Acknowledgements

My thanks and appreciation to the Peachpit team: Michael Nolan for encouraging me to write the third edition and getting it green-lighted, to my project editor Nancy Peterson for her guidance, insight, and patience, and to publisher Nancy Ruenzel for seven years of publishing my books.

On the development team, my thanks go to proofer Darren Meiss for his meticulous grammar, production editor Katerina Malone for her management skills, and Karin Arrigoni for the detailed work on the index.

A big thank you goes to my good friend and technical editor of this book, Curtis Blanton, for his detailed feedback and suggestions on the code and the explanations—a really great job, Curtis! Thanks also to programmers, Jeffrey Johnson and Isaac Shapira, for their advice and support.

Finally, a very special thanks goes to my wife Beth Bast who, as both development editor and compositor, has endlessly read and edited my drafts, created the book’s graphics, and laid out the entire book in InDesign. It has been an intense five months of work and her efforts and attention to detail are in every page of this book. Thanks, my love, I couldn’t have done it without you.

A big hug for my daughters, Jemma and Lucy, who have been very patient while their parents have focused on writing this book. We love you both.

—Charles Wyke-Smith
Charleston, South Carolina, September 24, 2012
About the Author

Charles Wyke-Smith has been involved in media production for his entire career. In the mid-80s, he co-founded PRINTZ Electronic Design, an early, all-computerized design studio in San Francisco. He has worked in management and consulting roles at Wells Fargo, ESPN Videogames, and Benefitfocus, where he was Director of User Experience. In 2009, he co-founded PeopleMatter, an HR platform for the services industries. He is currently CEO of a new startup, Bublish, a book discovery platform.

Charles is a performing musician and author of several Web development books, including *Stylin' with CSS*, *Codin' for the Web*, *Scriptin' with AJAX*, and *Visual Stylin' with CSS3*. He lives in Charleston, South Carolina with his wife and two daughters.
Contents

Acknowledgements • iii
About the Author • iv
Contents • v
Introduction • x

CHAPTER 1: HTML MARKUP AND DOCUMENT STRUCTURE • 1

The Basics of Markup • 2
Enclosing Tags for Text • 2
Non-Enclosing Tags for Referenced Content • 3
Attributes • 4
Headings and Paragraphs • 5
Compound Elements • 5
About Nested Tags • 6

Anatomy of an HTML Document • 7
An HTML Template • 7
Block and Inline Elements • 10
Nested Elements • 16

The Document Object Model • 20

Summary 22

CHAPTER 2: HOW CSS WORKS • 23

The Anatomy of a CSS Rule • 24
CSS Rule Naming Conventions • 26

Contextual Selectors • 28

Specialized Contextual Selectors • 32
Child Selector > • 32
Adjacent Sibling Selector + • 33
General Sibling Selector ~ • 33
The Universal Selector * • 34

IDs and Classes • 35
The Class Attribute • 35
The ID Attribute • 38
When to Use an ID and When to Use a Class • 39
IDs and Classes Summary • 41

Attribute Selectors • 41
The Attribute Name Selector • 41
The Attribute Value Selector • 42
Summary of Attribute Selectors • 42

Pseudo-Classes • 43
UI Pseudo-Classes • 43
Structural Pseudo-Classes • 46

Pseudo-Elements • 47

Inheritance • 49

The Cascade • 50
Sources of Styles • 50
The Cascade Rules • 52
Calculating Specificity • 53

Rule Declarations • 55
Word Values • 55
Numerical Values • 56
Color Values • 57

Summary • 61

CHAPTER 3: POSITIONING ELEMENTS • 62

Understanding the Box Model • 62
Box Border • 63
Box Padding • 66
Box Margin • 67
Collapsing Margins • 68
Setting Units for Margins • 69

How Big Is a Box? • 70

Floating and Clearing • 75
The Float Property • 76
Three Ways to Enclose Floated Elements • 78

The Position Property • 85
Static Positioning • 86
Relative Positioning • 87
CHAPTER 4: STYLIN’ FONTS AND TEXT • 108

Fonts • 108
Font-Family Property • 109
Font-Size Property • 112
Font-Style Property • 115
Font-Weight Property • 116
Font-Variant Property • 116
Font Property • 117

Text Properties • 117
Text-Indent Property • 118
Letter-Spacing Property • 119
Word-Spacing Property • 121
Text-Decoration Property • 122
Text-Align Property • 122
Line-Height Property • 123
Text-Transform Property • 124
Vertical-Align Property • 125
Web Fonts Demystified • 126
  Hosted Font Libraries • 127
  Packaged @font-face Kit • 128
  Generated @font-face Kit • 130

Stylin’ Text • 130
  Basic Text Layout • 131
  Stylin’ Text on a Grid • 135
  An Exercise in Classic Typography • 141

Summary • 150

CHAPTER 5: PAGE LAYOUTS • 151

Basic Layout Concepts • 151
  Layout Height and Layout Width • 152

Creating Columns • 153
  Setting Padding and Borders on Columns • 161

Three-Column, Fluid Center Layouts • 172
  A Three-Column, Fluid Center Layout with Negative Margins • 172
  A Three-Column, Fluid Center Layout with CSS3 Table Properties • 177

A Multi-Row, Multi-Column Layout • 179
  Real-World CSS Selectors • 182
  Inner Divs in Action • 184

Summary • 185

CHAPTER 6: INTERFACE COMPONENTS • 186

Creating Navigation Menus • 186
  A Vertical Menu • 186
  A Horizontal Menu • 189
  Drop-Down Menus • 191

Forms • 201
  The HTML Elements of a Form • 201
  Form Markup Strategies • 209
  Styling the Form • 210
  A Search Form • 221

A Popup Overlay • 224
  Stacking Context and z-index • 227
Introduction

This is an exciting time to be a Web designer. The Web is now the way we consume almost all media, as cable TV, CDs, and DVDs are replaced by on-demand Web-delivered services like Hulu, Netflix, Pandora, and Spotify.

We also have a variety of devices on which to consume this media—desktop computers, laptops, tablets, smartphones, and massive 60-inch flat screen displays.

Supporting the delivery of content across all these different devices and media is an emerging technology standard—centered around browsers that use HTML5, CSS3, and JavaScript.

When I wrote the second edition of *Stylin' with CSS* almost five years ago, a rigid and complex XML-based version of HTML, called XHTML, had become the standard. Because XHTML was unsuited to the free-wheeling and fast-moving world of Web development, Apple, Mozilla, and Opera formed the Web Hypertext Application Technology Working Group. The purpose of this organization was to revive the development of HTML that the World Wide Web Consortium had abandoned after HTML4 in favor of XHTML. The phoenix that arose from the ashes became known as HTML5, and in the last three years, the move from XHTML to HTML5 has occurred swiftly and with good reason.

HTML5 is designed for today’s multimedia Web, with a rich set of APIs (Application Programming Interfaces) that provide built-in support for video, audio, graphics, geo-location, data storage, and much more. HTML5 also offers many new elements for better of structuring documents (section, article, nav, and so on). Previously, semantically meaningless divs with identifying classes and IDs had been used for this purpose, which limited the portability and meaning of the markup.

During HTML’s transition from HTML4 to XHTML and then to HTML5, CSS3 has been steadily implemented by every browser. The visual-rendering toolbox that is CSS3 is a massive set of recommendations, so large that it has been divided into numerous modules so that different teams can plan how each module will work.
Finally, you can utilize long-awaited CSS3 features, such as gradients, transitions, transformations, shadows, and radiused corners, and be confident that the vast majority of users will see these features correctly rendered. For older browsers, where CSS3 is not fully supported, Modernizr, a JavaScript file that you link to your pages, enables you to detect support for specific CSS3 features. You can then provide fallbacks (alternative code) or polyfills (JavaScript code that simulates build-in CSS3 functionality) for those features. You can learn more about these fallbacks and polyfills in the Appendix section of this book.

Today, the Web is a much more user-friendly, and developer-friendly, place than it has ever been. Writing this latest version of *Stylin’ with CSS* has, as always, been a labor of hundreds of hours of coding and writing, countless late nights and endless cups of tea, but it has also felt like a kind of celebration. That’s because, behind the examples and exercises, this book describes a new state of the Web—the realization of a vision that has long been imagined. Thanks to the work and advocacy of Jeffrey Zeldman, Ian Hicks, and countless others, the long quest for Web standards appears to have been realized. It’s like inching your way up the side of a mountain and suddenly realizing you’re at the top. The fact that I no longer have to write (and waste pages of this book teaching you to write) hacks for older browsers for even the most simple layouts; that I can create shadows or radiused corners with a single line of CSS instead of using complex graphics and layers of divs; that every current browser displays my pages in a complete and consistent way—is a massive breakthrough.

So, in this book, I am looking forward to the future. Rather than spend pages showing you how to work around old browser incompatibilities as I did in previous editions, I am focusing on the vast scope of what is possible now and in the future with HTML5, CSS3, and modern browsers. Internet Explorer 9 and up, and Firefox, Chrome, Safari, and Opera (which automatically update users to the latest versions) all behave in remarkably consistent ways. The number of users on older browsers (specifically IE8 and below) decreases every day. I provide information in the Technical Notes in the Appendix about how to work with older browsers, but the main narrative of this book is to show you how CSS works today and in the future.
A Focus on Essential Techniques

You don't need to be a brilliant artist or a computer programmer to be successful with CSS, although both those abilities can be put to good use. What you do need to do is have a solid grounding in the workings of HTML and CSS, and understand some key techniques and best practices. This book is designed to provide you with this understanding and provide a strong foundation for building your skills. CSS3 is so extensive that there are many features of it I didn't even mention in this book. No matter, I believe that once you have read and worked through the examples in this book, you will be able to rapidly extend your knowledge and skills—that at least is the sincere objective in what I have written here.

Download My Code, Don't Rewrite It

All the code shown in this book is available for download at the book's Web site, www.stylinwithcss.com. I recommend you use the download code rather than copying the code from the book; not only is it much quicker and easier to do that, but I will also update the download code to correct any errors that are found. I will also run an errata section on the site and post any reported inaccuracies in the book. The site will have a new blog that I am starting, so please come by for information and maybe inspiration, and post comments and make suggestions for articles you would like to see.

Thanks for buying this book. I hope it's a big help to you, and good luck with your Web endeavors.
Chapter 4

Stylin’ Fonts and Text

Much of Web design deals with text, in paragraphs, headings, lists, menus, and forms, so understanding the CSS properties in this chapter is essential to making a site that looks professional. Almost more than any other factor, type makes the clearest visual statement about the quality of your site’s offerings. Graphics are the icing on the cake; typography is where good design begins.

In this chapter, you’ll learn about fonts and text, and the respective CSS properties you can use to style them. I’ll also introduce you to the wonderful world of Web fonts, which download to your user along with your pages. Now you no longer have to rely on the user having your font choices installed on his device, and you can be confident that every user will see your typography in the way you intend.

Let’s start with fonts.

Fonts

The fonts you specify in your Web pages can come from three sources.

- The fonts that are installed on the user’s device. (Until recently, these have been the only fonts reliably available to your Web pages.)
- Fonts that are hosted on third-party sites, most notably Typekit and Google, and linked to your page using the `link` tag.
- Fonts that are hosted on your Web server and served to the user’s browser along with the page, using the `@font-face` rule.

In the font property descriptions that follow, the examples will show the first of these sources: the fonts that are installed on the user’s computer. See *Web Fonts Demystified* later in this chapter for a discussion of the other two sources.
Now let’s look at the six properties that relate to font styling:

- **font-family**
- **font-size**
- **font-style**
- **font-weight**
- **font-variant**
- **font** (shorthand)

### Font-Family Property

Example: `h2 {font-family:times, serif;}`

`font-family` determines the font in which an element is displayed. Typically, you set a primary font for the entire page, and then only add `font-family` styles to elements that you want to display in a different font. To specify the font for the entire page, you set the `font-family` of the `body` element:

`body {font-family:verdana, sans-serif;}`
font-family is an inherited property, so its value is passed to all its descendants, which in the case of body is all the other elements in the markup.

Because fonts must either be on the user’s computer, or delivered over the Web, there is always a possibility that a particular font you specify might not be available to a page. For this reason, fonts are always specified in lists called font stacks.

**SPECIFYING INSTALLED FONTS USING FONT STACKS**

Fonts are installed in the operating system of a device, which allows all resident applications to share them. Only a limited set of fonts come installed in the typical operating system, and fonts can be added and removed by the user, so you can never be absolutely certain what fonts will be available to display your pages. Because of this, when stating the font in which you want text to display, you must also list additional “fallback” fonts in case your first choice isn’t available on the user’s system. This list of choices is called a font stack.

In short, font stacks ensure that the user sees your page text in the intended font if it is installed on her device, and if it is not, then in a font that you specify as an acceptable substitute.

body {font-family:"trebuchet ms", tahoma, sans-serif;}

This font stack effectively tells the browser “Display this document in Trebuchet MS, and if the system doesn’t have it, use Tahoma, and if neither is installed, use whatever generic sans-serif font is available.” It is very important to make the last item of a font-family declaration a generic declaration, typically “serif” or “sans-serif”, as a final fallback.

There are five generic font-family names:

**Serif**—serif fonts have small details at the terminals (tips) of the characters (like this text)

**Sans-serif**—sans-serif fonts have no details at the terminals (like the headings of this book)

**Monospace**—every monospace font character occupies the same amount of horizontal space (like the code examples in this book)
Cursive—cursive fonts look like handwriting (like the headline of *The Hound of the Baskervilles* example later in this chapter).

Fantasy—fonts that don't fit the other categories (typically the strange and bizarre)

The purpose of these generic fonts is to ensure, that if none of your choices are available that, at a minimum, your document displays in the right type (no pun intended) of font.

It’s worth taking some care when selecting the fonts that you put in a font stack. For example, Dreamweaver offers a list of selectable font stacks that pop up every time you type `font-family:` in your CSS file, but these fonts are not ideal substitutes for one another. For example, here is a font stack that Dreamweaver offers:

```plaintext
verdana, arial, helvetica, sans-serif;
```

Verdana is a bulky font that has a much larger x-height than Arial, so if a user does not have Verdana installed, your page will be displayed in Arial, a font that is smaller than the one you intended. More words will fit on each line, and the vertical height of text blocks may be shorter.

A good test is to view your pages with each font in the stack as the first choice so that you can see how the layout changes if it displays in one of the fallback fonts.

A better fallback for Verdana might be Tahoma, which has the same large x-height.

```plaintext
verdana, tahoma, sans-serif
```

For a stack of lighter sans-serif fonts, you might use

```plaintext
helvetica, arial, sans-serif
```

Here's a stack of serif fonts starting with a font that the user may not have.

```plaintext
{font-family:"hoefler text", georgia, times, serif;}
```

In a case like this, always complete the stack with fonts that are supplied with most computer's operating systems, here Georgia and Times, and end with the generic, serif.
Font-Size Property

Example: `h2 {font-size:18px;}`

Every HTML text element has a default font-size set by the browser style sheet, so when you set an element’s font-size, you are changing its font size from that default. Font sizing can appear to act unpredictably if you don’t understand how the inheritance of font sizes down the hierarchy is affected by which font size units you use. There are two types of units that you can use to set the font-size: absolute units, such as pixels or points, and relative units, such as percentages or ems. Let me explain the difference between them.

Font-size is an inherited property, so a change to the font size of an element will result in a proportional change of size in the font sizes of its descendant elements. This means that if you set the font-size of the body element to 200%, then the text of all the elements on your page will double in size.

This effect occurs because in the browser style sheet, all element font sizes are set in the relative unit, em. For example, the h1

---

So, Which Fonts Are Available to All Users’ Browsers?

That is a common question which has no definitive answer but you have a high probability that any Mac or PC will have these fonts installed:

<table>
<thead>
<tr>
<th>Serif</th>
<th>Sans-serif</th>
<th>Monospace</th>
</tr>
</thead>
<tbody>
<tr>
<td>Georgia</td>
<td>Arial</td>
<td>Courier New</td>
</tr>
<tr>
<td>Palatino/Book Antiqua</td>
<td>Arial Black</td>
<td>Lucida Console/Monaco</td>
</tr>
<tr>
<td>Times New Roman</td>
<td>Arial Narrow</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Tahoma</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Trebuchet MS</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Verdana</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Cursive</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Comic Sans MS</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Fantasy</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Impact</td>
</tr>
</tbody>
</table>

Because of the often obscure fonts on today’s phones and tablets, it is more important than ever to include fallback generic font families in your font stacks. If you want to specify a specific font, use a hosted Web font or one that is downloaded from your Web server—see Web Fonts Demystified, later in the chapter.
element is 2em, the h2 element is 1.5em, and p (paragraph) is 1em. By default, 1em is equivalent to 16 pixels—this is known as the font-size baseline. So by default, h1 is 32 pixels (16 × 2em = 32 pixels), h2 is 24 pixels, and p is 16 pixels.

If you set the body text to 20px, you are resetting the baseline, so now h1 would be 40px (20 pixels × 2em = 40 pixels), h2 would be 30 pixels, and p would be 20 pixels. However, font-size inheritance will not occur in descendant elements that have been sized with absolute units such as pixels—these elements will always display at their specified size.

Let’s learn more about font sizing by looking at each method of sizing fonts in turn.

**Absolute Font Sizing**

Sizing text with absolute units such as pixels, picas, or inches is simple; when you set the size of an element using absolute units, it stays that size no matter what font sizing is applied to its ancestors. The downside of absolute sizing is that if you decide to proportionally change the overall size of the text on your page, you have to change every absolute font-size in the style sheet; an absolutely-sized page requires more effort to fine tune.

In short, if you change the size of the body tag’s font, any absolutely-sized elements do not change size, but elements that have not been sized in your CSS will change proportionally to the size stated on body.

**Relative Font Sizing**

Sizing text with relative units such as percentages, ems, or rems is slightly more complex; when you set the size of an element using relative units, the size of the text is set relative to the size of the nearest “sized” ancestor.

Let’s consider this simple markup

```html
<body>
  <p>This is <strong>very important!</strong></p>
</body>
```
and this CSS

\[
p \{ \text{font-size:.75em;} \}
\]

\[
\text{strong} \{ \text{font-size:.75em;} \}
\]

In this example, the \textit{p} tag text would be 12 pixels (the \textit{body} tag’s 16 pixel baseline \times .75 = 12). Because \textbf{strong} is a child of \textit{p}, its point size would be 9 points. What you see is that relative sizes compound down through the hierarchy—\textbf{strong} is 16 pixels \times .75 \times .75 = 9 pixels. Relative units can take practice to master, as unlike absolute sizes, changing the relative font size of an element also changes all the child elements by the same proportion.

However, with relative sizing, you have the ability to tweak the size of all elements proportionally by resizing \textit{body}, or a number of elements by changing a shared ancestor element. This can be time saving as you experiment with your layout, but it also takes planning for the same reason; a change to an element’s \textit{font-size} affects all its descendant elements, too.

You cannot tweak font sizes like this if you work in absolute \textit{font-size} units—each absolutely-sized element must be resized individually. Of course, if you do size in absolute units, you can size an element without getting the often-unwanted “knock-on” effect of a change of size in its ancestors.

However, with today’s wide range of screen sizes, from massive monitors to tiny phones, the need for text that can be easily scaled makes relative sizing the preferred approach.

\section*{A Note on Rem Units}

The new relative rem (root em) unit is a CSS3 addition that is generating a lot of excitement in the Web community. When you size an element in rems, the size is relative, but only to the root HTML element. This gives you the best of both relative and absolute worlds; you can use relative sizing to proportionally change the overall font size by changing the font size of the HTML element, but unlike ems, font sizes are not compounded down through the hierarchy. Rems are supported by all the current browsers, but not by IE8 and earlier. The fallback is simple, however, and that is to provide absolute pixel sizing to browsers that don’t understand, and therefore ignore, rem declarations, like this
Let's now look at the other font-related CSS properties.

**Font-Style Property**

Values: *italic*, *oblique*, *normal*

Example: `h2 {font-style:italic;}`

*font-style* determines whether a font is italicized or not. You can also write *oblique* instead of *italic*—the result is the same.

There are only two useful settings for the *font-style* property: *italic* to make regular text italicized, and *normal* to make a section within italicized type regular “upright” text. In this example

```css
p {font-style:italic;}
span {font-style:normal;}
```

```html
<p>This is italicized text with <span>a piece of non-italic text</span> in the middle.</p>
```

the code produces the result in **Figure 4.1**.

Note that the main purpose of italic text is to indicate emphasis, as in “It's very hot today!” If you want to indicate emphasis, use the *em* tag, which styles the text as italic by default.

**The Normal Value**

*normal* causes any of the possible effects of a property not to be applied. Why might you want to do this?

The reason this option is available is so you can selectively override a default or a global property you have set. Headlines `h1` through `h6` are bold by default, so if you want to unbold the `h3` element, for example, you need to write `h3 {font-weight:normal;}`. If your style sheet states `a {font-variant:small-caps;}` so that all links are in small caps, and you want one special set of links to be in regular upper- and lowercase type, you might write a declaration such as `a.speciallink {font-variant:normal;}`.
Font-Weight Property

Possible values: 100, 200, and so on to 900, or lighter, normal, bold, and bolder.

Example: a {font-weight:bold;}

Despite all the numerical options listed here, browsers only display two visual results for all font-weight values—bold or normal. Because interpretation of the numerical values differs among browsers, you’ll see the switch from normal to bold at various values—typically around 400. It’s best to avoid using all values except bold and normal, as illustrated in Figure 4.2.

p.shows_weight {font-weight:bold;}
p.shows_weight .span {font-weight:normal;}

Note that the primary purpose of bold text is to indicate importance, as in “Danger!” Mark up important text with the strong tag, which styles the text as bold by default.

Font-Variant Property

Values: small-caps, normal

Example: blockquote {font-variant:small-caps;}

This property accepts just one value (besides normal), and that is small-caps. This causes all lowercase letters to be set in small caps, like this:

h3 {font-variant:small-caps;}

The code above produces the result in Figure 4.3.
I often use small-caps with the ::first-line pseudo-element as I demonstrate in *The Hound of the Baskervilles* example at the end of this chapter. Use this styling sparingly because text in all uppercase is harder to read as it lacks the visual cues provided by the ascenders and descendents of lowercase type.

**Font Property**

Example: `p {font: bold italic small-caps .9em helvetica, arial, sans-serif;}`

```html
<p>Here's a piece of text loaded up with every possible font property.</p>
```

The code above produces the result in *Figure 4.4*.

The font property is a shorthand styling that lets you apply all of the font properties in a single declaration, reducing the amount of CSS you have to write. You must follow two rules, however, so that the browser can interpret the properties correctly.

**Rule 1:** Values for font-size and font-family must always be declared.

**Rule 2:** The sequence for the values is as follows:

1. font-weight, font-style, font-variant, in any order, then
2. font-size, then
3. font-family

**Text Properties**

Now that you’ve looked at how to style font properties, it’s time to look at how to style text properties. If you want to indent a paragraph, create a superscript such as the 6 in $10^6$, create more space between each letter of a headline, and many other type formatting tasks, you will use the CSS text properties.
Here are the most useful text-related CSS properties:

- `text-indent`
- `letter-spacing`
- `word-spacing`
- `text-decoration`
- `text-align`
- `line-height`
- `text-transform`
- `vertical-align`

**Text-Indent Property**

Values: any length value (positive or negative)

Example: `p {text-indent:3em;}`

This property sets the start position of the text box in relation to the containing element. By default, that is the top-left corner of the container.

If you set a positive value to the `text-indent`, then the text moves to the right, creating an indented paragraph (Figure 4.5, example 1).

**Figure 4.5** These four examples illustrate the `text-indent` property.
However, if you set a negative value for `text-indent`, the first line hangs out to the left of the containing element, so make sure that there is a place for it to go. If there's an element to the left, the hanging text can overlap it, or if it's close to the edge of the browser window, it is clipped (Figure 4.5, example 2). The way to avoid this problem is always to specify a positive left margin value greater than the specified negative indent. In Figure 4.5 example 2, the negative indent is −1.5 ems, but in Figure 4.5, example 3, there is also a left margin value of 2 ems. Here is how this is written.

```
p { text-indent:-1.5em; margin-left:2em; border:1px solid red; }
```

Indents can help give text a professionally-styled look and also give the reader clear visual entry points into the text blocks. Remember to set indents and related margins in ems, as I have done here, so that the indent remains proportional to the line length if the user (or you) changes the font size.

**Letter-Spacing Property**

Values: any length values (positive or negative)

Example: `p { letter-spacing:.2em; }`

Positive `letter-spacing` values increase the overall space between letters, while negative values decrease it. Always use relative values such as ems for letter spacing, even if you are setting the font size in pixels, so that the spacing remains proportional if the font size...
Meet the Text Snake

A very important concept of how CSS manages text is that CSS puts an invisible box around the text inside an element. For example, if you put a block of text in a paragraph `<p>` element, CSS sees the actual text as a long line of text, even if it gets broken across multiple lines in order to fit in the container. To make this clear, in Figure 4.6, the border of the containing element (the paragraph) is in red, and the border of the text box is in green. Text properties are applied to the green text box.

![Figure 4.6](image)

**Figure 4.6** Text is contained within a long, skinny box that is often broken across multiple lines.

In this example, I mark up the text like this:

```html
<p><span>Here is a long paragraph…</span></p>
```

and apply the following styles:

```css
p {border:3px solid red;}
span {border:1px solid green;}
```

Note that the text box is broken across the lines and is only closed at the beginning of the first line and the end of the last line. Knowing this can help you get things looking the way you want faster. For example, if you want to indent the first line of a paragraph, you can use the text property `text-indent`, as I do in Figure 4.5, and then you are moving the start position of the text box. Subsequent lines are not indented because to CSS, it’s just one long piece of text.

If you want the whole paragraph indented, then you need to set the `margin-left` property of the paragraph; in other words, you have to push the whole container to the right. All you need to remember is that text properties are applied to the long, thin, snake-like inner text box, not the box of the containing element.
changes. Examples are shown in Figure 4.7. letter-spacing controls tracking, which is the typographical term for the letter spacing applied to all characters in a block of text. This contrasts with kerning, which is the term for adjusting the space between two specific characters.

![Figure 4.7](image)

The default letter spacing of a font appears looser as the text gets larger, so tightening the letter spacing of a headline adds refinement to your Web page. Note the text and headline I tightened in Figure 4.7 only have .05 em (a twentieth of an em) of letter spacing removed from between each character; much more and the letters would start to merge into each other.

**Word-Spacing Property**

Values: any length values (positive or negative)

Example: `p {word-spacing:.2em;}`

Word spacing is very similar to letter spacing except, as you might imagine, the space changes between each word rather than between each letter. CSS treats any character or group of characters with white space around them as a word. Second, even more than letter spacing, word spacing can easily be overdone and result in some very hard-to-read text (Figure 4.8).

![Figure 4.8](image)
Text-Decoration Property

Values: underline, overline, line-through, blink, none

Example: .retailprice {text-decoration:line-through;}

These values, with the exception of blink, are displayed in Figure 4.9. blink, which makes text flash on and off, is truly annoying, and should be used sparingly, or better yet, not at all.

Text-Align Property

Values: left, right, center, justify

Example: p {text-align:right;}

There are only four values for this property: left, right, center, and justify. The text aligns horizontally within the element. Note that center will also center a smaller fixed-width element or image horizontally within a larger element. Figure 4.10 shows the four possible text-align values in action.
Figure 4.10 The four text-align values.

This is text that is left aligned and is the usual way that text is aligned. It's easy to read because every line starts against the left edge of the containing element and that's why it's the default setting.

This is text that is right aligned and it can be somewhat hard to read because every line starts in a different place.

This is text that is centered. Small blocks of text look good centered but it's hard to read large amounts of text as every line starts a different distance from the edge of the page.

Justified text is both left and right aligned at the same time. This is achieved by mathematically increasing word space on a line-by-line basis. This can result in gaps of white space (known as rivers) throughout the text, if the containing element is narrow or if there are multiple numerous extended consecutive occurrences of exceptionally lengthy words like these, so that only a few fit on each line.

Line-Height Property

Values: any numerical value (value type is optional)

Example:  p {line-height:1.5;}

`line-height` is the CSS equivalent of leading (pronounced like the metal) in the world of print. Leading creates space between the lines of a block of text.

Line height is distributed above and below the text. For example, if you have a font size of 12 pixels and you set the line height to 20 pixels, the browser adds 4 pixels of space above and 4 pixels of space below to achieve the 20 pixel line height.

On a single line of text like a headline, `line-height` acts like another margin, and large headlines (h1 and h2, for example) have a significant amount of default line height. This is worth remembering, because sometimes you will find that even after removing margins and padding, you still can't eliminate all the space above and below a headline. To do this you need to reduce the line height also, sometimes to a height less than that of the text, i.e., less than 1.
As shown in Figure 4.11, the simplest way to change this default line height is to use the `font` shorthand property and write a compound value for both `font-size` and `line-height` in one. For example:

```css
div#intro {font: 1.2em/1.4 helvetica, arial, sans-serif;}
```

In this case, the leading is 1.4 times the font size of 1.2 ems. Note that you don't need any units, such as ems or pixels, specified for the `line-height` part of the value, just a number. In this case, CSS simply takes the calculated size of whatever number of onscreen pixels 1.2 ems works out to be and multiplies it by 1.4 to arrive at the line height. If you later increase the font size to 1.5 ems, the line height (leading) is still 1.4 times the calculated amount of 1.5 ems. Note if you specify a line height in a fixed unit, such as pixels, and you increase the font size, then the lines of text may start to overlap one another.

### Text-Transform Property

**Values:** none, uppercase, lowercase, capitalize

**Example:**

```css
p {text-transform: capitalize;}
```

`text-transform` changes the capitalization of text within an element. You can force a line of text to have initial letters capitalized, all text uppercase, or all text lowercase. Figure 4.12 illustrates these options.

- **capitalize** capitalizes the first letter of every word. This emulates the style of many headlines in ads, newspapers, and magazines, except that a human applying such styling tends to leave the capitalization off minor words such as “of,” “as,” and “and,” as in “Tom
and Jerry Go to Vegas.” CSS capitalization simply produces “Tom
And Jerry Go To Vegas.”

Vertical-Align Property
Values: any length value, sub, sup, top, middle, bottom

Example: span {vertical-align:60%;}

vertical-align moves text up or down with respect to the baseline,
but note that it only affects inline elements. If you want to vertically
align a block-level element, you must also set its display property
to inline. One of the most common uses is for superscript and
subscript numbers in formulas and mathematical expressions,
such as $x^4 y^5$ or $N_2O$. It’s also the correct way to style asterisks and
other markers within text to indicate footnotes. I don’t like the way
most browsers style sub- and superscripts by default—the font size
is too large and too high (or low, for subscript) for my liking. A little
styling can render better, and more consistent cross-browser, pro-
portions.

Here’s the HTML for this example

```html
<h4>Default <code>sub</code> and <code>sup</code> styles</h4>

<p>Enjoy mountain spring H<sub>2</sub>0. It's 10<sup>5</sup> times better than tap<sup>&dagger;</sup> water!</p>
<p class="customsmall"><sup>&dagger;</sup><em>This means water provided through a municipal distribution system</em></p>

<h4>Custom <code>sub</code> and <code>sup</code> styles</h4>
```
Enjoy mountain spring H₂O. It's 10⁵ times better than tap water!

This means water provided through a municipal distribution system.

While the HTML tags `<sup>` and `<sub>` create superscript or subscript text automatically, it's worth using `vertical-align` and `font-size` in combination to produce a more pleasing result (Figure 4.13). This covers the font and text properties of CSS. Now let's look at how fonts can be downloaded into your Web pages.

**Web Fonts Demystified**

A now widely implemented CSS feature is the capability to embed downloadable fonts into your pages, using the `@font-face` rule.

`@font-face` gives designers greatly expanded font options beyond the basic system fonts. Now you can ensure that the fonts you specify are available to your user's browser because they are downloaded to the browser from a Web server and you no longer have to rely on your font choices being installed on the user's device.

There are three ways to specify Web fonts:
• Use a hosted font library that delivers fonts to your Web pages, such as Google Web Fonts or Adobe’s Typekit

• Use a pre-packaged \texttt{@font-face} kit

• Generate an \texttt{@font-face} kit from one of your own fonts using Font Squirrel

Let’s start with the easiest method—accessing a hosted font library.

**Hosted Font Libraries**

The two largest hosted font libraries are Google Web Fonts, which offers free use of their 500-plus font collection, and Adobe’s Typekit, which offers subscription-based access to their collection of 739 font families. Both have easy-to-use interfaces.

Here’s how the process works on Google Web Fonts. Go to http://www.google.com/webfonts, find the font you want, click the \texttt{Add to Collection} button, then click \texttt{Use} at the bottom of the page (Figure 4.14). Google then generates a \texttt{link} tag with a reference to your selected fonts that you paste into the \texttt{head} of your HTML file.

![Figure 4.14 The font, Niconne, is added to my collection so Google will generate a link to this font.](image-url)
Multiple fonts can be linked in a single line of code. This `<link>` tag references the Anton, Niconne, and Prata fonts.

```html
<link href='http://fonts.googleapis.com/css?family=Anton|Niconne|Prata' rel='stylesheet' type='text/css'>
```

Once you have added the link in the `head` of the page, you can then use the fonts in the same way as you would use any other font. When your page is viewed, the font is served into your user's page directly from Google. For example

```css
h3 {font: 20px "Prata", serif;}
```

displays as shown in Figure 4.15.

Using a hosted font library offers a quick and reliable way to extend the otherwise limited palette of system fonts. You can add Google fonts to your pages in minutes, and by using them in your designs you can be virtually certain that your users will view your pages in the font that you intended.

**Packaged @font-face Kit**

The second method of embedding fonts in your pages uses the `@font-face` rule, which requires that the fonts are accessible from your site's, or a third-party, Web server. Fonts served in this way download to the browser when the first page that uses each font loads; after that, they are cached in the user's browser and don't have to download again. Note that the user can't use the font for any purpose except to display Web pages that include that font.

The `@font-face` approach involves more effort, but offers the possibility to use virtually any font you wish. Because of licensing restrictions, you must either purchase a font, or use a royalty-free font, that is licensed for embedding.

One issue with using `@font-face` is that different browsers require different font formats. The Firefox browser, and Webkit browsers such as Safari, Chrome, and mobile Safari iOS since v4.1, use OpenType (OTF) or TrueType (TTF) font formats. Internet Explorer uses the Embedded OpenType (EOT) format, and some other
browsers, such as mobile Safari pre-iOS 4.1, use the Scalable Vector Graphics (SVG) format. However, the different font formats are available in ready-to-use kits, or can be readily generated from a font on your computer (again, ensure you have a license to use the font in this way).

Font Squirrel (www.fontsquirrel.com) offers an extensive library of fonts in ready-to-use “font kits.” Each font kit includes the font in all the required font formats, and the related CSS code to ensure that each browser is served the correct format. FontSquirrel also has a converter that allows you to upload and convert any font into a font kit.

Here is an example of Font Squirrel’s `@font-face` CSS code for the font Ubuntu Titling Bold, but this format will work for fonts from other sources, too.

```css
@font-face {
  font-family: 'UbuntuTitlingBold';
  src: url('UbuntuTitling-Bold-webfont.eot');
  src: url('UbuntuTitling-Bold-webfont.eot?#iefix')
    format('embedded-opentype'),
    url('UbuntuTitling-Bold-webfont.woff')
    format('woff'),
    url('UbuntuTitling-Bold-webfont.ttf')
    format('truetype'),
    url('UbuntuTitling-Bold-webfont.'
        svg#UbuntuTitlingBold')
    format('svg');
  font-weight: normal;
  font-style: normal;
}
```

Once this code is added to the page you can reference it with a `font-family` rule in the normal way, using the `font-family` name that is stated in the `font-family` value of the `@font-face` rule.
Sometimes, you need to use a specific font in your design—a typical situation is that a client has a corporate font that you must use in the Web site you are designing. Today, as long as you have licensing rights to use that font as a Web font (check the font’s license agreement or check with the company that manufactured the font), you can convert that font into an @font-face kit at Font Squirrel (http://www.fontsquirrel.com/fontface/generator). Just follow the simple steps and in a few minutes, you will have downloaded an @font-face kit that is ready to go onto your server.

Before I move on to some examples of designing with type, I’ll make a couple of observations on embedded fonts. Until the day that all browsers manufacturers settle on one font format (which probably should be OpenType), you will have to deal with the complexities of multiple font formats. You can learn all about the preceding multi-font @font-face syntax, and how it ensures Internet Explorer gets the required .eot format font, at Fontspring’s Blog (http://www.fontspring.com/blog/fixing-ie9-font-face-problems). Fontspring also sells fonts that are licensed for use with @font-face.

Since the inception of the Web, designers have been limited, except with great effort, to fonts that are generally available on the PC and Mac operating systems. The long-awaited implementation of @font-face in all modern browsers, including IE9 and later, finally gives Web designers the access to the same smorgasbord of fonts available to their print brethren. The fallback for older browsers that don’t support @font-face is simple: Those users simply get the next font listed in the font stack, so be sure to list other more common fonts that are found on users’ systems after your preferred embedded ones.

If you want to gain a greater understanding of @font-face, I recommend you read Tim Brown’s blog article How to Use CSS @font-face (http://nicewebtype.com/notes/2009/10/30/how-to-use-css-font-face).

**Generated @font-face Kit**

**Stylin’ Text**

It’s time to put all your new-found knowledge about fonts and text into practice. I’m going to conclude this chapter with three examples of how you can create good-looking typography, from the quick and simple to the considered and sophisticated.
There is the notion of rhythm in typography, that defines the regular flow of the type down the page, generally achieved by working to an underlying grid. Good rhythm helps the eye move smoothly over the page.

Let’s start with some quick and basic text styling, and rather than use an underlying grid to organize our type, simply space each element proportionately to its type size. This exercise will allow you to see how to get a result fast if that’s what’s needed.

**Basic Text Layout**

As you saw in Chapter 1, the default browser stylings for headings, paragraphs, lists, and other text elements have a very wide range of sizes, and the vertical margins between them are too big. To illustrate how to style these defaults into a more pleasing presentation, here’s markup with some commonly used text elements.

```html
<article>
  <h1>CSS</h1>
  <p>CSS stands for Cascading Style Sheets. CSS controls the presentational aspects of your Web pages.</p>
  <h2>Block-Level Elements</h2>
  <p>Block-level elements stack down the page. They include:</p>
  <ul>
    <li>header</li>
    <li>section</li>
    <li>h1, h2, etc.</li>
  </ul>
  <h2>Inline Elements</h2>
  <p>Inline elements sit next to each other, if there is room. They include:</p>
  <ul>
    <li>img</li>
    <li>a</li>
  </ul>
</article>
```
<li><code>em</code></li>
</ul>

<blockquote>
<q>Typography maketh the Web site.</q><cite>CWS</cite>
</blockquote>

Figure 4.16 shows this markup displayed in a browser.

CSS
CSS stands for Cascading Style Sheets. CSS controls the presentational aspects of your Web pages.

Block-Level Elements
Block-level elements stack down the page. They include:
- header
- section
- h1, h2, etc.

Inline Elements
Inline elements sit next to each other, if there is room. They include:
- img
- a
- em

"Typography maketh the Web site," CWS

Here are some steps to quickly style this markup into a more pleasing layout. First, let's remove the margins that are creating all the space between the elements, set the overall font, and style the article tag that encloses all the text elements into a visual container that surrounds the text and centers it on the page.

```css
* {margin:0; padding:0;}
body {font:1.0em helvetica, arial, sans-serif;}
article {width:500px; margin:20px auto; padding:20px; border:2px solid #999;}
```
Figure 4.17 shows the displayed result in the browser.

Next, there needs to be some strategically-placed vertical space between the elements. Also, with the margins gone, the list bullets hang into the margin so I’ll fix that, too.

- **space around headings**
  ```css
  h1, h2, h3, h4, h5, h6 {line-height:1.15em; margin-bottom:.1em;}
  ```

- **space around other text elements**
  ```css
  p, ul, blockquote {line-height:1.15em; margin-bottom:.75em;}
  ```

- **indent on lists**
  ```css
  ul {margin-left:32px;}
  ```

Figure 4.18 Space has been added after the paragraphs.
As you can see in Figure 4.18, I have tightened up the `line-height` of all the elements, making each element’s `line-height` only slightly larger than the height of the text. This is because `line-height` is added equally above and below the text and I only want to add space below each element, which I do by applying margins. However, I have to leave some `line-height` or the adjacent lines of the paragraph text (and the headings if they run over to a second line) will touch.

Note there are only two settings for the margins, the exact amount of space of which is relative to each element’s font size. I give the headings very small bottom margins (equal to 15% of each one’s font size) so they sit close above elements that follow them. I give all the other text elements a larger bottom margin (equal to 75% of each one's font size) to create white space after them in the layout.

As a final step, I want to get a better balance between the headings, so that the bigger headings stand out and the smallest ones don’t get lost, and also increase the size of the inline `code` elements.

```
h1 {font-size:1.9em;}
h2 {font-size:1.6em;}
h3 {font-size:1.4em;}
h4 {font-size:1.2em;}
h5 {font-size:1em;}
h6 {font-size:.9em;}
p {font-size:.9em;}

code {font-size:1.3em;}
```

While this example is quick and basic, it shows that some minimal text styling can greatly improve the appearance of the page and readability of the content (Figure 4.19). Let’s now look at how to achieve a more sophisticated look through the use of grids.
Now with larger heading and code text, the page is more visually pleasing and helps the viewer understand the hierarchy of the information.

**Stylin’ Text on a Grid**

Using a grid to lay out your type provides a rhythm and visual flow to the page. Because I am looking at type in this chapter, I’ll focus on using a grid to create the *vertical* flow of text.

In this example, I’ll create a layout based on a vertical 18-pixel grid and every element will align with it. Because a graphic can be added into the background of an element, in this case body, I can temporarily add a simple spacing guide into the page.

Here I use Adobe Fireworks (you can use the graphics program of your choice) to make a white rectangle 100 by 18 pixels and add a 1 pixel gray line along the bottom. I save it in .png format (.jpg or .gif work just fine, too) with the name `grid_18px.png`. Figure 4.20 shows how it looks (shown on a pale blue background for clarity).

I add this image into the background of the `body` element

```css
body {background-image:url(images/grid_18px.png);}
```

and it tiles itself across and down the page (Figure 4.21).
With the horizontal lines of the grid in the background, I now start positioning the text elements, using the grid as a guide.

For this example, I use just a few common text elements but it’s easy, once you get the hang of how this works, to build a text style sheet with a full set of “grid-aligned” HTML text elements that you can use as the basis for all your sites.

I’ll start with a simple paragraph

```html
<p>In traditional typography, text is composed…</p>
```

and this CSS

```css
* {margin:0; padding:0;}
body {
  background-image:url(images/grid_18px.png);
  font:100% helvetica, arial, sans-serif;
  margin:0 40px 0;
}
p {
  font-size:13px;
  line-height:18px;
}
```

Note that I match the text’s line-height to the grid distance: 18 pixels. With all default margins and padding removed, I now know every line will be 18 pixels apart (Figure 4.22).
Next I add 4 pixels of padding to the container, body, to push this element down and align the baseline of its text with the grid. Once this first element aligns to the grid, it will be easy to get the elements that follow it to do the same. Actually, I’ll add 22 pixels (4 + 18) to also give an empty line of breathing space at the top, by adding this declaration to body.

```css
padding-top:22px;
```

While I’m at it, I’ll add this declaration onto the paragraph:

```css
p {
  font-size:13px;
  line-height:18px;
  margin-bottom:18px;
}
```

This will create exactly one empty grid line between each paragraph. Adding another paragraph will help show the effect of these two changes (Figure 4.23).
Now that the text and grid are aligned, and the paragraphs are correctly spaced, I’ll set font sizes for the other text elements. I start with the h3 tag, which I set at 18 pixels. Of course, it too will have a line-height of 18 pixels so that it occupies exactly one line of the grid. To test its spacing, I’ll insert it in the markup between the two paragraphs.

```html
<p>In traditional typography, text is composed…</p>
<h3>Type for Every Use</h3>
<p>The ubiquity of type has led typographers…</p>
```

Here’s the CSS for the new heading:

```css
h3 {font-size:18px; line-height:18px;}
```

As you can see, the headline sits a couple of pixels below the baseline but, surprisingly, does not push the following paragraph down by the same amount (Figure 4.24). The reason is, that while the headline’s line-height is correct, at this size and with this font the text is slightly offset within it. Here’s how to correct this.

**Figure 4.24** The baseline of the h3 element’s text sits slightly below the grid line.

**Figure 4.25** A small negative top margin and an equal positive bottom margin pulls the headline up into perfect alignment on the grid.
The negative top margin pulls the type up, and the same amount of positive bottom margin offsets this change to keep the element that follows exactly where it was (Figure 4.25).

A second and similar alignment technique is needed for those elements, usually headings, that are larger than the grid distance. To illustrate, I'll next add a 24 pixel \texttt{h1} headline. Obviously, 24-pixel text is going to occupy more than one line of the grid, so in this case I'll set the \texttt{line-height} to span two lines—36 pixels. I'll put the \texttt{h1} element where it usually appears—the first element on the page.

Let's start with this CSS:

\begin{verbatim}
\texttt{h1} \{font-size:24px; line-height:36px;\}
\end{verbatim}

This big headline sits uncomfortably between two lines (Figure 4.26). Its descenders will touch the paragraph text if I move it down onto the nearest line, so I'll move it up instead. With a little trial and error, I determine this distance to be 13 pixels.

\begin{verbatim}
\texttt{h1} \{font-size:24px; line-height:36px; margin-top:-13px; margin-bottom:13px;\}
\end{verbatim}

This \texttt{h1} now has some white space below it to set it off from the text (Figure 4.27). I could do this with the smaller headline too, but I think it looks better close to the element that follows it.
To finish this exercise, I’ll add some different sized headings, an unordered list, and a *blockquote* to show what a more complete page looks like once the grid is removed.

```css
* {margin:0; padding:0;}
body {font:100% helvetica, arial, sans-serif; background-image:url(images/grid_18px.png); margin:0 20px 0; padding:21px;}
p {font-size:14px; line-height:18px; margin-bottom:18px;}
h1 {font-size:24px; line-height:36px; margin-top:-13px; margin-bottom:13px;}
h2 {font-size:18px; line-height:18px; margin-top:-2px; margin-bottom:2px;}
h3 {font-size:16px; line-height:18px; margin-top:-2px; margin-bottom:2px;}
ul {margin-bottom:18px;}
li {font-size:13px; list-style-type:none; padding:0 20px; line-height:18px;}
a {color:#777; text-decoration:none;}
blockquote {font-size:12px; line-height:18px; padding-top:2px; margin-bottom:16px;}
```
As you can see in Figure 4.28, there's something pleasing about a page that's laid out on a grid. From a technical perspective, if you style your type on a grid-based layout for a site where the content will be managed by others, then the page will always lay out nicely, regardless of the order of the elements.

**An Exercise in Classic Typography**

To end this chapter, I'll lay out a small excerpt of *The Hound of the Baskervilles* (edited for the purpose of this example), using many of the font and text properties you have seen in this chapter. You'll see a number of techniques to achieve high-quality typography,
including the use of HTML entities, letter and word spacing, drop caps, a vertical grid (this time 24 pixels), and downloaded fonts.

The markup is quite simple: two headings, a number of paragraphs, and a blockquote.

```html
<h2>an excerpt from</h2>
<h1>The Hound of the Baskervilles</h1>

<p>Holmes stretched out his hand for the manuscript and flattened it upon his knee. "You will observe, Watson, the alternative use of the long s and the short. It is one of several indications which enabled me to fix the date." At the head was written: "Baskerville Hall," and below in large, scrawling figures: 1742.

<p>"It appears to be a statement of some sort."</p>

<p>"Yes— it is a statement of a certain legend which runs in the Baskerville family."</p>

<blockquote>
"Of the origin of the Hound of the Baskervilles there have been many statements, yet as I come in a direct line from Hugo Baskerville, and as I had the story from my father…"
</blockquote>

<p>When Dr. Mortimer had finished reading this singular narrative he pushed his spectacles up on his forehead and stared across at Mr. Sherlock Holmes.</p>

You can see in this markup that I have highlighted instances of the four different HTML entities that I am using for the punctuation, specifically left double-quote (&ldquo;) to open dialogue, right double-quote (&rdquo;) to close dialogue, ellipsis (…) for omission, and em dash (—) for the long dashes that indicate pauses or as a replacement for parentheses.

**STEP 1—SETTING THE FONTS AND THE UNDERLYING GRID**

I’m using the FontSquirrel font Crimson Roman for the overall text in this example. I downloaded the font kit, put it on my Web server
(and also stored it locally for development), and I added the provided @font-face rule to my style sheet. I can then specify it in a font-family rule.

@font-face {
  font-family: 'CrimsonRoman';
  src: url('fonts/Crimson-fontfacekit/Crimson-Roman-webfont.eot');
  src: url('fonts/Crimson-fontfacekit/Crimson-Roman-webfont.eot?#iefix') format('embedded-opentype'),
  url('fonts/Crimson-fontfacekit/Crimson-Roman-webfont.woff') format('woff'),
  url('fonts/Crimson-fontfacekit/Crimson-Roman-webfont.ttf') format('truetype'),
  url('fonts/Crimson-fontfacekit/Crimson-Roman-webfont.svg#CrimsonRoman') format('svg');
  font-weight: normal;
  font-style: normal;}

* {margin:0; padding:0;}

**HTML Entity Reference**

These Web sites provide tables that list the HTML entities.


The first URL above includes both HTML entity values, and the hex values that you need when adding entities as content in the ::before and ::after pseudo-elements. For example, a hex code shown in the table as &amp;#x201C; (left double-quote) needs to be modified for pseudo-element content, like this
e::before {content:'\201C';}

Note the backslash in front of the number. The hex value for the right double-quote is \201D.

Within the text of your HTML content, you must **always** replace all ampersands and all < (less-than) symbols with their HTML entities, which are & and > respectively. This is because “&” is reserved for the first character of HTML entities, and “<” is reserved for the first character of HTML tags. That’s what browsers expect to see when they encounter these characters.
I follow my standard procedure of removing all margins and padding, assigning the primary font, and adding left and right margins; I also add the temporary grid for aligning the type, as shown in Figure 4.29.

STEP 2—STYLING THE HEADINGS

I now work my way, element by element, down the page, aligning each element with the grid as I go. I want the first, minor headline to contrast with the main heading, which I plan to style in a large cursive font, so I’ll style this smaller heading in widely-spaced small capital letters.
For this heading I first use the `font-variant` property to convert the text to small caps and then apply the `word-spacing` and `letter-spacing` properties to get the look I want (Figure 4.30).

Next, I go to Google Web Fonts, where I find a cursive font called Pinyon that has a styling compatible with my subject matter. I copy the link tag generated by Google Web Fonts into the head of my HTML document so I can now reference the font in a `font-family` declaration. I again need my little negative/positive margin trick that I first showed in Figure 4.25 to pull the type into exact alignment with the grid. Figure 4.31 shows the result.

Note the `line-height` is set to a four-times multiple of the grid distance.

For this heading I first use the `font-variant` property to convert the text to small caps and then apply the `word-spacing` and `letter-spacing` properties to get the look I want (Figure 4.30).

```
<link href='http://fonts.googleapis.com/css?family=Pinyon+Script' rel='stylesheet' type='text/css'>

h1 {font-size:60px; line-height:96px; font-family:"Pinyon Script", cursive; margin:4px 0 -4px; text-align:center; font-weight:normal; position:relative;}
```

The Hound of the Baskervilles

FIGURE 4.30 The small heading is now aligned with the grid and styled with a mix of font and text properties.

FIGURE 4.31 The large headline is now styled and aligned with the grid.

Typography Resources

http://ilovetypography.com On this site you can follow the typographic musings of designer John Boardley and enjoy the unique typographic treatments on each page.

http://www.thinkingwithtype.com This is the Web site for the book, Thinking with Type by Ellen Lupton. The site features beautiful and classic typographic examples along with information on letterforms and type families.

http://webtypography.net This site calls itself The Elements of Typographic Style Applied to the Web—A practical guide to web typography. It is neatly organized by a Table of Contents that lists common typographic considerations and tips.
STEP 3—STYLING THE PARAGRAPH AND THE BLOCKQUOTE

The first paragraph is sitting a little high, so let’s set its font size, and most importantly, its line height.

```
p {font-size:18px; line-height:24px;}
```

While the first three paragraphs are now aligned with the grid, the subsequent paragraphs are not, because the line height of the blockquote that follows the first paragraph also needs to be aligned (Figure 4.32). The blockquote text is wrapped in an inline q (quote) tag, which by default, adds quote marks at its beginning and end. I’ll indent the containing blockquote, but I’ll set the font size and line height on the child quote element, because it contains the text.

```
blockquote {margin:0px 20%;}
q {font-size:18px; font-style:italic; line-height:24px;}
```

```
 Holmes stretched out his hand for the manuscript and flattened it upon his knee. “You will observe, Watson, the alternative use of the long s and the short. It is one of several indications which enabled me to fix the date.” At the head was written, “Baskerville Hall,” and below in large, scrawling figures: “1742.”

“It appears to be a statement of some sort.”

“Yes—it is a statement of a certain legend which runs in the Baskerville family.”

“Of the origin of the Hound of the Baskervilles there have been many statements, yet as I come in a direct line from Hugo Baskerville, and as I had the story from my father...”

When Dr. Mortimer had finished reading this singular narrative he pushed his spectacles up on his forehead and stared across at Mr. Sherlock Holmes.
```

FIGURE 4.32 Setting the paragraph line-height aligns it with the grid.

FIGURE 4.33 The block quote is now aligned with the grid.
Indenting and italicizing the quotation adds variation to the page. Notice that with the `blockquote` correctly positioned, the subsequent paragraphs fall into place, too (Figure 4.33).

**STEP 4 — ADDING A DROP CAP**

Next, I’ll add a distinctive drop cap on the first character of the first paragraph. A drop cap is a large letter that starts a paragraph. There are a number of variations on drop caps, but in this case, I’ll align its top edge with the top of the first line of the paragraph and its bottom edge with the baseline of the third line.

Typically, when you see this technique, the first letter is wrapped in a `span`, but this is not feasible or reliable in a site where text is supplied from a content management system. The technique I show here requires no modification to the markup.

I need to select the first letter of the first paragraph that follows the `h1` headline. I do this with a combination of two selectors: the `::first-letter` pseudo-element in combination with a sibling `+` selector. Once selected, that character can be enlarged and floated into position.

```
h1 + p::first-letter {font-family:times, serif; font-size:90px; float:left; border:1px solid;}
```

The first letter is now enlarged, but is not where I want it to be positioned (Figure 4.34). I’ve turned on the pseudo-element’s border as a guide, because what I really need to size and position is its box. The border shows me that this box is only large enough to force two lines of the paragraph to wrap around it because the box is inheriting its size and alignment from the paragraph. I need to set the pseudo-element’s `line-height` so its box encloses the drop cap.
The drop cap is now in place, but I'd like to see a visual transition between the large drop cap and the small paragraph text, so I'll add a small cap style to the entire first line of the paragraph text.

```
.step5stylingthefirstline

h1 + p::first-line {font-variant:small-caps; letter-spacing:.15em;}
```

The drop cap is now tightly surrounds the drop cap.

![Image](image1.png)

**Figure 4.35** The element box now tightly surrounds the drop cap.

With the box `line-height` increased, the third line now also wraps (Figure 4.35). All that is left to do is add a little padding to the top of the element to push it down and align the bottom edge of the letter with the baseline of the paragraph's third line.

```
.step5stylingthefirstline

h1 + p::first-letter {font-family:times, serif; font-size:90px; float:left; line-height:.65; border:1px solid; padding:.085em 3px 0 0;}
```

Note that I also add 3 pixels of right padding to give visual space between the drop cap and the paragraph text. Also, I no longer need the border so I remove it (Figure 4.36).

![Image](image2.png)

**Figure 4.36** The drop cap is now complete.

**Step 5—Styling the First Line**

The drop cap is now in place, but I'd like to see a visual transition between the large drop cap and the small paragraph text, so I'll add a small cap style to the entire first line of the paragraph text.
I again use the sibling selector, this time in combination with the `::first-line` pseudo-element to set the first line of the paragraph in small caps. By using the pseudo-element, rather than just typing the first few words in capital letters, the capitalization will adjust if the line length changes. You can see how nicely this styling connects the drop cap with the rest of the paragraph in Figure 4.37.

**STEP 6—FINISHING TOUCHES**

Without some space between the paragraphs, it’s hard to see where each one starts. In keeping with convention in books, rather than space the paragraphs apart, I’ll instead add a small indent to every paragraph that follows a paragraph; a paragraph that starts a sequence of paragraphs doesn’t need the indent. Also, I don’t like the anemic quotation marks around the quote, so I will update the default `::before` and `::after` pseudo-elements of the `q` tag and insert nicer ones from the Crimson font. Finally, I am done with the grid so I’ll remove that from the `body` tag (not shown).

```css
p + p {text-indent:14px;}
q::before {content: "\201C"}
q::after {content: "\201D"}
```

Both these small changes are worth comment. The indents are achieved with the sibling selector, which cause only paragraphs that follow a paragraph to be indented. The paragraph that begins “When Dr. Mortimer...” follows the `blockquote`, not a paragraph, and so does not get the indent, giving a solid lead off to that paragraph by aligning it with the margin (Figure 4.38).

The quotation marks added by the `::before` and `::after` pseudo-elements have to be defined with hex entities. I cannot use the normal HTML entities in the content value, as they don’t work in this situation; only hex entities work here, and only in a slightly modified state. See the sidebar, *HTML Entity Reference*, earlier in this chapter, for details.
With these final touches, the layout is complete. It may seem like a lot of work for a small excerpt like this, but of course, these styles could easily be applied to the entire book.

Summary

In this chapter, you have seen the many CSS properties associated with fonts and text, seen the different ways that fonts can be specified for your pages, and looked at various approaches to styling text. Next, I will expand the text layouts you have just seen, and look at techniques to create multi-column page layouts.
Index

+ (adjacent sibling selector), 33
& (ampersand), 19, 142
* (asterisk), 215
> (child selector), 32–33
> *("child-star" selector), 166
/ (forward slash), 2
= (general sibling selector), 33–34
# (hash symbol), 38–39, 58
* (star selector), 34–35, 166
- (tilde), 33
* (universal selector), 34–35
: (colon), 44
, (comma), 27
:: (double colon), 44
. (period), 36, 38
; (semicolon), 27
< > (angle brackets), 2

A
a (anchor) tag, 9
A (alpha) value, 61
abbr tag, 20
abbreviations, 20
absolute positioning, 87–88
absolute values, 56
action attribute, 206
Active state, 44
adjacent sibling selector (+), 33
Adobe Typekit, 127
::after pseudo-element, 48–49
alpha channels, 61
alpha (A) value, 61
alt attribute, 42
alt attribute text, 4
alt tags, 4, 42
&amp; entity, 19
ampersand (&), 19
ancestors, 21, 22
anchor (a) tag, 9
Android devices, 274
angle brackets (< >), 2
animation, 222
APIs (Application Programming Interfaces), x
Apple Retina screens, 275
Application Programming Interfaces (APIs), x
article elements, 154, 182
article tag, 2, 28–29
asterisk (*), 215
attribute name selector, 41–42
attribute selectors, 28, 41–43
attribute value selector, 42
attributes, 2
class, 35–41
input, 208
names, 41–42
overview, 4
type, 208
author style sheets, 51
auto, 70

B
background images, 95–96, 102
background properties, 94–103
background-attachment property, 100–101
background-break property, 102
background-clip property, 102, 194
background-color property, 95
background-image property, 95–96
background-origin property, 102
background-position property, 97–99
background-repeat property, 96–97
backgrounds, 93–109
attached, 100–101
color, 95
fallbacks, 295–296
gradients, 104–107
multiple, 102
position, 97–99
properties, 94–107
repeat options, 96–97
shorthand, 101
size, 99–100
background-size property, 99–100
::before pseudo-element, 48–49
block elements, 10–16, 93
block property, 10–16
block quotes, 146–147
blockquote tag, 18–19
blog links, 249, 258–260
body element, 16, 112
body tag, 8, 22, 110
bold text, 20, 115, 116
book area, 260–268, 281
borderBoxModel.js polyfill, 298
border property, 62, 63–66
borders
boxes, 62, 63–66
color, 95
columns, 161–171
transparent, 194, 229
box model, 62–70, 171
context
  positioning, 90–92
  stacking, 227–228
contextual selectors, 28–35, 41
controls, 202, 207–208, 209
CSS
  “Cascade” mechanism in, 50–54
  fallbacks, 295–296
  how it works, 23–61
  methods for adding, 25
  order, 292
  organizing, 292–293
  purpose of, 1, 23
  selectors, 182–184
  testing code, 293–294
  writing, 291–293
CSS declarations. See declarations
CSS rules
  anatomy of, 24–28
  Cascade mechanism, 52–54
  described, 24
  inheritance, 49–50
  multiple, 27–28
  naming conventions, 26–28
  order, 293
  organizing, 293
  sort rules, 52–54
  specificity, 52–54
CSS styles. See styles
CSS triangle, 228–230
CSS3
  browser support, xi, 295–298
  fallbacks, 295–296
  improved features, xi
  polyfills, 297–298
  table properties, 177–179
  transforms, 263, 264
CSS3 example
  book area, 260–268
  completed page, 232
  feature area, 249–260
  footers, 268–271
  headers, 236–248
  page structure, 231–236
  transitions, 221–223, 248
  web pages, 241–236
CSS3Pie polyfill, 297

D
debbuging, 293–294
declarations, 55–61
  multiple, 27, 291–292
organizing, 292
  structure, 26–27
  weight of, 52
descendents, 21, 22
devices. See smartphones; tablets
display properties, 10–16
display property, 62, 92–93
displaying elements, 92–93
DOCTYPE, 7
document flow, 10
Document Object Model. See DOM documents. See HTML documents
DOM (Document Object Model), 20–22
DOM hierarchy, 21–22, 29–30
don’t repeat yourself (DRY) principle, 268
double colon (::), 44
drop caps, 147–148
drop-down menus
  adding to menus, 246–248
  lists in, 191–201
  on touch screens, 288–289
DRY (don’t repeat yourself) principle, 268

E
eslastic layouts, 152
elements. See also tags
  attributes, 2
  backgrounds, 93–109
  block, 10–16, 93
  boxes. See boxes
  centering, 244–246, 271
  cleared, 75, 76, 80–85
  compound, 5–6
  display property, 92–93
  floated, 75, 76–85
  inline, 10–16, 93
  length, 56
  nested, 16–20, 236
  non-floated, 80–82
  order, 292
  oversized, 171
  positioning. See positioning elements
  pseudo-elements, 47–49, 149
  required, 7
  stacking order, 227–228
  visibility, 93
em (emphasis) tag, 6, 20, 31–32
em value, 56, 57
embedded styles, 25
emphasis (em) tag, 6, 20, 31–32
emphasis text, 20
enclosing floated elements, 71
enclosing tags, 2–3
error messages, 207, 215–216, 220, 258
ex value, 56, 57

F
fallbacks, 295–296
faux columns, 176
feature area, 249–260, 281
fieldset element, 206–207
files. See HTML documents
Firebug, 294
:first-child pseudo-class, 46
:first-letter pseudo-element, 47
:first-line pseudo-element, 47–48
fixed positioning, 89
fixed-width layouts, 151–152, 154
float property, 62, 75, 76–85
floated elements, 75, 76–85
float-slip, 161
fluid layouts
described, 152
fluid center, 172–179
smaller devices and, 274, 278–282
:focus pseudo-class, 45
font libraries, 127–128
font properties, 109–117
font property, 117
Font Squirrel, 127, 129
font stacks, 110–111, 129
@font-face kits, 127, 128–130
@fontface rule, 126, 128
font-family property, 109–112
fonts, 108–117. See also text
general sibling selector (~), 33–34
browser issues, 114, 128–129
color, 27
generic, 110–111, 112
Google Web Fonts, 127–128, 239
inherited, 49–50, 110, 119
installed, 110–112
names, 110–111
overview, 108
size, 112–115, 132
style, 115, 116
variants, 116–117
vs. text, 109
Web, 126–130
weight, 116
x-height, 56, 111
font-size property, 112–115, 117
font-style property, 115, 116
font-variant property, 116–117
font-weight property, 116
footers, 268–271
form data, 201
form element, 205–206, 212
form elements, 205–209
forms, 201–223
checkboxes, 203–204, 208, 210, 217
considerations, 201
controls, 202, 207–208, 209
error messages, 207, 215–216, 220
example, 201–206
fields, 207
fieldsets, 206–207
HTML elements of, 201–209
HTML5 and, 201
labels, 202, 207–208, 209, 210
legend, 202, 206–207, 212
markup strategies, 209–210
processing, 201, 206
radio buttons, 202, 204, 208–210, 217
required fields, 215–216
resources, 201, 206
search, 239–242
search field, 221–223
selects, 208, 209
sign-in, 253–258
styling, 210–220
submitting, 206
validating data, 201
forward slash (/), 2

G
general sibling selector (~), 33–34
Google Web Fonts, 127–128, 239
gradients
backgrounds, 104–107
linear, 104–105
radial, 104, 106–107

H
h1 tag, 21–22
hash symbol (#), 38–39, 58
hash tags, 38
head tag, 8
headers
element, 157–160
smaller devices, 279–287
styling, 236–248
headings, 3, 5, 12
headlines, 8–10, 144–145
hexadecimal notation, 58, 60
Hicks, Ian, x
hidden property, 93
high-resolution screens, 275
hooks, 182–183, 184
hosted font libraries, 127–128
hot area, 188–189
Hover state, 43
href (hyperlink reference) tag, 9
href value, 38
HSL color, 59–60
HTML attributes. See attributes
HTML code
considerations, 291
debugging, 293–294
hierarchy, 21
testing, 293–294
HTML comments tag, 8
HTML documents
anatomy of, 7–20
document flow, 10
structure of, 7
HTML elements. See elements
HTML entities, 19, 143
HTML markup
basics, 2–6
nesting examples, 18–20
planning, 232–236
presentational effect, 164
purpose of, 1
template, 7–10
html tag, 7
HTML tags. See tags
HTML templates, 7–10
HTML5
advantages of, x, 7
"bare-bones" template, 7–10
browser support, 295–298
closing tags, 3
considerations, 1–2, 253
forms and, 201
polyfills, 297–298
syntax overview, 3
htmlshiv.js polyfill, 297
hue, 59–60
hyperlink reference (href) tag, 9
hyperlinks. See links
I
"ICE" formula, 53–54
ID attribute, 35–41
ID selectors, 28
IDs, 35–41
in-page navigation links, 38
JavaScript-related, 39
names, 35
purpose of, 39, 41
vs. classes, 39–41
when to use, 39–40
IE9.js polyfill, 297
image tag (img), 4, 9
images
adding, 9–10, 13
alternative text for, 4
background, 95–96, 102
failure to load, 4
flexible, 274
floating text around, 76–78
as inline elements, 10, 13–15
source of, 4
img (image tag), 4, 9
!important declaration, 52
inheritance, 49–50, 110, 119
inherited values, 119
inline element boxes, 16
inline elements, 10–16, 93
inline property, 10–16
inline styles, 25
inline-block value, 245
inner divs, 164–168, 184–185
in-page navigation links, 38
input attributes, 208
input element, 203, 208
inset keyword, 241
interface components, 186–230
considerations, 186
forms. See forms
lists. See lists
menus. See menus
popup overlays (tooltips), 224–230, 263–268
selection path, 200–201
iPad, 272–274, 278–282
iPhone, 272–274, 275, 285–287
Irish, Paul, 129, 171, 297
italic text, 20, 115

J
JavaScript, 39, 295, 297
jQuery functions, 289

K
keyword colors, 57–58
labels, 202, 207–208, 209, 210
:last-child pseudo-class, 46
layouts. See page layouts
leading, 123–124
legends, 202, 206–207, 212
letter-spacing property, 119–121
line-height property, 117, 123–124, 134
Link state, 43
link tag, 276–277
linked styles, 25
links. See also URLs
  blog, 249, 258–260
color, 40, 44
hot area, 188–189
in-page navigation links, 38
pseudo-links, 43–47
states, 43–45
underlining and, 122
lists
  in drop-down menus, 191–201
  in horizontal menus, 189–191
  hot areas, 188–189
  nested, 191–201
  ordered, 5–6
  unordered, 5, 258
  in vertical menus, 186–189
luminance, 59–60

M
Marcotte, Ethan, 274
margins
  auto, 271
  boxes, 63, 67–70
  columns, 161, 172–177
markup. See HTML markup
media attribute, 276–277
media queries, 272, 274–278
@media rule, 274–276
media types, 276
menus, 186–201
  centering, 244–246
  described, 186
drop-down. See drop-down menus
element, 242–248
horizontal, 189–191
opacity, 247–248
selection path highlight, 200–201
submenus, 196–197
third level, 197–200
vertical, 186–189
meta tag, 8, 278
method attribute, 206
Meyer, Eric, 68
mobile devices. See smartphones; tablets
Modernizr, 101, 248, 288, 297
multiple background images, 102
multiple classes selector, 37–38

N
naming conventions, 26–28
nav tag, 1–2
navigation, 38, 234
navigation menus, 186–201
nested elements, 16–20, 236
nested tags, 6, 16–20
nesting examples, 18–20
non-enclosing tags, 3
non-floated elements, 80–82
normal value, 115
“not-first-child” selector, 187–188
:nth-child pseudo-class, 46–47
numbers, 35
numerical values, 55, 56–57

O
opacity, 247–248
opening tags, 2–3
overflow: hidden, 79–80, 171

P
p tag, 21–22
padding
  boxes, 63, 66–67, 68
  columns, 161–171
page layout, 151–185
  basic concepts, 151–153
columns. See columns
elastic layouts, 152
fixed-width layouts, 151–152, 154
fluid. See fluid layouts
inner divs, 184–185
layout height/width, 152–153
optimizing for smartphones, 282–287
optimizing for tablets, 278–282
templates, 7–10
text layout example, 131–135
page styles, 25
paragraphs, 5, 8–10, 12
parent elements, 21, 22, 80
percentage widths, 278
percentages, 57, 59
period (.), 36, 38
pixels, 56
polyfills, 170, 171, 288, 297–298
popup overlays, 224–230, 263–268
portrait format, 285–287
position property, 62, 85–92
positioning context, 90–92
positioning elements, 62–107
absolute positioning, 87–88
backgrounds. See backgrounds
boxes. See boxes
cleared elements, 75, 76, 80–85
display property, 92–93
fixed positioning, 89
floated elements, 75, 76–85
position property, 85–92
positioning context, 90–92
relative positioning, 87
static positioning, 86
types of positioning, 86–92
-prefix-free polyfill, 298
presentational markup, 164
properties. See also specific properties
animated, 222
background, 94–107
font, 109–117
normal value and, 115
table, 177–179
text, 109, 117–126
pseudo-classes, 43–47
pseudo-elements, 47–49, 149

Q
quirks mode, 70
quotation marks, 19, 149
quotations, 18, 147

R
radial gradients, 106–107
radio buttons, 202, 204, 208–210, 217
radiused corners, 239, 240, 295
referenced content, 3
relative positioning, 87
relative units, 56, 114
rem (root em) units, 114–115
reset.css style sheet, 68
Respond.js polyfill, 298
responsive design, 272–290
considerations, 272, 290
elements of, 274
media queries, 272, 274–278
portrait format, 285–287
smartphones, 272–274, 282–287
tables, 272–274, 278–282
touch screens, 288–289
returns, 291
RGB color, 58–59, 60
RGB percentages, 59
root em (rem) units, 114–115
root level tag, 7
rows, 47, 179–185, 218
rule declarations. See declarations
rules. See CSS rules
S
Safari Mobile scaling bug, 287
saturation, 59–60
scaling issues, 287
search engines, 48
search form, 221–223, 239–242
section tag, 21, 22
selection path, 200–201
selectivizr polyfill, 297
selectors. See also specific selectors
attribute, 28, 41–43
class, 28, 36
contextual, 28–35, 41
grouping, 27
ID, 28
multiple, 27
specificity of, 52–54
structure, 26–28
tips for using, 182–184
semantic markup, 1
semicolon (;), 27
shadows, 239, 241, 254
Shapira, Isaac, 289
shorthand styling, 64
siblings, 21, 22, 33–34
sign-in form, 253–258
smartphones
optimizing layout for, 282–287
overview, 272–274
SnapzProX, 278
sort order, 52–54
sort rules, 52–53
spaces
nesting and, 18
white space, 27, 291, 292
spacing
letter, 119–121
word, 121
span tag, 37
specificity, 52–54
stacking context, 227–228
star selector (*), 34–35, 166
states, links, 43–45
static positioning, 86
stop points, 105–106
strong tag, 20
structural pseudo-classes, 43, 46–47
style attribute, 25
style sheet styles, 25
style sheets
author, 51
linked, 25
tips for, 293
user, 51
style tag, 24, 26
styles
adding to documents, 25
considerations, 25
embedded, 25
fonts, 115, 116
forms, 210–220
inherited, 49–50
inline, 25
linked, 25
page vs. style sheet, 25
sources of, 50–51
stylinwithcss.com, xii, 23, 26, 183, 299
symbols, 35

table properties, 177–179
tables, 177–179
tablets
optimizing layout for, 278–282
overview, 272–274
tabs, 291
tags. See also attributes; elements; specific tags
closing, 3
considerations, 1
enclosing, 2–3
hierarchy, 21–22
names, 2
nested, 6, 16–20
non-enclosing, 3
opening, 2–3
root level, 7
tag-with-class selector, 36–37
:target pseudo-class, 45–46
technical notes, 291–299
testing code, 293–294
text, 117–150. See also fonts
abbreviations, 20
aligning horizontally, 122–123, 271
aligning vertically, 125–126
basic layout example, 131–135
bold, 20, 115, 116
browser issues, 114, 128–129
capitalization, 116–117, 124–125
captions, 224–229
centering, 167, 271
classic typography, 141–150
decoration values, 122
drop caps, 147–148
enclosing tags for, 2–3
floating around images, 76–78
indent, 118–119, 120
italics, 20, 115
leading, 123–124
letter spacing, 119–121
line height, 117, 123–124, 134
properties, 109, 117–126
rotated, 263, 264
shadows, 254
superscript/subscript, 125–126
underlined, 122
using grids, 135–141
vs. fonts, 109
word spacing, 121
word wrapping, 171
text boxes, 118, 120. See also boxes
text snake, 120
text-align property, 122–123, 271
text-decoration property, 122
text-indent property, 118–119
text-transform property, 124–125
tilde (~), 33
title area, 232–236
title attribute, 42
title tag, 7
tooltips, 224–230
transform property, 263, 264
transforms, 263, 264
transitions, 221–223, 248
transparent borders, 194
type attributes, 208
Typekit, 127
typographical characters, 19
typography resources, 145

U
UI components. See interface components
UI pseudo-classes, 43–46
universal selector (*), 34–35
URLs, 44, 187. See also links
user style sheets, 51

V
vendor specific prefixes (VSPs), 103
vertical centering, 248
vertical-align property, 125–126
viewpoint meta tag, 278
visibility property, 93, 248, 288
Visited state, 43
VSPs (vendor specific prefixes), 103

W
W3C site, 52
web browsers
  browser sniffing, 295
  considerations, xi
  CSS3 and, xi, 295–298
  fallbacks, 295–296
  fonts and, 114, 128–129
  HTML5 and, 295–298
  media queries and, 274–278
  missing features, 296
  older, xi, 295–298
  Safari Mobile, 287
  smaller screens and, 272
  support for, xi, 295–298
Web Developer extension, 14
Web fonts, 126–130
Web Inspector, 293–294
web pages
  book area, 260–268, 281
  CSS-enhanced, 231–271
  example, 231–236
  feature area, 249–260, 281
  footers, 268–271
  headers, 236–248
  optimizing for smartphones, 282–287
  optimizing for tablets, 278–282
  planning HTML markup, 232–236
  responsive design, 272–290
  on smartphones/tablets, 272–274
  structuring, 231–236
  title area, 232–236
Web sites
  box model resources, 63, 171
  color resources, 57, 61
  CSS3 transition resources, 222
  font resources, 111, 113, 127, 129, 130
  form resources, 201, 203, 206, 208
  HTML entity resources, 143
  HTML resources, 4, 19
  menu resources, 289
  mobile device resources, 272, 274, 275–278
  polyfill resources, 288, 298–299
  pseudo-element information, 44
  quirks mode, 70
  reset.css style sheet, 68
  stylinwithcss.com, xii, 23, 26, 183, 299
  typography resources, 145
  VSP information, 103
  XHTML resources, 3
Webkit redraw problem, 288
Web-safe color palette, 58
white space, 27, 291, 292
word values, 55–56
word-spacing property, 121
word-wrap, 171
wrapper elements, 154–161, 172–177
Wroblewski, Luke, 201

X
x-height, 56, 111
XHTML, 3

Z
Zeldman, Jeffrey, x, 41
z-index property, 227–228, 267