makes another request to the server for the image file, and then places it in the content flow.

The browser also sends requests to the server for style sheets (like *kitchen.css*), JavaScript files (*.js*), and other embedded media like audio and videos. The browser software (or more specifically, its rendering engine) brings the separate pieces together into the final page.

The assembly of the page generally happens in an instant, so it appears as though the whole page loads all at once. Over slow connections or if the page includes huge graphics or media files, the assembly process may be more apparent as images lag behind the text. The page may even need to be redrawn as new images, fonts, and style sheets arrive (although you can construct your pages in such a way that prevents this from happening).

Adding a Little Style

I want to direct your attention to one last key ingredient of our minimal page. Near the top of the HTML document there is a `link` element that points to the style sheet document *kitchen.css*. That style sheet includes a few lines of instructions for how the page should look in the browser. These are style instructions written according to the rules of Cascading Style Sheets (CSS). CSS allows designers to add visual style instructions (known as the document's presentation) to the marked-up text (the document's structure, in web design terminology). In **Part III**, you'll get to know the power of Cascading Style Sheets.

FIGURE 2-4 shows the Jen's Kitchen page without (top) and with (bottom) the style instructions. Browsers come equipped with default styles for every HTML element they support, so if an HTML document lacks custom style instructions, the browser will use its own. That's what you see in the screenshot on the top. Even just a few style rules can make big improvements to the appearance of a page.

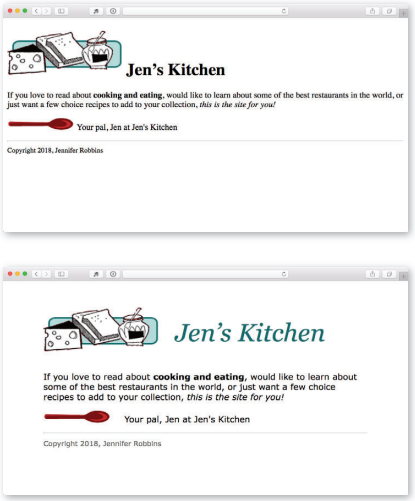


FIGURE 2-4. The Jen's Kitchen page without (top) and with (bottom) custom style rules.

Adding Behaviors with JavaScript

To make elements on the page *do* something, you use a scripting language called JavaScript (see **Note**). There are no scripts on the Jen's Kitchen page because I thought it best to keep things simple this early in the book, but know that JavaScript is an essential ingredient in modern websites.

Whereas HTML provides the structure and the CSS style sheet alters how things look, JavaScript adds a behavior component that controls how things work. Scripts may be standalone files on the server (with the *.js* suffix) or be written out right in the document. They may be triggered to run immediately when the page loads or be triggered by something the user does, like click or hover on an element or enter something in a form field.

You'll get a basic introduction to JavaScript in **Part IV** of this book.

NOTE

JavaScript is not required for the interactivity of links and web forms, which work using HTML alone.

PUTTING IT ALL TOGETHER

1. To wrap up our introduction to how the web works, let's trace a typical stream of events that occurs with every web page that appears on your screen (FIGURE 2-5). Request a web page by either typing its URL (for example, *http://jenskitchensite.com*) directly in the browser or by clicking a link on a page. The URL contains the information needed to target a specific document on a specific web server on the internet. In this case, it points to the default file (*index.html*) in the top directory.
2. Your browser sends an HTTP request to the server named in the URL and asks for the specific file. The request also includes information about what languages the user can read and what types of files the browser can accept. If the URL specifies a directory (not a file), it is the same as requesting the default file in that directory.
3. The server looks for the requested file and issues an HTTP response in the form of an HTTP header. The header includes information about the file, like the last modified date, the length of the file, and its Content-Type (for example, an *.html* file has the content type "text/html").
   a. If the page cannot be found, the server returns an error message. The message typically says "404 Not Found," although more hospitable error messages may be provided. Other error types are possible as well (see the sidebar **"HTTP Status Codes"**).
   b. If the document *is* found, the server retrieves the requested file and returns it to the browser. If the site is dynamic, the server assembles the page from stored data before returning it to the browser.
4. The browser parses the HTML document. If the page contains images (indicated by the HTML `img` element) or other external resources like scripts or style sheets, the browser contacts the server again to request each resource specified in the markup.
5. The browser inserts each image in the document flow where indicated by the `img` element, applies styles, and runs scripts. And *voilà*! The assembled web page is displayed for your viewing pleasure.

I should note that I've depicted a traditional and simplified scenario here to tell you how web pages are put together. These days, it is common for web pages to be generated from content management systems (CMSs) that keep content in databases and use templates to assemble the data into pages on the fly. In that case, in Step 3b, there is a more complicated process of assembling the file from various parts rather than just handing off an existing file.

Static vs. Dynamic Sites

Static websites consist of HTML files with fixed content that display the same information to every visitor. In other words, each page you see in the browser is a view of a single HTML file on the server. This book focuses on the creation of static web pages as they are straightforward and the best starting place for beginners.

By contrast, dynamic websites are generated with backend programming such as PHP or ASP. Each page is generated by the application on the fly. Dynamic sites access content and data from a database, and the final pages may be customized for each user. For extremely large sites with hundreds or thousands of pages, setting up and maintaining a dynamic site is considerably less work than creating and storing every page as a static HTML document individually.

HTTP Status Codes

Servers issue status codes in response to browser requests. The full list of status codes is quite long (you can read about them all at *en.wikipedia.org/wiki/List_of_HTTP_status_codes*), but here are a few common responses:

200	OK
301	Moved Permanently
302	Moved Temporarily
404	Not Found
410	Gone (no longer available)
500	Internal Server Error

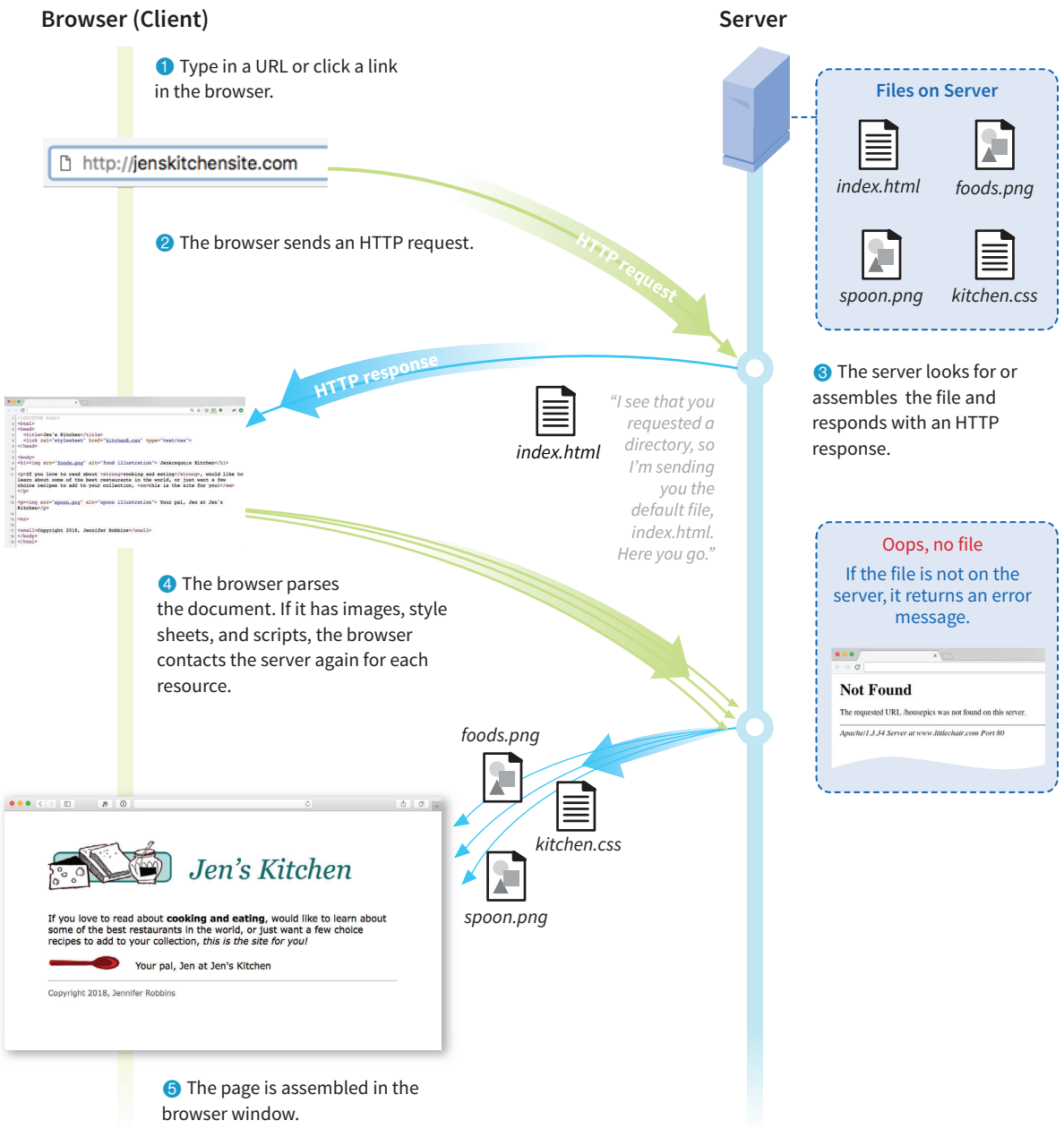


FIGURE 2-5. How browsers display web pages.

Getting Your Pages on the Web

If you would like more information about registering domain names and finding a server to host your site, download the article titled **"Getting Your Pages on the Web"** (PDF) at *learningwebdesign.com/articles/*.

TEST YOURSELF

Let's play a round of "Identify That Acronym!" The following are a few basic web terms mentioned in this chapter. Answers are in **Appendix A**.

1. HTML	_____	a. Home of Mosaic, the first graphical browser
2. W3C	_____	b. The location of a web document or resource
3. CERN	_____	c. The markup language used to describe web content
4. CSS	_____	d. Matches domain names with numeric IP addresses
5. HTTP	_____	e. A protocol for file transfer
6. IP	_____	f. Protocol for transferring web documents on the internet
7. URL	_____	g. The language used to instruct how web content looks
8. NCSA	_____	h. Particle physics lab where the web was born
9. DNS	_____	i. Internet Protocol
10. FTP	_____	j. The organization that monitors web technologies

CHAPTER
3

SOME BIG CONCEPTS YOU NEED TO KNOW

IN THIS CHAPTER

The web on mobile devices

The benefits of web standards

Progressive enhancement

Responsive Web Design

Accessibility

Site performance

As the web matures and the number of devices we access it from increases exponentially, our jobs as web designers and developers get significantly more complicated. Frankly, there's a lot more going on out there than I can fit in this book. In the chapters that follow, I will focus on the basic building blocks of web design—HTML elements, CSS styles, a taste of JavaScript, and web image production—that will give you a solid foundation for the further development of your skills.

Before we get to the nuts and bolts, I want to introduce some Big Concepts that every web designer needs to know. We'll look at ideas and concerns that inform our decisions and contribute to the contemporary web environment. I'll be referring back to the terminology introduced here frequently.

The heart of the matter is that as web designers, we never know exactly how the pages we create will be viewed. We don't know which of the dozens of browsers might be used, whether it is on a desktop computer or something more portable, how large the browser window will be, what fonts are installed, whether functionality such as JavaScript is enabled, how fast the internet connection is, whether the pages are being read by a screen reader, and so on. The Big Concepts in this chapter are primarily reactions to and methods for coping with the inescapable element of the Unknown in our medium. They include the following:

- The multitude of devices
- Web standards
- Progressive enhancement
- Responsive Web Design
- Accessibility
- Site performance

Because we're just getting started, I will keep the descriptions brief and fairly non-technical. My goal is that you have a basic understanding of what I mean by terms like "progressive enhancement" when you encounter them in lessons later. Many excellent articles and books have been written on each of these topics and their related production techniques, and I'll provide pointers to resources for further reading.

A MULTITUDE OF DEVICES

Until 2007, we could be relatively certain that our users were visiting our sites while sitting at their desks, looking at a large monitor, using a speedy internet connection. We had all more or less settled on 960 pixels as a good width for a web page based on the most common monitor size. Back then, our biggest concern was dealing with the dozen or so desktop browsers and jumping through a few extra hoops to support quirky old versions of Internet Explorer. And we thought we had it rough!

Although you could access web pages and web content on mobile phones prior to 2007, the introduction of the iPhone and Android smartphones as well as faster networks heralded a huge shift in how, when, and where we do our web surfing (particularly in the United States, which lagged behind Asia and the EU in mobile technology). Since then, we've seen the introduction of phones and tablets of all different dimensions, as well as web browsers on TVs, gaming systems, and other devices. And the diversity is only going to increase. I think mobile web design expert Brad Frost sums it up nicely in his illustrations in FIGURE 3-1.

The challenge of designing for all of these devices goes beyond addressing differing screen sizes. There is a world of difference between using a site over a broadband connection and over a slow cell network. Designers need to resist making assumptions about network speed and context based on the screen size. Just because it is a small screen doesn't mean it's a slow connection or

FIGURE 3-1. Brad Frost sums up the reality of device diversity nicely (*bradfrostweb.com*).

that the person is in a hurry. It's not uncommon to leisurely browse the web on a smartphone while sitting on the couch at home with a solid WiFi connection. And iPads with larger, high-resolution displays may be accessing the internet on pokey 3G connections. In other words, it's complicated!

For a lot of sites today, more people access the web via their mobile devices than on a desktop computer. Already, a significant portion of Americans use their mobile phones as their *only* access to the internet. That means it is critical to get the design and functionality right. We've made huge strides in serving a pleasing experience to users with handheld devices, and the technology for targeting their needs continues to head in the right direction.

What I want you to learn here is that the way you see your design as you're working on it on your nice desktop machine is not how it will be experienced by everyone. Some will see it much smaller. Some will see it load painfully slowly. Some may be looking at it on a TV across the room. All web design professionals should keep this fact in mind.

Resist making assumptions about network speed and context based on the screen size.

For Further Reading

- *Mobile First* by Luke Wroblewski (A Book Apart). Luke was way ahead of the curve in insisting that sites work well on mobile devices, and he shares his perspective in this little book, which is jam-packed with ideas.

Mobile Web?

You may hear people use the term "mobile web," but the truth is (as Stephen Hay put it in a tweet in 2011; see FIGURE 3-2), there is no Mobile Web any more than there is a Desktop Web, or a Tablet Web, or so on. There is just The Web, and it can be accessed from all manner of devices. As of this writing, "mobile web" is used as sort of a catchall term to describe our efforts to adapt our desktop design skills to accommodate a much wider variety of use cases. And, as we are finding out, there is more than one way to crack that nut.

FIGURE 3-2. Stephen Hay's tweet from January 2011. Read his follow-up article at *www.the-haystack.com/2011/01/07/there-is-no-mobile-web*.

STICKING WITH THE STANDARDS

So how do we deal with this diversity? A good start is to follow the standards documented by the World Wide Web Consortium (W3C). Sticking with web standards is your primary tool for ensuring your site is consistent on all standards-compliant browsers (that's approximately 99% of browsers in current use). It also helps make your content forward-compatible as web technologies and browser capabilities evolve. Another benefit is that you can tell your clients that you create "standards-compliant" sites, and they will like you more.

Sticking with web standards is your primary tool for ensuring your site is as consistent as possible.

The notion of standards compliance may seem like a no-brainer, but it used to be that everyone, including the browser makers, played fast and loose with HTML and scripting. The price we paid was incompatible browser implementations and the need to create sites twice to make them work for everyone. I talk more about web standards throughout this book, so I won't go into too much detail here. Suffice it to say that the web standards are your friends. Everything you learn in this book will start you off on the right foot.

For Further Reading

- The W3C site (*w3.org/standards*) is the primary resource for all web standards documents.
- The bible for standards compliance and how it makes good business sense is *Designing with Web Standards, 3rd Edition*, by Jeffrey Zeldman (New Riders). It's getting on in years, but the fundamentals are still solid.

PROGRESSIVE ENHANCEMENT

With a multitude of browsers comes a multitude of levels of support for the web standards. In fact, no browser has implemented all the standards 100%, and there are always new technologies that are slowly gaining steam. Furthermore, users can set their own browser preferences, so they may have a browser that supports JavaScript but have chosen to turn it off. The point here is that we are faced with a wide range of browser capabilities—from only basic HTML support to all the bells and whistles.

Progressive enhancement is one strategy for dealing with unknown browser capabilities (see **Note**). When designing with progressive enhancement, you start with a baseline experience that makes the content or core functionality available to even the most rudimentary browsers or assistive devices. From there, you layer on more advanced features for the browsers that can handle them. You might finish with some "nice to have" effects, like animation or wrapping text around images in interesting shapes, that enhance the experience for users with the most advanced browsers, but aren't really critical to the brand or message.

NOTE

Progressive enhancement is the flip side of an approach to browser diversity called graceful degradation, in which you design the fully enhanced experience first, then create a series of fallbacks for non-supporting browsers. Both methods have their place in modern development. You will find many fallback techniques suggested in this book to be sure less capable browsers are accommodated.

Progressive enhancement is an approach that informs all aspects of page design and production, including HTML, CSS, and JavaScript:

Authoring strategy

When an HTML document is written in logical order and its elements are marked up in a meaningful way, it will be usable on the widest range of browsing environments, including the oldest browsers, future browsers, and mobile and assistive devices. It may not look exactly the same, but the important thing is that your content is available. It also ensures that search engines like Google will catalog the content correctly. A clean HTML document with its elements accurately and thoroughly described is the foundation for accessibility.

Progressive enhancement is a strategy for coping with unknown browser capabilities.

Styling strategy

You can create layers of experience simply by taking advantage of the way browsers parse style sheet rules. Without going into too much technical detail, you can write a style rule that makes an element background red, but also include a style that gives it a cool gradient (a blend from one color to another) for browsers that know how to render gradients. Or you can use a cutting-edge CSS selector to deliver certain styles only to cutting-edge browsers. The knowledge that browsers simply ignore properties and rules they don't understand gives you license to innovate without bringing older browsers to their knees. You just have to be mindful of styling the baseline experience first, then adding improvements once the minimum requirements are met.

Scripting strategy

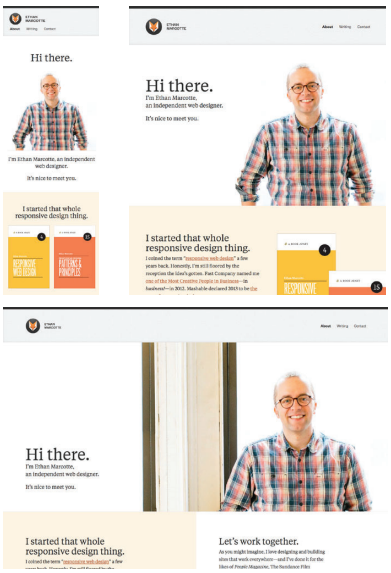
As with other web technologies, there are discrepancies in how browsers handle JavaScript (particularly on non-desktop devices), and some users opt to turn it off entirely. The first rule in progressive enhancement is to make sure basic functionality—such as linking from page to page or accomplishing essential tasks like data submission via forms—is intact even when JavaScript is off. In this way, you ensure the baseline experience, and enhance it when JavaScript is available.

For Further Reading

- There is no better introduction to the progressive enhancement approach than the book *Adaptive Web Design: Crafting Rich Experiences with Progressive Enhancement, 2nd Edition*, by Aaron Gustafson (New Riders).
- *The Uncertain Web: Web Development in a Changing Landscape* by Rob Larson (O'Reilly).
- Once you have more chops, the book *Designing with Progressive Enhancement* by Todd Parker, Patty Toland, Scott Jehl, and Maggie Costello Wachs (New Riders) is an excellent deep-dive into techniques and best practices. Read more about it at *filamentgroup.com/dwpe/*.

RESPONSIVE WEB DESIGN

Ethan Marcotte personal site
ethanmarcotte.com



NASA
nasa.gov

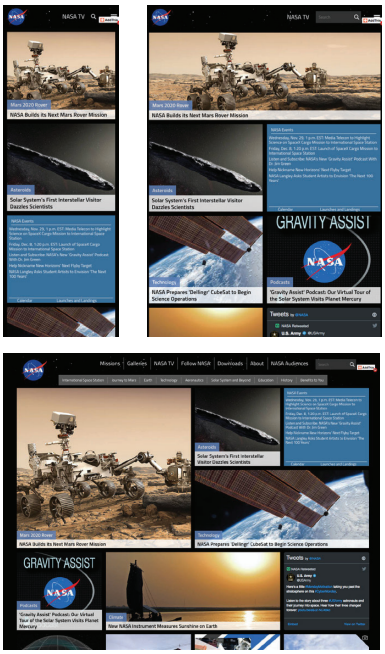


FIGURE 3-3. A responsive site's layout changes based on the size of the browser window.

By default, most browsers on small devices such as smartphones and tablets shrink a web page down to fit the screen and provide mechanisms for zooming and moving around the page. Although it technically works, it is not a great experience. The text is too small to read, the links are too small to tap, and all that zooming and panning around is distracting.

Responsive Web Design (RWD) is a strategy for providing appropriate layouts to devices based on the size of the viewport (browser window). The key to Responsive Web Design is serving a single HTML document (with one URL) to all devices, but applying different style sheets based on the screen size in order to provide the most optimized layout for that device. For example, when the page is viewed on a smartphone, it appears in one column with large links for easy tapping. But when that same page is viewed on a large desktop browser, the content rearranges into multiple columns with traditional navigation elements. It's like *magic*! (Except that it's actually just CSS.)

The web design community has been abuzz about responsive design since Ethan Marcotte first wrote about it and coined the phrase in his article "Responsive Web Design" on A List Apart in 2010 (*alistapart.com/articles/responsive-web-design/*). It's become one of the primary tools we use to cope with unknown viewport size.

FIGURE 3-3 shows some examples of responsive sites at the typical dimensions for a desktop monitor, tablet, and smartphone. You can see many more inspirational examples at the Media Queries gallery site (*mediaqueri.es*). Try opening one of the responsive sites in your browser and then resizing the window narrow and wide. Watch as the layout changes based on window size. *Très* cool.

Responsive Web Design helps with matters of layout, but it is not a solution to all mobile web design challenges. The fact is that providing the best experiences for your users and their chosen device may require optimizations that go beyond adjusting the look and feel. You can better address some problems by using the server to detect the device and its capabilities and then making decisions on what to send back.

For some sites and services, it may be preferable to build a separate mobile site (see the **"M-dot Sites"** sidebar) with a customized interface and feature set that takes advantage of phone capabilities like geolocation. That said, although responsive design won't fix everything, it is an important part of the solution for delivering satisfactory experiences on a wide variety of browsers.

For Further Reading

I'll cover Responsive Web Design in more detail in **Chapter 17, Responsive Web Design**, once you have more code experience under your belt. There you will find plenty of resources to continue your responsive design education.

M-dot Sites

Some companies and services choose to build an entirely separate site, with a unique URL, just for mobile devices. M-dot sites (named because their URLs typically begin with "m." or "mobile.") offer a reduced set of options and may also include mobile-specific features such as geolocation. A lot of the "extra" stuff (like promotions) from the desktop site is simply stripped away. (It makes you wonder what value it adds on the desktop.) A dedicated mobile site may be the best solution if you know that your mobile users have very different usage patterns than folks seated at a desk.

FIGURE 3-4 compares CVS's primary and m-dot sites as they appeared in early 2018. You can see that phone users are offered a more streamlined set of options. Other notable sites with dedicated mobile versions are Twitter and Facebook.

The point here is that Responsive Web Design is not a universal solution. For sites that feature mainly text content, a little layout adjustment may be all that is needed to deliver a good reading experience on all devices. For complex sites and web applications, a very different experience may be preferred.

The downside of a dedicated mobile site is that it is more than twice the work. It requires additional content planning, design templates, production time, and ongoing maintenance. But if it means giving your visitors the functionality they need, it is worth the investment.

It is possible that you have a business for which mobile use is so distinct from desktop use that a separate mobile site makes sense, but in general, m-dot sites are fading away in favor of RWD. Google is helping to speed this process along by encouraging all m-dot sites to migrate to RWD before the launch of their "mobile-first index" in 2018 (*webmasters.googleblog.com/2016/11/mobile-first-indexing.html*). If search result rankings are a concern, you may get more mileage from going responsive.

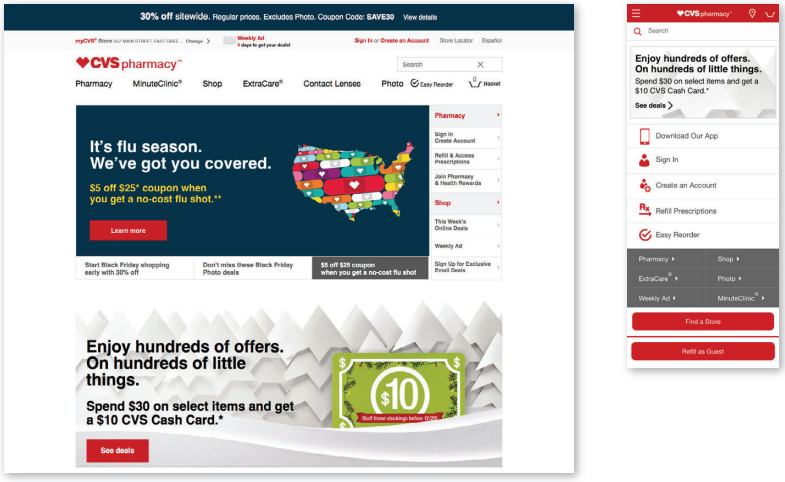


FIGURE 3-4. A comparison of the desktop site and the dedicated mobile site for the same business.

Responsive Web Design is a strategy for dealing with unknown screen size.

ONE WEB FOR ALL (ACCESSIBILITY)

We've been talking about the daunting number of browsers in use today, but so far, we've only addressed visual browsers controlled with mouse pointers or fingertips. It is critical, however, to keep in mind that people access the web in many different ways—with a keyboard, mouse, voice commands, screen readers, Braille output, magnifiers, joysticks, foot pedals, and so on. Web designers must build pages in a manner that creates as few barriers as possible to getting to information, regardless of the user's ability and the device used to access the web. In other words, you must design for accessibility.

Although intended for users with disabilities such as poor vision or limited mobility, the techniques and strategies developed for accessibility also benefit other users with less-than-optimum browsing experiences. Accessible sites are also more effectively indexed by search engines such as Google. Making your site accessible is well worth the extra effort.

There are four broad categories of disabilities that affect how people interact with their computers and the information on them:

Vision impairment

People with low or no vision may use an assistive device such as a screen reader, Braille display, or a screen magnifier to get content from the screen. They may also simply use the browser's text zoom function to make the text large enough to read.

Mobility impairment

Users with limited or no use of their hands may use special devices such as modified mice and keyboards, foot pedals, voice commands, or joysticks to navigate the web and enter information.

Auditory impairment

Users with limited or no hearing will miss out on audio aspects of multimedia, so it is necessary to provide alternatives, such as transcripts for audio tracks or captions for video.

Cognitive impairment

Users with memory, reading comprehension, problem solving, and attention limitations benefit when sites are designed simply and clearly. These qualities are helpful to anyone using your site.

The W3C started the Web Accessibility Initiative (WAI) to address the need to make the web usable for everyone. The WAI site (*www.w3.org/WAI*) is an excellent starting point for learning more about web accessibility. One of the documents produced by the WAI to help developers create accessible sites is the Web Content Accessibility Guidelines (WCAG and WCAG 2.0). You can read them all at *www.w3.org/WAI/intro/wcag.php*. The US government based its Section 508 accessibility guidelines on the Priority 1 points of the WCAG (see the sidebar **"Government Accessibility Requirements: Section 508"**). All

sites benefit from these guidelines, but if you are designing a government site, adherence is a requirement.

Another W3C effort is the WAI-ARIA (Accessible Rich Internet Applications) spec, which addresses the accessibility of web applications that include dynamically generated content, scripting, and advanced interface elements that are particularly confounding to assistive devices. The ARIA Recommendation defines a number of roles for content and widgets that authors can explicitly apply using the `role` attribute. Roles include `menubar`, `progressbar`, `slider`, `timer`, and `tooltip`, to name just a few. For the complete list of roles, go to *www.w3.org/TR/wai-aria/#role_definitions*.

For Further Reading

The following resources are good starting points for further exploration of web accessibility:

- The Web Accessibility Initiative (WAI), *www.w3.org/WAI*
- WebAIM: Web Accessibility in Mind, *www.webaim.org*
- *Accessibility Handbook: Making 508 Compliant Websites* by Katie Cunningham (O'Reilly)
- *Universal Design for Web Applications: Web Applications That Reach Everyone* by Wendy Chisholm and Matt May (O'Reilly)

US Government Accessibility Requirements: Section 508

If you create a site receiving federal funding from the US government, you are required by law to comply with the Section 508 guidelines, which ensure that electronic information and technology are available to people with disabilities. State and other publicly funded sites may also be required to comply.

The following guidelines, excerpted from the Section 508 Standards at *www.section508.gov*, provide a good checklist for basic accessibility for all websites:

1. A text equivalent for non-text elements shall be provided (e.g., via the "alt" attribute or in element content).
2. Equivalent alternatives for any multimedia presentation shall be synchronized with the presentation.
3. Web pages shall be designed so that all information conveyed with color is also available without color—for example, from context or markup.
4. Documents shall be organized so they are readable without requiring an associated style sheet.
5. Row and column headers shall be identified for data tables.
6. Markup shall be used to associate data and header cells for tables with two or more levels of row or column headers.
7. Pages shall be designed to avoid causing the screen to flicker with a frequency greater than 2 Hz and lower than 55 Hz.
8. When pages utilize scripting languages to display content, or to create interface elements, the information provided by the script shall be identified with functional text that can be read by assistive technology.
9. When a web page requires that an applet, plug-in, or other application be present on the client system to interpret page content, the page must provide a link to a plug-in or applet that complies with §1194.21(a) through (l).
10. When electronic forms are designed to be completed online, the form shall allow people using assistive technology to access the information, field elements, and functionality required for completion and submission of the form, including all directions and cues.
11. A method shall be provided that permits users to skip repetitive navigation links.
12. When a timed response is required, the user shall be alerted and given sufficient time to indicate more time is required.

THE NEED FOR SPEED (SITE PERFORMANCE)

Although the number of users accessing the internet on slow dial-up connections is shrinking (3–5% in the US as of this writing), the percentage of folks using mobile phones to access the web is increasing dramatically; and for some sectors, such as social media and search, mobile has already exceeded desktop usage. If you have a smartphone, then you know how frustrating it is to wait for a web page to fully display over a cellular data connection.

Site performance is critical regardless of how your users access your site. A study by Google in 2009[1] showed that the addition of just 100 to 400 milliseconds to their search results page resulted in reduced searches (–0.2 to –0.6%). Amazon.com showed that reducing page load times by just 100ms resulted in a 1% increase in revenue.[2] Other studies show that users expect a site to load in under 2 seconds, and nearly a third of your audience will leave your site for another if it doesn't. Furthermore, those people aren't likely to come back. Google has added site speed to its search algorithm, so if your site is a slowpoke, it's not likely to show up in that coveted first screen of results. The takeaway here is that site performance (down to the millisecond!) matters a lot.

There are many things you can do to improve the performance of your site, and they fall under two broad categories: limiting file sizes and reducing the number of requests to the server. The following list only scratches the surface for site optimization, but it gives you a general idea of what can be done:

- Optimize images so they are the smallest file size possible without sacrificing quality. You'll learn image optimization techniques in **Chapter 24, Image Asset Production**.
- Streamline HTML markup, avoiding unnecessary levels of nested elements.
- Minimize HTML and CSS documents by removing extra character spaces and line returns.
- Keep JavaScript to a minimum.
- Add scripts in such a way that they load in parallel with other page assets and don't block rendering.
- Don't load unnecessary assets (such as images, scripts, or JavaScript libraries).
- Reduce the number of times the browser makes requests of the server (known as HTTP requests).

1 "Speed Matters," *googleresearch.blogspot.com/2009/06/speed-matters.html*.

2 Statistic from "Make Data Matter," PowerPoint presentation by Greg Linden of Stanford University (2006).

Every trip to the server in the form of an HTTP request takes a few milliseconds, and those milliseconds can add up. All those little Twitter widgets, Facebook Like buttons, and advertisements can make dozens of server requests each. You may be surprised to see how many server requests even a simple site makes.

If you'd like to see for yourself, you can use the Network tool available with the Developer tools in Chrome, Safari, or Firefox. The Network tool displays each request to the server and how many milliseconds it took. Here's how you use it in Chrome (but all the browsers work similarly):

1. Launch the Chrome browser and go to any web page.
2. Go to the **View** menu and select **Developer → Developer Tools**. A panel will open at the bottom of the browser.
3. Select the Network tab in the tools view and load a web page. The chart (commonly referred to as a waterfall chart) shows you all the requests made and assets downloaded. The columns on the right show the amount of time each request took in milliseconds. At the bottom of the chart, you can see a summary of the number of requests made and the total amount of data transferred.

FIGURE 3-5 shows a portion of the performance waterfall chart for *oreilly.com*. You can poke around any site on the web this way. It can be very educational.

I won't address site performance in deep technical detail in this book, but I do want you to remember the importance of keeping file sizes as small as possible and eliminating unnecessary server requests in your web design work.

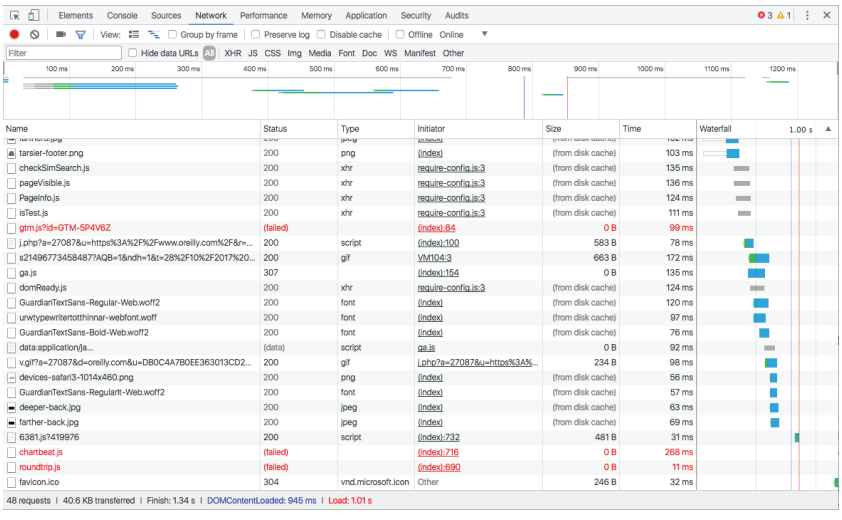


FIGURE 3-5. Waterfall charts such as this one created by the Chrome Network developer tool show the individual server requests made by a web page and the amount of time each request takes.

More Site Performance Tools

Try some of these tools for testing site performance:

- WebPageTest (*webpagetest.org*) is a tool that was originally developed for AOL, but is now available for all to use for free under an open source license. Just type in a URL, and WebPagetest returns a waterfall diagram, screenshot, and other statistics.
- Google's PageSpeed Insights (*developers.google.com/speed/pagespeed/insights/*) is another service that analyzes the performance of any site you point it to. It also generates suggestions for making your page load faster.
- Yahoo!'s freely available YSlow tool (*yslow.org*) analyzes a site according to 23 rules of web performance, and then gives the site a grade and suggestions for improvement.

For Further Reading

There are other techniques that are too technical for this book (and frankly, for me), and I figure if you are reading this book, you are probably not quite ready to become a site performance wizard. But when you are ready to take it on, here are some resources that should help:

- Lara Hogan has assembled a list of performance-related studies, tools, and resources at *larahogan.me/design*. You can also read her book, *Designing for Performance* (O'Reilly), there for free.
- *High Performance Mobile Web: Best Practices for Optimizing Mobile Web Apps* by Maximiliano Firtman (O'Reilly) covers optimization methods and tools to check your progress.
- Google's site *Make the Web Faster* (*code.google.com/speed/*) is an excellent first stop for learning about site optimization. It compiles a number of excellent tutorials and articles as well as tools for measuring site speed.

TEST YOURSELF

Here are a few questions that check your knowledge of the Big Concepts. If you are stumped, you can find the answers in **Appendix A**.

1. List at least two unknown factors you need to consider when designing and developing a site.

2. Match the technology or practice on the left with the problem it best addresses:

1. ____ Progressive enhancement	a. Assistive reading and input devices
2. ____ Server-side detection	b. Slow connection speeds
3. ____ Responsive design	c. All levels of browser capabilities
4. ____ WAI-ARIA	d. Determining which device is being used
5. ____ Site performance optimization	e. A variety of screen sizes

3. Web accessibility strategies take into account four broad categories of disabilities. Name at least three, and provide a measure you might take to ensure content is accessible for each.

4. When would you use a waterfall chart?

II

HTML FOR STRUCTURE

CHAPTER

4

CREATING A SIMPLE PAGE

(HTML OVERVIEW)

IN THIS CHAPTER

Part I provided a general overview of the web design environment. Now that we've covered the big concepts, it's time to roll up our sleeves and start creating a real web page. It will be an extremely simple page, but even the most complicated pages are based on the principles described here.

In this chapter, we'll create a web page step-by-step so you can get a feel for what it's like to mark up a document with HTML tags. The exercises allow you to work along.

This is what I want you to get out of this chapter:

- Get a feel for how markup works, including an understanding of elements and attributes.
- See how browsers interpret HTML documents.
- Learn how HTML documents are structured.
- Get a first glimpse of a style sheet in action.

Don't worry about learning the specific text elements or style sheet rules at this point; we'll get to those in the following chapters. For now, just pay attention to the process, the overall structure of the document, and the new terminology.

A WEB PAGE, STEP-BY-STEP

You got a look at an HTML document in **Chapter 2, How the Web Works**, but now you'll get to create one yourself and play around with it in the browser. The demonstration in this chapter has five steps that cover the basics of page production:

Step 1: Start with content. As a starting point, we'll write up raw text content and see what browsers do with it.

Step 2: Give the document structure. You'll learn about HTML element syntax and the elements that set up areas for content and metadata.

Step 3: Identify text elements. You'll describe the content using the appropriate text elements and learn about the proper way to use HTML.

Step 4: Add an image. By adding an image to the page, you'll learn about attributes and empty elements.

Step 5: Change how the text looks with a style sheet. This exercise gives you a taste of formatting content with Cascading Style Sheets.

By the time we're finished, you'll have written the document for the page shown in FIGURE 4-1. It's not very fancy, but you have to start somewhere.

FIGURE 4-1. In this chapter, we'll write the HTML document for this page in five steps.

We'll be checking our work in a browser frequently throughout this demonstration—probably more than you would in real life. But because this is an introduction to HTML, it's helpful to see the cause and effect of each small change to the source file along the way.

LAUNCH A TEXT EDITOR

In this chapter and throughout the book, we'll be writing out HTML documents by hand, so the first thing we need to do is launch a text editor. The text editor that is provided with your operating system, such as Notepad (Windows) or TextEdit (Macintosh), will do for these purposes. Other text editors are fine as long as you can save plain-text files with the *.html* extension. If you have a visual web-authoring tool such as Dreamweaver, set it aside for now. I want you to get a feel for marking up a document manually (see the sidebar **"HTML the Hard Way"**).

HTML the Hard Way

I stand by my method of teaching HTML the old-fashioned way—by hand. There's no better way to truly understand how markup works than typing it out, one tag at a time, and then opening your page in a browser. It doesn't take long to develop a feel for marking up documents properly.

Although you may choose to use a visual or drag-and-drop web-authoring tool down the line, understanding HTML will make using your tools easier and more efficient. In addition, you will be glad that you can look at a source file and understand what you're seeing. It is also crucial for troubleshooting broken pages or fine-tuning the default formatting that web tools produce.

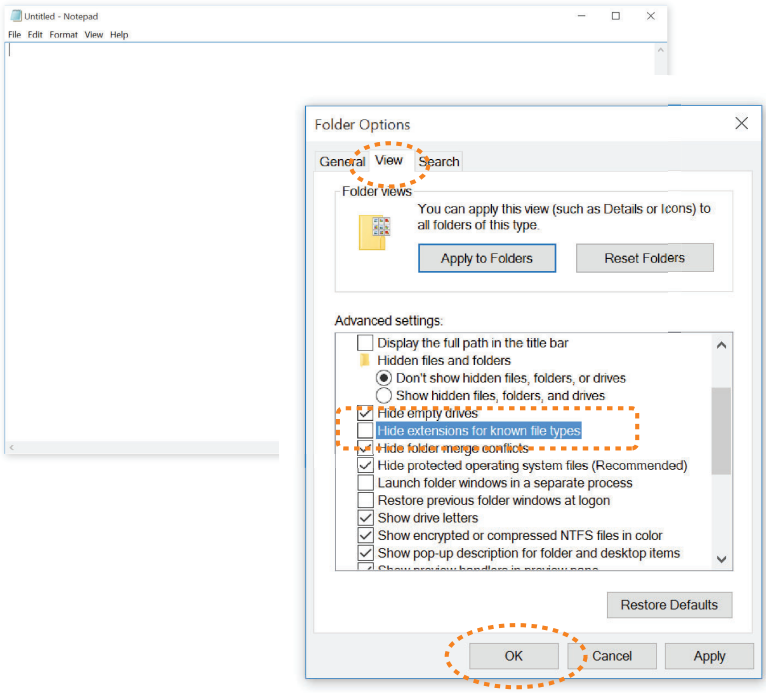
And for what it's worth, professional web developers tend to mark up content manually for better control over the code and the ability to make deliberate decisions about what elements to use.

This section shows how to open new documents in Notepad and TextEdit. Even if you've used these programs before, skim through for some special settings that will make the exercises go more smoothly. We'll start with Notepad; Mac users can jump ahead.

Creating a New Document in Notepad (Windows)

These are the steps to creating a new document in Notepad on Windows 10 (FIGURE 4-2):

1. Search for "Notepad" to access it quickly. Click on Notepad to open a new document window, and you're ready to start typing. ❶
2. Next, make the extensions visible. This step is not required to make HTML documents, but it will help make the file types clearer at a glance. Open the File Explorer, select the View tab, and then select the Options button on the right. In the Folder Options panel, select the View tab again. ❷
3. Find "Hide extensions for known file types" and uncheck that option. ❸
4. Click OK to save the preference ❹, and the file extensions will now be visible.



❶ Click on Notepad to open a new document.

❷ Open the File Explorer, select the View tab, and then select the Options button on the right (not shown). Select the View tab.

❸ Uncheck "Hide extensions for known file types."

❹ Click OK to save the preference, and the file extensions will now be visible.

FIGURE 4-2. Creating a new document in Notepad.

Creating a New Document in TextEdit (macOS)

By default, TextEdit creates rich-text documents—that is, documents that have hidden style-formatting instructions for making text bold, setting font size, and so on. You can tell that TextEdit is in rich-text mode when it has a formatting toolbar at the top of the window (plain-text mode does not). HTML documents need to be plain-text documents, so we'll need to change the format, as shown in this example (FIGURE 4-3):

1. Use the Finder to look in the Applications folder for TextEdit. When you've found it, double-click the name or icon to launch the application.
2. In the initial TextEdit dialog box, click the New Document button in the bottom-left corner. If you see the text formatting menu and tab ruler at the top of the Untitled document, you are in rich-text mode ❶. If you don't, you are in plain-text mode ❷. Either way, there are some preferences you need to set.
3. Close that document, and open the Preferences dialog box from the TextEdit menu.
4. Change these preferences:

   On the New Document tab, select Plain text ❸. Under Options, deselect all of the automatic formatting options ❹.

   On the Open and Save tab, select Display HTML files as HTML Code ❺ and deselect "Add '.txt' extensions to plain text files" ❻. The rest of the defaults should be fine.
5. When you are done, click the red button in the top-left corner.
6. Now create a new document by selecting File → New. The formatting menu will no longer be there, and you can save your text as an HTML document. You can always convert a document back to rich text by selecting Format → Make Rich Text when you are not using TextEdit for HTML.

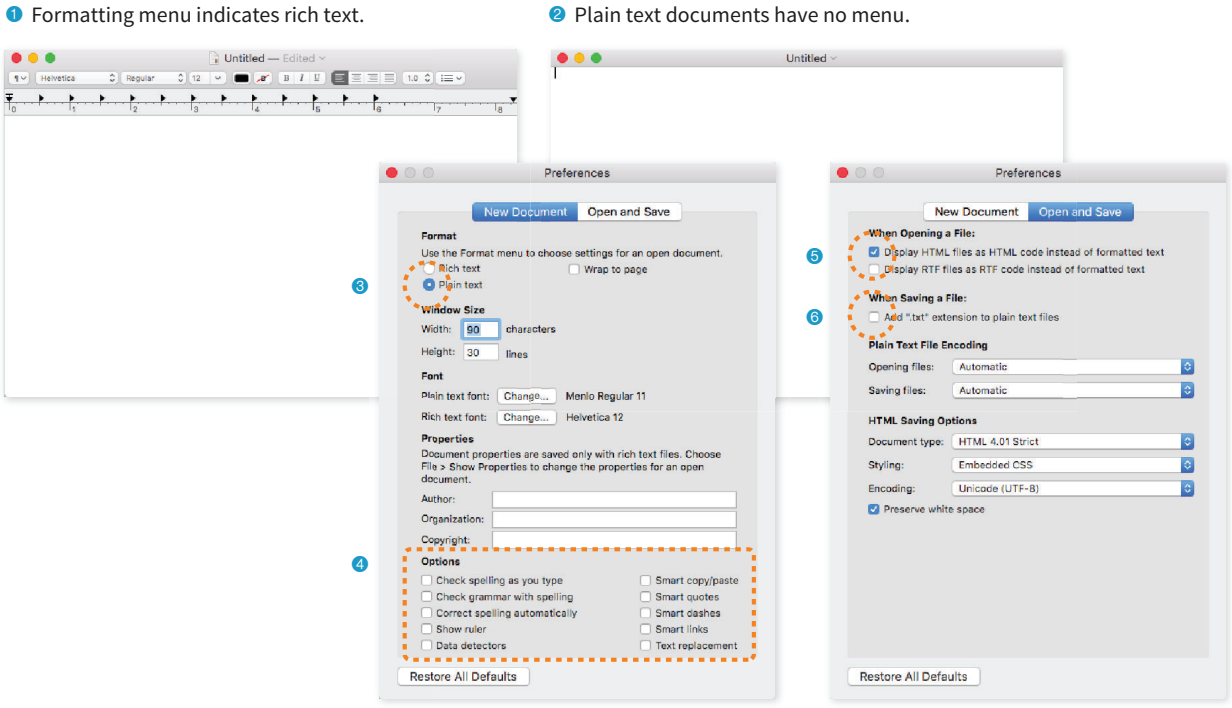


FIGURE 4-3. Launching TextEdit and choosing "Plain text" settings in the Preferences.

STEP 1: START WITH CONTENT

Now that we have our new document, it's time to get typing. A web page is all about content, so that's where we begin our demonstration. EXERCISE 4-1 walks you through entering the raw text content and saving the document in a new folder.

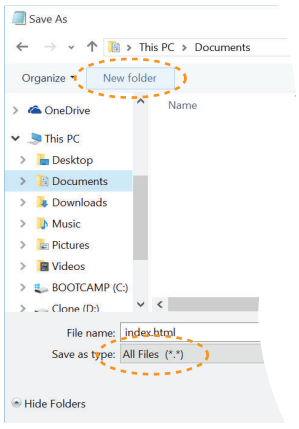
EXERCISE 4-1. Entering content

1. Type the home page content below into the new document in your text editor. Copy it exactly as you see it here, keeping the line breaks the same for the sake of playing along. The raw text for this exercise is also available online at *learningwebdesign.com/5e/materials/*.

```
Black Goose Bistro

The Restaurant
The Black Goose Bistro offers casual lunch and dinner fare in a relaxed
atmosphere. The menu changes regularly to highlight the freshest local
ingredients.

Catering
You have fun. We'll handle the cooking. Black Goose Catering can handle
events from snacks for a meetup to elegant corporate fundraisers.

Location and Hours
Seekonk, Massachusetts;
Monday through Thursday 11am to 9pm; Friday and Saturday, 11am to
midnight
```

2. Select "Save" or "Save as" from the File menu to get the Save As dialog box (FIGURE 4-4). The first thing you need to do is create a new folder (click the New Folder button on both Windows and Mac) that will contain all of the files for the site. The technical name for the folder that contains everything is the local root directory.

Windows 10



MacOS 10

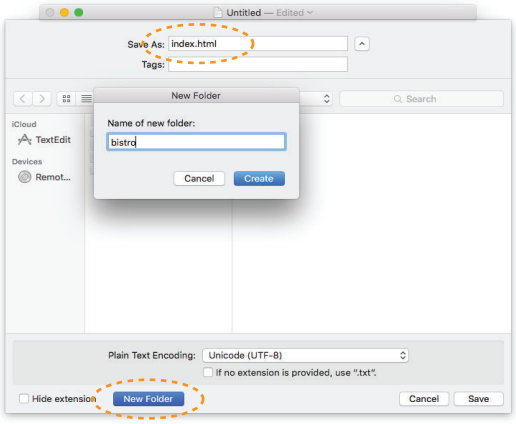


FIGURE 4-4. Saving *index.html* in a new folder called *bistro*.

Name the new folder *bistro*, and save the text file as *index.html* in it. The filename needs to end in *.html* to be recognized by the browser as a web document. See the sidebar **"Naming Conventions"** for more tips on naming files.

3. Just for kicks, let's take a look at *index.html* in a browser.

   Windows users: Double-click the filename in the File Explorer to launch your default browser, or right-click the file for the option to open it in the browser of your choice.

   Mac users: Launch your favorite browser (I'm using Google Chrome) and choose Open or Open File from the File menu. Navigate to *index.html*, and then select the document to open it in the browser.

4. You should see something like the page shown in FIGURE 4-5. We'll talk about the results in the following section.

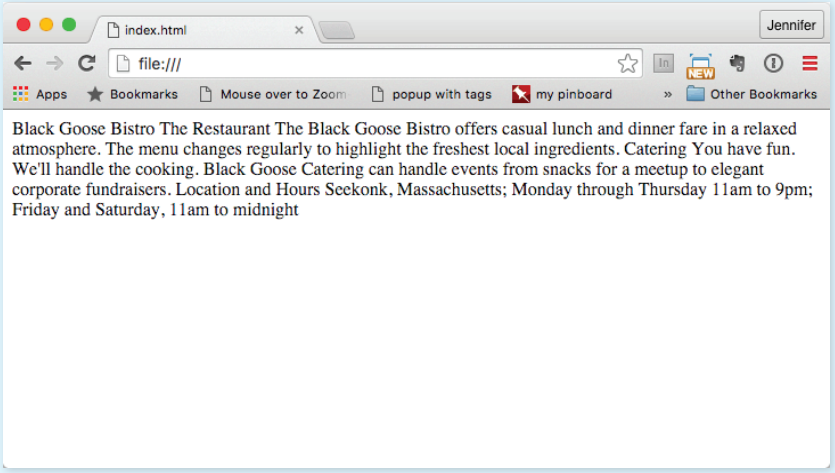


FIGURE 4-5. A first look at the content in a browser.

Naming Conventions

It is important that you follow these rules and conventions when naming your files:

Use proper suffixes for your files. HTML files must end with *.html* or *.htm*. Web graphics must be labeled according to their file format: *.gif*, *.png*, *.jpg* (*.jpeg* is also acceptable, although less common), or *.svg*.

Never use character spaces within filenames. It is common to use an underline character or hyphen to visually separate words within filenames, such as *robbins_bio.html* or *robbins-bio.html*.

Avoid special characters such as **?**, **%**, **#**, **/**, **:**, **;**, **•**, etc. Limit filenames to letters, numbers, underscores, hyphens, and periods. It is also best to avoid international characters, such as the Swedish å.

Filenames may be case-sensitive, depending on your server configuration. Consistently using all lowercase letters in filenames, although not required, is one way to make your filenames easier to manage.

Keep filenames short. Long names are more likely to be misspelled, and short names shave a few extra bytes off the file size. If you really must give the file a long, multiword name, you can separate words with hyphens, such as *a-long-document-title.html*, to improve readability.

Self-imposed conventions. It is helpful to develop a consistent naming scheme for huge sites—for instance, always using lowercase with hyphens between words. This takes some of the guesswork out of remembering what you named a file when you go to link to it later.

Learning from Step 1

Our page isn't looking so good (FIGURE 4-5). The text is all run together into one block—that's not how it looked when we typed it into the original document. There are a couple of lessons to be learned here. The first thing that is apparent is that the browser ignores line breaks in the source document. The sidebar **"What Browsers Ignore"** lists other types of information in the source document that are not displayed in the browser window.

Second, we see that simply typing in some content and naming the document *.html* is not enough. While the browser can display the text from the file, we haven't indicated the *structure* of the content. That's where HTML comes in. We'll use markup to add structure: first to the HTML document itself (coming up in Step 2), then to the page's content (Step 3). Once the browser knows the structure of the content, it can display the page in a more meaningful way.

STEP 2: GIVE THE HTML DOCUMENT STRUCTURE

We have our content saved in an HTML document—now we're ready to start marking it up.

The Anatomy of an HTML Element

Back in **Chapter 2** you saw examples of elements with an opening tag (** ** for a paragraph, for example) and a closing tag (** **). Before we start adding tags to our document, let's look at the anatomy of an HTML element (its syntax) and firm up some important terminology. A generic container element is labeled in FIGURE 4-6.

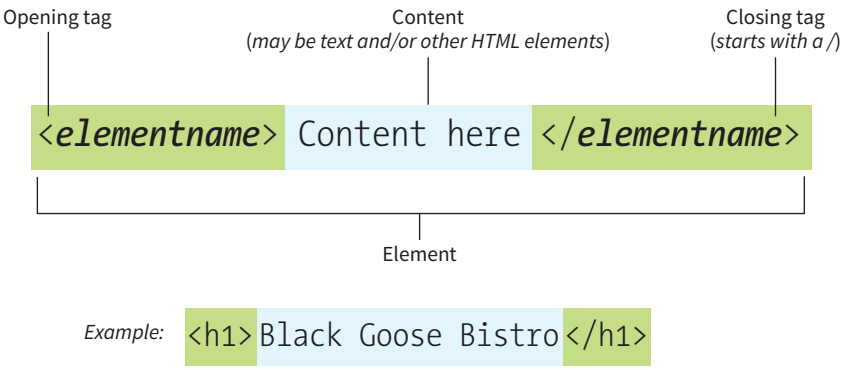


FIGURE 4-6. The parts of an HTML container element.

What Browsers Ignore

The following information in the source document will be ignored when it is viewed in a browser:

Multiple-character (white) spaces

When a browser encounters more than one consecutive blank character space, it displays a single space. So if the document contains

```
long,      long      ago
```

the browser displays:

```
long, long ago
```

Line breaks (carriage returns).

Browsers convert carriage returns to white spaces, so following the earlier "ignore multiple white spaces" rule, line breaks have no effect on formatting the page.

Tabs

Tabs are also converted to character spaces, so guess what? They're useless for indenting text on the web page (although they may make your code more readable).

Unrecognized markup

Browsers are instructed to ignore any tag they don't understand or that was specified incorrectly. Depending on the element and the browser, this can have varied results. The browser may display nothing at all, or it may display the contents of the tag as though it were normal text.

Text in comments

Browsers do not display text between the special **<!--** and **-->** tags used to denote a comment. See the upcoming **"Adding Hidden Comments"** sidebar.

Elements are identified by tags in the text source. A tag consists of the element name (usually an abbreviation of a longer descriptive name) within angle brackets (< >). The browser knows that any text within brackets is hidden and not displayed in the browser window.

The element name appears in the opening tag (also called a start tag) and again in the closing (or end) tag preceded by a slash (/). The closing tag works something like an "off" switch for the element. Be careful not to use the similar backslash character in end tags (see the tip **"Slash Versus Backslash"**).

The tags added around content are referred to as the markup. It is important to note that an element consists of both the content *and* its markup (the start and end tags). Not all elements have content, however. Some are empty by definition, such as the **img** element used to add an image to the page. We'll talk about empty elements a little later in this chapter.

One last thing: capitalization. In HTML, the capitalization of element names is not important (it is not case-sensitive). So **<img>**, **<Img>**, and **<IMG>** are all the same as far as the browser is concerned. However, most developers prefer the consistency of writing element names in all lowercase (see **Note**), as I will be doing throughout this book.

Basic Document Structure

FIGURE 4-8 shows the recommended minimal skeleton of an HTML document. I say "recommended" because the only element that is *required* in HTML is the **title**. But I feel it is better, particularly for beginners, to explicitly organize documents into metadata (**head**) and content (**body**) areas. Let's take a look at what's going on in this minimal markup example.

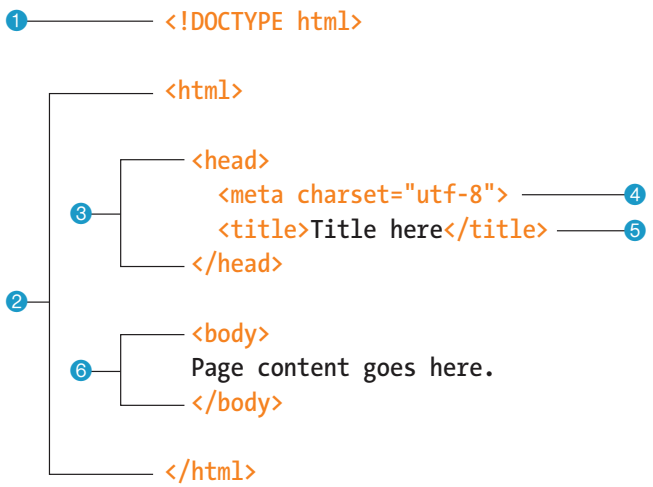


FIGURE 4-8. The minimal structure of an HTML document includes **head** and **body** contained within the **html** root element.

■ MARKUP TIP

Slash Versus Backslash

HTML tags and URLs use the slash character (/). The slash character is found under the question mark (?) on the English QWERTY keyboard (key placement on keyboards in other countries may vary).

It is easy to confuse the slash with the backslash character (\), which is found under the bar character (|); see FIGURE 4-7. The backslash key will not work in tags or URLs, so be careful not to use it.

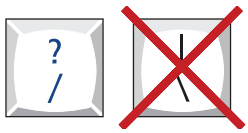


FIGURE 4-7. Slash versus backslash keys.

NOTE

There is a stricter version of HTML called XHTML that requires all element and attribute names to appear in lowercase. HTML5 has made XHTML all but obsolete except for certain use cases when it is combined with other XML languages, but the preference for all lowercase element names has persisted.

1. I don't want to confuse things, but the first line in the example isn't an element at all. It is a document type declaration (also called DOCTYPE declaration) that lets modern browsers know which HTML specification to use to interpret the document. This DOCTYPE identifies the document as written in HTML5.
2. The entire document is contained within an `html` element. The `html` element is called the root element because it contains all the elements in the document, and it may not be contained within any other element.
3. Within the `html` element, the document is divided into a head and a body. The `head` element contains elements that pertain to the document that are not rendered as part of the content, such as its title, style sheets, scripts, and metadata.
4. `meta` elements provide document metadata, information about the document. In this case, it specifies the character encoding (a standardized collection of letters, numbers, and symbols) used in the document as Unicode version UTF-8 (see the sidebar **"Introducing Unicode"**). I don't want to go into too much detail on this right now, but know that there are many good reasons for specifying the `charset` in every document, so I have included it as part of the minimal document markup. Other types of metadata provided by the `meta` element are the author, keywords, publishing status, and a description that can be used by search engines.
5. Also in the `head` is the mandatory `title` element. According to the HTML specification, every document must contain a descriptive title.
6. Finally, the `body` element contains everything that we want to show up in the browser window.

Are you ready to start marking up the Black Goose Bistro home page? Open the *index.html* document in your text editor and move on to EXERCISE 4-2.

Introducing Unicode

All the characters that make up languages are stored in computers as numbers. A standardized collection of characters with their reference numbers (code points) is called a coded character set, and the way in which those characters are converted to bytes for use by computers is the character encoding. In the early days of computing, computers used limited character sets such as ASCII that contained 128 characters (letters from Latin languages, numbers, and common symbols). The early web used the Latin-1 (ISO 8859-1) character encoding that included 256 Latin characters from most Western languages. But given the web was "worldwide," it was clearly not sufficient.

Enter Unicode. Unicode (also called the Universal Character Set) is a super-character set that contains over 136,000 characters (letters, numbers, symbols, ideograms, logograms, etc.) from all active modern languages. You can read all about it at *unicode.org*. Unicode has three standard encodings—UTF-8, UTF-16, and UTF-32—that differ in the number of bytes used to represent the characters (1, 2, or 3, respectively).

HTML5 uses the UTF-8 encoding by default, which allows wide-ranging languages to be mixed within a single document. It is always a good idea to declare the character encoding for a document with the `meta` element, as shown in the previous example. Your server also needs to be configured to identify HTML documents as UTF-8 in the HTTP header (information about the document that the server sends to the user agent). You can ask your server administrator to confirm the encoding of the HTML documents.

EXERCISE 4-2. Adding minimal structure

1. Open the new *index.html* document if it isn't open already and add the DOCTYPE declaration:

```
<!DOCTYPE html>
```

2. Put the entire document in an HTML root element by adding an **<html>** start tag after the DOCTYPE and an **</html>** end tag at the very end of the text.
3. Next, create the document head that contains the title for the page. Insert **<head>** and **</head>** tags before the content. Within the **head** element, add information about the character encoding **<meta charset="utf-8">**, and the title, "Black Goose Bistro", surrounded by opening and closing **<title>** tags.
4. Finally, define the body of the document by wrapping the text content in **<body>** and **</body>** tags. When you are done, the source document should look like this (the markup is shown in color to make it stand out):

```
<!DOCTYPE html>
<html>

<head>
  <meta charset="utf-8">
  <title>Black Goose Bistro</title>
</head>

<body>
Black Goose Bistro

The Restaurant
The Black Goose Bistro offers casual lunch and
dinner fare in a relaxed atmosphere. The menu
changes regularly to highlight the freshest local
ingredients.

Catering
You have fun. We'll handle the cooking. Black Goose
Catering can handle events from snacks for a meetup
to elegant corporate fundraisers.

Location and Hours
Seekonk, Massachusetts;
Monday through Thursday 11am to 9pm; Friday and
Saturday, 11am to midnight
</body>
</html>
```

5. Save the document in the *bistro* directory, so that it overwrites the old version. Open the file in the browser or hit Refresh or Reload if it is open already. FIGURE 4-9 shows how it should look now.

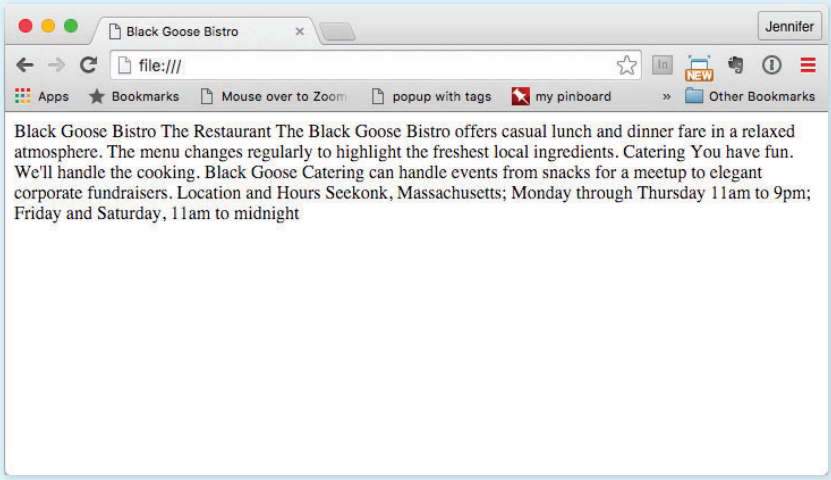


FIGURE 4-9. The page in a browser after the document structure elements have been defined.

Not much has changed in the bistro page after setting up the document, except that the browser now displays the title of the document in the top bar or tab (FIGURE 4-9). If someone were to bookmark this page, that title would be added to their Bookmarks or Favorites list as well (see the sidebar **"Don't Forget a Good Title"**). But the content still runs together because we haven't given the browser any indication of how it should be structured. We'll take care of that next.

STEP 3: IDENTIFY TEXT ELEMENTS

With a little markup experience under your belt, it should be a no-brainer to add the markup for headings and subheads (`h1` and `h2`), paragraphs (`p`), and emphasized text (`em`) to our content, as we'll do in EXERCISE 4-3. However, before we begin, I want to take a moment to talk about what we're doing and not doing when marking up content with HTML.

Mark It Up Semantically

The purpose of HTML is to add meaning and structure to the content. It is *not* intended to describe how the content should look (its presentation).

Your job when marking up content is to choose the HTML element that provides the most meaningful description of the content at hand. In the biz, we call this semantic markup. For example, the most important heading at the beginning of the document should be marked up as an `h1` because it is the most important heading on the page. Don't worry about what it looks like... you can easily change that with a style sheet. The important thing is that you choose elements based on what makes the most sense for the content.

In addition to adding meaning to content, the markup gives the document structure. The way elements follow each other or nest within one another creates relationships between them. You can think of this structure as an outline (its technical name is the DOM, for Document Object Model). The underlying document hierarchy gives browsers cues on how to handle the content. It is also the foundation upon which we add presentation instructions with style sheets and behaviors with JavaScript.

Although HTML was intended to be used strictly for meaning and structure since its creation, that mission was somewhat thwarted in the early years of the web. With no style sheet system in place, HTML was extended to give authors ways to change the appearance of fonts, colors, and alignment using markup alone. Those presentational extras are still out there, so you may run across them if you view the source of older sites or a site made with old tools. In this book, however, I'll focus on using HTML the right way, in keeping with the contemporary standards-based, semantic approach to web design.

OK, enough lecturing. It's time to get to work on that content in EXERCISE 4-3.

Don't Forget a Good Title

A `title` element is not only required for every document, but it is also quite useful. The title is what is displayed in a user's Bookmarks or Favorites list and on tabs in desktop browsers. Descriptive titles are also a key tool for improving accessibility, as they are the first things a person hears when using a screen reader (an assistive device that reads the content of a page aloud for users with impaired sight). Search engines rely heavily on document titles as well.

For these reasons, it's important to provide thoughtful and descriptive titles for all your documents and avoid vague titles, such as "Welcome" or "My Page." You may also want to keep the length of your titles in check so they are able to display in the browser's title area. Knowing that users typically have a number of tabs open or a long list of Bookmarks, put your most uniquely identifying information in the first 20 or so characters.

The purpose of HTML is to add meaning and structure to the content.

EXERCISE 4-3. Defining text elements

1. Open the document *index.html* in your text editor, if it isn't open already.
2. The first line of text, "Black Goose Bistro," is the main heading for the page, so we'll mark it up as a Heading Level 1 (**h1**) element. Put the opening tag, **<h1>**, at the beginning of the line and the closing tag, **</h1>**, after it, like this:

```
<h1>Black Goose Bistro</h1>
```

3. Our page also has three subheads. Mark them up as Heading Level 2 (**h2**) elements in a similar manner. I'll do the first one here; you do the same for "Catering" and "Location and Hours."

```
<h2>The Restaurant</h2>
```

4. Each **h2** element is followed by a brief paragraph of text, so let's mark those up as paragraph (**p**) elements in a similar manner. Here's the first one; you do the rest:

```
<p>The Black Goose Bistro offers casual lunch and dinner fare in
a relaxed atmosphere. The menu changes regularly to highlight the
freshest local ingredients.</p>
```

5. Finally, in the Catering section, I want to emphasize that visitors should just leave the cooking to us. To make text emphasized, mark it up in an emphasis element (**em**) element, as shown here:

```
<p>You have fun. <em>We'll handle the cooking.</em> Black Goose
Catering can handle events from snacks for a meetup to elegant
corporate fundraisers.</p>
```

6. Now that we've marked up the document, let's save it as we did before, and open (or reload) the page in the browser. You should see a page that looks much like the one in FIGURE 4-10. If it doesn't, check your markup to be sure that you aren't missing any angle brackets or a slash in a closing tag.

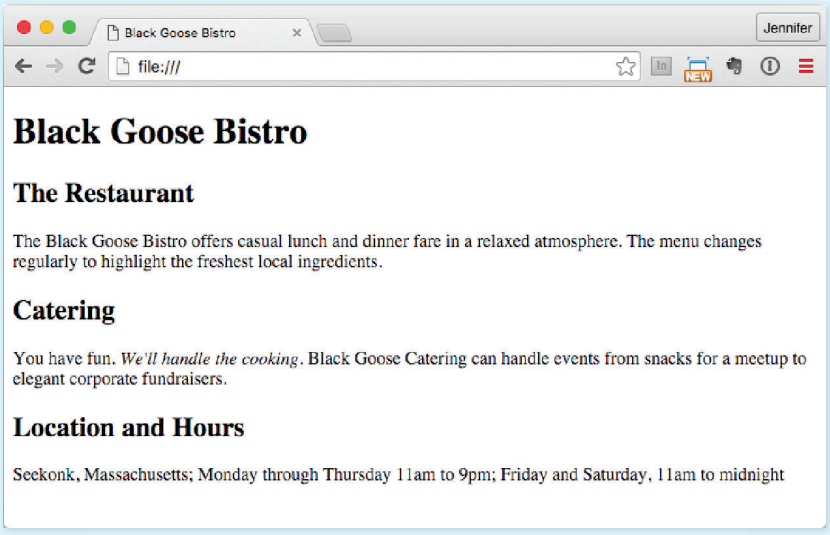


FIGURE 4-10. The home page after the content has been marked up with HTML elements.

Now we're getting somewhere. With the elements properly identified, the browser can now display the text in a more meaningful manner. There are a few significant things to note about what's happening in FIGURE 4-10.

Block and Inline Elements

Although it may seem like stating the obvious, it's worth pointing out that the heading and paragraph elements start on new lines and do not run together as they did before. That is because by default, headings and paragraphs display as block elements. Browsers treat block elements as though they are in little rectangular boxes, stacked up in the page. Each block element begins on a new line, and some space is also usually added above and below the entire element by default. In FIGURE 4-11, the edges of the block elements are outlined in red.

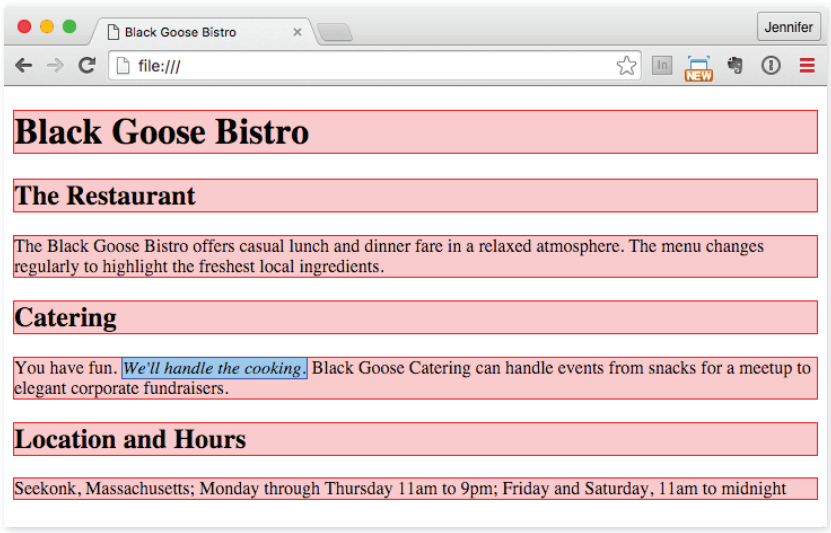


FIGURE 4-11. The outlines show the structure of the elements in the home page.

By contrast, look at the text we marked up as emphasized (`em`, outlined in blue in FIGURE 4-11). It does not start a new line, but rather stays in the flow of the paragraph. That is because the `em` element is an inline element (also called a text-level semantic element or phrasing element). Inline elements do not start new lines; they just go with the flow.

Default Styles

The other thing that you will notice about the marked-up page in FIGURES 4-10 and 4-11 is that the browser makes an attempt to give the page some

Adding Hidden Comments

You can leave notes in the source document for yourself and others by marking them up as comments. Anything you put between comment tags (`<!-- -->`) will not display in the browser and will not have any effect on the rest of the source:

```
<!-- This is a comment -->
<!-- This is a
  multiple-line comment
  that ends here. -->
```

Comments are useful for labeling and organizing long documents, particularly when they are shared by a team of developers. In this example, comments are used to point out the section of the source that contains the navigation:

```
<!-- start global nav -->
<ul>
  …
</ul>
<!-- end global nav -->
```

Bear in mind that although the browser will not display comments in the web page, readers can see them if they "view source," so be sure that the comments you leave are appropriate for everyone.

visual hierarchy by making the first-level heading the biggest and boldest thing on the page, with the second-level headings slightly smaller, and so on.

How does the browser determine what an `h1` should look like? It uses a style sheet! All browsers have their own built-in style sheets (called user agent style sheets in the spec) that describe the default rendering of elements. The default rendering is similar from browser to browser (for example, `h1`s are always big and bold), but there are some variations (the `blockquote` element for long quotes may or may not be indented).

If you think the `h1` is too big and clunky as the browser renders it, just change it with your own style sheet rule. Resist the urge to mark up the heading with another element just to get it to look better—for example, using an `h3` instead of an `h1` so it isn't as large. In the days before ubiquitous style sheet support, elements were abused in just that way. You should always choose elements based on how accurately they describe the content, and don't worry about the browser's default rendering.

We'll fix the presentation of the page with style sheets in a moment, but first, let's add an image to the page.

STEP 4: ADD AN IMAGE

What fun is a web page with no images? In EXERCISE 4-4, we'll add an image to the page with the `img` element. Images will be discussed in more detail in **Chapter 7, Adding Images**, but for now, they give us an opportunity to introduce two more basic markup concepts: empty elements and attributes.

Empty Elements

So far, nearly all of the elements we've used in the Black Goose Bistro home page have followed the syntax shown in FIGURE 4-6: a bit of text content surrounded by start and end tags.

A handful of elements, however, do not have content because they are used to provide a simple directive. These elements are said to be empty. The image element (`img`) is an example of an empty element. It tells the browser to get an image file from the server and insert it at that spot in the flow of the text. Other empty elements include the line break (`br`), thematic breaks (`hr`, a.k.a. "horizontal rules"), and elements that provide information about a document but don't affect its displayed content, such as the `meta` element that we used earlier.

FIGURE 4-12 shows the very simple syntax of an empty element (compare it to FIGURE 4-6).

```
<element-name>
```

Example: The **br** element inserts a line break.

```
<p>1005 Gravenstein Highway North<br>Sebastopol, CA 95472</p>
```

FIGURE 4-12. Empty element structure.

Attributes

Let's get back to adding an image with the empty **img** element. Obviously, an **<img>** tag is not very useful by itself—it doesn't indicate which image to use. That's where attributes come in. Attributes are instructions that clarify or modify an element. For the **img** element, the **src** (short for "source") attribute is required, and specifies the location (URL) of the image file.

The syntax for an attribute is as follows:

```
attributename="value"
```

Attributes go after the element name, separated by a space. In non-empty elements, attributes go in the opening tag only:

```
<element attributename="value">
<element attributename="value">Content</element>
```

You can also put more than one attribute in an element in any order. Just keep them separated with spaces:

```
<element attribute1="value" attribute2="value">
```

FIGURE 4-13 shows an **img** element with its required attributes labeled.

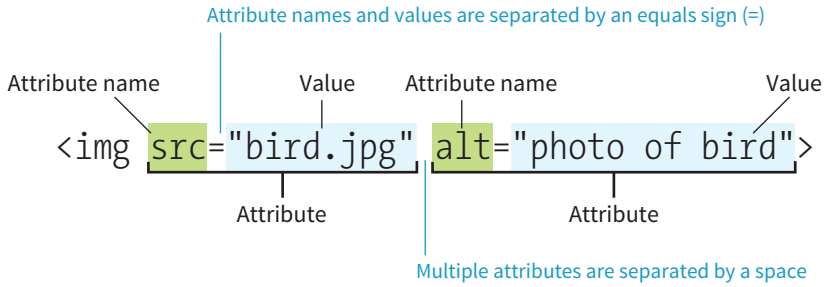


FIGURE 4-13. An **img** element with two attributes.

What Is That Extra Slash?

If you poke around in source documents for existing web pages, you may see empty elements with extra slashes at the end, like so: **<img />**, **
, **<meta />, and **<hr />**. That indicates the document was written according to the stricter rules of XHTML. In XHTML, all elements, including empty elements, must be closed (or terminated, to use the proper term). You terminate empty elements by adding a trailing slash before the closing bracket. The preceding character space is not required but was used for backward compatibility with browsers that did not have XHTML parsers, so **<img/>**, **
**, and so on are valid.

Attributes are instructions that clarify or modify an element.

Here's what you need to know about attributes:

- Attributes go after the element name in the opening tag only, never in the closing tag.
- There may be several attributes applied to an element, separated by spaces in the opening tag. Their order is not important.
- Most attributes take values, which follow an equals sign (=). In HTML, some attribute values are single descriptive words. For example, the **`checked`** attribute, which makes a form checkbox checked when the form loads, is equivalent to **`checked="checked"`**. You may hear this type of attribute called a Boolean attribute because it describes a feature that is either on or off.
- A value might be a number, a word, a string of text, a URL, or a measurement, depending on the purpose of the attribute. You'll see examples of all of these throughout this book.
- Wrapping attribute values in double quotation marks is a strong convention, but note that quotation marks are not required and may be omitted. In addition, either single or double quotation marks are acceptable as long as the opening and closing marks match. Note that quotation marks in HTML files need to be straight ("), not curly (").
- The attribute names and values available for each element are defined in the HTML specifications; in other words, you can't make up an attribute for an element.
- Some attributes are required, such as the **`src`** and **`alt`** attributes in the **`img`** element. The HTML specification also defines which attributes are required in order for the document to be valid.

Now you should be more than ready to try your hand at adding the **`img`** element with its attributes to the Black Goose Bistro page in EXERCISE 4-4. We'll throw a few line breaks in there as well.

EXERCISE 4-4. Adding an image

1. If you're working along, the first thing you'll need to do is get a copy of the image file on your hard drive so you can see it in place when you open the file locally. The image file is provided in the materials for this chapter (*learningwebdesign.com/5e/materials*). You can also get the image file by saving it right from the sample web page online at *learningwebdesign.com/5e/materials/ch04/bistro*. Right-click (or Control-click on a Mac) the goose image and select "Save to disk" (or similar) from the pop-up menu, as shown in FIGURE 4-14. Name the file *blackgoose.png*. Be sure to save it in the *bistro* folder with *index.html*.
2. Once you have the image, insert it at the beginning of the first-level heading by typing in the **`img`** element and its attributes as shown here:

```
<h1><img src="blackgoose.png" alt="logo">Black Goose Bistro</h1>
```

Windows: Right-click on the image to access the pop-up menu.

Mac: Control-click on the image to access the pop-up menu. The options may vary by browser.

FIGURE 4-14. Saving an image file from a page on the web.

The **src** attribute provides the name of the image file that should be inserted, and the **alt** attribute provides text that should be displayed if the image is not available. Both of these attributes are required in every **img** element.

3. I'd like the image to appear above the title, so add a line break (**br**) after the **img** element to start the headline text on a new line.

```
<h1><img src="blackgoose.png" alt="logo"><br>Black Goose Bistro</h1>
```

4. Let's break up the last paragraph into three lines for better clarity. Drop a **
** tag at the spots you'd like the line breaks to occur. Try to match the screenshot in FIGURE 4-15.
5. Now save *index.html* and open or refresh it in the browser window. The page should look like the one shown in FIGURE 4-15. If it doesn't, check to make sure that the image file, *blackgoose.png*, is in the same directory as *index.html*. If it is, then check to make sure that you aren't missing any characters, such as a closing quote or bracket, in the **img** element markup.

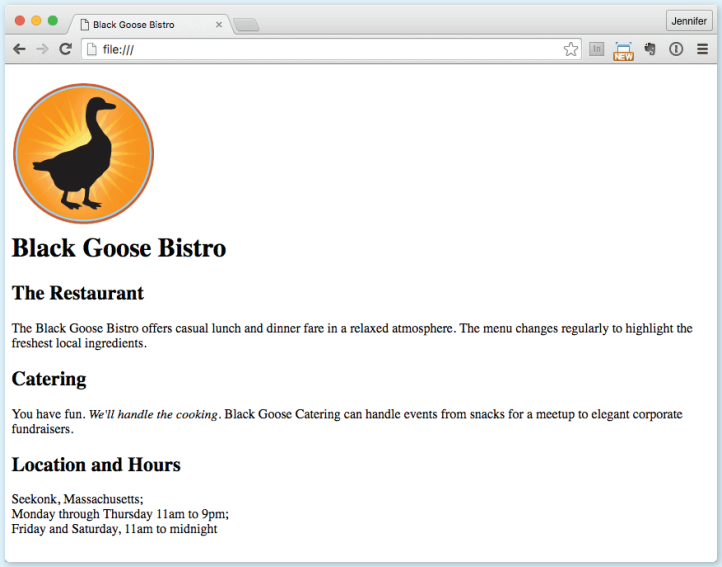


FIGURE 4-15. The Black Goose Bistro page with the logo image.

STEP 5: CHANGE THE LOOK WITH A STYLE SHEET

Depending on the content and purpose of your website, you may decide that the browser's default rendering of your document is perfectly adequate. However, I think I'd like to pretty up the Black Goose Bistro home page a bit to make a good first impression on potential patrons. "Prettying up" is just my way of saying that I'd like to change its presentation, which is the job of Cascading Style Sheets (CSS).

In EXERCISE 4-5, we'll change the appearance of the text elements and the page background by using some simple style sheet rules. Don't worry about understanding them all right now. We'll get into CSS in more detail in **Part III**. But I want to at least give you a taste of what it means to add a "layer" of presentation onto the structure we've created with our markup.

EXERCISE 4-5. Adding a style sheet

1. Open *index.html* if it isn't open already. We're going to use the **style** element to apply a very simple embedded style sheet to the page. This is just one of the ways to add a style sheet; the others are covered in **Chapter 11, Introducing Cascading Style Sheets**.
2. The **style** element is placed inside the document **head**. Start by adding the **style** element to the document as shown here:

```
<head>
  <meta charset="utf-8">
  <title>Black Goose Bistro</title>
  <style>

  </style>
</head>
```

3. Next, type the following style rules within the **style** element just as you see them here. Don't worry if you don't know exactly what's going on (although it's fairly intuitive). You'll learn all about style rules in **Part III**.

```
<style>
body {
  background-color: #faf2e4;
  margin: 0 10%;
  font-family: sans-serif;
  }
h1 {
  text-align: center;
  font-family: serif;
  font-weight: normal;
  text-transform: uppercase;
  border-bottom: 1px solid #57b1dc;
  margin-top: 30px;
}
h2 {
  color: #d1633c;
  font-size: 1em;
}
</style>
```

4. Now it's time to save the file and take a look at it in the browser. It should look like the page in FIGURE 4-16. If it doesn't, go over the style sheet to make sure you didn't miss a semicolon or a curly bracket. Look at the way the page looks with our styles compared to the browser's default styles (FIGURE 4-15).

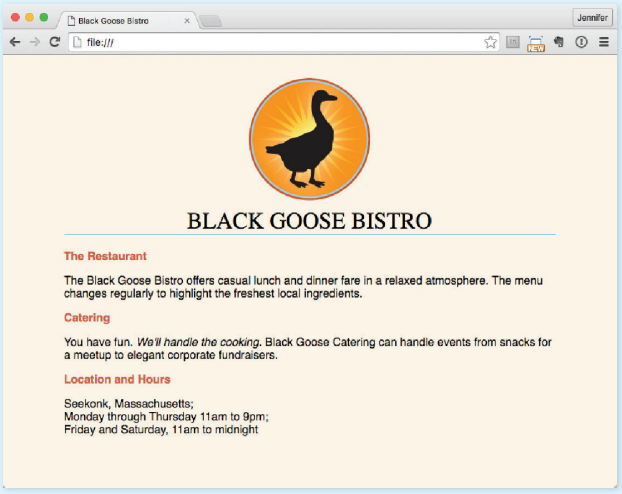


FIGURE 4-16. The Black Goose Bistro page after CSS style rules have been applied.