VISUAL QUICKSTART GUIDE

HTML and CSS

Eighth Edition

ELIZABETH CASTRO
BRUCE HYSLOP

LEARN THE QUICK AND EASY WAY!

VISUAL QUICKSTART GUIDE

HTML and CSS

Eighth Edition

ELIZABETH CASTRO • BRUCE HYSLOP

Peachpit Press

Visual QuickStart Guide
HTML and CSS, Eighth Edition
Elizabeth Castro and Bruce Hyslop

Peachpit Press
www.peachpit.com

To report errors, please send a note to errata@peachpit.com.

Peachpit Press is a division of Pearson Education.

Editor: Clifford Colby
Development editor: Robyn G. Thomas
Production editor: David Van Ness
Copyeditor: Scout Festa
Technical editor: Aubrey Taylor
Compositor: David Van Ness
Indexer: Valerie Haynes Perry
Cover design: RHDG / Riezebos Holzbaur Design Group, Peachpit Press
Interior design: Peachpit Press
Logo design: MINE™ www.minesf.com

ISBN-13: 978-0-321-92883-2
ISBN-10: 0-321-92883-0

9 8 7 6 5 4 3 2 1

Printed and bound in the United States of America

Dedication

To family.

To those I know who endured difficult challenges, demonstrating courage and perseverance all the way.

Acknowledgments

One of my favorite parts of working on this book has been the people I've been able to work with. All are dedicated, professional, good-natured, and good-humored folks who made it a real pleasure. The book wouldn't be the same without their contributions.

A grateful, sincere thank you goes out to:

Nancy Aldrich-Ruenzel and Nancy Davis, for their continued trust in me.

Cliff Colby, for his support, for bringing the team together, and for keeping things light.

Robyn Thomas, for making the engine go, improving copy, tracking all the details, being flexible, and providing encouragement.

Scout Festa, for her skill in simplifying language, for her watchful eye, and for helping to keep things consistent and polished.

Aubrey Taylor, for all the great suggestions and technical feedback. They were very helpful, and readers are better off for them.

David Van Ness, for leading the charge in making it all look great and for all his efforts in refining the layouts.

Valerie Haynes Perry, for compiling the all-important index, which will be the first destination for many readers in search of information.

The marketing, sales, and other folks at Peachpit, for working hard to make the book available to readers.

Natalia Ammon, for the wonderful design of the example webpage that adorns the pages of Chapters 11 and 12, and other spots. You can see more of her work at www.nataliaammon.com.

Zach Szukala, for recommending Natalia.

Scott Boms, Ian Devlin, Seth Lemoine, Erik Vorhes, and Brian Warren, for their contributions to the previous edition.

Victor Gavenda, for providing access to necessary software.

Dan Cederholm, Ethan Dunham, Paul Irish, Mark James, Randy Jensen, Julie Lamba, Fontfabric, Google, and Namecheap, for allowing me to use screen shots or design assets (as the case may be).

C.R. Freer, for working her camera magic.

My family and friends, for providing inspiration and breaks, for being patient, and for not disowning me while I was holed up writing for months.

Robert Reinhardt, as always, for getting me started in writing books and for having a swell beard.

The Boston Bruins, for providing a lot of playoffs thrills during my infrequent breaks.

The numerous folks in the web community who have shared their expertise and experiences for the betterment of others. (I've cited many of you throughout the book.)

To you readers, for inspiring me to recall when I began learning HTML and CSS so that I may explain them in ways I hope you find helpful. Thank you for choosing this book as part of your journey in contributing to the web. Happy reading!

And, lastly, I would like to give a special thank you to Elizabeth Castro, who created this title in the 1990s. She has taught countless readers how to build webpages over many editions and many years. Because the web has given me so much, I'm genuinely appreciative of the opportunity to teach readers via this title as well.

—Bruce

Contents at a Glance

This page intentionally left blank

Table of Contents

Introduction

Whether you are just beginning your venture into building websites or have built some before but want to ensure that your knowledge is current, you've come along at a very exciting time.

How we code and style webpages, the browsers in which we view the pages, and the devices on which we visit the web have all advanced substantially the past few years. Once limited to browsing the web from our desktop computers or laptops, we can now take the web with us on any number of devices: phones, tablets, and, yes, laptops and desktops.

Which is as it should be, because the web's promise has always been the dissolution of boundaries—the power to share and access information from any metropolis, rural community, or anywhere in between and on any web-enabled device. In short, the web's promise lies in its universality. And its reach continues to expand as technology finds its way to communities that were once shut out.

Better still, the web belongs to everyone, and anyone is free to create and launch a site. This book shows you how. It is ideal for the beginner with no knowledge of HTML or CSS who wants to begin to create webpages. You'll find clear, easy-to-follow instructions that take you through the process of creating pages step by step. And the book is a helpful guide to keep handy. You can look up topics in the table of contents or index and consult just those subjects about which you need more information.

HTML and CSS in Brief

At the root of the web's success is a simple, text-based markup language that is easy to learn and that any device with a basic web browser can read: HTML. Every webpage requires at least some HTML; it wouldn't be a webpage without it.

As you will learn in greater detail as you read this book, HTML is used to define your content, and CSS is used to control how your content and webpage will look. Both HTML pages and CSS files (*style sheets*) are text files, making them easy to edit. You can see snippets of HTML and CSS in "How This Book Works," near the end of this introduction.

You'll dive into learning a basic HTML page right off the bat in Chapter 1, and you'll begin to learn how to style your pages with CSS in Chapter 7. See "What this book will teach you" later in this introduction for an overview of the chapters and a summary of the primary topics covered.

The word *HTML* is all encompassing, representing the language in general. *HTML5* is used when referring to that specific version of HTML, such as when discussing a feature that is new in HTML5 and doesn't exist in previous versions. The same approach applies to usage of the terms *CSS* (general) and *CSS3* (specific to CSS3).

HTML and HTML5

It helps to know some basics about the origins of HTML to understand HTML5.

HTML began in the early 1990s as a short document that detailed a handful of elements used to build webpages. Many of those elements were for content such as headings, paragraphs, lists, and links to other pages. HTML's version number has increased as the language has evolved with the introduction of other elements and adjustments to its rules. The most current version is HTML5.

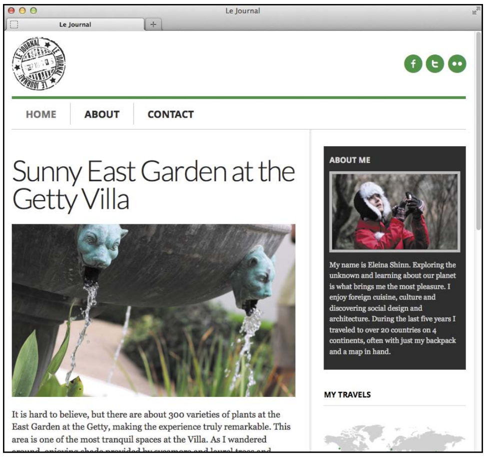
HTML5 is a natural evolution of earlier versions of HTML and strives to reflect the needs of both current and future websites. It inherits the vast majority of features from its predecessors, meaning that if you coded HTML before HTML5 came on the scene, you already know a lot of HTML5. This also means that much of HTML5 works in both old and new browsers; being backward compatible is a key design principle of HTML5 (see www.w3.org/TR/html-design-principles/).

HTML5 also adds a bevy of new features. Many are straightforward, such as additional elements (**`article`**, **`main`**, **`figure`**, and many more) that are used to describe content. Others are complex and aid in creating powerful web applications. You'll need a firm grasp of creating webpages before you can graduate to the more complicated features that HTML5 provides, which is why this book focuses on the former. HTML5 also introduces native audio and video playback to your webpages, which the book also covers.

CSS and CSS3

The first version of CSS didn't exist until after HTML had been around for a few years, becoming official in 1996. Like HTML5 and its relationship to earlier versions of HTML, CSS3 is a natural extension of the versions of CSS that preceded it.

CSS3 is more powerful than earlier versions of CSS and introduces numerous visual effects, such as drop shadows, rounded corners, gradients, and much more. (See "What this book will teach you" for details of what's covered.)



Sunny East Garden at the Getty Villa

It is hard to believe, but there are about 300 varieties of plants at the East Garden at the Getty, making the experience truly remarkable. This area is one of the most tranquil spaces at the Villa. As I wandered

My name is Eleina Shinn. Exploring the unknown and learning about our planet is what brings me the most pleasure. I enjoy foreign cuisine, culture and discovering social design and architecture. During the last five years I traveled to over 20 countries on 4 continents, often with just my backpack and a map in hand.

A The desktop version of Firefox

Browser Version Numbers

Like HTML and CSS, browsers have version numbers. The higher the number, the more recent it is.

For instance, Safari 7 is more recent than Safari 6, which is more recent than Safari 5. Internet Explorer 10 is more recent than Internet Explorer 9. But Internet Explorer 10 is not more recent than Safari 7.

This is true because Microsoft, Apple, and the other browser vendors do not collectively coordinate either their version numbers or when they will all release new versions. Chrome and Firefox release new versions every six weeks so naturally have much higher version numbers than the other browsers, which are updated roughly once a year at best.

Regardless of who is releasing what and when, the latest version of a browser will have better support for HTML and CSS (and other) features than the previous versions do, as you would expect.

Web Browsers

We all use a web browser to visit websites, whether on a computer A, a phone, or another device. However, the browser you use might be different than the one someone else uses.

Windows comes preinstalled with Internet Explorer, Microsoft's browser. OS X comes preinstalled with Safari, Apple's browser. There are other browsers you may download for free and use instead, such as Chrome (by Google), Firefox (by Mozilla) A, and Opera (by Opera Software)—and that's just for the desktop.

On mobile devices, you'll find the mobile version of Safari (for iPhone, iPad, and iPod touch); various default Android browsers; Chrome for Android; Firefox for Android; Opera Mini; and more.

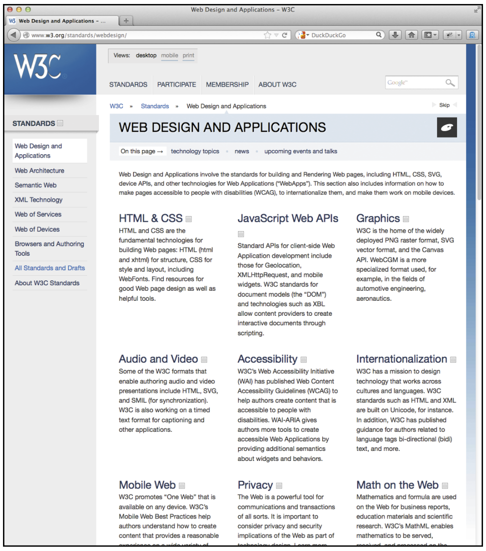
I'll refer to various browsers throughout the book. For the most part, the latest version of each one has similar support for the HTML and CSS features you'll learn about. But sometimes a feature doesn't work on one or more browsers (or works differently). I'll note those cases and typically offer a way to handle them. This mostly pertains to Internet Explorer 8, the oldest browser that is still relevant enough to be of concern. (Its usage is dropping, so that could change in 2014 or so.)

"Testing Your Pages" in Chapter 20 provides information about how to acquire various browsers, which ones are the most important for testing your webpages, and how to test your pages.

Web Standards and Specifications

You might be wondering who created HTML and CSS in the first place, and who continues to evolve them. The World Wide Web Consortium (W3C)—directed by the inventor of the web and HTML, Tim Berners-Lee—is the organization responsible for shepherding the development of web standards.

The W3C releases *specifications* (or *specs*, for short) that document these web standards. They define the parameters of languages like HTML and CSS. In other words, specs standardize the rules. Follow the W3C's activity at www.w3.org A.


Web Design and Applications – W3C
Web Design and Applications – ...
www.w3.org/standards/webdesign/
DuckDuckGo
W3C
Views: desktop mobile print
STANDARDS
PARTICIPATE
MEMBERSHIP
ABOUT W3C
Google
W3C » Standards » Web Design and Applications
Skip
STANDARDS
Web Design and Applications
Web Architecture
Semantic Web
XML Technology
Web of Services
Web of Devices
Browsers and Authoring Tools
All Standards and Drafts
About W3C Standards
On this page → technology topics • news • upcoming events and talks

WEB DESIGN AND APPLICATIONS

Web Design and Applications involve the standards for building and Rendering Web pages, including HTML, CSS, SVG, device APIs, and other technologies for Web Applications ("WebApps"). This section also includes information on how to make pages accessible to people with disabilities (WCAG), to internationalize them, and make them work on mobile devices.

HTML & CSS

HTML and CSS are the fundamental technologies for building Web pages: HTML (html and xhtml) for structure, CSS for style and layout, including WebFonts. Find resources for good Web page design as well as helpful tools.

JavaScript Web APIs

Standard APIs for client-side Web Application development include those for Geolocation, XMLHttpRequest, and mobile widgets. W3C standards for document models (the "DOM") and technologies such as XBL allow content providers to create interactive documents through scripting.

Graphics

W3C is the home of the widely deployed PNG raster format, SVG vector format, and the Canvas API. WebCGM is a more specialized format used, for example, in the fields of automotive engineering, aeronautics.

Audio and Video

Some of the W3C formats that enable authoring audio and video presentations include HTML, SVG, and SMIL (for synchronization). W3C is also working on a timed text format for captioning and other applications.

Accessibility

W3C's Web Accessibility Initiative (WAI) has published Web Content Accessibility Guidelines (WCAG) to help authors create content that is accessible to people with disabilities. WAI-ARIA gives authors more tools to create accessible Web Applications by providing additional semantics about widgets and behaviors.

Internationalization

W3C has a mission to design technology that works across cultures and languages. W3C standards such as HTML and XML are built on Unicode, for instance. In addition, W3C has published guidance for authors related to language tags bi-directional (bidi) text, and more.

Mobile Web

W3C promotes "One Web" that is available on any device. W3C's Mobile Web Best Practices help authors understand how to create content that provides a reasonable

Privacy

The Web is a powerful tool for communications and transactions of all sorts. It is important to consider privacy and security implications of the Web as part of

Math on the Web

Mathematics and formula are used on the Web for business reports, education materials and scientific research. W3C's MathML enables mathematics to be served,

A The W3C site is the industry's primary source of web standards specifications.

The W3C and WHATWG

For a variety of reasons, another organization—the Web Hypertext Application Technology Working Group (WHATWG)—is developing most of the HTML5 specification. The W3C incorporates WHATWG's work into its official version of the in-progress spec. You can find the WHATWG at www.whatwg.org.

If you want to dig into various specs (recommended!), here are the latest versions:

- HTML5 (W3C):
  http://www.w3.org/TR/html5/
- HTML5.1 (W3C):
  http://www.w3.org/TR/html51/
- HTML Living Standard (WHATWG):
  http://www.whatwg.org/specs/web-apps/current-work/multipage/

The HTML Living Standard includes newer features under development (and very much in flux) and informs the W3C's HTML5.1 spec.

There are too many CSS specs to list, but you can see them at http://www.w3.org/standards/techs/css#w3c_all.

Differences Between HTML4 and HTML5

If you have prior experience with HTML4 and are wondering what is different in HTML5, the W3C has created just the document for you: http://www.w3.org/TR/html5-diff/.

I call out many of the differences at various points in the book. They aren't particularly important to know for those of you who are new to HTML, because HTML5 is what virtually everyone uses now. But you might find the W3C's document interesting to peruse regardless.

With standards in place, we can build our pages from the agreed-upon set of rules, and browsers can be built to display our pages with those rules in mind. (On the whole, browsers implement the standards well. Older versions of Internet Explorer, especially Internet Explorer 8, have some issues.)

Specifications go through several stages of development before they are considered final, at which point they are dubbed a *Recommendation* (www.w3.org/2005/10/Process-20051014/tr).

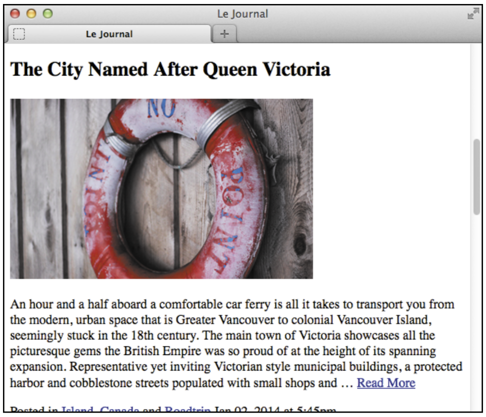
Parts of the HTML5 and CSS3 specs are still being finalized, but that doesn't mean you can't use them. It just takes time (literally years) for the standardization process to run its course. Browsers begin to implement a spec's features long before it becomes a Recommendation, because that informs the spec development process itself. So browsers already include a wide variety of features in HTML5 and CSS3, even though they aren't Recommendations yet.

On the whole, the features covered in this book are well entrenched in their respective specs, so the risk of their changing prior to becoming a Recommendation is minimal. Developers have been using many HTML5 and CSS3 features for some time. So can you.

Progressive Enhancement: A Best Practice

I began the introduction by speaking of the universality of the web—the notion that the web should be accessible to all. *Progressive enhancement* helps you build sites with universality in mind. It is not a language, but rather an approach to building sites that Steve Champeon promoted beginning in 2003 (http://en.wikipedia.org/wiki/Progressive_enhancement).

The idea is simple but powerful: Start your site with HTML content and basic behavior that is accessible to all visitors A. To the same page, add your design with CSS B and additional behavior with JavaScript (a programming language). These components are kept separate but work together.

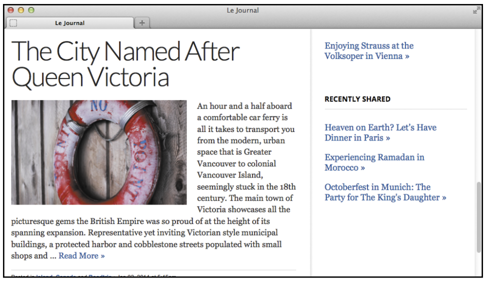


The City Named After Queen Victoria

An hour and a half aboard a comfortable car ferry is all it takes to transport you from the modern, urban space that is Greater Vancouver to colonial Vancouver Island, seemingly stuck in the 18th century. The main town of Victoria showcases all the picturesque gems the British Empire was so proud of at the height of its spanning expansion. Representative yet inviting Victorian style municipal buildings, a protected harbor and cobblestone streets populated with small shops and … Read More

Posted in Island, Canada and Roadtrip Jan 02, 2014 at 5:45pm

A A basic HTML page with no custom CSS applied to it. Primarily, only very old browsers would display it this way. The page may not look great, but the information is accessible—and that's what's important.



The City Named After Queen Victoria

An hour and a half aboard a comfortable car ferry is all it takes to transport you from the modern, urban space that is Greater Vancouver to colonial Vancouver Island, seemingly stuck in the 18th century. The main town of Victoria showcases all the picturesque gems the British Empire was so proud of at the height of its spanning expansion. Representative yet inviting Victorian style municipal buildings, a protected harbor and cobblestone streets populated with small shops and ... Read More »

Enjoying Strauss at the Volksoper in Vienna »

RECENTLY SHARED

Heaven on Earth? Let's Have Dinner in Paris »

Experiencing Ramadan in Morocco »

Octoberfest in Munich: The Party for The King's Daughter »

B The same page as viewed in a browser that supports CSS. It's the same information, just presented differently. (The content on the right side would be visible in A if you were to scroll down the page.)

More Examples

Take an early peek at Chapter 12 if you're interested in seeing how the principle of progressive enhancement helps you build a website that adapts its layout based on a device's screen size and browser capabilities. It can look great on mobile, desktop, and beyond.

Or see Chapter 14 for how older browsers can display simplified designs while modern browsers display ones enhanced with CSS3 effects.

Elsewhere in the book, you'll learn other techniques that allow you to build progressively enhanced webpages.

The result is that browsers capable of accessing basic pages will get the simplified, default experience A. Even browsers from the inception of the web more than 20 years ago can display this page; so too can the oldest or simplest of mobile phones with web browsers. And *screen readers*, software that reads webpages aloud to visually impaired visitors, will be able to navigate it easily.

Meanwhile, modern browsers capable of viewing more-robust sites will see the enhanced version B. The capabilities of yet other (somewhat older) browsers might fall somewhere in between; so, too, could the way they display the page. The experience on your site doesn't have to be the same for everyone, as long as your content is accessible.

In essence, the idea behind progressive enhancement is that everyone wins.

]]>

Is This Book for You?

This book assumes no prior knowledge of building websites. So in that sense, it is for the absolute beginner. You will learn both HTML and CSS from the ground up. In the course of doing so, you will also learn about features that are new in HTML5 and CSS3, with an emphasis on many that designers and developers are using today in their daily work.

But even if you are familiar with HTML and CSS, you still stand to learn from this book, especially if you want to get up to speed on the new elements in HTML5, several CSS3 effects, responsive web design, and various best practices.

What this book will teach you

The chapters are organized like so:

- Chapters 1 through 6 and 15 through 18 cover the principles of creating HTML pages and most of the HTML elements at your disposal, with clear examples demonstrating how and when to use each one.
- Chapters 7 through 14 dive into CSS, all the way from creating your first style rule to applying enhanced visual effects with CSS3.
- Chapter 19 shows you how to add pre-written JavaScript to your pages.
- Chapter 20 tells you how to test and debug your pages before putting them on the web.
- Chapter 21 explains how to secure your own domain name and then publish your site on the web for all to see.

Covered topics include the following:

- Creating, saving, and editing HTML and CSS files.
- What it means to write semantic HTML and why it is important.
- How to separate your page's HTML content, CSS presentation, and JavaScript behavior—a key aspect of progressive enhancement.
- Structuring your content in a meaningful way by using HTML elements that have been around for years as well as ones that are new in HTML5.
- Linking from one webpage to another, or from one part of a page to another part.
- Adding images to your pages and optimizing them for the web. This includes creating images targeted for Apple's Retina display and other high-pixel-density screens.
- Improving your site's accessibility with ARIA (Accessible Rich Internet Applications) landmark roles and other good coding practices.
- Styling text (size, color, bold, italics, and more) and adding background colors and images.
- Implementing a multi-column webpage layout.
- Building a responsive webpage. That is, a page that shrinks or expands to fit your visitor's screen and with a layout that adapts in other ways as you wish. The result is a page that's appropriate for mobile phones, tablets, laptops, desktop computers, and other web-enabled devices.

- Adding custom web fonts to your pages with **`@font-face`** and using fonts from services like Font Squirrel and Google Fonts.
- Using CSS3 effects such as opacity, background alpha transparency, gradients, rounded corners, drop shadows, shadows inside elements, text shadows, and multiple background images.
- Taking advantage of CSS generated content and using sprites to minimize the number of images your page needs, making it load faster for your visitors.
- Building forms to solicit input from your visitors, including using some of the new form input types in HTML5.
- Including media in your pages with the HTML5 **`audio`** and **`video`** elements for modern browsers, and a Flash fallback audio or video player for older browsers.
- And more.

These topics are complemented by many dozens of code samples that demonstrate how to implement the features based on best practices in the industry.

What this book *won't* teach you

Alas, with so many developments in the world of HTML and CSS in recent years, we had to leave out some topics. With a couple of exceptions, we stuck to omitting items that you would likely have fewer occasions to use, are still subject to change, lack widespread browser support, require JavaScript knowledge, or are advanced subjects.

Some of the topics not covered include the following:

- The HTML5 **`details`**, **`summary`**, **`menu`**, **`command`**, **`output`**, and **`keygen`** elements. The W3C has included some of these on their list of features that might not make the cut when HTML5 is finalized in 2014. The others are used infrequently at best.
- The HTML5 **`canvas`** element, which allows you to draw graphics, create games, and more. Also, Scalable Vector Graphics (SVG). Both are mentioned briefly in Chapter 17, with links to more information.
- The HTML5 APIs and other advanced features that require JavaScript knowledge or are otherwise not directly related to the new HTML5 elements.
- CSS3 transforms, animations, and transitions. See www.htmlcssvqs.com/resources/ for links to learn more.
- CSS3's new layout methods, such as FlexBox, Grid, and more. They are poised to change the way we lay out pages once the specs shake out and browser support is stronger. See Zoe Mickley Gillenwater's presentation at www.slideshare.net/zomigi/css3-layout, or see Peter Gasston's article at www.netmagazine.com/features/pros-guide-css-layouts.

How This Book Works

Nearly every section of the book contains practical code examples that demonstrate real-world use (A and B). Typically, they are coupled with screen shots that show the results of the code when you view the webpage in a browser C.

Most of the screen shots are of the latest version of Firefox that was available at the time. However, this doesn't imply a recommendation of Firefox over any other browser. The code samples will look similar in any of the latest versions of Chrome, Internet Explorer, Opera, or Safari.

The code and screen shots are accompanied by descriptions of the HTML elements or CSS properties in question, both to increase your understanding of them and to give the samples context.

In many cases, you may find that the descriptions and code samples are enough for you to start using the HTML and CSS features. But if you need explicit guidance on how to use them, step-by-step instructions are provided as well.

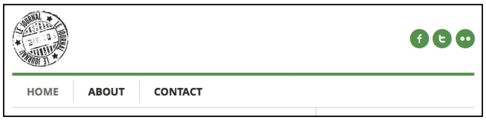
Finally, most sections contain tips that relay additional usage information, best practices, references to related parts of the book, links to relevant resources, and more.

A You'll find a snippet of HTML code on many pages, with the pertinent sections highlighted. An ellipsis (**...**) represents additional code or content that was omitted for brevity. Often, the omitted portion is shown in a different code figure.

```
...
<body>
<header class="masthead" role="banner">
     ...
        <nav role="navigation">
           <ul class="nav-main">
              <li><a href="/" class="current-page">Home</a></li>
              <li><a href="/about/">About</a></li>
              <li><a href="/contact/">Contact</a></li>
           </ul>
        </nav>
     ...
</header>
...
</body>
</html>
```

B If CSS code is relevant to the example, it is shown in its own box, with the pertinent sections highlighted.

```
body {
     font-family: Georgia, "Times New Roman",
     ➝ serif;
}

/* Site Navigation */
.nav-main {
     list-style: none;
     padding: .45em 0 .5em;
}

.nav-main li {
     border-left: 1px solid #c8c8c8;
}

.nav-main a {
     color: #292929;
     font-size: 1.125em;
     font-weight: bold;
}
```



C Screen shots of one or more browsers demonstrate how the code affects the page.

Conventions used in this book

The book uses the following conventions:

- Text that is a placeholder for a value you would create yourself is italicized. Most placeholders appear in the step-by-step instructions. For example, "Type **`padding:`** ***`x`*****`;`**, where ***x*** is the amount of desired space to be added.
- Code that you should actually type or that represents HTML or CSS code appears in **`this font`**.
- An arrow (➝) in a code figure indicates a continuation of the previous line—the line has been wrapped to fit in the book's column B. The arrow is not part of the code itself, so it's not something you would type. Instead, type the line continuously, as if it had not wrapped to another line.
- The first occurrence of a word is italicized when it is defined.
- *IE* is often used as a popular abbreviation of *Internet Explorer*. For instance, IE9 is synonymous with Internet Explorer 9.
- *Modern browsers* collectively refers to the versions of browsers with solid support for the latest HTML5 and CSS3 features. Generally, this includes recent versions of the browsers discussed in the "Web Browsers" section of this introduction, but not IE8.
- Whenever a plus sign (+) follows a browser version number, it means "the version listed plus subsequent versions." For instance, IE8+ refers to Internet Explorer 8 and all versions after it.

Companion Website

The book's companion website contains the table of contents, every complete code example featured in the book (plus some additional ones that wouldn't fit), links to resources cited in the book (as well as additional ones), a list of errata, and more.

The URLs for some of the key pages on the site follow:

- Home page:
  www.htmlcssvqs.com
- Code examples:
  www.htmlcssvqs.com/8ed/examples/

You can view the code examples directly from the site or download them to your computer—all the HTML and CSS files are yours for the taking.

In some cases, I've included additional comments in the code to explain more about what it does or how to use it. A handful of the code samples in the book are truncated for space considerations, but the complete versions are on the website.

Please feel free to use the code as you please, modifying it as needed for your own projects.

I hope you find the site helpful!

4

Text

Unless a site is heavy on videos or photo galleries, most content on webpages is text. This chapter explains which HTML semantics are appropriate for different types of text, especially (but not solely) for text within a sentence or phrase.

For example, the **`em`** element is specifically designed for indicating emphasized text, and the **`cite`** element's purpose is to cite works of art, movies, books, and more.

Browsers typically style many text elements differently than normal text. For instance, both the **`em`** and **`cite`** elements are italicized. Another element, **`code`**, which is specifically designed for formatting lines of code from a script or program, displays in a monospace font by default.

How content will look is irrelevant when deciding how to mark it up. So, you shouldn't use **`em`** or **`cite`** just because you want to italicize text. That's the job of CSS.

Instead, focus on choosing HTML elements that describe the content. If by default a browser styles it as you would yourself with CSS, that's a bonus. If not, just override the default formatting with your own CSS.

In This Chapter

Adding a Paragraph

HTML does not recognize the returns or other extra whitespace that you enter in your text editor. To start a new paragraph in your webpage, you use the **p** element (A and B).

To create a new paragraph:

1. Type **<p>**.
2. Type the contents of the new paragraph.
3. Type **</p>** to end the paragraph.

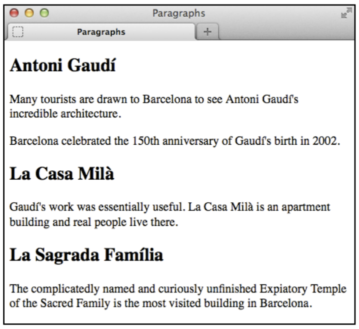
TIP You can use styles to format paragraphs (and other page text) with a particular font, size, or color (and more). For details, consult Chapter 10.

TIP To control the amount of space between lines within a paragraph, consult "Setting the Line Height" in Chapter 10. To control the amount of space before or after a paragraph, consult "Setting the Margins Around an Element" or "Adding Padding Around an Element," both of which are in Chapter 11.

TIP You can justify paragraph text or align it to the left, right, or center with CSS (see "Aligning Text" in Chapter 10).

A Unsurprisingly, **p** is one of the most frequently used HTML elements. (Note: In practice, I would wrap an **article** around this particular content. I omitted it to make the example generic and to avoid giving the impression that **p** elements must always be nested in an **article**.)

```
...
<body>

<h1>Antoni Gaudí</h1>
<p>Many tourists are drawn to Barcelona
➝ to see Antoni Gaudí's incredible
➝ architecture.</p>

<p>Barcelona celebrated the 150th
➝ anniversary of Gaudí's birth in
➝ 2002.</p>

<h2 lang="es">La Casa Milà</h2>
<p>Gaudí's work was essentially useful.
➝ <span lang="es">La Casa Milà</span> is
➝ an apartment building and real people
➝ live there.</p>

<h2 lang="es">La Sagrada Família</h2>
<p>The complicatedly named and curiously
➝ unfinished Expiatory Temple of the
➝ Sacred Family is the most visited
➝ building in Barcelona.</p>

</body>
</html>
```



Antoni Gaudí

Many tourists are drawn to Barcelona to see Antoni Gaudí's incredible architecture.

Barcelona celebrated the 150th anniversary of Gaudí's birth in 2002.

La Casa Milà

Gaudí's work was essentially useful. La Casa Milà is an apartment building and real people live there.

La Sagrada Família

The complicatedly named and curiously unfinished Expiatory Temple of the Sacred Family is the most visited building in Barcelona.

B Here you see the typical default rendering of paragraphs. By default, browsers provide vertical space between headings and paragraphs, and between paragraphs themselves. As with all content elements, you have full control over the formatting with CSS.

Specifying Fine Print

The **small** element represents side comments such as fine print, which, according to the HTML5 spec, "typically features disclaimers, caveats, legal restrictions, or copyrights. Small print is also sometimes used for attribution or for satisfying licensing requirements."

The **small** element is intended for brief portions of inline text, not for text spanning multiple paragraphs or other elements (A and B).

To specify fine print:

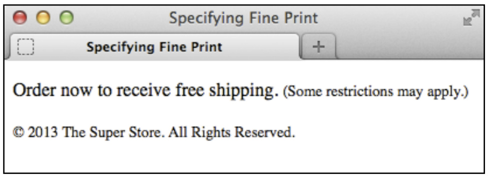
1. Type **<small>**.
2. Type the text that represents a legal disclaimer, note, attribution, and so on.
3. Type **</small>**.

TIP Be sure to use small only because it's appropriate for your content, not because you want to reduce the text size, as happens in some browsers B. You can always adjust the size with CSS (even making it larger if you'd like). See "Setting the Font Size" in Chapter 10 for more information.

TIP The small element is a common choice for marking up your page's copyright notice (A and B). It's meant for short phrases like that, so don't wrap it around long legal notices, such as your Terms of Use or Privacy Policy pages. Those should be marked up with paragraphs and other semantics, as necessary.

A The **small** element denotes brief legal notices in both instances shown. The second one is a copyright notice contained in a page-level **footer**, a common convention.

```
...
<body>

<p>Order now to receive free shipping.
<small>(Some restrictions may apply.)
→ </small></p>

...

<footer role="contentinfo">
     <p><small>&copy; 2013 The Super
     → Store. All Rights Reserved.
     → </small></p>
</footer>

</body>
</html>
```


Specifying Fine Print

Order now to receive free shipping. (Some restrictions may apply.)

© 2013 The Super Store. All Rights Reserved.

B The **small** element may render smaller than normal text in some browsers, but the visual size is immaterial to whether you should mark up your content with it.

Marking Important and Emphasized Text

The **`strong`** element denotes important text, whereas **`em`** represents stress emphasis. You can use them individually or together, as your content requires (A and B).

To mark important text:

1. Type **`<strong>`**.
2. Type the text that you want to mark as important.
3. Type **`</strong>`**.

To emphasize text:

1. Type **`<em>`**.
2. Type the text that you want to emphasize.
3. Type **`</em>`**.

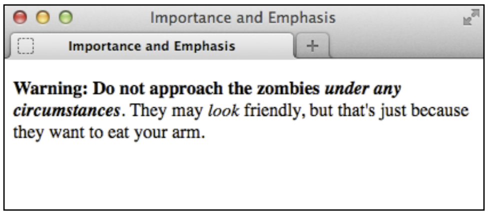
TIP Do not use the `b` and `i` elements as replacements for `strong` and `em`, respectively. Although they may look similar in a browser, their meanings are very different (see the sidebar "The `b` and `i` Elements: Redefined in HTML5").

TIP Just as when you emphasize words in speech, where you place `em` in a sentence affects its meaning. For example, `<p><em>Run</em> over here.</p>` and `<p>Run over <em>here</em>.</p>` convey different messages.

TIP The importance of `strong` text increases each time it's a child of another `strong`. The same is true of the level of emphasis for `em` text in another `em`. For example, "due by April 12th" is marked as more important semantically than the other `strong` text in this sentence: `<p><strong>Remember that entries are <strong>due by March 12th</strong>.</strong></p>`.

A The first sentence has both **`strong`** and **`em`**, whereas the second has **`em`** only.

```
...
<body>

<p><strong>Warning: Do not approach the
→ zombies <em>under any circumstances</em>
→ </strong>. They may <em>look</em>
→ friendly, but that's just because they want
→ to eat your arm.</p>

</body>
</html>
```



Warning: Do not approach the zombies *under any circumstances*. They may *look* friendly, but that's just because they want to eat your arm.

B Browsers typically display **`strong`** text in boldface and **`em`** text in italics. If **`em`** is a child of a **`strong`** element (see the first sentence in A), its text will be both italicized and bold.

TIP You can style any text as bold or italic with CSS, as well as negate the browser's default styling of elements like `strong` and `em` B. For details, consult "Creating Italics" and "Applying Bold Formatting" in Chapter 10.

TIP If you had experience with HTML before HTML5, you may know that at that time `strong` represented text with stronger emphasis than `em` text. In HTML5, however, `em` is the only element that indicates emphasis, and `strong` has shifted to importance.

The `b` and `i` Elements: Redefined in HTML5

HTML5 focuses on semantics, not on an element's presentation. The **`b`** and **`i`** elements are hold-overs from the earliest days of HTML, when they were used to make text bold or italic (CSS didn't exist yet). They fell out of favor in HTML 4 and XHTML 1 because of their presentational nature. Coders were encouraged to use **`strong`** instead of **`b`**, and **`em`** instead of **`i`**. It turns out, though, that **`em`** and **`strong`** are not always semantically appropriate. HTML5 addresses this by redefining **`b`** and **`i`**.

Some typographic conventions in traditional publishing fall through the cracks of available HTML semantics. Among them are italicizing certain scientific names (for example, "The *Ulmus americana* is the Massachusetts state tree."), named vehicles (for example, "We rode the *Orient Express*."), and foreign (to English) language phrases (for example, "The couple exhibited a *joie de vivre* that was infectious."). These terms aren't italicized for emphasis, just stylized per convention.

Rather than create several new semantic elements to address cases like these (and further muddy the waters), HTML5 takes a practical stance by trying to make do with what is available: **`em`** for all levels of stress emphasis, **`strong`** for importance, and **`b`** and **`i`** for the through-the-cracks cases. HTML5 emphasizes that you use **`b`** and **`i`** only as a last resort when another element (such as **`strong`**, **`em`**, **`cite`**, and others) won't do.

The b Element in Brief

HTML5 redefines the **`b`** element this way:

The **`b`** element represents a span of text to which attention is being drawn for utilitarian purposes without conveying any extra importance and with no implication of an alternate voice or mood, such as key words in a document abstract, product names in a review, actionable words in interactive text-driven software, or an article lede.

For example:

```
<p>The <b>XR-5</b>, also dubbed the <b>Extreme Robot 5</b>, is the best robot we've ever
➝ tested.</p>
```

The **`b`** element renders as bold by default.

The `i` Element in Brief

HTML5 redefines the **`i`** element this way:

The **`i`** element represents a span of text in an alternate voice or mood, or otherwise offset from the normal prose in a manner indicating a different quality of text, such as a taxonomic designation, a technical term, an idiomatic phrase or short span of transliterated prose from another language, a thought, or a ship name in Western texts.

Here are some examples:

```
<p>The <i lang="la">Ulmus americana</i> is the Massachusetts state tree.</p>
<p>We rode the <i>Orient Express</i>.<p>
<p>The couple exhibited a <i lang="fr">joie de vivre</i> that was infectious.<p>
```

The **`i`** element displays in italics by default.

Creating a Figure

No doubt you've seen figures in printed newspapers, magazines, reports, and more. Typically, figures are referenced from the main text on a page (like a news story). This very book has them on most pages.

Prior to HTML5, there wasn't an element designed for this use, so developers cobbled together solutions on their own. This often involved the less-than-ideal, non-semantic **div** element. HTML5 has changed that with **figure** and **figcaption** (A and B). A **figure** element may contain a chart, a photo, a graph, an illustration, a code segment, or similar self-contained content.

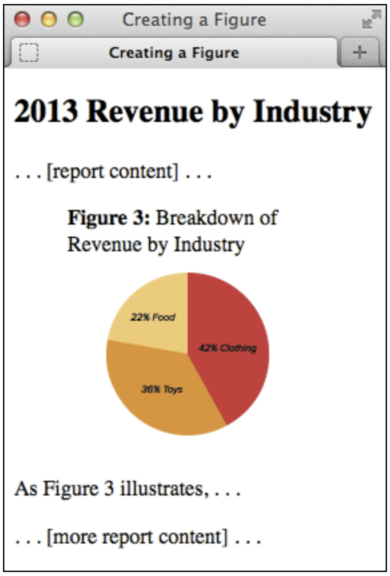
You may refer to a **figure** from other content on your page (as shown in A and B), but it isn't required. The optional **figcaption** is a **figure**'s caption or legend and may appear either at the beginning or at the end of a **figure**'s content.

To create a figure and figure caption:

1. Type **<figure>**.
2. Optionally, type **<figcaption>** to begin the figure's caption.
3. Type the caption text.
4. Type **</figcaption>** if you created a caption in steps 2 and 3.
5. Create your figure by adding code for images, videos, data tables, and so on.
6. If you didn't include a **figcaption** before your **figure**'s content, optionally follow steps 2–4 to add one after the content.
7. Type **</figure>**.

A This **figure** has a chart image, though more than one image or other types of content (such as a data table or video) are allowed as well. The **figcaption** element isn't required, but it must be the first or last element in a **figure** if you do include it. A **figure** doesn't have a default styling aside from starting on its own line in modern browsers B. (Note: **figure**s aren't required to be in an **article**, but it's probably suitable in most cases.)

```
...
<body>
...
<article>
    <h1>2013 Revenue by Industry</h1>

    <p>... [report content] ...</p>

    <figure>
       <figcaption><b>Figure 3:</b>
       → Breakdown of Revenue by
       → Industry</figcaption>

       <img src="chart-revenue.png"
       → width="180" height="143" alt=
       → "Revenue chart: Clothing 42%,
       → Toys 36%, Food 22%" />
    </figure>

    <p>As Figure 3 illustrates, ... </p>

    <p>... [more report content] ...</p>
</article>
...
</body>
</html>
```



2013 Revenue by Industry

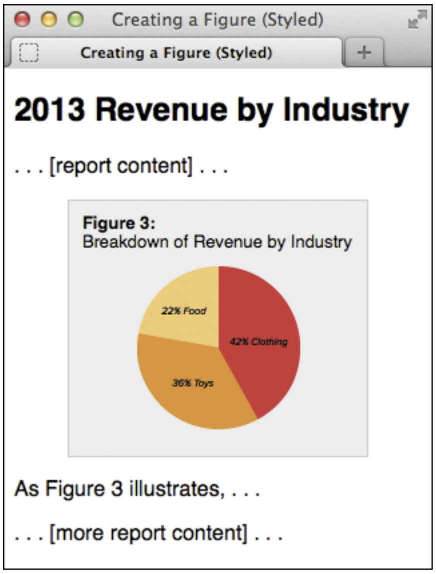
. . . [report content] . . .

Figure 3: Breakdown of Revenue by Industry

As Figure 3 illustrates, . . .

. . . [more report content] . . .

B The **figure** of the chart and caption appears within the **article** text. The figure is indented because of the browser's default styling (see the last tip).



2013 Revenue by Industry

. . . [report content] . . .

Figure 3:
Breakdown of Revenue by Industry

As Figure 3 illustrates, . . .

. . . [more report content] . . .

C You can differentiate your **figure** from the surrounding text with just a little bit of CSS. This simple example is available at www.htmlcssvqs.com/8ed/figure-styled/.

TIP Typically, `figure` is part of the content that refers to it **A**, but it could also live elsewhere on the page or on another page, such as in an appendix.

TIP The `figure` element may include multiple pieces of content. For instance, **A** could include two charts: one for revenue and another for profits. You can even nest one `figure` inside another one. Keep in mind, though, that regardless of how much content a `figure` has, only one `figcaption` is allowed per `figure`.

TIP Don't use `figure` simply as a means to embed all instances of self-contained bits of content within text. Oftentimes, the `aside` element may be appropriate instead (see "Specifying an Aside" in Chapter 3).

TIP See "Quoting Text" to learn how to use `figure` with a `blockquote` element.

TIP You can't use the `figcaption` element unless it's in a `figure` with other content.

TIP `figcaption` text doesn't have to begin with "Figure 3" or "Exhibit B." It could just as well be a brief description of the content, like a photo caption.

TIP Modern browsers apply left and right margins of 40px to a `figure` by default **C**. You can change that with the `margin-left` and `margin-right` CSS properties. For example, `margin-left: 0;` would make the figure flush left. Also, you can make the text containing a `figure` wrap around it with `figure { float: left;}` (so the text will wrap around the right side) or `figure { float: right;}` (so the text will wrap around the left side). You may need to set a `width` to the `figure` as well so it doesn't occupy too much horizontal real estate. CSS coverage begins in Chapter 7, and the `float` and `width` properties are demonstrated in Chapter 11.

Indicating a Citation or Reference

Use the **cite** element for a citation or reference to a source. Examples include the title of a play, script, or book; the name of a song, movie, photo, or sculpture; a concert or musical tour; a specification; a newspaper or legal paper; and more (A and B).

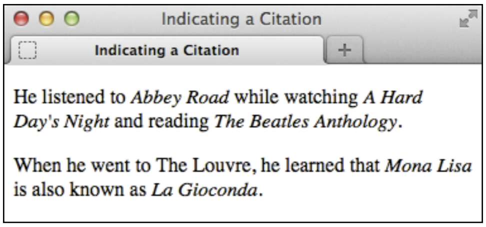
To cite a reference:

1. Type **<cite>**.
2. Type the reference's name.
3. Type **</cite>**.

TIP For instances in which you are quoting from the cited source, use the blockquote or q elements, as appropriate, to mark up the quoted text (see "Quoting Text"). To be clear, cite is only for the source, not for what you are quoting from it.

A In this example, the **cite** element marks up the titles of an album, a movie, a book, and a work of art. (Note: The **lang="it"** in the last instance declares that the language of the **cite** text is Italian.)

```
...
<body>

<p>He listened to <cite>Abbey Road</cite>
→ while watching <cite>A Hard Day's Night
→ </cite> and reading <cite>The Beatles
→ Anthology</cite>.

<p>When he went to The Louvre, he learned
→ that <cite>Mona Lisa</cite> is also
→ known as <cite lang="it">La Gioconda
→ </cite>.</p>

</body>
</html>
```



He listened to *Abbey Road* while watching *A Hard Day's Night* and reading *The Beatles Anthology*.

When he went to The Louvre, he learned that *Mona Lisa* is also known as *La Gioconda*.

B The **cite** element renders in italics by default.

HTML5 and Using the cite Element for Names

Amid a good amount of disagreement from the development community, HTML5 explicitly declares that using **cite** for a reference to a person's name is invalid, even though previous versions of HTML allowed it and many developers and designers used it that way.

The HTML 4 spec provides the following example (I've changed the element names from uppercase to lowercase):

```
As <cite>Harry S. Truman</cite> said,
<q lang="en-us">The buck stops here.</q>
```

In addition to instances like that, sites have often used **cite** for the name of people who leave comments in blog postings and articles (the default WordPress theme does, too).

Many developers have made it clear that they intend to continue to use **cite** on names associated with quotes in their HTML5 pages, because they find the alternatives that HTML5 provides unacceptable (namely, the **span** and **b** elements). Jeremy Keith made the case vociferously in http://24ways.org/2009/incite-a-riot/.

Quoting Text

There are two special elements for marking text quoted from a source. The **blockquote** element represents a stand-alone quote (generally a longer one, but not necessarily) (A and B) and displays on its own line by default C. Meanwhile, the **q** element is for short quotes, like those within a sentence D.

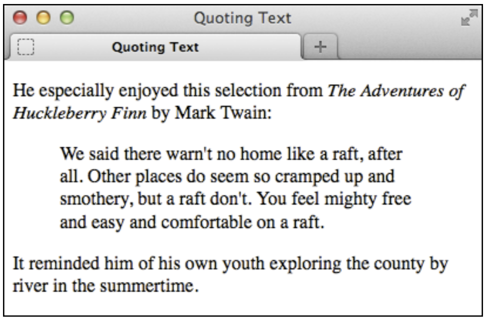
continues on next page

A A **blockquote** can be as short or as long as you need. Optionally, include the **cite** attribute—not to be confused with the **cite** element shown in the first paragraph—to provide the location of the quoted text.

```
...
<body>

<p>He especially enjoyed this selection from
→ <cite>The Adventures of Huckleberry Finn
→ </cite> by Mark Twain:</p>

<blockquote cite="http://www.
→ marktwainbooks.edu/the-adventures-of-
→ huckleberry-finn/">
     <p>We said there warn't no home like a
     → raft, after all. Other places do seem
     → so cramped up and smothery, but a
     → raft don't. You feel mighty free and
     → easy and comfortable on a raft.</p>
</blockquote>

<p>It reminded him of his own youth exploring
→ the county by river in the summertime.</p>

</body>
</html>
```

B If you'd like to provide attribution, it must be outside the **blockquote**. You could place the attribution in a **p**, but the most explicit way to associate a quote with its source is with a **figure** and **figcaption**, as shown (see "Creating a Figure").

```
...

<figure>
     <blockquote>
     I want all my senses engaged. Let
     → me absorb the world's variety and
     → uniqueness.
     </blockquote>
     <figcaption>— Maya Angelou</figcaption>
</figure>

...
```



He especially enjoyed this selection from *The Adventures of Huckleberry Finn* by Mark Twain:

> We said there warn't no home like a raft, after all. Other places do seem so cramped up and smothery, but a raft don't. You feel mighty free and easy and comfortable on a raft.

It reminded him of his own youth exploring the county by river in the summertime.

C Browsers typically indent **blockquote** text by default, and don't display the **cite** attribute value. (See the second tip for a related recommendation.) The **cite** element, on the other hand, is supported by all browsers and typically renders in italics, as shown. All of these defaults can be overridden with CSS.

D Here we see two **q** examples. Add the **lang** attribute to the **q** element if the quoted text is in a different language than the page's default (as specified by the **lang** attribute on the **html** element).

```
...
<body>

<p>And then she said, <q>Have you read
→ Barbara Kingsolver's <cite>High Tide in
→ Tucson</cite>? It's inspiring.</q></p>

<p>She tried again, this time in French:
→ <q lang="fr">Avez-vous lu le livre <cite
→ lang="en">High Tide in Tucson</cite> de
→ Kingsolver? C'est inspirational.</q></p>

</body>
</html>
```

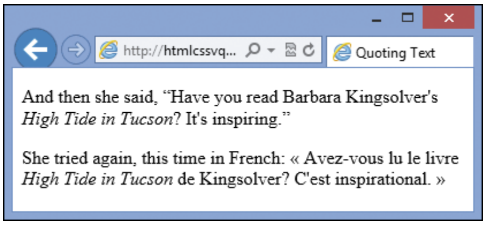
Browsers are supposed to enclose **q** element text in language-specific quotation marks automatically, but the results are mixed **E**. Be sure to read the tips to learn about alternatives to using the **q** element.

To quote a block of text:

1. Type **<blockquote** to begin a block quote.
2. If desired, type **cite="*url*"**, where ***url*** is the address of the source of the quote.
3. Type **>** to complete the start tag.
4. Type the text you wish to quote, surrounding it with paragraphs and other elements as appropriate.
5. Type **</blockquote>**.

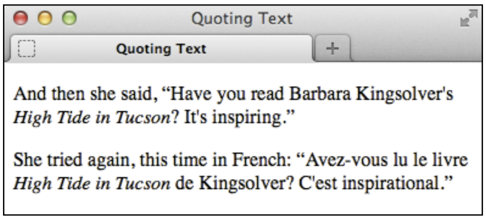
To quote a short phrase:

1. Type **<q** to begin quoting a word or phrase.
2. If desired, type **cite="*url*"**, where ***url*** is the address of the source of the quote.
3. If the quote's language is different than the page's default language (as specified by the **lang** attribute on the **html** element), type **lang="*xx*"**, where ***xx*** is the code for the language the quote will be in. This code is *supposed* to determine the type of quote marks that will be used (“ ” for English, « » for many European languages, and so on), though browser support for this rendering can vary.
4. Type **>** to complete the start tag.
5. Type the text that should be quoted.
6. Type **</q>**.



And then she said, “Have you read Barbara Kingsolver's *High Tide in Tucson*? It's inspiring.”

She tried again, this time in French: « Avez-vous lu le livre *High Tide in Tucson* de Kingsolver? C'est inspirational. »



And then she said, “Have you read Barbara Kingsolver's *High Tide in Tucson*? It's inspiring.”

She tried again, this time in French: “Avez-vous lu le livre *High Tide in Tucson* de Kingsolver? C'est inspirational.”

E Browsers are supposed to add language-specific quotation marks around **q** elements automatically. In this example, that means curly double quotes for English and guillemets for French. IE (shown on top) and Chrome do this correctly. Firefox (shown on bottom) is correct for English but not French. Opera and Safari do neither, rendering straight quotes instead, including for French. Inconsistencies like these limit the usefulness of the **q** element.

TIP If your `blockquote` contains only a single paragraph or phrase, you don't have to enclose it in a `p` within the `blockquote`.

TIP You can use the optional `cite` attribute on `blockquote` and `q` to provide a URL to the source you are quoting. Although historically browsers haven't displayed the `cite` attribute's URL C, in theory it can be handy for search engines or other automated tools that gather quotes and their references. If you would like visitors to have access to it, you could repeat the URL in a link (via the `a` element) in your content. Less effectively, you could expose `cite`'s value via JavaScript (search online for sample code).

TIP The `q` element is invalid for a quote that extends beyond one paragraph. Instead, use `blockquote`.

TIP Be sure you don't use `q` simply because you want quotation marks around a word or phrase. For instance, `<p>Every time I hear the word <q>soy</q>, I jump for joy.</p>` is improper because "soy" isn't a quote from a source. In that case, simply type quotation marks around the word.

TIP You can nest `blockquote` and `q` elements. For example, `<p>The short story began, <q>When she was a child, she would say, <q>Howdy, stranger!</q> to everyone she passed.</q></p>`. Nested `q` elements should display the appropriate quotation marks automatically—for example, in English the outer quotes should be double and the inner ones should be single. Since outer and inner quotations are treated differently in languages, add the `lang` attribute to `q` as needed D. Unfortunately, browsers are inconsistent with nested `q` elements much like they are for non-nested ones E.

TIP Because of cross-browser issues with `q` E, many (most likely the majority of) coders choose to simply type the desired quotation marks or use character entities instead of the `q` element.

Specifying Time

You can mark up a time, date, or duration with the **time** element, which is new in HTML5. It allows you to represent this information in a variety of ways (A and C).

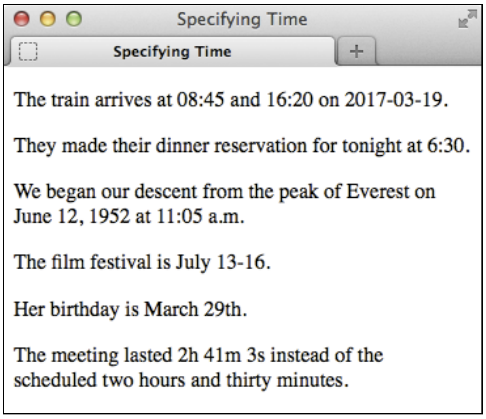
The text content inside **time** (that is, **<time>text</time>**) appears on the screen for us humans (B and D), whereas the value of the optional **datetime** attribute is intended for the machines among us. It requires a specific format; the sidebar "Understanding the Valid Time Format" covers the basics, and the first tip explains another case when the format is required.

To specify a time, date, or duration:

1. Type **<time** to begin a **time** element.
2. If desired, type **datetime="*time*"** where ***time*** is in the approved machine-readable format (see the sidebar) that represents the text you'll enter in step 4.
3. Type **>** to complete the start tag.
4. Type the text that reflects the time, the date, or the duration that you want to display in the browser. (See the first tip if you did not include **datetime** in step 2.)
5. Type **</time>**.

A As shown in the first example, the simplest form of the **time** element lacks a **datetime** attribute. But it does provide the times and date in the valid machine-readable format as required when **datetime** is omitted. The remaining examples show that the text between the **time** tags doesn't need to match the valid format when **datetime** is present (the last example shows one case of each approach).

```
...
<body>

<p>The train arrives at <time>08:45</time>
→ and <time>16:20</time> on <time>
→ 2017-03-19</time>.</p>

<p>They made their dinner reservation for
→ <time datetime="2013-11-20T18:30:00">
→ tonight at 6:30</time>.</p>

<p>We began our descent from the peak of
→ Everest on <time datetime="1952-06-12T
→ 11:05:00">June 12, 1952 at 11:05 a.m.
→ </time></p>

<p>The film festival is <time datetime=
→ "2014-07-13">July 13</time>-<time
→ datetime="2014-07-16">16</time>.</p>

<!-- Example with no year -->
<p>Her birthday is <time datetime="03-29">
→ March 29th</time>.</p>

<!-- Example of durations -->
<p>The meeting lasted <time>2h 41m 3s
→ </time> instead of the scheduled <time
→ datetime="2h 30m">two hours and thirty
→ minutes</time>.</p>

</body>
</html>
```



The train arrives at 08:45 and 16:20 on 2017-03-19.

They made their dinner reservation for tonight at 6:30.

We began our descent from the peak of Everest on June 12, 1952 at 11:05 a.m.

The film festival is July 13-16.

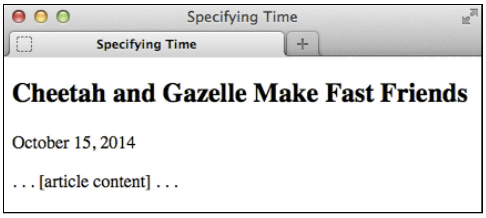
Her birthday is March 29th.

The meeting lasted 2h 41m 3s instead of the scheduled two hours and thirty minutes.

B Only the **`time`** text displays in browsers, not the **`datetime`** value.

C This shows how you might include a date for a blog post or news article. As is required for all cases of **`datetime`**, its value represents the text content in a machine-readable format.

```
...
<body>

<article>
     <h1>Cheetah and Gazelle Make Fast
     → Friends</h1>
     <p><time datetime="2014-10-15">October
     → 15, 2014</time></p>

     ... [article content] ...
</article>

</body>
</html>
```



Cheetah and Gazelle Make Fast Friends

October 15, 2014

. . . [article content] . . .

D As expected, the date is below the heading.

TIP **If you omit the `datetime` attribute, the text content inside `time` must follow the machine-friendly format rather than being "free-form." In other words, the first example in A could not be coded as `<p>The train arrives at <time>8:45 a.m.</time> and <time>4:20 p.m.</time> on <time>April 20th, 2015</time>.</p>` because the `time` text doesn't follow the format in any of the three instances. However, when you do include `datetime`, you're free to represent the date, time, or duration in the text content as you wish, as seen in the other examples in A.**

TIP **The `datetime` attribute doesn't do anything on its own but could be used for syncing dates and times between web applications and the like (for example, think of a calendar application). That's why it requires a standard, machine-readable format; it allows these programs to share information by speaking the same "language."**

TIP **You may not nest a `time` element inside another one or place any other elements (just text) in a `time` element that lacks a `datetime` attribute.**

TIP **The `time` element allowed an optional attribute named `pubdate` in an earlier iteration of HTML5 (remember that the language is still evolving). However, `pubdate` is no longer part of HTML5. I mention this in case you come across it in an older tutorial or book (such as the seventh edition of this book!) and wonder if you should use it (you shouldn't).**

Understanding the Valid Time Format

The **`datetime`** attribute—or a **`time`** element without **`datetime`**—must provide the desired date and/or time in a specific machine-readable format. I've simplified it below:

`YYYY-MM-DDThh:mm:ss`

For example (local time):

`1985-11-03T17:19:10`

This means "November 3, 1985, at 10 seconds after 5:19 p.m. local time." The hours portion uses a 24-hour clock, hence **`17`** instead of **`05`** for 5 p.m. If you include a time, the seconds are optional. (You may also provide time with milliseconds in the format of **`hh:mm.sss`**. Note the period before the milliseconds.)

The format is a little different when representing a duration. There are a couple of syntax options, but this is the simplest to follow:

***n*h *n*m *n*s**

(Where ***n*** is the number of hours, minutes, and seconds, respectively.)

The last example in A shows it in action.

Global Dates and Times and Time Zone Offsets

If you'd like, you can represent your dates and times in a global context instead of a local one. (Or simply the time by omitting the date.) Add a **`Z`** at the end to mark the time zone as UTC (Coordinated Universal Time), the primary global time standard. (See https://en.wikipedia.org/wiki/Coordinated_Universal_Time.)

For example (global date and time in UTC):

`1985-11-03T17:19:10Z`

Or, you can specify a time-zone offset from UTC by omitting **`Z`** and preceding the offset with **`–`** (minus) or **`+`** (plus).

For example (global date and time with offset from UTC):

`1985-11-03T17:19:10-03:30`

This means "November 3, 1985, at 10 seconds after 5:19 p.m. Newfoundland Standard Time (NST)," because NST is minus three and a half hours from UTC. A list of time zones by UTC offsets is available at http://en.wikipedia.org/wiki/List_of_time_zones_by_UTC_offset.

Just as a reminder, if you do include **`datetime`**, it doesn't require the full complement of information I just described, as the examples in A show.

Explaining Abbreviations

Abbreviations abound, whether as Jr., M.D., or even good ol' HTML. You can use the **abbr** element to mark up abbreviations and explain their meaning (A through C). You don't have to wrap every abbreviation in **abbr**, only when you think it would be helpful for visitors to be given the expanded meaning.

To explain abbreviations:

1. Type **<abbr**.
2. Optionally, next type **title="*expansion*"**, where ***expansion*** is the words represented by the abbreviation.
3. Type **>**.
4. Then type the abbreviation itself.
5. Finally, finish up with **</abbr>**.
6. Optionally, type a space and **(*expansion*)**, where ***expansion*** is the words represented by the abbreviation.

TIP It's common practice to include an abbreviation's expansion (by way of a title or a parenthetical) only the first time it appears on a page.

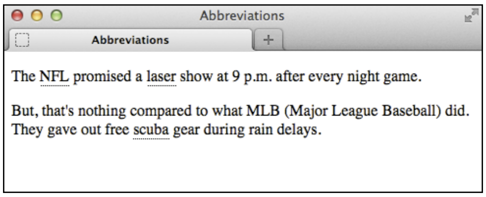
TIP A parenthetical abbreviation expansion is the most explicit way to describe an abbreviation, making it available to the widest set of visitors A. For instance, users on touchscreen devices like smartphones and tablets may not be able to hover on an abbr element to see a title tool tip. So if you provide an expansion, consider putting it in parentheses whenever possible.

TIP If you use an abbreviation in its plural form, make the expansion plural as well.

continues on next page

A Use the optional **title** attribute to provide the expanded version of an abbreviation. Alternatively, and arguably preferably, you could place the expansion in parentheses after the abbreviation. Or mix and match. Most people will be familiar with words like laser and scuba, so marking them up with **abbr** and providing titles isn't really necessary, but I've done it here for demonstration purposes.

```
...
<body>

<p>The <abbr title="National Football
→ League">NFL</abbr> promised a <abbr
→ title="light amplification by
→ stimulated emission of radiation">
→ laser</abbr> show at 9 p.m. after every
→ night game.</p>

<p>But, that's nothing compared to what
→ <abbr>MLB</abbr> (Major League
→ Baseball) did. They gave out free
→ <abbr title="self-contained underwater
→ breathing apparatus">scuba</abbr> gear
→ during rain delays.</p>

</body>
</html>
```



The NFL promised a laser show at 9 p.m. after every night game.

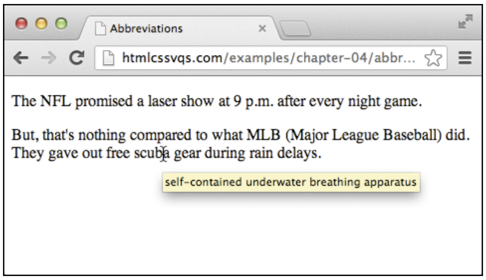
But, that's nothing compared to what MLB (Major League Baseball) did. They gave out free scuba gear during rain delays.

B When abbreviations have a **title** attribute, Firefox and Opera draw attention to them with dots underneath the text. You can instruct other browsers C to do the same with CSS; see the tips.

TIP As a visual cue to sighted users, Firefox and Opera display `abbr` with a dotted bottom border if it has a `title` B. If you'd like to replicate that effect in other browsers, add the following to your style sheet: `abbr[title] { border-bottom: 1px dotted #000; }`. Browsers provide the `title` attribute's contents as a tool tip C regardless of whether the `abbr` is styled with a border.

TIP If you don't see the dotted bottom border under your `abbr`, try adjusting the parent element's CSS `line-height` property (see Chapter 10).

TIP HTML had an `acronym` element before HTML5, but coders were often confused by the difference between an abbreviation and an acronym, so HTML5 eliminated the `acronym` element in favor of `abbr` for all instances.



The NFL promised a laser show at 9 p.m. after every night game.

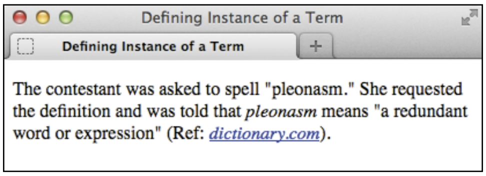
But, that's nothing compared to what MLB (Major League Baseball) did. They gave out free scuba gear during rain delays.

self-contained underwater breathing apparatus

C Browsers display the **`title`** of abbreviations as a tool tip when you hover the pointer over text marked up with **`abbr`**. (This figure also demonstrates an example of a browser—Chrome in this case—that doesn't style abbreviations with a **`title`** any differently than regular text by default.)

Ⓐ Note that although *pleonasm* appears twice in the example, **dfn** marks only the second one, because that's when I defined the term. Similarly, if I were to use pleonasm subsequently in the document, I wouldn't use **dfn**. Although browsers style **dfn** text differently than normal text Ⓑ, what's important is that the term is marked up differently. Also, you don't have to use the **cite** element each time you use **dfn**, just when you reference a source.

```
...
<body>

<p>The contestant was asked to spell
→ "pleonasm." She requested the definition
→ and was told that <dfn>pleonasm</dfn>
→ means "a redundant word or expression"
→ (Ref: <cite><a href="http://dictionary.
→ reference.com/browse/pleonasm" rel=
→ "external">dictionary.com</a></cite>).</p>

</body>
</html>
```



The contestant was asked to spell "pleonasm." She requested the definition and was told that *pleonasm* means "a redundant word or expression" (Ref: *dictionary.com*).

Ⓑ Typically, the **dfn** element renders in italics by default, as does **cite**.

Proximity of a Term and Its Definition

The location of a term marked with **dfn** relative to the location of its definition is important. HTML5 states, "The paragraph, description list group, or section that is the nearest ancestor of the **dfn** element must also contain the definition(s) for the term given by the **dfn** element." Simplified, this means that the **dfn** and its definition should be near each other, which makes sense. This is the case in both Ⓐ and the example given in the fourth tip; the **dfn** and its definition are in the same paragraph.

Defining a Term

In the print world, it's customary to differentiate a term visually when you define it. Typically, this is done with italics; subsequent uses of the term are not italicized.

In HTML, when you define a term, you differentiate it *semantically* with the **dfn** element. You wrap its tags only around the term you're defining, not around the definition Ⓐ. And just as in print convention, subsequent uses of the term are not marked with **dfn**, because you aren't defining them again. (HTML refers to the point where you define a term as the "defining instance of a term.")

To mark the defining instance of a term:

1. Type **<dfn>**.
2. Type the term you wish to define.
3. Type **</dfn>**.

TIP You can also use dfn in a description list (the dl element). See "Creating Description Lists" in Chapter 15.

TIP Use dfn only when defining a term, not simply because you want to italicize text. CSS allows you to style any text in italics (see "Creating Italics" in Chapter 10).

TIP dfn may also enclose another phrasing element, like abbr, when appropriate. For example, `<p>A <dfn><abbr title="Junior">Jr.</abbr></dfn> is a son with the same full name as his father.</p>`.

TIP If you use the optional title attribute on a dfn, it should have the same value as the dfn term. If you nest a single abbr in dfn and the dfn has no text of its own, the optional title should be on the abbr only, as in the previous tip.

Creating Superscripts and Subscripts

Letters or numbers that are raised or lowered slightly relative to the main body text are called superscripts and subscripts, respectively A. HTML includes elements for defining both kinds of text. Common uses for superscripts include marking trademark symbols, exponents, and footnotes B. Subscripts are common in chemical notation.

To create superscripts or subscripts:

1. Type **`<sub>`** to create a subscript or **`<sup>`** to create a superscript.
2. Type the characters or symbols that represent the subscript or superscript.
3. Type **`</sub>`** or **`</sup>`**, depending on what you used in step 1, to complete the element.

TIP Most browsers automatically reduce the font size of sub- or superscripted text by a few points.

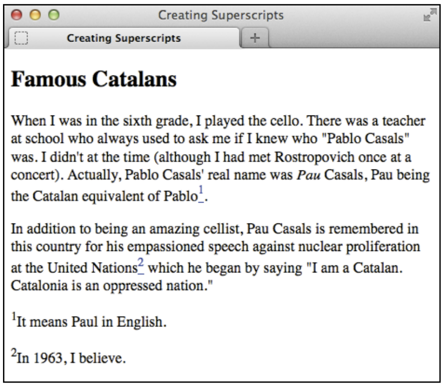
TIP Superscripts are the ideal way to mark up certain foreign-language abbreviations—such as M^{lle} for *Mademoiselle* in French or 3^{a} for tercera in Spanish—or to mark up numerics like 2nd and 5th.

TIP One proper use of subscripts is for writing out chemical molecules, such as $\text{H}_2\text{0}$. For example, `<p>I'm parched. Could I please have a glass of H<sub>2</sub>0?</p>`.

TIP Super- and subscripted characters gently spoil the even spacing between lines. In B, for example, notice that there is more space between lines 4 and 5 of the first paragraph and lines 2 and 3 of the second than between the other lines. CSS comes to the rescue, though; see the sidebar to learn how to fix this.

A One use of the **`sup`** element is to indicate footnotes. I placed the footnotes in a **`footer`** within the **`article`** rather than in the page at large because they are associated. I also linked each footnote number within the text to its footnote in the footer so visitors can access them more easily. Note, too, that the **`title`** attribute on the links provides another cue.

```
...
<body>

<article>
     <h1>Famous Catalans</h1>
     <p>... Actually, Pablo Casals' real
     → name was <i>Pau</i> Casals, Pau
     → being the Catalan equivalent of Pablo
     → <a href="#footnote-1" title="Read
     → footnote 1"><sup>1</sup></a>.</p>

     <p>... Pau Casals is remembered in this
     → country for his empassioned speech
     → against nuclear proliferation at the
     → United Nations <a href="#footnote-2"
     → title="Read footnote 2"><sup>2</sup>
     → </a> ...</p>

     <footer>
        <p id="footnote-1"><sup>1</sup>It
        → means Paul in English.</p>
        <p id="footnote-2"><sup>2</sup>In
        → 1963, I believe.</p>
     </footer>
</article>

</body>
</html>
```



Famous Catalans

When I was in the sixth grade, I played the cello. There was a teacher at school who always used to ask me if I knew who "Pablo Casals" was. I didn't at the time (although I had met Rostropovich once at a concert). Actually, Pablo Casals' real name was *Pau* Casals, Pau being the Catalan equivalent of Pablo[1].

In addition to being an amazing cellist, Pau Casals is remembered in this country for his empassioned speech against nuclear proliferation at the United Nations[2] which he began by saying "I am a Catalan. Catalonia is an oppressed nation."

[1]It means Paul in English.

[2]In 1963, I believe.

B The **`sup`** elements display higher than text in the same line. In the process, unfortunately, they change the spacing between lines (see the last tip).

Fixing the Spacing Between Lines When Using **sub** or **sup**

With a little bit of CSS, you can fix the line height discrepancies caused by the **sub** and **sup** elements. The code below comes from Nicolas Gallagher and Jonathan Neal's excellent **normalize.css** (http://necolas.github.com/normalize.css/). They didn't invent the method that follows; they borrowed it from https://gist.github.com/413930, which includes a full explanation of what this CSS does, so I encourage you to give it a look.

I also recommend checking out **normalize.css**, which you can use on your own projects. It helps you achieve a consistent baseline display of elements across browsers and is documented thoroughly (see "Resetting or Normalizing Default Styles" in Chapter 11).

```
/*
* Prevents sub and sup affecting line-height in all browsers
* gist.github.com/413930
*/
sub,
sup {
    font-size: 75%;
    line-height: 0;
    position: relative;
    vertical-align: baseline;
}
sup {
    top: -0.5em;
}
sub {
    bottom: -0.25em;
}
```

You may need to adjust this CSS a bit to level out the line heights, depending on your content's font size, but this should give you a very good start at the least. You'll learn about creating style sheets and how to add this CSS to your site in Chapter 8.

Adding Author Contact Information

You might think the **address** element is for marking up a postal or street address, but it isn't (except for one circumstance; see the first tip). In fact, there isn't an HTML element explicitly designed for that purpose.

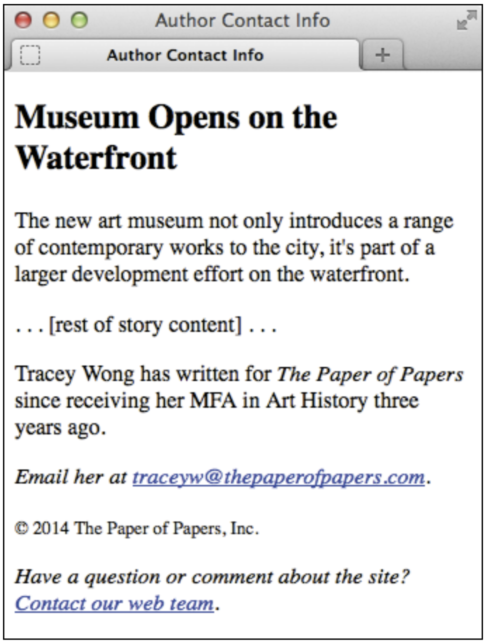
Instead, **address** defines the contact information for the author, people, or organization responsible for either a part of a webpage (such as a news article, product review, or report) or a whole page (A and B). Which of those is true depends on where **address** appears. The first step describes each scenario.

To provide the author's contact information:

1. If you want to provide author contact information for an **article**, place the cursor within that **article** (see the first instance in A). Alternatively, place the cursor within the **body** (or, more commonly, the page-level **footer**) if you want to provide author contact information for the page at large (see the second instance in A).
2. Type **<address>**.
3. Type the author's email address, a link to a page with contact information, and so on.
4. Type **</address>**.

A This page has two **address** elements: one for the **article**'s author and the other in a page-level **footer** for the people who maintain the whole page. Note that the **address** for the **article** contains only contact information. Although the background information about Tracey Wong is also in the **article**'s **footer**, it's outside the **address** element.

```
...
<body>
<main role="main">
<article>
     <h1>Museum Opens on the Waterfront</h1>
     <p>The new art museum not only
     → introduces a range of contemporary
     → works to the city, it's part of
     → larger development effort on the
     → waterfront.</p>

     ... [rest of story content] ...

     <!-- the article's footer with address
     → information for the article -->
     <footer>
        <p>Tracey Wong has written for
        → <cite>The Paper of Papers</cite>
        → since receiving her MFA in art
        → history three years ago.</p>
        <address>
        Email her at <a href="mailto:
        → traceyw@thepaperofpapers.com">
        → traceyw@thepaperofpapers.com
        → </a>.
        </address>
     </footer>
</article>
</main>

<!-- the page's footer with address
→ information for the whole page -->
<footer role="contentinfo">
     <p><small>&copy; 2014 The Paper of
     → Papers, Inc.</small></p>
     <address>
     Have a question or comment about the
     → site? <a href="site-feedback.html">
     → Contact our web team</a>.
     </address>
</footer>
</body>
</html>
```



Museum Opens on the Waterfront

The new art museum not only introduces a range of contemporary works to the city, it's part of a larger development effort on the waterfront.

. . . [rest of story content] . . .

Tracey Wong has written for *The Paper of Papers* since receiving her MFA in Art History three years ago.

Email her at traceyw@thepaperofpapers.com.

© 2014 The Paper of Papers, Inc.

Have a question or comment about the site? Contact our web team.

B The **address** element renders in italics by default. (The text "The Paper of Papers" is also italicized, but it is enclosed in the **cite** element, covered in "Indicating a Citation or Reference" in this chapter.)

TIP Most of the time, contact information takes the form of the author's email address or a link to a page with more contact information. The contact information could very well be the author's postal address, in which case marking it up with address would be valid. But if you're creating the Contact Us page for your business and want to include your office locations, it would be incorrect to code those with address. The example in "Creating a Line Break" shows one way to mark up a postal or street address.

TIP The address element pertains to the nearest article it is contained in, or to the page's body if address isn't nested within an article. It's customary to place address in a footer element when noting author contact information for the page at large, like the second instance of address in A.

TIP An address in an article provides contact information for the author of that article A, not for any articles nested within that article, such as user comments.

TIP The address element may contain only author contact information, not anything else such as the document or article's last modified date A. Additionally, HTML5 forbids nesting any of the following elements inside address: h1–h6, article, address, aside, footer, header, hgroup, nav, and section.

TIP See Chapter 3 to learn more about the article and footer elements.

Noting Edits and Inaccurate Text

Sometimes you may want to indicate content edits that have occurred since the previous version of your page. There are two elements for noting edits: the **ins** element represents content that has been added, and the **del** element marks content that has been removed (A through D). You may use them together or individually.

Meanwhile, the **s** element notes content that is no longer accurate or relevant (it's not for edits) (E and F).

To mark newly inserted text:

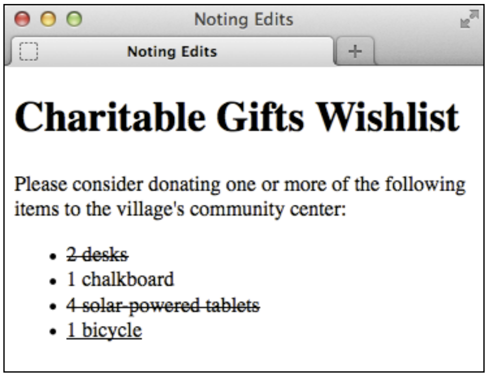
1. Type **<ins>**.
2. Type the new content.
3. Type **</ins>**.

To mark deleted text:

1. Place the cursor before the text or element you wish to mark as deleted.
2. Type **<del>**.
3. Place the cursor after the text or element you wish to mark as deleted.
4. Type **</del>**.

A One item (the bicycle) has been added to this gift list since it was previously published, and purchased items have been removed, as noted by the **del** elements. You are not required to use **del** each time you use **ins**, or vice versa. Browsers differentiate the contents of each element visually by default B.

```
...
<body>

<h1>Charitable Gifts Wishlist</h1>

<p>Please consider donating one or more
→ of the following items to the village's
→ community center:</p>

<ul>
     <li><del>2 desks</del></li>
     <li>1 chalkboard</li>
     <li><del>4 solar-powered tablets
     → </del></li>
     <li><ins>1 bicycle</ins></li>
</ul>

</body>
</html>
```



Charitable Gifts Wishlist

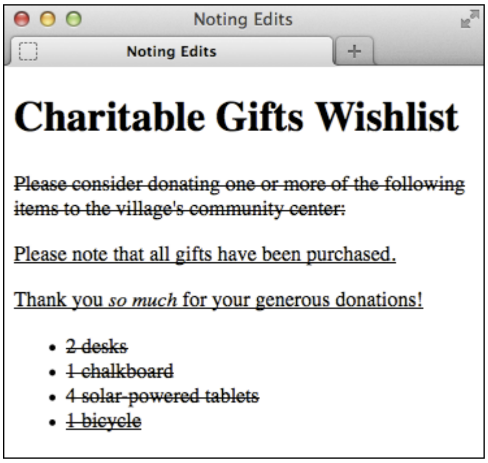
Please consider donating one or more of the following items to the village's community center:

- ~~2 desks~~
- 1 chalkboard
- ~~4 solar-powered tablets~~
- 1 bicycle

B Browsers typically display a line through deleted text, and they typically underline inserted text. You can change these treatments with CSS.

C Both **del** and **ins** are rare in that they can surround both phrasing content ("inline" content, in pre-HTML5 parlance) and blocks of content like entire paragraphs or lists, as shown here.

```
...
<body>

<h1>Charitable Gifts Wishlist</h1>
<del>
     <p>Please consider donating one or more of the following items to the village's community
     ➝ center:</p>
</del>

<ins>
     <p>Please note that all gifts have been purchased.</p>
     <p>Thank you <em>so much</em> for your generous donations!</p>
</ins>

<del>
     <ul>
        <li><del>2 desks</del></li>
        <li>1 chalkboard</li>
        <li><del>4 solar-powered tablets</del></li>
        <li><ins>1 bicycle</ins></li>
     </ul>
</del>

</body>
</html>
```



Charitable Gifts Wishlist

~~Please consider donating one or more of the following items to the village's community center:~~

Please note that all gifts have been purchased.

Thank you *so much* for your generous donations!

- ~~2 desks~~
- ~~1 chalkboard~~
- ~~4 solar-powered tablets~~
- ~~1 bicycle~~

D Just as before, browsers indicate which content has been deleted or inserted.

To mark text that is no longer accurate or relevant:

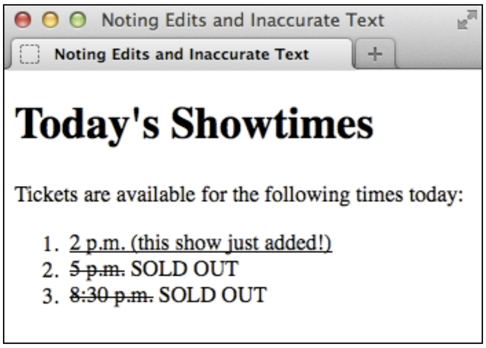
1. Place the cursor before the text you wish to mark as no longer accurate or relevant.
2. Type `<s>`.
3. Place the cursor after the text you wish to mark.
4. Type `</s>`.

TIP Both `del` and `ins` support two attributes: `cite` and `datetime`. The `cite` attribute (not the same as the `cite` *element*) is for providing a URL to a source that explains why an edit was made. For example, `<ins cite="http://www.movienews.com/ticket-demand-high.html">2 p.m. (this show just added!)</ins>`. Use the `datetime` attribute to indicate the time of the edit. (See "Specifying Time" to learn about `datetime`'s acceptable format.) Browsers don't display the values you assign to either of these attributes, so their use isn't widespread with `del` and `ins`, but feel free to include them to add context to your content. The values could be extracted with JavaScript or a program that parses through your page.

TIP Use `del` and `ins` anytime you want to inform your visitors of your content's evolution. For instance, you'll often see them used in a web development or design tutorial to indicate information that was learned since it was initially posted, while maintaining the copy as it originally stood for completeness. The same is true of blogs, news sites, and so on.

E This example shows an ordered list (the **`ol`** element) of show times. The time slots for which ticket availability is no longer relevant have been marked with the **`s`** element. You can use **`s`** around any phrases, not just around text within list items (**`li`** elements), but you cannot use it around a whole paragraph or other "block-level" element like you can with **`del`** and **`ins`**.

```
...
<body>

<h1>Today's Showtimes</h1>
<p>Tickets are available for the following
→ times today:</p>

<ol>
     <li><ins>2 p.m. (this show just added!)
     → </ins></li>
     <li><s>5 p.m.</s> SOLD OUT</li>
     <li><s>8:30 p.m.</s> SOLD OUT</li>
</ol>

</body>
</html>
```


Noting Edits and Inaccurate Text
Noting Edits and Inaccurate Text

Today's Showtimes

Tickets are available for the following times today:

1. 2 p.m. (this show just added!)
2. ~~5 p.m.~~ SOLD OUT
3. ~~8:30 p.m.~~ SOLD OUT

F The **`s`** element renders as a strikethrough by default in browsers.

TIP Text marked with the `ins` element is generally underlined by default B. Since links are often underlined as well (if not in your site, then in many others), this may be confusing to visitors. You may want to use CSS to change how inserted passages (or links) are displayed (see Chapter 10).

TIP Text marked with the `del` element is generally struck out B. Why not just erase it and be done with it? It depends on whether you think it's important to indicate what's been removed. Striking out content makes it easy for sighted users to know what has changed. (Also, screen readers could announce the content as having been removed, but their support for doing so has historically been lacking.)

TIP Only use `del`, `ins`, and `s` for their semantic value. If you wish to underline or strike out text purely for cosmetic reasons, you can do so with CSS (see "Decorating Text" in Chapter 10).

TIP HTML5 notes that "The `s` element is not appropriate when indicating document edits; to mark a span of text as having been removed from a document, use the `del` element." You may find the distinction a little subtle at times. It's up to you to decide which is the appropriate semantic choice for your content.

Marking Up Code

If your content contains code samples or file names, the **code** element is for you (A and B).

The examples show **code** used in a sentence. To show a standalone block of code (outside of a sentence), wrap the **code** element with a **pre** element to maintain its formatting (see "Using Preformatted Text" for an example).

To mark up code or a file name:

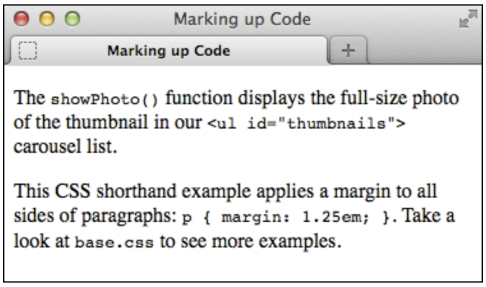
1. Type **<code>**.
2. Type the code or file name.
3. Type **</code>**.

TIP You can change the default monospaced font applied to code B with CSS (see Chapter 10).

TIP See "A Webpage's Text Content" in Chapter 1 regarding character entities A.

A The **code** element indicates that the text is code or a file name. If your code needs to display **<** or **>** signs, use the **&lt;** and **&gt;** character entities, respectively (see the last tip). Here, the second instance of **code** demonstrates this. If you were to use **<** and **>**, the browser would treat your code as an HTML element, not as text to display.

```
...
<body>

<p>The <code>showPhoto()</code> function
→ displays the full-size photo of the
→ thumbnail in our <code>&lt;ul id=
→ "thumbnail"&gt;</code> carousel list.</p>

<p>This CSS shorthand example applies a
→ margin to all sides of paragraphs: <code>p
→ { margin: 1.25em; }</code>. Take a look
→ at <code>base.css</code> to see more
→ examples.</p>

</body>
</html>
```



The showPhoto() function displays the full-size photo of the thumbnail in our <ul id="thumbnails"> carousel list.

This CSS shorthand example applies a margin to all sides of paragraphs: p { margin: 1.25em; }. Take a look at base.css to see more examples.

B The **code** element's text even looks like code because of the monospaced default font.

Other Computer and Related Elements: kbd, samp, and var

The **kbd**, **samp**, and **var** elements see infrequent use, but you may have occasion to take advantage of them in your content.

The kbd Element

Use **kbd** to mark up user input instructions.

```
<p>To log into the demo:</p>
<ol>
    <li>Type <kbd>tryDemo</kbd> in the User Name field</li>
    <li><kbd>TAB</kbd> to the Password field and type <kbd>demoPass</kbd></li>
    <li>Hit <kbd>RETURN</kbd> or <kbd>ENTER</kbd></li>
</ol>
```

Like **code**, **kbd** renders as a monospaced font by default.

The samp Element

The **samp** element indicates sample output from a program or system.

```
<p>Once the payment went through, the site returned a message reading,
➝<samp>Thanks for your order!</samp></p>
```

samp also renders as a monospaced font by default.

The var Element

The **var** element represents a variable or placeholder value.

```
<p>Einstein is best known for <var>E</var>=<var>m</var><var>c</var><sup>2</sup>.</p>
```

var can also be a placeholder value in content, like a Mad Libs sheet in which you'd put `<var>adjective</var>`, `<var>verb</var>`, and so on.

var renders in italics by default.

Note that you can use **math** and other **MathML** elements in your HTML5 pages for advanced math-related markup. See http://dev.w3.org/html5/spec-author-view/mathml.html for more information.

Using Preformatted Text

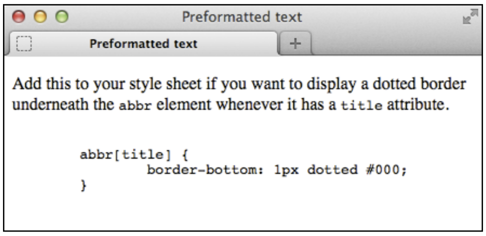
Usually, browsers collapse all extra returns and spaces and automatically break lines of text according to the width of the browser window. Preformatted text lets you maintain and display the original line breaks and spacing that you've inserted in the text. It is ideal for computer code examples **A**, though you can also use it for text (hello, ASCII art!).

To use preformatted text:

1. Type `<pre>`.
2. Type or paste the text that you wish to display as is, with all the necessary spaces, returns, and line breaks. Unless it is code, do not mark up the text with any HTML, such as **p** elements.
3. Type `</pre>`.

A The **pre** element is ideal for text that contains important spaces and line breaks, like the bit of CSS code shown here. Note, too, the use of the **code** element to mark up pieces of code or code-related text outside of **pre** (see "Marking Up Code" for more details).

```
...
<body>

<p>Add this to your style sheet if you want
→ to display a dotted border underneath the
→ <code>abbr</code> element whenever it has
→ a <code>title</code> attribute.</p>

<pre>
     <code>
     abbr[title] {
        border-bottom: 1px dotted #000;
     }
     </code>
</pre>

</body>
</html>
```


Preformatted text

Add this to your style sheet if you want to display a dotted border underneath the `abbr` element whenever it has a `title` attribute.

```
        abbr[title] {
                border-bottom: 1px dotted #000;
        }
```

B Notice that the indentation and line breaks are maintained in the **pre** content.

Presentation Considerations with `pre`

Be aware that browsers typically disable automatic word wrapping of content inside a **`pre`**, so if the text is too wide, it might affect your layout or force a horizontal scrollbar. The following CSS rule enables wrapping within **`pre`** in many browsers, but not in Internet Explorer 7 and below. (In the vast majority of cases, those versions are too old to worry about.)

```
pre {
    white-space: pre-wrap;
}
```

On a related note, in most cases I don't recommend you use the **`white-space: pre;`** CSS declaration on an element such as **`div`** as a substitute for **`pre`**. Whitespace can be crucial to the semantics of content, especially code, and only **`pre`** always preserves it. (Also, if the user has disabled CSS in his or her browser, the formatting will be lost.)

Please see CSS coverage beginning in Chapter 7. Text formatting, in particular, is discussed in Chapter 10.

TIP Preformatted text is typically displayed with a monospaced font like Courier or Courier New B. You can use CSS to change the font, if you like (see Chapter 10).

TIP If what you want to display—such as a code sample in a tutorial—contains HTML elements, you'll have to substitute each < and > around the element name with their appropriate character entities: `&lt;` and `&gt;` respectively (see "Marking Up Code" for an example). Otherwise the browser may try to display those elements.

TIP Be sure to validate your pages to see if you've nested HTML elements in `pre` when you shouldn't have (see "Validating Your Code" in Chapter 20).

TIP The `pre` element isn't a shortcut for avoiding marking up your content with proper semantics and then styling the way it looks with CSS. For instance, if you want to post a news article you wrote in a word processor, don't simply copy and paste it into a `pre` because you like the spacing the way it is. Instead, wrap your content in `p` (and other relevant text elements) and write CSS to control the layout as desired.

TIP `pre`, like a paragraph, always displays on a new line by default B.

Highlighting Text

We've all used a highlighter pen at some point or another. Maybe it was when studying for an exam or going through a contract. Whatever the case, you used the highlighter to mark key words or phrases.

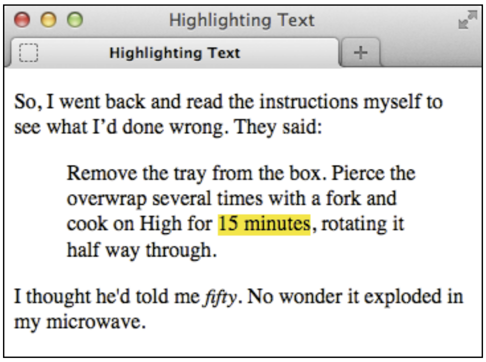
HTML5 replicates this with the new **`mark`** element. Think of **`mark`** as a semantic version of a highlighter pen. In other words, what's important is that you're noting certain words; how they appear isn't important. Style its text with CSS as you please (or not at all), but use **`mark`** only when it's pertinent to do so.

No matter when you use **`mark`**, it's to draw the reader's attention to a particular text segment. Here are some use cases for it:

- To highlight a search term when it appears in a search results page or an article. When people talk about **`mark`**, this is the most common context. Suppose you used a site's search feature to look for "solar panels." The search results or each resulting article could use **`<mark>solar panels</mark>`** to highlight the term throughout the text.
- To call attention to part of a quote that wasn't highlighted by the author in its original form (A and B). This is akin to the real-world task of highlighting a textbook or contract.
- To draw attention to a code fragment (C and D).

A Although **`mark`** may see its most widespread use in search results, here's another valid use of it. The phrase "15 minutes" was not highlighted in the instructions on the packaging. Instead, the author of this HTML used **`mark`** to call out the phrase as part of the story.

```
...
<body>

<p>So, I went back and read the instructions
→ myself to see what I'd done wrong. They
→ said:</p>

<blockquote>
    <p>Remove the tray from the box. Pierce
    → the overwrap several times with a
    → fork and cook on High for <mark>15
    → minutes</mark>, rotating it half way
    → through.</p>
</blockquote>

<p>I thought he'd told me <em>fifty</em>. No
→ wonder it exploded in my microwave.</p>

</body>
</html>
```



So, I went back and read the instructions myself to see what I'd done wrong. They said:

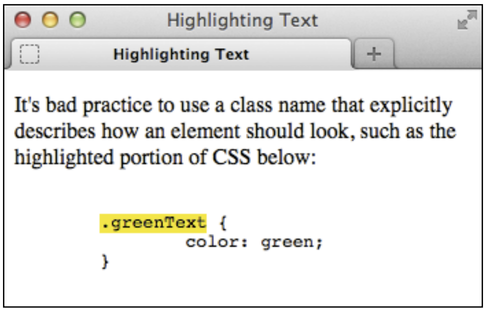
> Remove the tray from the box. Pierce the overwrap several times with a fork and cook on High for 15 minutes, rotating it half way through.

I thought he'd told me *fifty*. No wonder it exploded in my microwave.

B Browsers with native support of the **`mark`** element display a yellow background behind the text by default. Older browsers don't, but you can tell them to do so with a simple rule in your style sheet (see the tips).

C This example uses **mark** to draw attention to a code segment.

```
...
<body>

<p>It's usually bad practice to use a class
→ name that explicitly describes how an
→ element should look, such as the
→ highlighted portion of CSS below:</p>

<pre>
     <code>
        <mark>.greenText</mark> {
           color: green;
        }
     </code>
</pre>

</body>
</html>
```



It's bad practice to use a class name that explicitly describes how an element should look, such as the highlighted portion of CSS below:

```
        .greenText {
                color: green;
        }
```

D The code noted with **mark** is called out.

To highlight text:

1. Type **`<mark>`**.
2. Type the word or words to which you want to call attention.
3. Type **`</mark>`**.

TIP The `mark` element is not the same as either `em` (which represents stress emphasis) or `strong` (which represents importance). Both are covered earlier in this chapter.

TIP Since `mark` is new in HTML5, older browsers don't render a background color by default. You can instruct them to do so by adding `mark { background-color: yellow; }` to your style sheet.

TIP Be sure not to use `mark` simply to give text a background color or other visual treatment. If all you're looking for is a means to style text and there's no proper semantic HTML element to contain it, use the `span` element (covered later in this chapter), perhaps with a `class` assigned to it, and style it with CSS.

Creating a Line Break

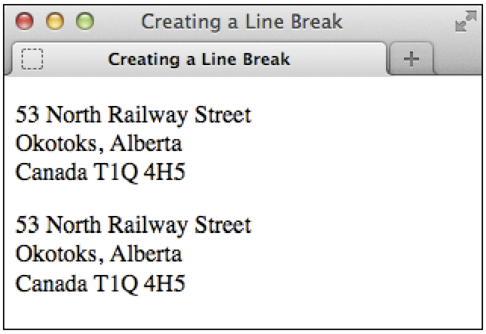
Browsers automatically wrap text according to the width of the block or window that contains content. It's best to let content flow like this in most cases, but sometimes you'll want to force a line break manually. You achieve this with the **br** element.

Using **br** is a last resort tactic because it mixes presentation with your HTML instead of leaving all display control to your CSS. For instance, never use **br** to simulate spacing between paragraphs. Instead, mark up the two paragraphs with **p** elements and define the spacing between the two with the CSS **margin** property (see the second tip).

So when might **br** be OK? Well, the **br** element is suitable for creating line breaks in poems, in a street address (A and B), and occasionally in other short lines of text that should appear one after another.

A The same address appears twice, but I coded them a little differently for demonstration purposes. Remember that the returns in your code are always ignored, so both paragraphs shown display the same way B.

```
...
<body>

<p>53 North Railway Street<br />
Okotoks, Alberta<br />
Canada T1Q 4H5</p>

<p>53 North Railway Street <br />Okotoks,
→ Alberta <br />Canada T1Q 4H5</p>

</body>
</html>
```



53 North Railway Street
Okotoks, Alberta
Canada T1Q 4H5

53 North Railway Street
Okotoks, Alberta
Canada T1Q 4H5

B Each **br** element forces the subsequent content to a new line. Without them, the entire address would display on one line, unless the browser were narrow enough to force wrapping.

To insert a line break:

Type **`<br />`** (or **`<br>`**) where the line break should occur. There is no separate end **`br`** tag because it's what's known as an *empty* (or *void*) *element*; it lacks content.

TIP **Typing `br` as either `<br />` or `<br>` is perfectly valid in HTML5.**

TIP **CSS allows you to control the space between lines in a paragraph (see "Setting the Line Height" in Chapter 10) and between the paragraphs themselves (see "Setting the Margins Around an Element" in Chapter 11).**

TIP **The hCard microformat (http://microformats.org/wiki/hcard) is for representing people, companies, organizations, and places in a semantic manner that's human- and machine-readable. You could use it to represent a street address instead of using the provided example A.**

Creating Spans

The **span** element, like **div**, has absolutely no semantic meaning. The difference is that **span** is appropriate around a word or phrase only, whereas **div** is for blocks of content (see “Creating Generic Containers” in Chapter 3).

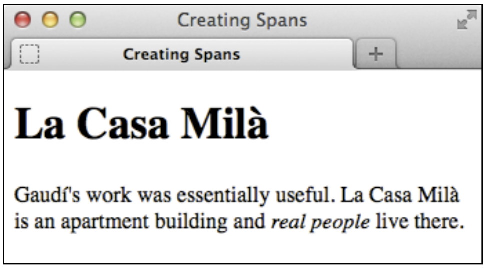
The **span** element is useful when you want to apply any of the following to a snippet of content for which HTML doesn’t provide an appropriate semantic element:

- Attributes, like **class**, **dir**, **id**, **lang**, **title**, and more (A and B)
- Styling with CSS
- Behavior with JavaScript

Because **span** has no semantic meaning, use it as a last resort when no other element will do.

A In this case, I want to specify the language of a portion of text, but there isn’t an HTML element whose semantics are a fit for “La Casa Milà” in the context of a sentence. The **h1** that contains “La Casa Milà” before the paragraph is appropriate semantically because the text is the heading for the content that follows. So for the heading, I simply added the **lang** attribute to the **h1** rather than wrap a **span** around the heading text unnecessarily for that purpose. (The **lang** attribute allows you to declare the language of the element’s text.)

```
...
<body>

<h1 lang="es">La Casa Milà</h1>

<p>Gaudí's work was essentially useful.
→ <span lang="es">La Casa Milà</span> is
→ an apartment building and <em>real people
→ </em> live there.</p>

</body>
</html>
```



La Casa Milà

Gaudí's work was essentially useful. La Casa Milà is an apartment building and *real people* live there.

B The **span** element has no default styling.

To add a span:

1. Type **`<span`**.
2. If desired, type **`id="`*`name`*`"`**, where ***name*** uniquely identifies the spanned content.
3. If desired, type **`class="`*`name`*`"`**, where ***name*** is the name of the class that the spanned content belongs to.
4. If desired, type other attributes (such as **`dir`**, **`lang`**, or **`title`**) and their values.
5. Type **`>`** to complete the start **`span`** tag.
6. Create the content you wish to contain in the **`span`**.
7. Type **`</span>`**.

TIP A `span` doesn't have default formatting B, but just as with other HTML elements, you can apply your own with CSS.

TIP You may apply both a `class` and `id` attribute to the same `span` element, although it's more common to apply one or the other, if at all. The principal difference is that `class` is for a group of elements, whereas `id` is for identifying individual, unique elements on a page.

TIP Microformats often use `span` to attach semantic class names to content as a way of filling the gaps where HTML doesn't provide a suitable semantic element. You can learn more about them at http://microformats.org.

Other Elements

This section covers other elements that you can include within your text, but which typically have fewer occasions to be used or have limited browser support (or both).

The u element

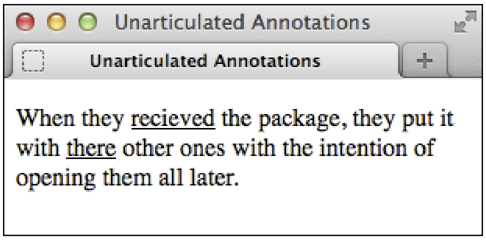
Like **b**, **i**, **s**, and **small**, the **u** element has been redefined in HTML5 to disassociate it from its past as a non-semantic, presentational element. In those days, the **u** element was for underlining text. Now, it's for unarticulated annotations (sounds a little befuddling, I know). HTML5 defines it thus:

> The **u** element represents a span of text with an unarticulated, though explicitly rendered, non-textual annotation, such as labeling the text as being a proper name in Chinese text (a Chinese proper name mark), or labeling the text as being misspelt.

Here is an example of how you could use **u** to note misspelled words:

```
<p>When they <u class="spelling">
➝recieved</u> the package, they put
➝it with <u class="spelling">there
➝</u> other ones with the intention
➝of opening them all later.</p>
```

The **class** is entirely optional, and its value (which can be whatever you'd like) doesn't render with the content to explicitly indicate a spelling error. But you could use it to style misspelled words differently (though **u** still renders as underlined text by default). Or you could add a **title** attribute with a note such as "[sic]"—a convention in some languages to indicate a misspelling.



When they recieved the package, they put it with there other ones with the intention of opening them all later.

A Like links, **u** elements are underlined by default, which can cause confusion unless you change one or both with CSS.

Use **u** only when an element like **cite**, **em**, or **mark** doesn't fit your desired semantics. Also, it's best to change its styling if **u** text will be confused with linked text, which is also underlined by default A.

The wbr element

HTML5 introduces a cousin of **br** named **wbr**. It represents "a line break opportunity." Use it in between words or letters in a long, unbroken phrase (or, say, a URL) to indicate where it could wrap if necessary to fit the text in the available space in a readable fashion. So unlike **br**, **wbr** doesn't force a wrap; it just lets the browser know where it *can* force a line break if needed.

Here are a couple of examples:

```
<p>They liked to say, "FriendlyFleas
→ andFireFlies<wbr /> FriendlyFleasa
→ ndFireFlies<wbr />FriendlyFleasand
→ FireFlies<wbr />" as fast as they
→ could over and over.</p>
```

```
<p>His favorite site is this<wbr />
→ is<wbr />a<wbr />really<wbr />
→ really<wbr />longurl.com.</p>
```

You can type **wbr** as either **<wbr />** or **<wbr>**. As you might have guessed, you won't find many occasions to use **wbr**. Additionally, browser support is inconsistent as of this writing. Although **wbr** works in current versions of Chrome and Firefox, Internet Explorer and Opera simply ignore it.

The `ruby`, `rp`, and `rt` elements

A *ruby annotation* is a convention in East Asian languages, such as Chinese and Japanese, and is typically used to show the pronunciation of lesser-known characters. These small annotative characters appear either above or to the right of the characters they annotate. They are often called simply *ruby* or *rubi*, and the Japanese ruby characters are known as *furigana*.

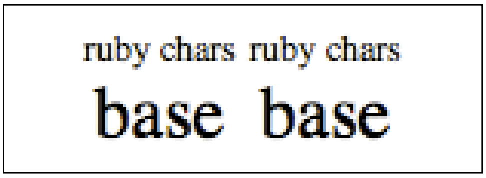
The **`ruby`** element, as well as its **`rt`** and **`rp`** child elements, is HTML5's mechanism for adding them to your content. **`rt`** specifies the ruby characters that annotate the base characters. The optional **`rp`** element allows you to display parentheses around the ruby text in browsers that don't support **`ruby`**.

The following example demonstrates this structure with English placeholder copy to help you understand the arrangement of information both in the code and in supporting B and non-supporting C browsers. The area for ruby text is highlighted:

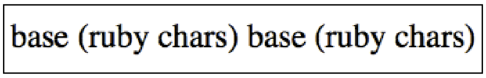
```
<ruby>
    base <rp>(</rp><rt>ruby chars
    → </rt><rp>)</rp>
    base <rp>(</rp><rt>ruby chars
    → </rt><rp>)</rp>
</ruby>
```

Now, a real-world example with the two Chinese base characters for "Beijing," and their accompanying ruby characters D:

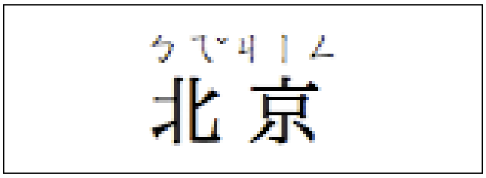
```
<ruby>
    北 <rp>(</rp><rt>ㄅㄟˇ</rt><rp>)
    → </rp>
    京 <rp>(</rp><rt>ㄐㄧㄥ</rt><rp>)
    → </rp>
</ruby>
```


ruby chars ruby chars
base base

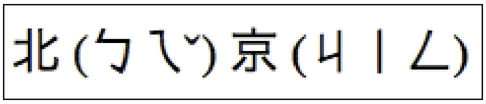
B A supporting browser will display the ruby text above the base (or possibly on the side) without parentheses because it ignores the **`rp`** elements.


base (ruby chars) base (ruby chars)

C A non-supporting browser displays the **`rt`** content in parentheses in the normal flow of content.


ㄅㄟˇ ㄐㄧㄥ
北 京

D Now, the ruby markup for "Beijing" as seen in a supporting browser.


北(ㄅㄟˇ)京(ㄐㄧㄥ)

E In a non-supporting browser, the content could be harder to understand without the parentheses.

You can see how important the parentheses are for browsers that don't support **`ruby`** **E**. Without them, the base and ruby text would run together, clouding the message.

TIP At the time of this writing, Firefox and Opera lack basic `ruby` support (all the more reason to use `rp` in your markup). The Firefox add-on HTML Ruby (https://addons.mozilla.org/en-US/firefox/addon/html-ruby/) provides support for Firefox in the meantime.

TIP You can learn more about ruby characters at http://en.wikipedia.org/wiki/Ruby_character.

The bdi and bdo elements

If your HTML pages ever mix left-to-right characters (like Latin characters in most languages) and right-to-left characters (like characters in Arabic or Hebrew), the **`bdi`** and **`bdo`** elements may be of interest.

But first, a little backstory. The base directionality of your content defaults to left-to-right unless you set the **`dir`** attribute on the **`html`** element to **`rtl`**. For instance, **`<html dir="rtl" lang="he">`** specifies that the base directionality of your content is right-to-left and that the base language is Hebrew.

Just as I've done with **`lang`** in several examples throughout the book, you may also set **`dir`** on elements within the page when the content deviates from the page's base setting. So if the base were set to English (**`<html lang="en">`**) and you wanted to include a paragraph in Hebrew, you'd mark it up as **`<p dir="rtl" lang="he">...</p>`**.

With those settings in place, the content will display in the desired directionality most of the time; Unicode's bidirectional ("bidi") algorithm takes care of figuring it out.

The **`bdo`** ("bidirectional override") element is for those occasions when the algorithm *doesn't* display the content as intended, and you need to override it. Typically, that's the case when the content in the HTML source is in visual order instead of logical order.

Visual order is just what it sounds like—the HTML source code content is in the same order in which you want it displayed. *Logical order* is the opposite for a right-to-left language like Hebrew; the first character going right to left is typed first, then the second character (in other words, the one to the left of it), and so on.

In line with best practices, Unicode expects bidirectional text in logical order. So if it's visual instead, the algorithm will still reverse the characters, displaying them opposite of what is intended. If you aren't able to change the text in the HTML source to logical order (for instance, maybe it's coming from a database or a feed), your only recourse is to wrap it in a **`bdo`**.

To use **`bdo`**, you must include the **`dir`** attribute and set it to either **`ltr`** (left-to-right) or **`rtl`** (right-to-left) to specify the direction you want. Continuing our earlier example of a Hebrew paragraph within an otherwise English page, you would type **`<p lang="he"><bdo dir="rtl">...</bdo></p>`**. The **`bdo`** element is appropriate for phrases or sentences within a paragraph. You wouldn't wrap it around several paragraphs.

The **bdi** element, new in HTML5, is for cases when the content's directionality is unknown. You don't have to include the **dir** attribute, because it's set to auto by default. HTML5 provides the following example, which I've modified slightly:

> This element is especially useful when embedding user-generated content with an unknown directionality.

In this example, usernames are shown along with the number of posts that the user has submitted. If the **bdi** element were not used, the username of the Arabic user would end up confusing the text (the bidirectional algorithm would put the colon and the number "3" next to the word "User" rather than next to the word "posts").

```
<ul>
    <li>User <bdi>jcranmer</bdi>:
    → 12 posts.</li>
    <li>User <bdi>hober</bdi>:
    → 5 posts.</li>
    <li>User <bdi>إيان</bdi>:
    → 3 posts.</li>
</ul>
```

TIP If you want to learn more on the subject of incorporating right-to-left languages, I recommend reading the W3C's article "Creating HTML Pages in Arabic, Hebrew, and Other Right-to-Left Scripts" (www.w3.org/International/tutorials/bidi-xhtml/).

The `meter` element

The **`meter`** element is another that is new thanks to HTML5. At first glance, it seems very similar to the **`progress`** element, covered next, which is for indicating "the completion progress of a task" (to quote the spec).

In contrast, you can use **`meter`** to indicate a fractional value or a measurement within a known range. In plain language, it's the type of gauge you use for the likes of voting results (for example, "30% Smith, 37% Garcia, 33% Hawkins"), the number of tickets sold (for example, "811 out of 850"), a numerical test grade (for example, "91 out of 100"), and disk usage (for example, "74 GB out of 256 GB").

HTML5 suggests (but doesn't require) that browsers could render a **`meter`** not unlike a thermometer on its side—a horizontal bar with the measured value colored differently than the maximum value (unless they're the same, of course). Firefox, one of the browsers that supports **`meter`** so far, does just that F. For non-supporting browsers, you can style **`meter`** to some extent with CSS or enhance it further with JavaScript.

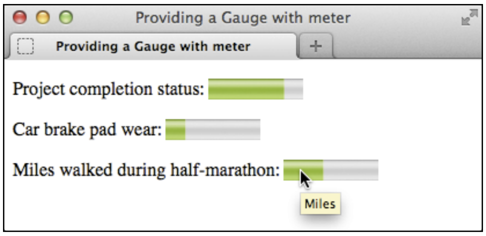
Although it's not required, it's best to include text inside **`meter`** that reflects the current measurement for non-supporting browsers to display G.

Here are some **`meter`** examples (as seen in F and G):

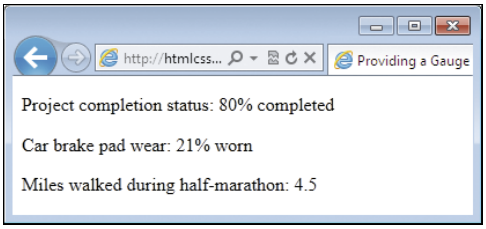
```
<p>Project completion status: <meter
→ value="0.80">80% completed</meter>
→ </p>
```

```
<p>Car brake pad wear: <meter low=
→ "0.25" high="0.75" optimum="0"
→ value="0.21">21% worn</meter></p>
```

```
<p>Miles walked during half-marathon:
→ <meter min="0" max="13.1" value="5.5"
→ title="Miles">4.5</meter></p>
```


Providing a Gauge with meter
Providing a Gauge with meter
Project completion status:
Car brake pad wear:
Miles walked during half-marathon:
Miles

F A browser, like Firefox, that supports **`meter`** displays the gauge automatically, coloring it based on the attribute values. It doesn't display the text in between **`<meter>`** and **`</meter>`**. As seen in the last example, if you include **`title`** text, it displays when you hover over the meter.



Project completion status: 80% completed

Car brake pad wear: 21% worn

Miles walked during half-marathon: 4.5

G IE9 doesn't support **`meter`**, so instead of a colored bar, it displays the text content inside the **`meter`** element. You can change the look with CSS.

The **`meter`** element doesn't have defined units of measure, but you can use the **`title`** attribute to specify text of your choosing, as in the last example. As is usual with **`title`** text, browsers display it as a tooltip F.

TIP `meter` supports several attributes. The `value` attribute is the only one that's required. The `min` and `max` attributes default to 0 and 1.0, respectively, if omitted. The `low`, `high`, and `optimum` attributes work together to split the range into low, medium, and high segments. The number assigned to `optimum` indicates the optimum position within the range, such as "0 brake pad wear" in one of the examples. Set `optimum` in between if neither a low nor a high value is optimal.

TIP At the time of this writing, browser support of `meter` is still evolving: It's not supported by Internet Explorer, mobile Safari (iOS devices), or Android's browser. This partially explains why you don't yet see it much in the wild. Feel free to use it, but just understand that these browsers will render the `meter` text rather than the visual gauge by default G. See http://caniuse.com/#feat=progressmeter for the latest browser support.

TIP The style of the gauge that each supporting browser displays may vary.

TIP Some people have experimented with styling `meter` CSS for both supporting and non-supporting browsers. Search online for "style HTML5 meter with CSS" to see some of the results (note that some use JavaScript).

TIP The `meter` element is not for marking up general measurements—such as height, weight, distance, or circumference—that have no known range. For example, you cannot use it for the following because the number of miles walked isn't gauged against a range: `<p>I walked <meter value="4.5">4.5 </meter> miles yesterday.</p>`.

TIP Be sure not to mix up your uses of the `meter` and `progress` elements.

The **progress** element

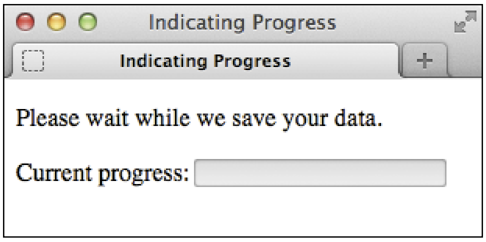
The **progress** element is yet another of the new elements in HTML5. As stated earlier, it indicates the completion progress of a task. Think of a progress bar, like the kind you might see in a web application to indicate progress while it is saving or loading a large amount of data.

As with **meter**, supporting browsers automatically display a progress bar based on the values of the attributes H. And again like **meter**, it's usually best to include text (for example, "0% saved," as shown in the example) inside **progress** to reflect the current progress for older browsers to display I, even though it's not required.

Here's an example:

```
<p>Please wait while we save your
➝ data.</p>

<p>Current progress: <progress
➝ max="100" value="0">0% saved
➝ </progress></p>
```

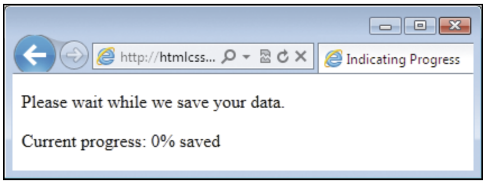
A full discussion of **progress** is beyond the scope of this book, since typically you would dynamically update both the **value** attribute and the inner text with JavaScript as the task progresses (for example, to indicate that it's 37% completed). The visual results are the same whether you do that with JavaScript or code it that way in the HTML initially; for example, `<progress max="100" value="37">37% saved</progress>` J. Of course, nonsupporting browsers would display it similarly to I.



Please wait while we save your data.

Current progress:

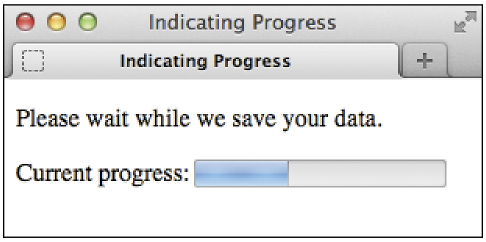
H A browser, like Firefox, that supports **progress** displays the progress bar automatically, coloring it based on the value. It doesn't display the text in between **<progress>** and **</progress>**. The **value** attribute is set to **0** in this example, so the bar indicates no progress.



Please wait while we save your data.

Current progress: 0% saved

I IE9 doesn't support **progress**, so instead of a colored bar, it displays the text content inside the element. You can change the look with CSS.



Please wait while we save your data.

Current progress:

J The **progress** bar in Firefox when the **value** attribute is set to **37** programmatically with JavaScript (or directly in the HTML), assuming **max="100"**. The blue area reflects the amount of progress.

TIP The `progress` element supports three attributes, all of which are optional: `max`, `value`, and `form`. The `max` attribute specifies the total amount of work for the task and must be greater than 0. The `value` attribute specifies the amount completed relative to the task. Assign the `form` attribute to the `id` of a `form` element on the page if you want to associate the `progress` element with a `form` it isn't nested within.

TIP Here's a small taste of how to modify a `progress` element with JavaScript. Let's assume that the element had been coded with an `id` of your choosing, like this:

```
<progress max="100" value="0" id=
→ "progressBar">0% saved</progress>
```

JavaScript such as the following would give you access to the element:

```
var bar = document.getElementById
→ ('progressBar');
```

Then you could get or set the value via `bar.value` as needed. For example, `bar.value = 37;` would set it to 37, and the appearance of the `progress` element would change accordingly.

TIP The `progress` element is supported by the most current version of all desktop browsers as of this writing. IE9 and prior, mobile Safari, and Android browsers don't support it. See http://caniuse.com/#feat=progressmeter for the latest support information.

TIP The style of the `progress` bar that each supporting browser displays may vary, though you can style it yourself to some extent with CSS.

Index