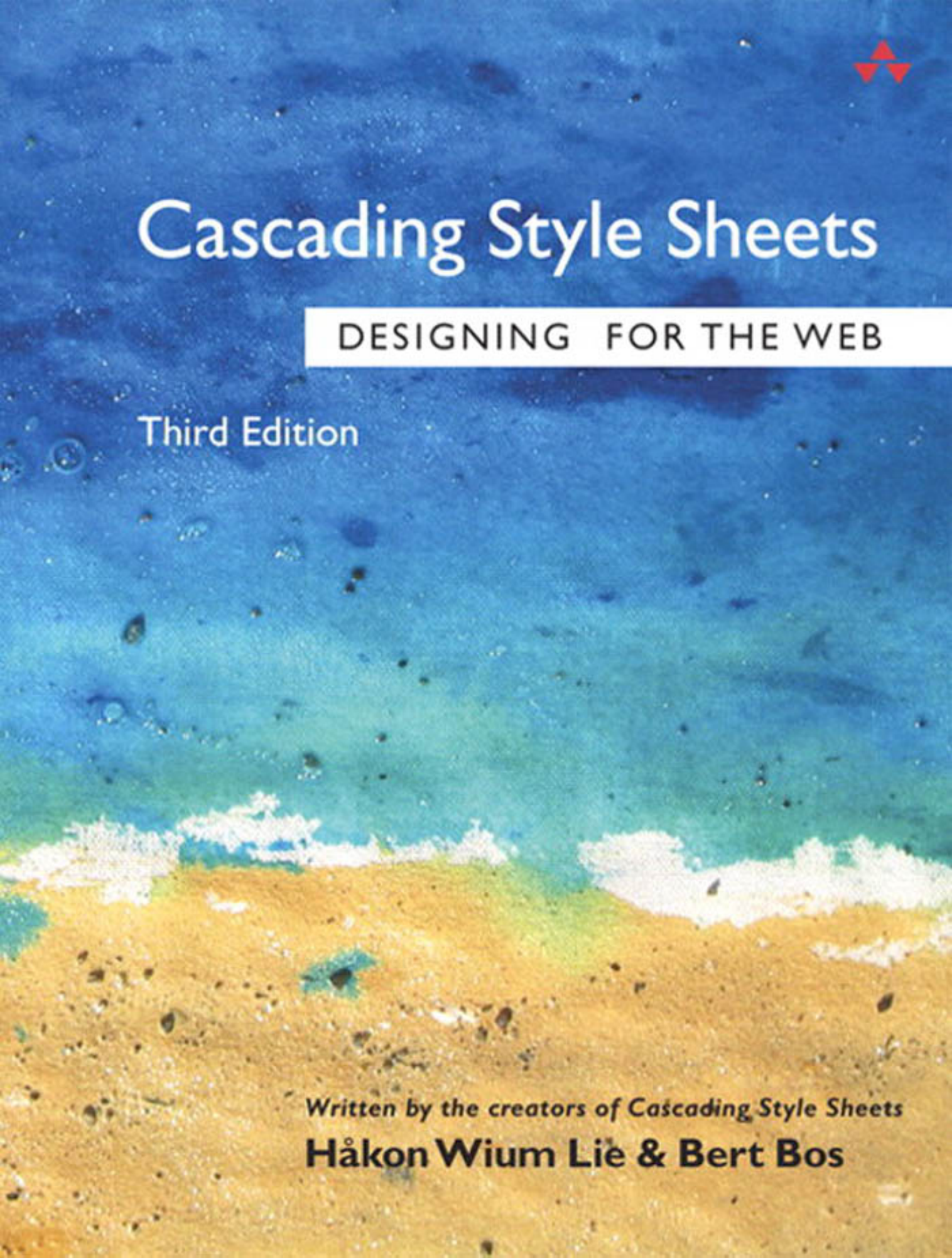
Cascading Style Sheets
DESIGNING FOR THE WEB
Third Edition
Written by the creators of Cascading Style Sheets
Håkon Wium Lie & Bert Bos

Cascading Style Sheets

DESIGNING FOR THE WEB

Third Edition

This page intentionally left blank

Cascading Style Sheets

DESIGNING FOR THE WEB

Third Edition

Håkon Wium Lie
Bert Bos

Addison-Wesley
Upper Saddle River, NJ • Boston • Indianapolis • San Francisco
New York • Toronto • Montreal • London • Munich • Paris • Madrid
Capetown • Sydney • Tokyo • Singapore • Mexico City

Library of Congress Catalog Number: 2004116047

ISBN 0-321-19312-1

Text printed in the United States on recycled paper at R.R. Donnelley and Sons Company in Crawfordsville, Indiana.

First printing, April 2005

° ° ° ° °

Foreword 2005

Watson and Crick. Fred and Ginger. Bert and Håkon. The first of these immortal duos cracked the secret of life. The second pair dazzled Depression-era audiences with gravity defying dance routines that remain the very icon of charm and grace. As for the third pair – the authors of the book you now hold in your hands – the achievement that ensures their immortality benefits everyone who uses the Web or creates content for it.

For it was Bert and Håkon's gift, not only to recognize the Web's importance long before many of us had so much as heard the word "modem," but also to recognize the danger posed by the medium's lack of a standard visual formatting language which could be abstracted from the semantics of a page's structure and content. In the early (and not-so-early) days, an unholy hodge-podge of proprietary, nonstandard tags was used to cobble page layouts together. All such Web pages were fat and slow, and the way they were built made their content unavailable to far too many would-be readers, users, or customers.

The authors of this book solved all those problems at a single stroke: namely, the invention of Cascading Style Sheets (CSS), a standard layout language for the Web. The key to their invention – the biggest thing Bert and Håkon did for us – is that CSS separates presentation from underlying structure and semantics. CSS takes the visual instructions out of HTML, where they never belonged anyway, and sticks them into one or more lean, cache-able, documents that are powerful enough to present your site one way in a traditional graphic browser, another way in a phone browser, a third way in kiosk mode, and a fourth for your printer.

If you designed or built Web pages during the 1990s, you might not have learned much about CSS. After all, in those days, the CSS standard was still in its infancy. Besides, certain browser makers (no names please) were almost pathologically reluctant to support CSS completely or even accurately. In fact, some didn't even bother to accurately or completely support HTML.

But times have changed. So has CSS: in its latest incarnation as CSS 2.1, it packs surprising power and flexibility, allowing us as designers to create almost any layout we can imagine. As to today's Web browsers, while they are not perfect (and some are more imperfect than others), their CSS support has traveled light years since the 1990s. This improved support allows anyone who understands CSS to create lean, fast-loading, content-rich Web pages that score as big with search engines as they do with readers. Best of all, the content of your pages is available to more people because it is accessible to more types of browsers and devices.

It's pretty clear: if you design or program Web pages, or create Web content, or own or manage a website, you need to know how CSS works. And who better to show you than the dynamic duo that invented CSS in the first place?

In this updated edition to their original best-selling classic, the co-creators of CSS clearly, logically, and painlessly explain the hows and whys and ins and outs of the visual formatting language that is their gift to us. The examples are simple enough for novices yet detailed enough for experts. If you missed the previous edition, you are in for a treat. If, like the rest of us, you own a dog-eared copy of the previous edition, you will appreciate how this new edition makes sense of CSS 2.1, clearing up points that have sometimes confused even the experts.

The Web would be a poorer place without Messieurs Bos and Lie. Your shelf will be richer for the addition of this book. Rely on it. Study it. Savor it.

Jeffrey Zeldman
New York
November 2004

° ° ° ° °

Foreword 1999

This Foreword was written by Robert Cailliau for the first edition of this book, which appeared in 1997. The following text is the slightly revised version published in the second edition from 1999.

When the Web was in its infancy, seven years ago or so, I felt greatly relieved at the final removal of all the totally unsolvable problems of fixed format presentation. In the young Web, there were no more pagination faults, no more footnotes, no silly word breaks, no fidgeting the text to gain that extra line that you sorely needed to fit everything on one page. In the window of a Web page on the NeXTStep system, the text was always clean. Better than that: I decided what font it came out in, how big I wanted the letters, what styles I chose for definition lists and where tabs went.

Then we descended into the Dark Ages for several years, because the Web exploded into a community that had no idea that such freedom was possible, but worried about putting on the remote screen exactly what they thought their information should look like. I've read recommendations against using structured markup because you have no control over what comes out the other side. Sad.

You have by now understood that I'm firmly in the camp of those who think that quality of content comes first, and presentation comes later. But of course, I'm not entirely right here: presentation is important. Mathematical formulas are always presented in a two-dimensional layout.

Fortunately, SGML's philosophy allows us to separate structure from presentation, and the Web DTD, HTML, is no exception. Even in the

NeXTStep version of 1990, Tim Berners-Lee provided for style sheets, though at a rudimentary level (we had other things to do then!).

Today, style sheets are becoming a reality again, this time much more elaborate. This is an important milestone for the Web, and we should stop for a minute to reflect on the potential benefits and pitfalls of this technology.

I followed the CSS effort from its inception – mostly over cups of coffee with Håkon at CERN – and I've always had one concern: is it possible to create a powerful enough style sheet "language" without ending up with a programming language?

The CSS described in this book shows that you can create some quite stunning presentations without programming. While the programmer in me may be a little disappointed, the minimalist in me is comforted. In fact, I'll never need this much freedom and special effects, but then I'm not a graphic artist. Anything that needs more complication effectively becomes an image, and should be treated as such. I feel therefore that the middle part of the spectrum between pure ASCII text and full images is effectively covered by the power of CSS, without introducing the complexity of programming.

You have here a book on presentation. But it is presentation of information that should also remain structured, so that your content can be effectively used by others, while retaining the specific visual aspects you want to give it. Use CSS with care. It is the long-awaited salt on the Web food: a little is necessary, too much is not good cooking.

The efforts of the authors have finally brought us what we sorely needed: the author's ability to shape the content without affecting the structure. This is good news for the Web!

Robert Cailliau
CERN, Geneva
January 1999

° ° ° ° °

Table of contents

Preface

Since its introduction in 1996, Cascading Style Sheets (CSS) has revolutionized Web design. Now, in 2004, most Web pages use CSS, and many designers base their layouts entirely on CSS. To do so successfully requires a good understanding of how CSS works. The purpose of this book is to describe how designers can take full advantage of CSS 2.1, which is the newly released update of the specification.

CSS's journey from an idea to a specification – and then on to a specification designers can rely on – has been long and arderous. The creator of the CSS Zen Garden (described in Chapter 11, "From HTML extensions to CSS") describes it this way:

> Littering a dark and dreary road lay the past relics of browser-specific tags, incompatible DOMs, and broken CSS support. Today, we must clear the mind of past practices. Web enlightenment has been achieved thanks to the tireless efforts of folk like the W3C, WaSP, and the major browser creators.

Indeed, we believe that the web is a more enlightened place now that CSS has matured to a stage where it can be used for advanced layouts in a range of browsers. This book tells you all you need to know to start using CSS.

ACKNOWLEDGMENTS

Creating a lasting specification for the Web is not a job for one person. That's why the two authors joined forces. Then, we found out two wasn't enough, and a W3C Working Group (which includes W3C technical staff and W3C member representatives) was formed. The CSS 2.1 specification is the product of that working group. We are indebted to Tantek Çelik and Ian Hickson, who are co-editors of the CSS 2.1 specification, and to the other members of the group. In particular, we want to thank David Baron, Jim Bigelow, Kimberly Blessing, Frederick Boland, Ada Chan, Don Day, Michael Day, Elika Etemad, Daniel Glazman, David Hyatt, Björn Höhrmann, and Kevin Lawver.

CSS 2.1 is only the most recent version of the CSS specification. The foundation for CSS was laid by CSS1 and CSS2, and many people helped write, maintain, and promote those documents. In roughly chronological order:

- David Raggett and Steven Pemberton were influential in establishing the concept of style sheets for the Web.
- Eric Meyer created the test suite for the CSS1 specification and has promoted CSS tirelessly ever since.
- Todd Fahrner created the *W3C Core Styles* (described in Chapter 14), the "acid" test, and the *Ahem* font.
- Ian Jacobs and Chris Lilley were co-editors of the CSS2 specification.
- Brian Wilson provided extensive documentation of CSS and its implementations.
- Jeffrey Zeldman has publicized widely on why standards, including CSS, are good for you.
- Dave Shea created the wonderful CSS Zen Garden.
- Michael Day and Xuehong Liu for making it possible to print CSS. Thanks to the Prince formatter, we were able to produce the book ourselves using the same methods we preach in this book.

Lastly, our thanks to Tim Berners-Lee, without whom none of this would have been possible.

Håkon Wium Lie & Bert Bos
Oslo/Antibes
February 2005

TRADEMARK NOTICE

Chapter 3

The amazing em unit and other best practices

This chapter is about writing style sheets with style. By showing you case studies and how they are constructed, we give you a sense of how CSS can encode the visual presentation you want to achieve. More importantly, if you follow the guidelines in this chapter, your documents will behave well on a wide range of Web devices. For example, they will scale gracefully from one screen size to another.

The foremost tool for writing scalable style sheets is the *em* unit, and it therefore goes on top of the list of guidelines that we compile throughout this chapter: *Use ems to make scalable style sheets.* Named after the letter "M," the em unit has a long-standing tradition in typography where it has been used to measure horizontal widths. For example, the long dash (—) often found in American texts is known as an "em dash" because historically, it has had the same width as the letter "M." Its narrower cousin (–), often found in European texts, is similarly referred to as "en dash."

Use ems to make scalable style sheets!

The meaning of "em" has changed over the years. Not all fonts have the letter "M" in them (for example, Chinese), but all fonts have a height. The term has therefore come to mean the height of the font – not the width of the letter "M."

In CSS, the em unit is a general unit for measuring lengths (for example, page margins and padding around elements). You can use it both horizontally and vertically, and this shocks traditional typographers who have always used the em exclusively for horizontal measurements. By extending the em unit to also work vertically, it has become a very powerful unit – so powerful that you seldom have to use other units of length.

Let's look at a simple example where we use the em unit to set font sizes:

```
<HTML>
  <STYLE>
    H1 { font-size: 2em }
  </STYLE>
  <BODY>
    <H1>Movies</H1>
  </BODY>
</HTML>
```

When used to specify font sizes, the em unit refers to the font size of the parent element. So, in the previous example, the font size of the **H1** element is set to be twice the font size of the **BODY** element. To find what the font size of the **H1** element will be, we need to know the font size of **BODY**. Because this isn't specified in the the style sheet, the browser must find it from somewhere else – a good place to look is in the user's preferences. So, if the user sets the normal font size to 10 points, the size of the **H1** element is 20 points. This makes document headlines stand out relative to the surrounding text. Therefore: *Always use ems to set font sizes!*

Always use ems to set font sizes!

Designers who come from a desktop-publishing background may be inclined to skip the indirection that em introduces and specify directly that the font size should be 20 points. This is possible in CSS (see the description of the **font-size** property in Chapter 5, "Fonts") but using em is a better solution. Say, for example, that a sight-impaired user sets his normal font size to `20pt` (20 points). If the font size of **H1** is `2em`, as we recommend, **H1** elements will scale accordingly and be displayed in 40 points. If, however, the style sheet sets the font size to be `20pt`, there will be no scaling of fonts and the size of headlines will have the same size as the surrounding text.

The usefulness of the em unit isn't limited to font sizes. Figure 3.1 shows a page design where all lengths – including the padding and margins around elements – are specified in ems.

Let's first consider the *padding*. In CSS, padding is space around an element that is added to set the element apart from the rest of the content. The color of the padding is always the same as the background color of the element it surrounds. In Figure 3.1, the menu on the right has been given a padding with this rule:

```
DIV.menu { padding: 1.5em }
```

Figure 3.1 All lengths on this page are specified using ems.

By specifying the padding width in ems, the width of the padding is relative to the font size of the **DIV** element. As a designer, you don't really care what the exact width of the padding is on the user's screen; what you care about is the proportions of the page you are composing. If the font size of an element increases, the padding around the element should also increase. This is shown in Figure 3.2 where the font size of the menu has increased while the proportions remain constant.

Outside the menu's padding is the margin area. The *margin area* ensures that there is enough space around an element so that the page doesn't appear cramped. This rule sets the margin around the menu:

```
DIV.menu { margin: 1.5em }
```

Figure 3.2 identifies the margin area. Again, the use of ems ensures scalable designs.

Another use of ems can be found in this book where the indent of the first line of most paragraphs is set to 1.8 em. The same value is used for the left margin of code examples, such as this:

```
P { text-indent: 1.8em }
PRE { margin-left: 1.8em }
```

So, if ems are so great, why does CSS have other units as well? There are cases when it makes sense to use other units. For example, here is a

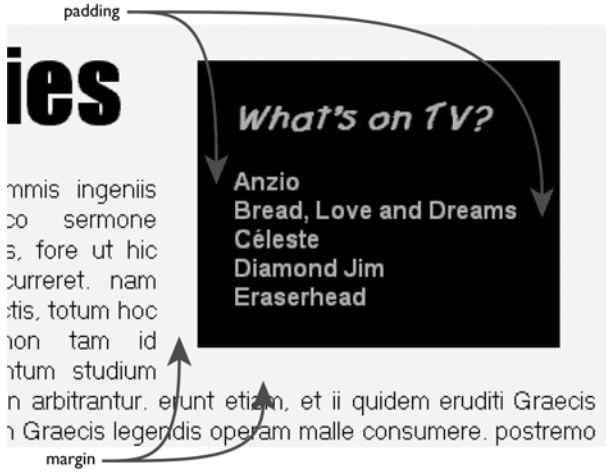

Figure 3.2 Because margins and padding are specified in ems, they scale relative to the font size.

case where percentages may work just as well, if not better: setting the margins of the **BODY** element. Remember that everything that is displayed in an HTML page is inside **BODY**, so setting the margins of that element sets the overall shape of the page. You could give the page nice wide margins on both sides with these two rules:

```
BODY {
  margin-left: 15%;
  margin-right: 10%
}
```

This makes the text 75% of the total width, and the left margin a bit wider than the right one. Try it! Your page immediately looks more professional. Percentage values set on the **BODY** element are typically calculated with respect to the browser window. So, in the previous example, the text will cover 75% of the browser window.

Use relative units for lengths!

Both ems and percentages are *relative units,* which means that they are computed with respect to something. We can distill a general rule from this: *Use relative units for lengths.* But, how about the *absolute units* in CSS – inches, centimeters, points, and picas – why are they in there at all if you never recommend to use them?

Cases may arise when you'll need to use absolute units. Say, for example, that you are creating your wedding invitations using XML and CSS. You have carefully crafted tags such as <BESTMAN> and <RSVP/>, and you plan to distribute the invitations through the Web. However, some parts of your families are not yet connected and require printed

invitations – on handmade paper, of course, and with proper margins. And 12 point fonts, exactly. This is the time to pull out the ~~obsolete~~ absolute length units: *Only use absolute length units when the physical characteristics of the output medium are known.* In practice, this happens only when you hand-tailor a style sheet for a specific printer paper size. In all other cases, you are better off using relative length units.

Only use absolute length units when the physical characteristics of the output medium are known!

A common presentation on the Web is to move elements to the sides of the page. Typically, this is achieved by using a table for layout purposes. Although you can use CSS to describe table layout (see Appendix A, "HTML 4.0 quick reference"), there is a simpler way to "put stuff on the side." In HTML, images can float; i.e., they move over to the side while allowing text to "wrap around" them. In CSS, all elements – not just images – can float. The menu in Figures 3.1 and 3.2 is an example of a floating element that has been set to float to the right side of the page. To achieve this effect, you must complete two steps. First, the element must be declared to be floating using the **float** property. Second, the element must be given an appropriate width (in ems, of course). This is done through the **width** property. Here are the two rules needed:

```
DIV.menu {
  float: right;
  width: 15em;
}
```

Use floating elements instead of tables!

By using floating text elements instead of tables, your markup can remain simple while achieving many of the visual effects that are often accomplished with tables in HTML. Thus, we have another guideline: *Use floating elements instead of tables.* Simpler markup isn't the only reason why floating elements are good replacements for tables. Flexible layout schemes are another. By changing a few lines in the style sheet which generated the page shown in Figure 3.1, we can, for example, move the menu to the left side (see Figure 3.3). Also, many text-only browsers have problems displaying tables because content within the table doesn't come in its logical order.

Put content in its logical order!

This brings us to the next guideline: *Put content in its logical order*. Although CSS allows you to move text around on the screen by means of floats and other ways of positioning, do not rely on that. By putting content in its logical order, you ensure that your document makes sense in browsers that don't support CSS. That includes browsers that work in text mode, such as Lynx, older browsers that date from before

Figure 3.3 By changing a few lines in the style sheets, you can achieve a different design.

CSS, browsers whose users have turned off style sheets, or browsers that don't work visually at all, such as voice browsers and Braille browsers. Voice browsers may actually support CSS because CSS can also describe the style of spoken pages, but aural CSS (not described in this book) doesn't allow text to be spoken out of order.

Make sure your documents are legible without style sheets!

Even a browser that supports CSS may sometimes fail to load the style sheet because of a network error. Therefore, you should always *make sure your documents are legible without style sheets.* Documents must be legible to humans, but also to Web robots and other software that try to index, summarize, or translate your documents. Also, think of the future: Five years from now, the style sheet may be lost; in 50 years, there may not be a browser that knows CSS; and in 500 years…

Test your documents on several browsers!

A good way to make sure your documents are really legible is to *test your documents on several browsers.* Alas, not all browsers that claim to support CSS do so according to W3C's specification. How much effort you should put into testing your style sheets depends on the target audience of your documents. If you publish on a closed intranet where everyone uses the same browser, your testing job will be easy. If, on the other hand, your documents are openly available on the Web, testing can be a time-consuming task. One way to avoid doing all the testing yourself is to use one of the W3C Core Styles, which are freely available on the Web (see Chapter 14, "External style sheets").

Realize that your document will end up on systems that have different fonts. CSS specifies five so-called *generic fonts* that are guaranteed to exist in all browsers: `serif`, `sans-serif`, `monospace`, `cursive`, and `fantasy`. When specifying a font family in CSS, you have the option of supplying a list to increase the chance of finding a specified font at the user's system. The last font family in the list should always be a generic font. So *always specify a fallback generic font.* This book, for example, has been set in Gill Sans. But, not everybody has a copy of that font, so we actually specified the font as

Always specify a fallback generic font!

```
BODY { font-family: Gill Sans, sans-serif }
```

This code says that the font for the document's body is Gill Sans when available, or any other sans serif font when not. Depending on your browser and your machine's configuration, you may get Helvetica, Arial, or something similar. You can learn more about setting fonts in Chapter 5.

Know when to stop!

A word of warning at the end: *Know when to stop.* Be critical when designing your style sheet. Just because you can use 10 different fonts and 30 different colors on the same page doesn't mean you have to – or should. Simple style sheets often convey your message better than overloaded ones. That single word of red in a page of black gets more attention than any of the words on a page with a dozen different fonts and colors. If you think a piece of your text deserves more attention, give it larger margins, maybe even on all four sides. A little extra space can do wonders.