treehouse

CSS3
Foundations

Ian Lunn

treehouse™

CSS3 Foundations

treehouse™

CSS3 Foundations

Ian Lunn

WILEY
A John Wiley and Sons, Ltd, Publication

This edition first published 2013

Registered office

John Wiley & Sons Ltd, The Atrium, Southern Gate, Chichester, West Sussex, PO19 8SQ, United Kingdom

For details of our global editorial offices, for customer services and for information about how to apply for permission to reuse the copyright material in this book please see our website at www.wiley.com.

978-1-118-35654-8

A catalogue record for this book is available from the British Library.

ISBN 978-1-118-35654-8 (paperback); ISBN 978-1-118-42516-9 (ebook); 978-1-118-42514-5 (ebook); 978-1-118-42515-2 (ebook)

Printed in the U.S. at Command Web Missouri

About the Author

IAN LUNN (DEVON, UK) is a Freelance Front End Developer, passionate about building the future of the web using technologies such as CSS3 and HTML5. With a Higher National Diploma in Internet Technology, Ian combines his education with expertise in CSS, HTML, JavaScript, and WordPress to build creative and effective websites and applications. Ian's enthusiasm for utilizing cutting edge technologies is matched by his dedication for advocating them; sharing his knowledge with the community, in the form of blog tutorials and open source projects.

Follow @IanLunn on Twitter (www.twitter.com/IanLunn) where he shares links and ideas about web design and development.

Publisher's Acknowledgements

Some of the people who helped bring this book to market include the following:

Editorial and Production

VP Consumer and Technology Publishing Director:
Michelle Leete

Associate Director–Book Content Management:
Martin Tribe

Associate Publisher:
Chris Webb

Assistant Editor:
Ellie Scott

Development Editor:
Brian Herrmann

Copy Editor:
Chuck Hutchinson

Technical Editor:
Nick Elliott

Editorial Manager:
Jodi Jensen

Senior Project Editor:
Sara Shlaer

Editorial Assistant:
Leslie Saxman

Marketing

Associate Marketing Director:
Louise Breinholt

Marketing Executive:
Kate Parrett

Composition Services

Compositor:
Andrea Hornberger
Jennifer Mayberry

Proofreaders:
Linda Seifert

Indexer:
Ty Koontz

Contents

part 2: Learning CSS Syntax and Adding Presentational Styles

CHAPTER THREE

CHAPTER FOUR

part 6: Preparing for Multiple Browsers and Devices

Introduction

YOU'RE READING *CSS3 FOUNDATIONS* at an exciting time. Right now, the way in which the web is styled is undergoing changes, and with the constant advancement of technology, that doesn't look to end any time soon. The web is already a beautiful place, but with more features allowing you to style a page than ever, the possibilities of what can be created are endless.

Who Should Read This Book?

CSS3 Foundations is for those completely new to styling web pages, but also makes as a great reference for those familiar with CSS too—particularly when you want to remind yourself of the newest CSS features that you may not have committed to memory.

Ideally, you have a basic understanding of HTML (Hypertext Markup Language) but it's not essential. The HTML that makes up the web page built upon throughout *CSS3 Foundations* is provided for you and described in Chapter 2. So, whether you've chosen to start learning CSS before HTML, or vice versa, you'll be gently eased into both technologies.

If you're a hobbyist, somebody looking to make a career change into the wonderful world of the web industry, an owner of a website that is need of restyling, or a web designer/developer looking to upgrade your skills to the most recent techniques and methods in use today, *CSS3 Foundations* is for you.

What You Will Learn

CSS3 Foundations aims to get you not just hitting the ground running, but hopping, stepping, and jumping too!

You'll go from the very basics of styling a web page; changing the background color, the size of text, and so on, all the way through to more advanced topics such as animations and media queries, which allow you to change a page's layout for different sizes of device.

Although the easiest topics come first and slowly progress to reach the more advanced chapters, the way in which *CSS3 Foundations* is laid out represents the workflow I, and many other people in the web industry use to create a website. So, as well as learning CSS, you'll also learn a "best practice" approach that can be applied to real-world projects, to make pages not just great to look at, but easy to update and robust enough to be viewed on a wide range of devices too (such as desktop computer and mobile phone).

You'll also learn methods and techniques not described in the CSS specification (the official technical document that explains CSS), which web designers and developers have relied on for years to make the most of CSS.

How to Use This Book

In Chapter 2, I'll introduce you to a fictitious company that has asked you to create a web page—there's no pay but the experience is golden! You'll be able to download the necessary project files to get started. This download contains updated project files for each chapter (which are sometimes broken into milestones for the bigger chapters). I'll let you know when a milestone is reached and you can either use the project files to compare to your own work, or switch to that milestone's project files in case you didn't follow the steps that came before it.

Depending on your current experience with CSS, you may decide to jump ahead to the chapters that interest you most -- that's fine, the beginning of each chapter will let you know which project files to start with.

Once you've finished reading *CSS3 Foundations,* it'll make a great desktop companion for when you need to remind yourself on how to use a feature or solve a particular problem. Each property description includes useful information for quick reference, such as the property's initial value, browser compatibility, and so on. If you're unfamiliar with those terms, I'll explain those too!

Using This Book with Treehouse

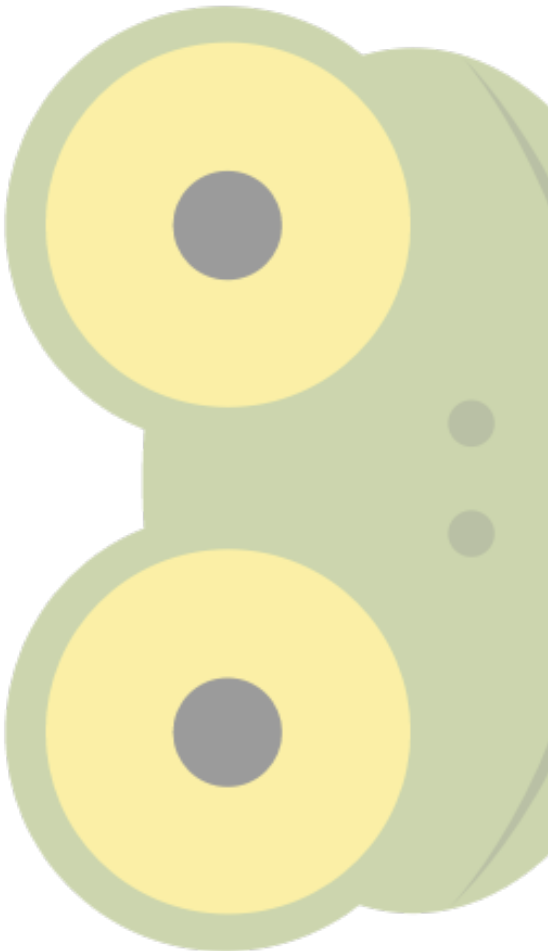
By no means do you need to be a member of Treehouse to make the most out of *CSS3 Foundations,* but the two sit nicely together. Although both aim to be extensive, there's certainly always something extra to learn.

What's more, Treehouse's video courses are often followed by fun, interactive challenges that put what you've learned to the test. You can redo a challenge as many times as you like until you feel comfortable with the topic being covered.

Treehouse also has a great community (which I am pleased to be a part of), so if at any time you have a question or want to show off your CSS skills, come and join in!

part 1

Introduction



chapter one

Understanding CSS and the Modern Web

IN THIS CHAPTER, you'll learn what the modern web is and why CSS is so important to it.

What Is the Modern Web?

Most importantly, the web today is what it has always been—accessible information. Unlike the early days of the web, though—when it was just a collection of text files—the modern web has grown to support many differing media formats and now, more than ever, many differing ways to access information.

No longer do you just sit at a desktop computer to "log on." Nowadays, you sit on a beach reading the news on a tablet device, you go to a coffee shop with laptop in hand to chat with friends in different countries, and you try to refrain from laughing at pictures of cats playing keyboards while viewing a smart phone on a train journey. The information on the web is practically infinite (more content is created than you could ever consume), and the way in which you access that information continues to grow.

The modern web is an exciting media to be a part of. It is continuously growing and so too are the technologies behind it.

What Is CSS?

Cascading Style Sheets (CSS) is a simple language defining styles that can be applied to HTML. Where HTML describes the structure of a web page, CSS describes its presentation.

An international community called the World Wide Web Consortium (W3C) writes and maintains the CSS specifications that define and standardize the way in which people should write the CSS language and browser vendors (the people who make web browsers) should implement it. Because the CSS specification has grown since its introduction in 1996, the latest version, CSS3, has been broken up into modules so that each defines a part of CSS, making the overall specification easier to maintain.

You can find the CSS specifications at `www.w3.org/Style/CSS/`. Because these specifications are very much technical and in depth, you may find them off-putting. I know I do! Although they are useful to refer to from time to time, by no means do you need to read them. *CSS3 Foundations* takes a much more simple and friendly approach to your understanding of CSS3.

Before you take a closer look at what CSS can offer, you should first understand the current state of CSS. You may have noticed I referred to both CSS and CSS3. What's the difference?

CSS refers to all three levels of the specification: CSS Level 1 (CSS1), CSS Level 2 (CSS2), and CSS Level 3 (CSS3). Each level of CSS builds on its predecessor. CSS2 had a shaky start and many issues came to light, leading to a revision of this specification and the release of CSS2.1. So, CSS3 contains aspects of its predecessor CSS2.1, and CSS2.1 contains aspects of CSS2 and CSS1.

Although each CSS level builds on its predecessor, where relevant, a level recommends a particular feature from its predecessor no longer be used and thus deprecated. This means that when referring to CSS3, one is actually referring to all the features made available throughout CSS, except those that have been deprecated.

In *CSS3 Foundations*, you learn CSS3, which includes not just the newest of features, but also those from the previous CSS levels that experienced designers and developers have relied on for years.

The Role of CSS

The main purpose of CSS is to separate structure (HTML) from presentation (CSS). Figure 1-1 shows a web page that consists only of HTML, without any CSS.

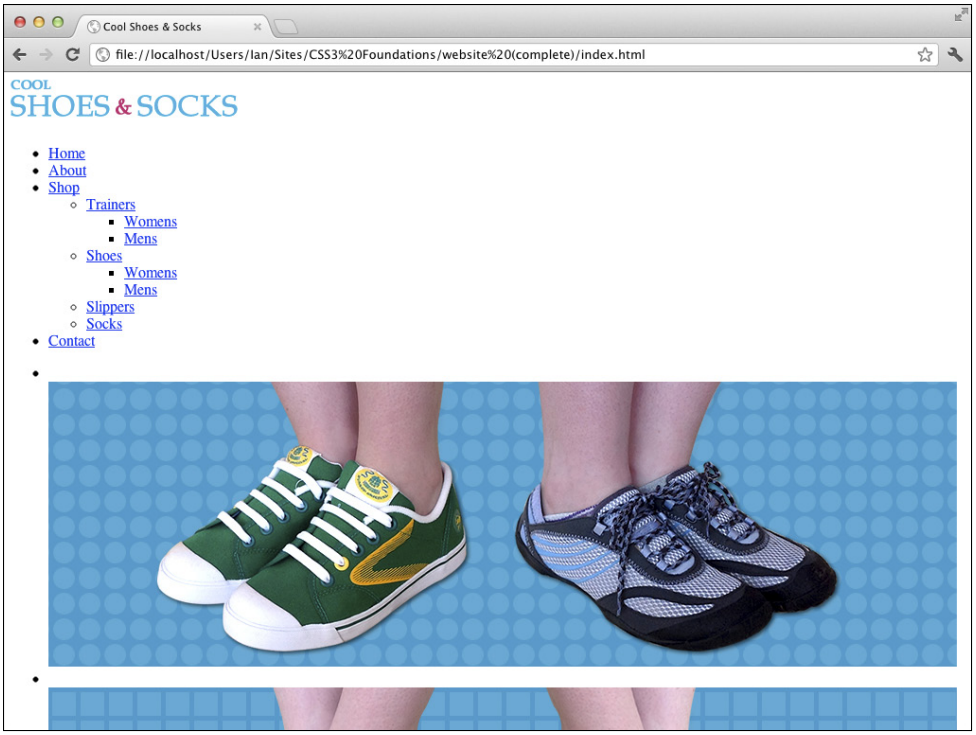


FIGURE 1-1 A web page without any CSS.

Figure 1-2 shows the same web page *with* CSS applied to it—a huge difference, one that makes the page much more attractive.

In the early days of styling web pages, structure and presentation were mixed together. Presentation was directly applied to structure, meaning that maintaining pages became an arduous task. If, for example, somebody decided that the main title on each page of a website should be changed from black to blue and that site consisted of 10 pages, you had to change the style for that title 10 times.

By separating structure and presentation, you gain numerous advantages:

- A Cascading Style Sheet can be shared across multiple web pages.
- Sites are easier to maintain and become more flexible.
- The styles applied to a web page can be tailored to suit multiple devices/environments.

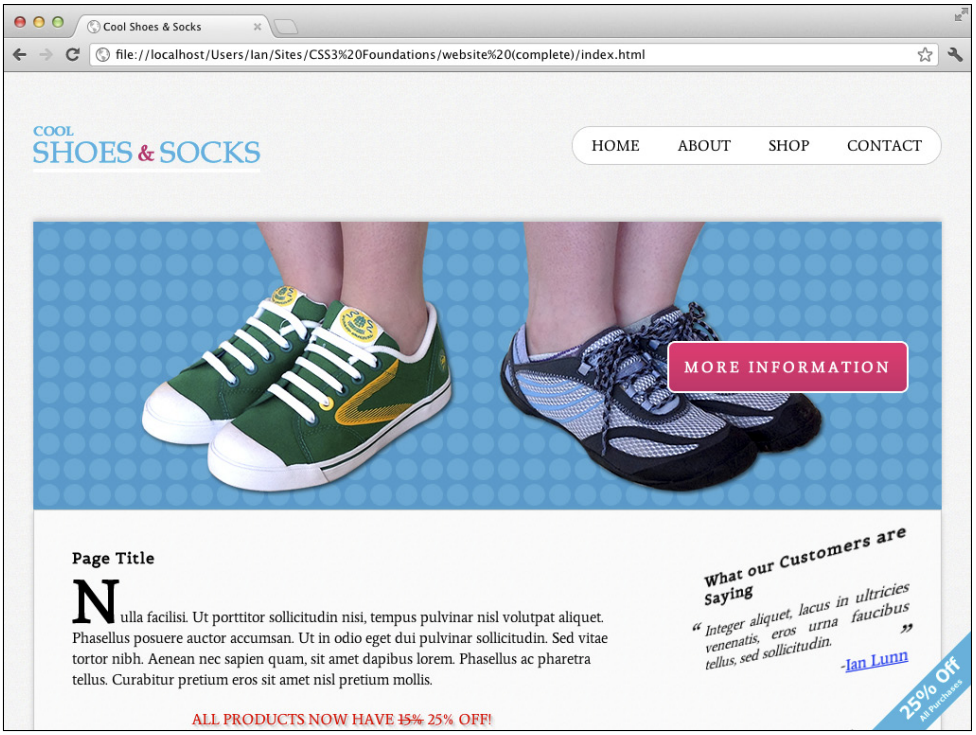


FIGURE 1-2 A web page with CSS.

In the modern web, the role of CSS is more important than ever. As technology has advanced and the ways in which you access the web are more diverse, CSS must evolve to accommodate both users wanting to access information over the web and developers wanting to present that information in a particular way. I say *developers,* but really I mean anyone. The web is not just for the technically minded. Open information is for everyone, and CSS aims to be as easy and open to use as possible, allowing anyone to style information.

The most important aspect of CSS is backward compatibility. Because each new level of the specification builds on the last, CSS remains backward compatible. Although features may become deprecated, they still work in existing browsers because new levels only add new functionality or refine existing definitions. By working backward first, the specification then has the opportunity to pave a solid path into the future.

CSS3 offers many new features to push the web forward and help improve the accessibility and presentation of information. With the emergence of multiple ways to access the web, one of CSS3's most important features is the Media Queries module, which builds on media types.

When CSS2 was developed, it became apparent there was a need to serve different styles based on how web pages were being accessed, so media types were introduced. These media types allowed for specifying different styles for particular devices, such as screen (intended for color computer displays), print (intended for printers and printed documents), and handheld (intended for small screen devices with limited bandwidth).

With such a varied range of handheld devices on the market today, defining those devices within the handheld category certainly isn't descriptive enough. Likewise, desktop displays differ greatly in features too.

CSS3's Media Queries module builds on media types, allowing a developer to query particular aspects (such as width and height) of the device and environment on which a web page is being viewed and specify styles to be applied only when certain conditions are met.

The specification also adds new presentational features such as gradients and rounded corners to make easier some of the tasks developers have been doing for years, more ways to select the elements to be styled (called Selectors), and web fonts, which sees the number of fonts you can safely use go from tens to thousands—if not tens of thousands!

Working Draft and the CSS Process

At the time of writing, the overall CSS3 specification is incomplete. To make reading and implementing it easier, the specification is broken up into modules. While some of these modules are complete, others are still being written or tested. Modules go through varying stages to determine how far from being complete they are and also serve as an indication of whether you should use them:

- **Working Draft**—A module that is incomplete and still being worked on.
- **Last Call**—A module that is considered to be complete. This module will move toward Candidate Recommendation unless significant issues arise.
- **Candidate Recommendation**—All known issues are resolved and the features of the module are ready to be implemented into web browsers.
- **Proposed Recommendation**—Features of the module are implemented in at least two web browsers and the module is reviewed by W3C Members one last time.
- **Recommendation**—The module is complete.

When implementing features from modules that are Working Draft or Last Call, those features should be used with a vendor prefix; this ensures you are using experimental features in a way that prevents incompatibilities should there be changes to modules in the future.

You'll learn exactly what a vendor prefix is and how to use them in Chapter 5.

More information about the CSS process can be found at `www.w3.org/Style/2011/CSS-process#rectrack`.

All in all, CSS3 makes creating web pages easier and more exciting than ever. So, at the time of writing, as the specification is in Working Draft status, can you use it yet?

Absolutely! Modern browsers already have great support for CSS3 because the sooner you start using it, the sooner you can provide feedback and improve the future of CSS. The W3C consists not only of member organizations and its full-time staff but the public, too. By using CSS3 today, you can have your say in its future. Just by using a feature of CSS3 is giving your stamp of approval to it.

Throughout *CSS3 Foundations*, you learn to use CSS3 in a safe way that is compatible with both existing browsers that have some or no CSS3 support and future browsers, in the case of features being redefined prior to the specification reaching its Recommendation status.

Modern Browsers

A web browser is, at its core, an engine that renders HTML, CSS, and other technologies, turning them into a functional web page. Browsers also offer features such as bookmarking, history management, developer tools and many others, including support for add-ons that allow for extending a browser's capabilities.

So, which browser should you use to create your website? All of them! Okay, you don't need to download them all right now, but it is important to test the websites you build in a range of browsers (and devices).

Today's Major Browsers

Five major browsers are available today. These browsers are created by organizations—often referred to as browser vendors or vendors for short—and are made free to download and use.

- Google Chrome (`www.google.com/chrome`)
- Apple Safari (`www.apple.com/safari`)
- Mozilla Firefox (`www.mozilla.org/firefox/new`)
- Opera (`www.opera.com`, both browser and vendor are referred to as Opera)
- Microsoft Internet Explorer (`http://windows.microsoft.com/en-US/internet-explorer/downloads/ie`)

I personally like to create websites using Chrome. I feel its web developer tools are the most useful for the way I work, and it is often ahead of the rest when implementing the latest CSS features.

Throughout the book, I use Chrome as my main browser (and test other browsers in Chapter 15). I recommend avoiding Microsoft Internet Explorer versions 7–9 as a browser to use while creating a web page because its features aren't as consistent as the others listed, but if you'd like to use Safari, Firefox, or Opera, feel free. The development tools, which I use and refer to, are similar between browsers, and I don't use a feature if it doesn't exist in another browser.

At the time of writing, Microsoft Internet Explorer 10 is available as a preview but hasn't officially been released.

Browser Engines (Layout Engines)

Knowing which engine these browsers support is also useful:

- **WebKit (**`www.webkit.org/`**)**—Both Google Chrome and Apple Safari use this open source engine. Although these browsers use the same engine, they may have implemented it in a slightly different way, so you shouldn't expect one to render the page exactly the same as the other. Always test in both to make sure. That said, in most cases, consistency between these browsers is very good. Android Browser also uses WebKit, which together with Safari (used on Apple's iPhone, iPad, and iPod products) makes it the most-used browser engine on mobile devices.
- **Gecko (**`www.developer.mozilla.org/en/Gecko`**)**—Mozilla Firefox uses Gecko, another open source project.
- **Presto (**`www.opera.com/docs/specs/`**)**—Available only as a part of Opera products.
- **Trident (**`www.en.wikipedia.org/wiki/Trident_(layout_engine)`**)**—Available only as a part of Microsoft's products.

These engines are used not only by desktop and mobile browsers but also by other products such as e-mail clients and word editing software.

What Is Open Source Software?

Open source software is that which is made available for use and modification completely free. This is a great way to open up a project to everybody and, in turn, enhance the project by having many people improving it at the same time. *CSS3 Foundations* uses a few open source projects, and no doubt, you'll come across many on your own in the future.

Browser Usage Statistics

Although browsers are free to download and use, the web browser market is very much a competitive place. As mentioned, vendors are all trying to push the web forward, but they also stand to gain a lot from people using their particular browser. Some vendors have other products that they want to profit from. The bigger their share of the market, the more they are able to push these products and services.

Tables 1-1 and 1-2 present the usage statistics provided by `www.netmarketshare.com` in July 2012 (the time of writing).

Table 1-1 Desktop Browser Usage

Browser	Total Market Share
Internet Explorer	54.02%
Firefox	20.06%
Chrome	19.08%
Safari	4.73%
Opera	1.60%
Others (below 1%)	0.34%

Table 1-2 Mobile/Tablet Browser Usage

Browser	Total Market Share
Safari	65.79%
Android Browser	19.17%
Opera Mini	10.45%
BlackBerry	1.45%
Others (individually below 1%)	2.56%

Older Browsers on the Modern Web

Unfortunately, the modern web—with all its fancy new features—comes with excess baggage in the form of older web browsers that are still in use today. Although the W3C, browser vendors, and the web community/industry are working to push the web forward, there are many web users who are either unaware of or unable to embrace this change.

Before looking at which of the browsers still in use today could be deemed as being old, Table 1-3 shows the breakdown of market share into individual version numbers.

Table 1-3 Desktop Browser Usage by Version Number

Browser	Total Market Share
Internet Explorer 8.0	26.65%
Internet Explorer 9.0	17.96%
Chrome 19.0	15.56%
Firefox 12	7.76%
Firefox 13	6.33%
Internet Explorer 6.0	5.92%
Safari 5.1	3.63%
Internet Explorer 7.0	3.10%
Firefox 3.6	1.38%
Opera 11.x	1.04%

Like me, you may find these statistics to be surprising. Internet Explorer 8 has the biggest market share, over 25%? There are a few reasons for this. Internet Explorer comes packaged with Microsoft Windows, which is the most-used operating system. Many people access the web via Internet Explorer unaware that other browsers even exist. They also don't upgrade their browser because either they don't know that option is available or the technology they have is too outdated to do so. You should note that Internet Explorer 9 is available only to users of Windows Vista and Windows 7. Many users are still using Windows XP, which is restricted to Internet Explorer 8. This may explain why it is the number one web browser, despite Internet Explorer 9 being on the market since 2011.

Here's where I drop the bomb: Internet Explorer 6, 7, and 8 are old browsers, despite all having significant market share. Although each is superior to the next, their support of standardized CSS is poor. Versions 6 and 7 implement CSS 2.1 but not particularly well. Version 8 has good support for CSS 2.1 but little for CSS3.

There is light ahead though. Although Internet Explorer 6 has around a 6% share according to these statistics, many of its users are Asian based, so if your website is not targeting that audience, you may find those statistics greatly drop. Internet Explorer 6 is such an old browser, and so troublesome to the future of the web that even Microsoft is trying to make its users aware that they can upgrade, with the site `www.ie6countdown.com`. This site also confirms that Asia (in particular, China, with a 22.4% share) greatly contributes to that 7% global statistic.

Aside from Internet Explorer versions 6, 7, and 8, the only other old browser with a share greater than 1% is Mozilla Firefox 3.6. It has good support for CSS2.1 and CSS3 and is not a problem browser by any means, but at the time of writing, Firefox is in the double figures, at

version 14. You would be forgiven for thinking you didn't have to test in 3.6 then because it's so outdated, but testing in Firefox 3.6 is something I highly recommend due to its having a market share higher than 1% (generally a browser with a market share less than 1% is deemed as being redundant), and you do just that toward the end of *CSS3 Foundations*.

So how do you deal with older browsers because they are still very much a part of the modern web?

It is important to remember that the web is open information and it is the person's job that is making that information available as accessible to as many as possible. You shouldn't ignore users of old web browsers, but it's unrealistic to provide them the same levels of features you would provide somebody using a modern web browser. You would still like them to be able to see the beauty of the page you've created, just at the level their browser can support.

Say you've used some of CSS3's features such as rounded corners and background gradients. An older browser doesn't understand these features, so it can't display them. Instead, that browser just shows square corners and a solid background, as shown in Figure 1-3 and Figure 1-4.

FIGURE 1-3 An element with rounded corners.

FIGURE 1-4 The same element viewed in a browser that doesn't support CSS3's rounded corners.

This is known as *graceful degradation*. In a browser that doesn't support CSS3, it simply falls back to what it does support. The user of that browser still sees a nicely styled web page and is none the wiser to the fact that in a modern browser, he could be seeing a few extra niceties. Nothing appears broken to him, and the way in which he is interacting with your web page isn't affected.

There is also another technique similar to this known as *progressive enhancement*, but what's the difference? Although graceful degradation starts with modern browsers in mind first and then provides workarounds for older browsers later, progressive enhancement does the opposite: it starts with an older browser and then later adds new features for modern browsers.

Which approach should you use? With all the work being done to push the web forward, it would be a shame for that to go to waste. By using graceful degradation, you are putting the future of the web first but making sure there is support for the part of the web that is yet to catch up, too.

In the case of Internet Explorer 6, you may decide to just ignore it completely, simply because it's such an old and troublesome browser to provide fixes for. Rather than spending *x* number of hours fixing it, in Chapter 15, you'll apply basic CSS to Internet Explorer 6, which still makes the content of the web page perfectly accessible, it just won't be styled as much. This saves you work (and if you're creating a web page for a client, it saves money), but the user can still see all the information on the page because, after all, CSS is just presentation.

Of course, if you're creating web pages for a paying client, now or in the future, you would need to discuss browser support. By analyzing the latest usage statistics for browsers, as well as the client's existing statistics on their own website (should they already have one), you can determine what browsers your client's users are using to view the site and aim to support those browsers with the site you're creating.

Throughout *CSS3 Foundations*, I advise you when a feature isn't available in an older browser, and you learn to test and examine alternatives for these browsers in Chapter 15.

Tools for Building and Styling the Modern Web

As the web has grown, so too has the number of tools available to build it. You can easily become overwhelmed by choice, but you should remember that, at its core, the web is just text. To start building a web page, you need only a web browser and a text editor.

Tools range in price anywhere from completely free to several hundred dollars, but by no means are the most expensive ones necessarily the best. You can get started with some great tools having spent no money whatsoever.

Web Developer Tools

Each of the latest web browsers has its own version of web developer tools that make creating websites easier and more efficient.

Web developer tools allow you to quickly look at HTML elements and the CSS applied to them. You can also make live alterations to CSS within a browser's web developer tools, often a quicker way to experiment with styles. Figure 1-5 shows Google Chrome's web developer tools.

You use these tools later in the book, but for now, you might want to familiarize yourself with opening them in your chosen browser.

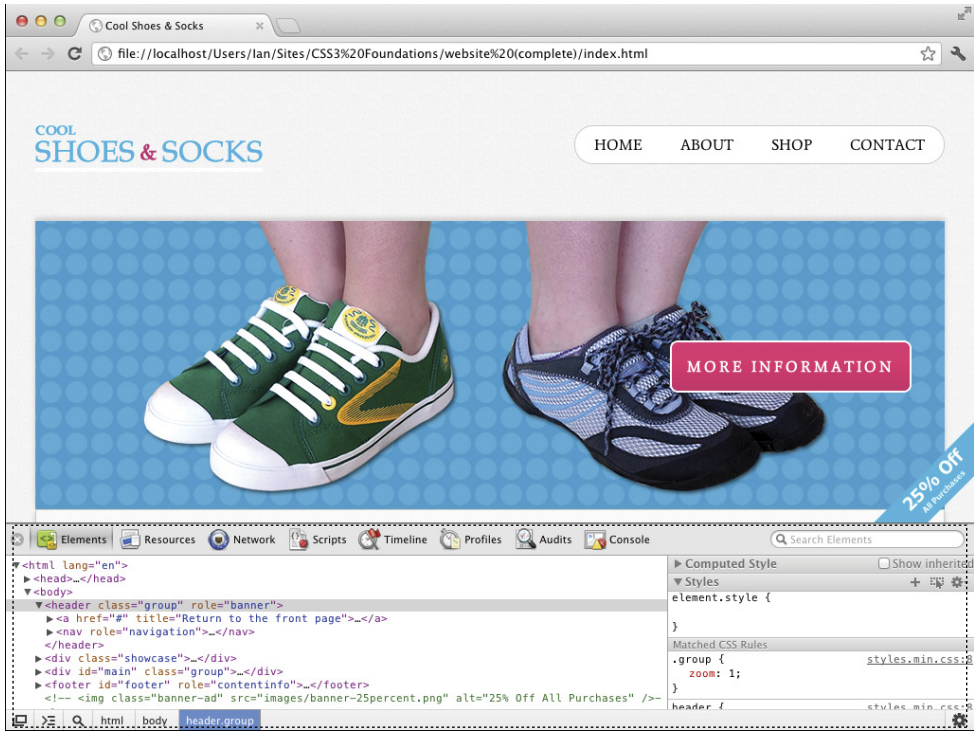


FIGURE 1-5 The web developer tools open in Google Chrome.

Opening Web Developer Tools in Desktop Browsers

Browser	Web Developer Tool's Name	Mac Shortcut	Mac Menu Navigation	Windows Shortcut	Windows Menu Navigation
Chrome	Developer Tools	Option + Command + I	View ➜ Developer ➜ Developer Tools	Ctrl + Shift + I	Settings icon (wrench) ➜ Tools ➜ Developer Tools
Safari	Web Inspector	Option + Command + I	Develop ➜ Show Web Inspector	Ctrl + Alt + I	Menu icon (page) ➜ Develop ➜ Show Web Inspector*
Firefox	Firebug (Add-on; see following note)	N/A	View ➜ Firebug	F12	Firefox ➜ Web Developer ➜ Firebug

Browser	Web Developer Tool's Name	Mac Shortcut	Mac Menu Navigation	Windows Shortcut	Windows Menu Navigation
Opera	Dragonfly	Option + Command + I	View ➜ Developer Tools ➜ Opera Dragonfly	Ctrl + Shift + I	View ➜ Developer Tools ➜ Opera Dragonfly
Internet Explorer	F12 Developer Tools	N/A	N/A	F12	Settings icon (cog) ➜ F12 Developer Tools

*Safari's Develop menu must first be switched on under Preferences ➜ Advanced.

Prior to version 10, Firefox didn't come with built-in developer tools. I highly recommend an add-on called Firebug both in the absence of tools prior to version 10 and in place of the built-in tools available from version 10 onward. Firebug has more features than the built-in tools and is more consistent with those available in other browsers.

To install Firebug, in Firefox, click Tools ➜ Add-Ons to open the Add-Ons Manager. Search for Firebug and then click the Install button. After it is downloaded, click Restart Now (or close and reopen Firefox manually) to complete the installation.

Text Editors

A text editor is the tool used to write CSS. Text editors also allow for writing HTML, JavaScript, and many other languages that make up the web.

Text editors allow you to do just that—edit text. You would expect them all to be quite similar, but actually, text editors come with many differing features. Most come with features that speed up your writing of CSS (and other web technologies). Some come with File Transfer Protocol (FTP) clients and What You See Is What You Get (WYSIWYG) editors.

A WYSIWYG editor allows you to build a website without your having any knowledge of code. Although this idea sounds great, in reality, it's not! When you are using these tools—dragging elements into the positions you want—the editor is given the responsibility of generating the code for you. These types of editors are rarely up to date with the latest coding standards and can never do as good a job of coding a website as you or I. Generated code is almost always bloated (more code than needed) and doesn't exercise the best practices you learn in this book, leading to a poor experience for the visitors to your website. Should your text editor also come with a WYSIWYG editor (such as Adobe Dreamweaver), that's fine; just choose to ignore it and make use of the text editor feature instead.

If you don't already have one, I recommend using a basic text editor to begin with. TextEdit (for Mac users) or Notepad (for Windows users) already come installed on your computer and are perfectly acceptable to use. They don't offer any features beyond editing your text, but that's all you need to start.

Throughout *CSS3 Foundations,* I use a text editor for Mac called Espresso (`www.macrabbit.com/espresso/`).

Summary

The web could exist without CSS, but it would never be as accessible or pretty as it is today. Its importance—in particular the new features introduced in CSS3—grows significantly as the ways in which people access the web change. CSS continues to advance, with many exciting features being introduced. Now is a great time to jump in and start learning how to use it.

With the necessary tools to begin creating CSS and web pages, in the next chapter you will get started with the project files built on throughout *CSS3 Foundations*. You'll take the page seen in Figure 1-1, and turn it into the styled page seen in Figure 1-2.

chapter two

Getting Started

AS YOU MAKE your way through *CSS3 Foundations*, you learn CSS from the basics to the more advanced topics, all the while adding to an overall project that will eventually become a fully featured web page.

Often, I find the best way to learn new things is to create a scenario with particular restrictions and an end goal. In doing this, you can focus on reaching that end goal by limiting the number of choices you have to make (an infinite number of choices is often scary and off-putting, so it's always good to define boundaries!). This approach to learning also means you have to make some real-world decisions to get the best possible outcome for the project.

So, without further ado, I'd like to introduce you to the fictitious company Cool Shoes & Socks (CSS for short—get it?).

The folks at Cool Shoes & Socks have asked you to style the company's website for them. This web-savvy company, like the name says, sells cool shoes and socks. The staff have given you the go-ahead to use CSS3 on the website; their only caveat is that you

must make the website gracefully degrade in browsers that don't support CSS3. The company is also very aware that the web is accessed on many different types of devices (and its underfunded research department has determined socks will be able to access the web in the near future), so having a Responsive Web Design is very important to everyone involved.

Cool Shoes & Socks has already had the HTML of the site written and handed it to you to begin implementing CSS and modifying the HTML should that be necessary.

Getting Started with the Project Files

In a moment you will download the Cool Shoes & Socks project files that are built on throughout *CSS3 Foundations*. At specific milestones in the book, the most up-to-date project files are made available for you to reference or in case you want to skip a section and start at a different point. These milestones are marked in the book, along with the project file number and a link to where you can download it.

Downloading the Project Files

Go to `www.wiley.com/go/treehouse/css3foundations` in your web browser, where you can download the project files. The project files are contained within a Zip file that you should unzip into the location of your choosing.

Example, I unzipped the files to the following directories:

- Windows: C:/Users/Ian/Sites/CSS3 Foundations/
- Mac: Users/Ian/Sites/CSS3 Foundations/

Folder Structure and Good Practices

When you unzip the downloaded file, you are presented with folders named *website* followed by a chapter and milestone number, such as *(ch02-00)*, representing the first milestone of Chapter 2 (the one you begin with). Each website folder contains the project files that are worked on throughout *CSS3 Foundations*. My personal recommendation is to keep the files within the website folder. So, my folder structure looks like this:

Users/Ian/Sites/CSS3 Foundations/website (ch02-00)/

Often, you might want to save files that aren't part of the website but relate to it (such as notes, instructions from a client, images that have yet to be edited), so it's a good idea to have a *website* folder within the project folder to keep it separate from those extra files.

Within the *website (ch02-00)* folder, you find the following folders and files:

- css
- fonts
  - Average-OFL.txt
  - Average-Regular.eot
  - Average-Regular.woff
  - Belgrano-OFL.txt
  - Belgrano-Regular.eot
  - Belgrano-Regular.woff
- images
  - banner-25percent.png
  - banner-shoes.jpg
  - banner-socks.jpg
  - banner-trainers.jpg
  - bdr-footer.png
  - bg-blog.png
  - bg-body.jpg
  - bg-footer.jpg
  - bg-newsletter.png
  - check.png
  - icon-newsletter.png
  - logo-small.png
  - logo.png
  - socks-horses.png
- index.html

A good practice is to separate external files (those separate from the main HTML files, known as *assets* or *dependent files*) into their own folders. Being organized in this way speeds up your workflow and makes managing a website easier.

Another good practice is to follow consistent naming conventions for your images. For example, bg-body.jpg and bg-footer.jpg are so called because they are applied as backgrounds (*bg,* meaning background). Multiple versions of the company logo exist, all beginning with the prefix *logo-*, which keeps them together alphabetically. As a website grows, using naming conventions such as these makes finding your files quicker.

Of course, these are only my personal preferences; you may like to use different naming conventions or folder structures that work better for you.

Understanding the HTML Template

The web page template (index.html) is written in HTML5. Because you will build on and style one page in *CSS3 Foundations*, links to other pages are only included to represent and demonstrate a real-world web page. `href` links are given the value `#`, which means the page only refers to itself. Should you wish to add more pages in the future, feel free to change the links. index.html in all of its glory:

```
<!DOCTYPE html>
<html lang="en">
  <head>
      <meta charset="utf-8" />
      <title>Cool Shoes &amp; Socks</title>
      <!--[if lt IE 9]>
      <script src="http://html5shiv.googlecode.com/svn/trunk/html5.
  js"></script>
      <![endif]-->
  </head>
  <body>
      <header id="header" class="group" role="banner">
          <a href="#" title="Return to the front page">
              <img class="logo" src="images/logo.png" alt="Cool
  Shoes &amp; Socks" />
          </a>

          <nav role="navigation">
              <ul>
                  <li>
                      <a href="#" title="Return to the front
  page">Home</a>
                  </li>
                  <li>
                      <a href="#" title="About Company
  Name">About</a>
                  </li>
                  <li class="products">
                      <a href="#" title="Visit our shop">Shop</a>
                  </li>
```

```
                <li>
                    <a href="#" title="Contact us">Contact</a>
                </li>
            </ul>
        </nav>
    </header>

    <div class="showcase">
        <img src="images/banner-trainers.jpg" alt="Some of our
cool trainers" />
        <a class="button purchase" href="#" title="Purchase
product">
            More Information
        </a>
    </div>

    <div id="main" class="group">
        <div id="content" role="main">
            <h1>Page Title</h1>
            <p>Nulla facilisi. Ut porttitor sollicitudin nisi,
tempus pulvinar nisl volutpat aliquet. Phasellus posuere auctor
accumsan. Ut in odio eget dui pulvinar sollicitudin. Sed vitae
tortor nibh. Aenean nec sapien quam, sit amet dapibus lorem.
Phasellus ac pharetra tellus. Curabitur pretium eros sit amet
nisl pretium mollis.</p>
            <p class="offer">
                All products now have <span class="strike">15%</
span> 25% off!
            </p>
            <p>Nulla facilisi. Ut porttitor sollicitudin nisi,
tempus pulvinar nisl volutpat aliquet.</p>
            <img class="product" src="images/socks-horses.png"
alt="Horse Socks" />
            <p>Nulla facilisi. Ut porttitor sollicitudin nisi,
tempus pulvinar nisl volutpat aliquet. Phasellus posuere auctor
accumsan. Ut in odio eget dui pulvinar sollicitudin. Sed vitae
tortor nibh. Aenean nec sapien quam, sit amet dapibus lorem.
Phasellus ac pharetra tellus. Curabitur pretium eros sit amet
nisl pretium mollis.</p>
            <p>Nulla facilisi. Ut porttitor sollicitudin nisi,
tempus pulvinar nisl volutpat aliquet.</p>
            <div class="benefits">
                <h3>Why Choose Cool Shoes &amp;amp; Socks?</h3>
                <ul>
                    <li>World's Largest Range of Shoes &amp;
Socks</li>
```

```
                    <li>Next-Day Free Delivery</li>
                    <li>Money Back Guarantee</li>
                </ul>
            </div>
        </div>

        <div class="sidebar" role="complementary">
            <aside>
                <h3>What our Customers are Saying</h3>
                <ul class="testimonials">
                    <li>
                        <blockquote>
                            <p>Integer aliquet, lacus in
ultricies venenatis, eros urna faucibus tellus, sed
sollicitudin.</p>
                            <cite>
                                <a href="http://www.ianlunn.co.
uk/" title="Visit Ian Lunn's website">Ian Lunn</a>
                            </cite>
                        </blockquote>
                    </li>
                </ul>
            </aside>

            <aside class="post">
                <h3>
                    <img src="images/bg-blog.png" alt="Latest
Blog Post" />
                    Latest Blog Post
                </h3>
                <p>Nulla facilisi. Ut porttitor sollicitudin
nisi, tempus pulvinar nisl volutpat aliquet. Phasellus posuere
auctor accumsan. Ut in odio eget dui pulvinar sollicitudin. Sed
vitae tortor nibh. Aenean nec sapien quam, sit amet dapibus
lorem. Phasellus ac pharetra tellus. Curabitur pretium eros sit
amet nisl pretium mollis.</p>
                <p>Nulla facilisi. Ut porttitor sollicitudin
nisi, tempus pulvinar nisl volutpat aliquet. Phasellus posuere
auctor accumsan. Ut in odio eget dui pulvinar sollicitudin. Sed
vitae tortor nibh. Aenean nec sapien quam, sit amet dapibus
lorem. Phasellus ac pharetra tellus. Curabitur pretium eros sit
amet nisl pretium mollis.</p>
            </aside>
```

```
            <aside>
                <form id="newsletter" action="#">
                    <h3>Newsletter</h3>
                    <p>Sign up for news and special offers
  delivered to your inbox monthly.</p>
                    <label>Name: </label>
                    <input type="text" placeholder="Your Name" />

                    <label>Email: </label>
                    <input type="email" placeholder="Your Email
  Address" />

                    <input class="button" id="submit-newsletter"
  type="submit" value="Sign Up" />
                </form>
            </aside>
        </div>
    </div>
    <footer id="footer" role="contentinfo">
        <ul class="container">
            <li class="small-logo">
                <a href="#" title="Return to the front page">
                    Cool Shoes &amp;amp; Socks
                </a>
            </li>
            <li class="back-to-top">
                <a href="#header" title="Go back to the top of
  the page">
                    Back to Top
                </a>
            </li>
        </ul>
    </footer>
  </body>
</html>
```

So, what does this code mean?

The first line, `<!DOCTYPE html>`, specifies the type of markup the *document* (the technical term for a web page) will use; in this case, it uses HTML.

Next is the `<head>`, which is the location where you place metadata about the web page. The template's `<head>` contains the charset (which tells the browser what character encoding to use) and the title of the page. You also use `<head>` to add resources such as stylesheets and

JavaScript, among other metadata. Because the template is built using HTML5 elements—which isn't supported by Internet Explorer 6, 7, and 8—a script is referenced here that makes those browsers understand HTML5. Conditional comments are used to make the script apply only to Internet Explorer versions below version 9. Conditional comments are explained in Chapter 15.

```
<head>
    <meta charset="utf-8" />
    <title>Cool Shoes &amp;amp; Socks</title>
    <!--[if lt IE 9]>
        <script src="http://html5shiv.googlecode.com/svn/trunk/
  html5.js">
        </script>
    <![endif]-->
</head>
```

The `<body>` tags follow the closing `</head>` tag. Here, you place the structure and content of the page:

```
<body>
    ...
</body>
```

Inside the `<body>` tags is the header of the web page:

```
<header id="header" class="group" role="banner">
    <a href="#" title="Return to the front page">
        <img class="logo" src="images/logo.png" alt="Cool Shoes
  &amp;amp; Socks" />
    </a>

    <nav role="navigation">
        <ul>
            <li>
                <a href="#" title="Return to the front page">Home</
  a>
            </li>
            <li>
                <a href="#" title="About Company Name">About</a>
            </li>
            <li class="products">
                <a href="#" title="Visit our shop">Shop</a>
            </li>
            <li>
                <a href="#" title="Contact us">Contact</a>
```

```
            </li>
        </ul>
    </nav>
</header>
```

The header `<header>` is given the ID `header`, the class `group` and the role `banner`. IDs, classes, roles, and other additional information within element tags are known as *attributes*. In CSS, you can select elements based on their attributes.

The role attribute lets screen readers (software used by people with disabilities such as visual impairment) know that this particular section of the page is a banner, defined as being "a region that contains mostly site-oriented content, rather than page-specific content" (`www.w3.org/TR/wai-aria/roles#banner`). The roles model is another specification being developed by the W3C, which aims to improve the accessibility of web pages for persons with disabilities. For more information, see `www.w3.org/TR/wai-aria/`.

The header also contains the Cool Shoes & Socks logo and site navigation.

Underneath the header is a *product showcase*:

```
<div class="showcase">
    <img src="images/banner-trainers.jpg" alt="Some of our cool
  trainers" />
    <a class="button purchase" href="#" title="Purchase product">
        More Information
    </a>
</div>
```

The product showcase contains one large image and a call to action that links to more information about the product. A *call to action (CTA)* is an area of the page that aims to grab the attention of web page visitors and lead them through to achieving a particular goal. The product showcase is the place where Cool Shoes & Socks can show off its best products and latest offers; the CTA aims to lead visitors through to finding out more information about the products and potentially buying some shoes or socks. Employing calls to action is a classic technique used across many sites on the web. A CTA often stands out from the rest of the page to draw visitors' eyes to it, leading them to complete an action. Using CSS, you can make this button stand out from the rest of the page to achieve that goal.

After the product showcase is the main content of the page. First, look at the main structure of this section:

```
<div id="main" class="group">
    <div id="content" role="main">
        ...
```

```
    </div>

    <div class="sidebar" role="complementary">
        <aside>
            …
        </aside>
    </div>
</div>
```

The line `<div id="main" class="group">` contains the content of the page as well as a sidebar that holds customer testimonials, the latest blog posts and a newsletter sign up form. The preceding code shows only the structure of this main section. As you make your way through *CSS3 Foundations*, you learn how to create columns, placing the content on the left, with the sidebar to its right. Again, this code uses role attributes to assist screen readers.

Finally, prior to the closing of the `</body>` and `</html>` tags, you see the pages footer:

```
<footer id="footer" role="contentinfo">
    <ul class="container">
        <li class="small-logo">
            <a href="#" title="Return to the front page">Cool Shoes
  &amp;amp; Socks</a>
        </li>
        <li class="back-to-top">
            <a href="#header" title="Go back to the top of the
  page">Back to Top</a>
        </li>
    </ul>
</footer>
```

The footer holds a link to the front of the page and a link to the top of the page.

Figure 2-1 shows index.html when viewed in a web browser. The template doesn't much resemble a web page yet, does it? All the information is there and so too is the navigation, but it doesn't look that great.

In Figure 2-1, notice that the content of the page is gibberish. This sort of text is known as ***Lorem Ipsum.*** It's dummy text that can be used to fill out a web page with content, prior to real content being put in place. By using dummy text, you can see how a page will look and function before you get any real content.

FIGURE 2-1 The index.html file when viewed in Google Chrome.

Getting Started with CSS

To create a new stylesheet file, follow these steps:

1. In your text editor, create a new file and name it styles.css.
2. Save the file in the css folder of your website project.

You must now reference that stylesheet in the web page.

Adding CSS to a Page

As you saw in the HTML template, the `<head>` is the place where you add metadata and reference the resources on which the page relies.

In the `<head>` section of index.html, reference the stylesheet you just created as follows:

```
<head>
    <meta charset="utf-8" />
    <title>Cool Shoes &amp;amp; Socks</title>
    <link rel="stylesheet" href="css/styles.css" type="text/css"
  media="all" />
    <!--[if lt IE 9]>
        <script src="http://html5shiv.googlecode.com/svn/trunk/
  html5.js">
        </script>
        <![endif]-->
</head>
```

Here, you add a link for which the relationship (`rel`) to this page is a stylesheet. The hypertext reference (`href`) points to the place where that stylesheet is located within your project files and is of type `text/css`. This stylesheet will be utilized across all media types. More on media types shortly. Note that when a page is using the HTML5 doctype `<!DOCTYPE html>`, specifying the type is optional.

Often, you require multiple stylesheets per page. To do this, you can simply add another `<link>`:

```
<link rel="stylesheet" href="css/styles.css" type="text/css"
  media="all" />
<link rel="stylesheet" href="css/styles2.css" type="text/css"
  media="all" />
```

Alternatively, within a stylesheet, you can use the `@import` rule, which is a reference from one stylesheet to another and can be added anywhere within a stylesheet:

```
@import url("styles2.css");
@import url("styles3.css");
```

Which approach is better? `<link>` performs better over `@import` (you can find out more about the performance impact of `@import` at: `www.stevesouders.com/blog/2009/04/09/dont-use-import/`) but on a live web page (on the web and viewable by everyone), neither approach is best!

When you're creating a web page, separating styles into multiple sheets is a good idea to make managing styles easier. However, when a site is live, ideally, you should have only one stylesheet. This is best practice, making a web page more efficient. Whether you choose to use multiple `<link>` references within the HTML or use `@import` in your main stylesheet is entirely your choice. Just remember for good practice, you should merge all stylesheets into one when it's time for a web page to go live. This is something you'll do in Chapter 17.

Using Media Types

As mentioned in Chapter 1, you can apply a stylesheet to specific media types, such as screen, print, and handheld. Note that if a media attribute isn't specified, a browser treats that referenced stylesheet as applying to all media types, like so:

```
<link rel="stylesheet" href="css/styles.css" type="text/css" />
```

This is a safe (it works in all browsers) and slightly quicker way of referencing a stylesheet.

What about multiple stylesheets for different media types? No problem. Assume styles.css is the main CSS file applied to all media types, but you decide you want to add an extra stylesheet that is only applied when you print the web page. You can do that like so:

```
<link rel="stylesheet" href="css/styles.css" type="text/css" />
<link rel="stylesheet" href="css/print.css" type="text/css"
  media="print" />
```

Here, the first `<link>` references a stylesheet to be applied to all media types (because you haven't specified a media type). However, the second `<link>` references a stylesheet specifically for print. The main stylesheet still applies to all media types (including print), but the additional print stylesheet applies only to the print media type.

Sometimes styles within the print stylesheet can override the ones in the main stylesheet. Most of the time this will be intentional, but sometimes you get conflicts. The next chapter covers specificity and importance that determines which styles should be applied when style conflicts occur.

Much like using the `@import` rule for adding additional stylesheets into another stylesheet, you can also use the `@media` rule for defining styles for specific media types within a stylesheet.

Assume you have just one main stylesheet, like the following:

```
<link rel="stylesheet" href="css/styles.css" type="text/css" />
```

Rather than adding another `<link>` to reference a print stylesheet, within styles.css, you can add the following with the main stylesheet:

```
@media print{
    /*print styles here*/
}
```

Any styles within this `@media` rule are applied only to printed web pages. Anything outside this rule is applied to all media types.

You may find using media rules more desirable because it means referencing only one stylesheet, which is more efficient.

Inline Styles

In the earlier days of the web, it was possible to use inline styles, forgoing the need for a referenced stylesheet. In fact, you still are able to use inline styles today, *but* it's advisable that you don't.

Inline styles are those applied directly to HTML, as follows:

```
<body style="color: blue; margin: 10px;">
```

By applying styles in this way, you lose all the benefits previously mentioned. No longer can styles be shared across pages, pages become more difficult to maintain, and styles can't be tailored to suit differing types of devices and environments.

Although the web has, fortunately, moved away from inline styles, e-mail hasn't. E-mails are also created using HTML and CSS, but unlike the web, the standards and technologies haven't moved forward.

E-mail clients are in the unfortunate position of relying on old technologies with a lack of standardization, and because of that, they often don't support the CSS used in today's modern web. Therefore, if you want to create an HTML/CSS-based e-mail, it is better to use inline styles.

User Agent Stylesheets

Before you begin applying styles to a page, it's important to understand that browsers apply their own default stylesheet to every web page—often referred to as a *user agent stylesheet*.

User agent stylesheets add default styles to HTML elements. These styles make an HTML page without any custom CSS applied to it instantly readable. When creating a web page using these stylesheets, you don't have to apply all the default CSS styles as defined by the CSS specification to elements yourself.

Figures 2-2 and 2-3 show the web page in Chrome and Internet Explorer 9, respectively. You may notice differences between the two. For example, the logo in Internet Explorer 9 has a blue border around it, but in Chrome it doesn't. The reason is that these two browsers use different user agent stylesheets. Microsoft, the vendor of Internet Explorer 9, made the decision that a linked image should have a border around it; otherwise, you have no visual indication that it's a link. Microsoft makes a fair point; but in most cases, this border is undesirable, and furthermore, no other modern browser does that.

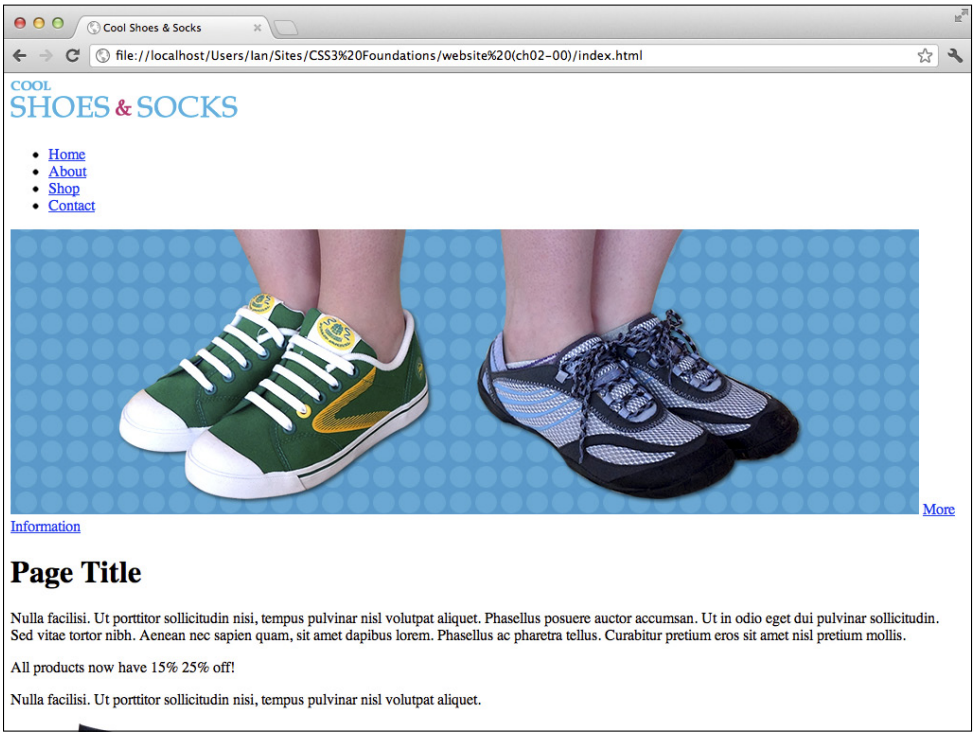


FIGURE 2-2 The index.html file when viewed in Google Chrome.

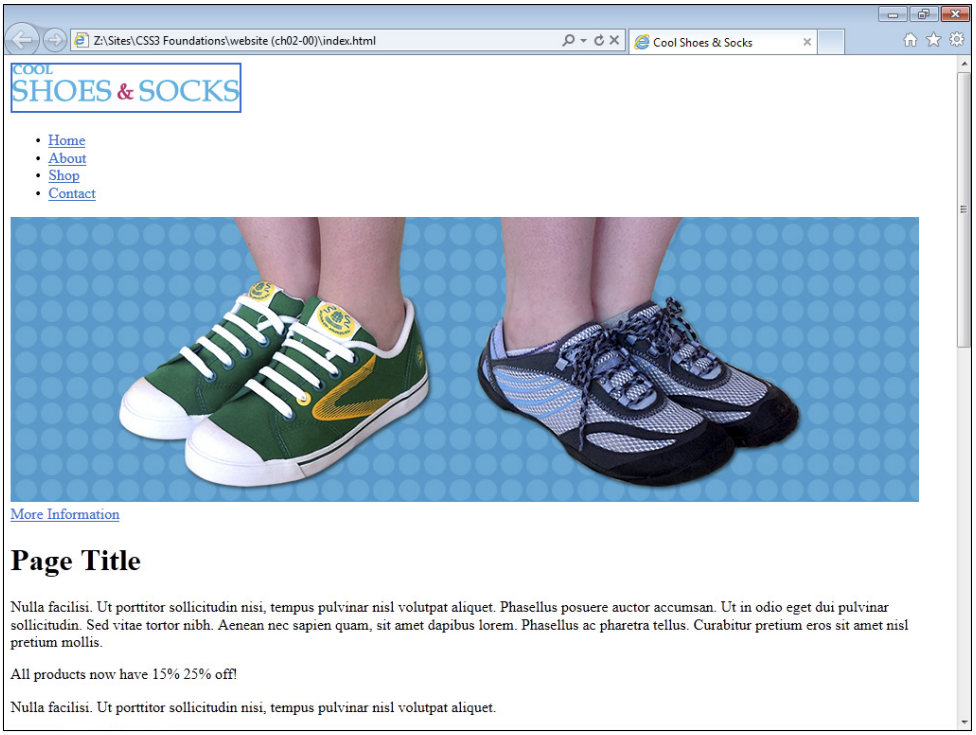


FIGURE 2-3 The index.html file when viewed in Internet Explorer 9.

In modern browsers, user agent stylesheets are similar, making creating cross-browser-compatible (pages that work in all browsers) pages easier, but particularly in older browsers, consistency isn't so good. You can be thankful that there's an easy solution to fix these inconsistencies.

Using a CSS Reset for Better Browser Consistency

A CSS Reset is a set of styles that overwrite user agent styles, making the default styles as consistent as possible across all browsers. Eric Meyer (`www.meyerweb.com/eric/`), an industry leader, has worked on improving this technique over many years, producing a small set of styles that make writing CSS much easier from the start.

By using a CSS Reset, you must add a few default styles back into your own CSS, because they are almost certainly required, but that is a small task when you consider how much easier it is to begin with consistent styles across all browsers.

To add a CSS Reset to the web page, follow these steps:

1. Copy and paste Eric Meyer's CSS Reset from `www.meyerweb.com/eric/tools/css/reset/`.
2. Create a new file and name it reset.css.
3. Paste the CSS Reset into reset.css.
4. Save reset.css into your css folder (which should already contain styles.css).
5. In styles.css, add the following:

   ```
   @import url("reset.css");
   ```

6. Save styles.css.

You've now added an additional stylesheet and imported it into the main stylesheet. Remember, using multiple stylesheets when you create your web page makes for a more efficient workflow, but before you make a website live, it's good practice to add all stylesheets into one. If you'd like to do that now, simply paste Eric Meyer's CSS Reset into the top of styles.css instead of a new file.

Eric Meyer's CSS Reset as of July 2012:

```
/* http://meyerweb.com/eric/tools/css/reset/
   v2.0 | 20110126
   License: none (public domain)
*/

html, body, div, span, applet, object, iframe,
h1, h2, h3, h4, h5, h6, p, blockquote, pre,
a, abbr, acronym, address, big, cite, code,
del, dfn, em, img, ins, kbd, q, s, samp,
small, strike, strong, sub, sup, tt, var,
b, u, i, center,
dl, dt, dd, ol, ul, li,
fieldset, form, label, legend,
table, caption, tbody, tfoot, thead, tr, th, td,
article, aside, canvas, details, embed,
figure, figcaption, footer, header, hgroup,
menu, nav, output, ruby, section, summary,
time, mark, audio, video {
    margin: 0;
    padding: 0;
    border: 0;
```

```
    font-size: 100%;
    font: inherit;
    vertical-align: baseline;
}
/* HTML5 display-role reset for older browsers */
article, aside, details, figcaption, figure,
footer, header, hgroup, menu, nav, section {
    display: block;
}
body {
    line-height: 1;
}
ol, ul {
    list-style: none;
}
blockquote, q {
    quotes: none;
}
blockquote:before, blockquote:after,
q:before, q:after {
    content: '';
    content: none;
}
table {
    border-collapse: collapse;
    border-spacing: 0;
}
```

You come to understand this CSS in the following chapters.

After you add a CSS Reset, compare the web page in Chrome and Internet Explorer 9 again (see Figures 2-4 and 2-5). They're much more consistent now, right? The blue border has been removed from the logo in Internet Explorer 9 and the spacing between elements is almost exactly the same.

In these figures, you may notice that a lot of the spacing between elements has been removed, but you add that again shortly.

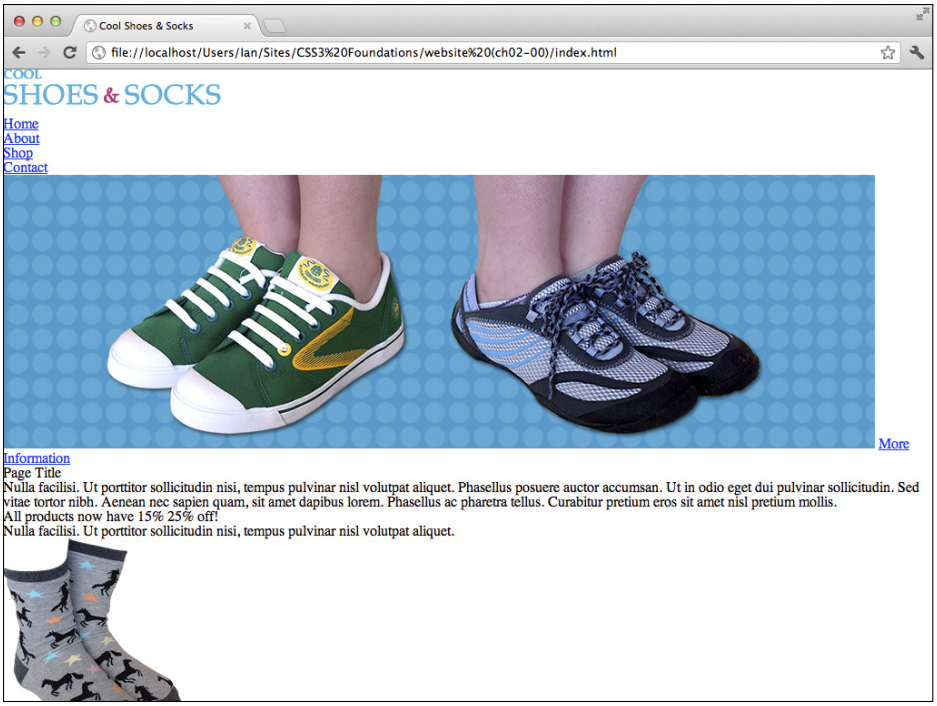


FIGURE 2-4 The index.html file in Chrome with a CSS Reset applied to it.

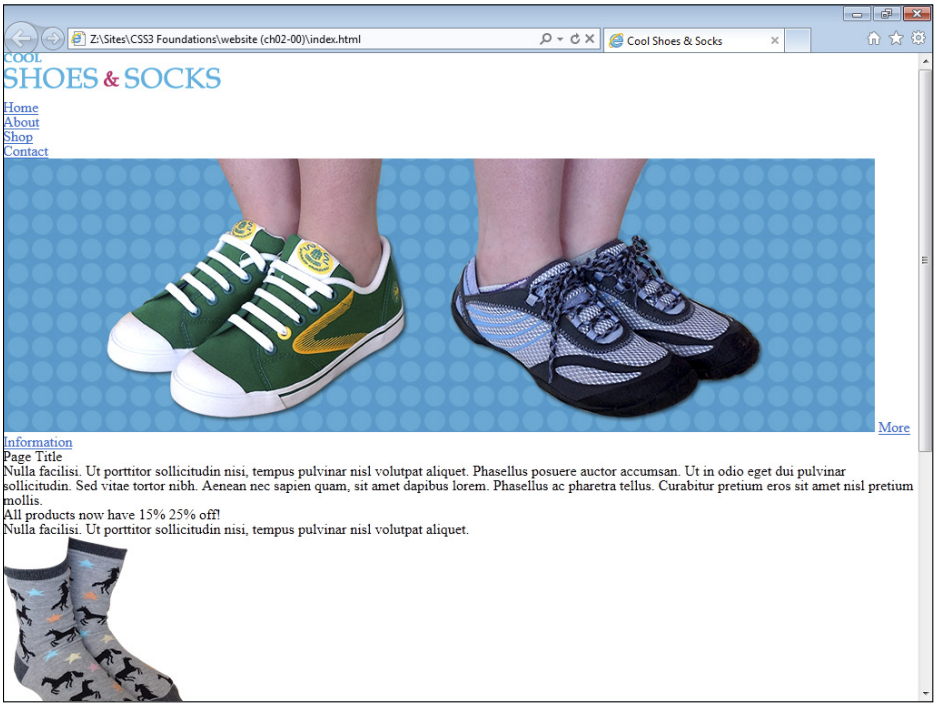


FIGURE 2-5 The index.html file in Internet Explorer 9 with a CSS Reset applied to it.

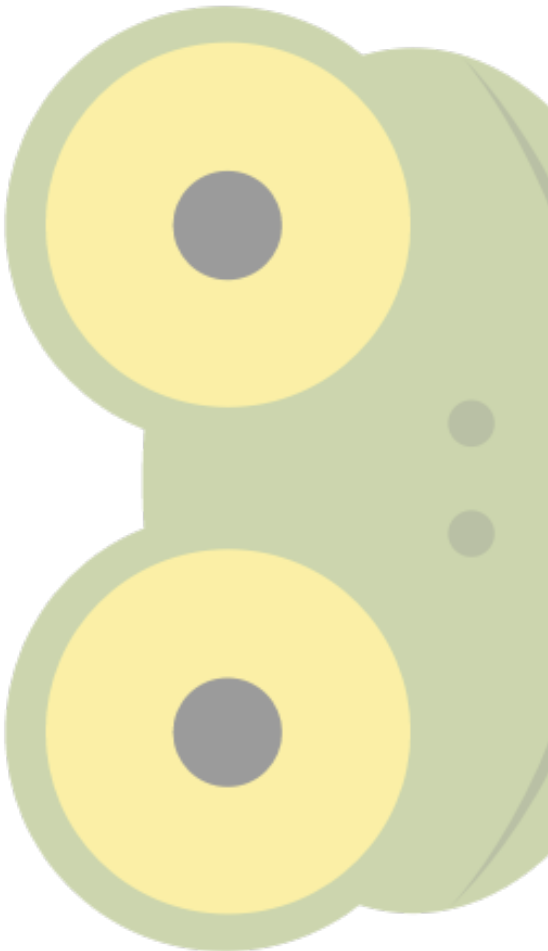
Summary

In this chapter, you've set up the project files that are worked on throughout *CSS3 Foundations*. By adding a CSS Reset to the project, you've given yourself a more consistent base to begin working from.

In the next chapter, you'll learn about the syntax of CSS (the rules of how it should be written), and start selecting elements to be styled using Selectors.

part 2

Learning CSS Syntax and Adding Presentational Styles



chapter three

Mastering the Power of CSS Selectors

IN THIS CHAPTER, you will learn to master the power of CSS selectors, which allow you to select any HTML element to be styled. Selectors have a very friendly learning curve and are easy to start with but can become more advanced as and when you are ready.

Writing Your First Styles

CSS is a collection of *rule sets* (also called *rules*), which consist of a selector and a declaration.

A *selector* is used to select the element you want to apply styles to, and a *declaration* is the combination of a property and a value for that element, as shown in Figure 3-1. You learn about property values in the next chapter and properties from Chapter 5 onward.

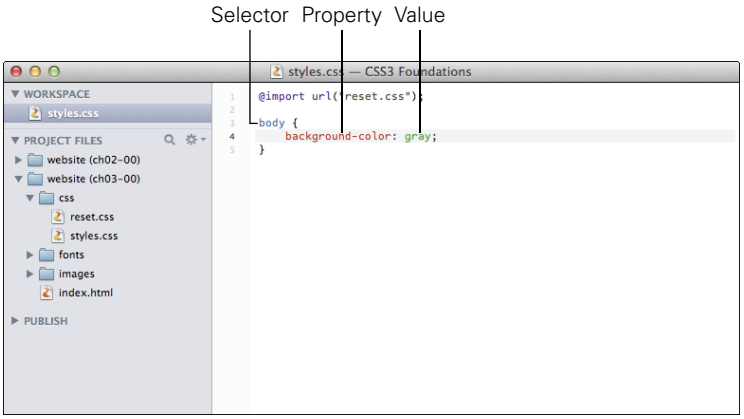


FIGURE 3-1 A rule set that changes the background color of the <body> element, written in Espresso text editor.

You end a declaration with a semicolon; this way, you are able to add multiple declarations within the same rule set.

Project Files Update (ch03-00): If you haven't followed the previous instructions and are comfortable working from here onward or would like to reference the project files up to this point, you can download them from `www.wiley.com/go/treehouse/css3foundations`.

Go ahead and add the style shown in Figure 3-1 to the page.

1. Open styles.css and add the following:

```
body {
    background-color: gray;
}
```

2. Save styles.css.

To see the affect of this rule, open index.html in your browser of choice. Whenever you save a modification made to styles.css, index.html, or any other file, you should refresh the browser to see the most up-to-date version of the page.

The rule you just added to styles.css tells the browser to select the <body> HTML element and apply the background color gray to it.

What if you want to add another style to the <body> element? Rather than add another body selector, you can simply define a second declaration within this rule, as shown here:

1. Inside the `body` selector, below the `background-color` declaration, add the following:

```
background-image: url("../images/bg-body.jpg");
```

2. Save styles.css.

Note that in CSS, a word followed by parentheses is known as a *function*. A function requires at least one *argument* (depending on the function being used), which, in the case of the `url()` function, is the path to an image. In Chapter 5 you will learn more about `background-image` and what exactly those two dots mean in the path.

You now see a background image applied to the body of the page, as shown in Figure 3-2. What happened to the background color? Don't worry; it's still there! The browser sees that you've added both a background color and background image to the same selector. The browser knows that a background image has more detail than a background color and is the preferred choice. So why add a background color if it's hidden behind an image? In some cases, the browser may not download images (a user can turn them off, for example, or a technical issue may prevent the image from being downloaded and displayed). If this happens, the page still has a background color similar to that of the image. Also, an image may be too small to cover the whole background; the areas that it doesn't cover show the background color.

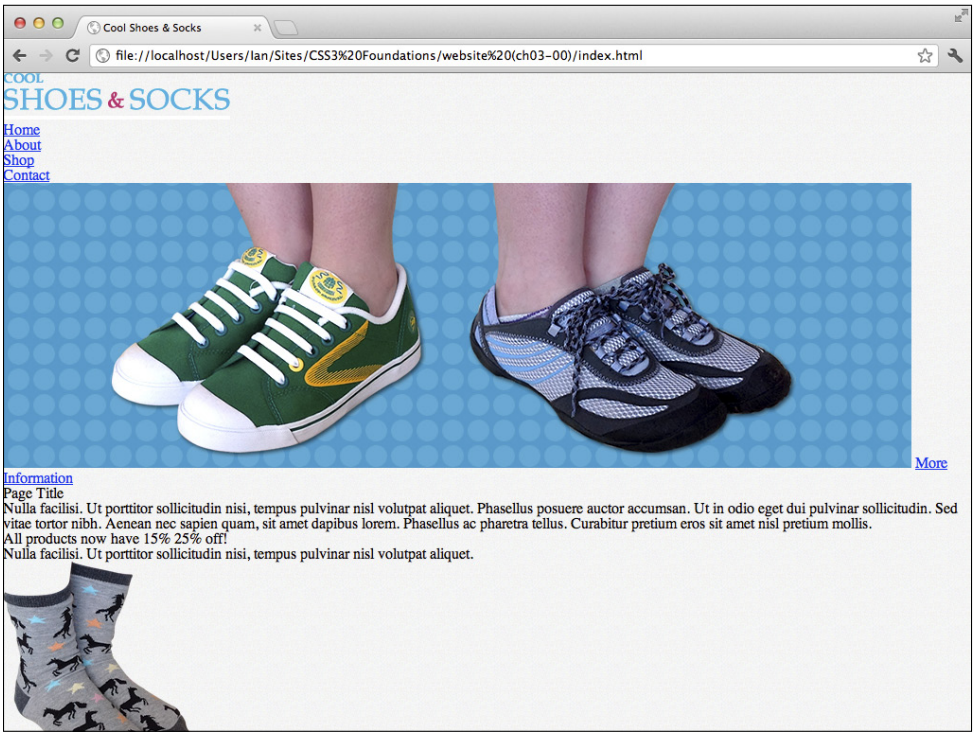


Figure 3-2 The page with a background image applied to it.

If you've had a good look through the project files, you may have noticed that the image bg-body.jpg is small (only 40 × 40 pixels), yet the background image when viewed in the browser covers the whole page. The reason is that a background image, by default, repeats, so when viewing the page in a browser, the background image is repeated multiple times to cover the whole page. More on background images in Chapter 5.

Inheritance and the Relationship Between Elements

Just before you begin selecting elements to style, it's a good idea to understand CSS inheritance and the relationship between elements.

When applying a style to an element, depending on the property being used, that style may be inherited by other elements. The following rule set makes text in the `<body>` element blue:

```
body {
    color: blue;
}
```

Assume this is applied to the following HTML (a simplified version of index.html):

```
<body>
    <div id="main" class="group">
        <h1>Page Title</h1>
        <p>Nulla facilisi. Ut porttitor sollicitudin nisi, tempus
  pulvinar nisl volutpat aliquet.</p>
    </div>
</body>
```

Although the `<body>` element doesn't contain any text itself, the text within `<h1>` and `<p>` will still have the blue color applied to them. This is because the `color` property is inherited. From Chapter 5 onward, *CSS3 Foundations* explains properties in greater depth and points out whether each is inherited.

What is meant by inherited, though? Because `<body>` in the HTML example contains the `<h1>` and `<p>` elements, they are known as the *descendants* of `<body>`. An inherited property is one that is passed down and applied to descendant elements, unless that element overrides it with a property of its own.

Descendants can also be referred to as *children*, but only when an element is the direct descendant of another. For example: the <h1> and <p> elements are descendants of <body> and children of . Likewise, is a child of <body>.

An element that contains other elements may be called the *parent*, and two elements at the same level (such as the <h1> and <p> in the example) are called *siblings*.

More advanced selectors allow you to select an element based on its relationship with others, which you will see shortly.

Selectors

Selectors are the conditions of a CSS rule set. The true power of selectors comes from their capability to be combined, allowing you to create very specific conditions, applying styles only to the exact element or elements you want.

Universal Selector

The *universal selector* is represented by an asterisk (*) and selects every element that makes up a web page:

```
* {
    color: black;
}
```

In this example, every element is given a black text color. Note that this isn't a very effective method of applying this style, though. Because styles are inherited, rather than apply a style to *every* element, you could apply it to a body selector instead, which would give the exact same result, but a little quicker. Because each rule set is an action the browser carries out to turn HTML and CSS into a beautiful page, this sort of consideration means a browser can work faster and the user can see a page quicker. So, because it's not always the most efficient approach to styling multiples elements, universal selectors tend to be used with a combination of other selectors to make them apply only to specific elements rather than all of them. More on combining selectors shortly.

Type Selector

When you added a background color to the <body> element, you used a type selector to do that. A *type selector* is the name of an HTML element. A type selector can be used as follows:

1. In styles.css, add the following rule set:

```
h1 {
    font-weight: bold;
}
```

2. Save styles.css.

Here, you select the <h1> element—a type selector. This declaration makes the <h1> bold. Changing the font style is covered in more detail in Chapter 11.

Unless otherwise specified, add new rule sets below others you've already added.

ID and Class Selectors

IDs and classes, which are attributes of an HTML element, allow you to be more specific when styling elements.

As you saw in the project files, some elements already have IDs and class attributes. The main content area, for example, is a with an ID of `main` and class of `group`:

```
<div id="main" class="group">
```

If you use a type selector for this element to make its background color white, for example, you add the following to the CSS:

```
div {
    background-color: white;
}
```

The problem here is that although the main content area is now white, so too is every other on the page. By using an ID selector instead, you can select just that particular .

Go ahead and change the main content area's background color to white:

1. In styles.css, add the following:

```
#main {
    background-color: white;
}
```

2. Save styles.css.

A Note on Using and Naming Classes

The purpose of a class is modularity. When writing styles, you should ask yourself: "Could I use this style elsewhere?" If elements are to share similar styles, rather than write out two similar rule sets, you can use a class and apply that class to both elements.

When creating classes, also put some thought into their names. Although you're free to call a class whatever you like, consider whether the name of a class represents its purpose. If you come back to work on your web page in six months (or somebody else does), will your class names make sense then?

These are best practice approaches that you learn to use throughout *CSS3 Foundations*.

The hash or pound symbol (#) represents an ID. An ID should be used only once per page. Consequently, it allows you to be very specific when applying styles.

Similar to the ID is the less specific *class selector*. When elements on a page (or across a website) need to share the same styles, you should use a class selector. Unlike IDs, classes can be used on a page as many times as you want.

You will eventually apply quite a few styles to the call to action button described in Chapter 2 (remember its purpose is to attract the user's eye) and will use it multiple times on the same page. Because of this, it makes sense to use the `button` class for elements representing a button.

Style any element with the `button` class like so:

1. In styles.css, add the following:

```
.button {
    background-color: hotpink;
    color: white;
}
```

2. Save styles.css.

The full-stop, period, or dot represents a class. Here, you make the background color hot pink and make the text of the button white. You take a closer look at these property values, known as *color keywords*, in the next chapter.

Grouping Selectors

Sometimes you may want to apply a style to more than one element. You recently made the text of the `<h1>` element bold but what if you want the subtitle elements such as `<h2>`, `<h3>`, and `<h4>` to be bold too? You could write a rule set that selects each element individually, but a quicker way is to group those selectors instead.

Cross Browser Compatibility for Advanced Selectors

Up to now, the selectors demonstrated are supported in all browsers in use today. However, the more advanced selectors have a varying level of support, which will be stated in the description of each selector.

That said, this lack of support for some advanced selectors can be corrected using a free script called Selectivizr, found at `www.selectivizr.com`. When completed, the Cool Shoes & Socks page will only use selectors with full browser support. Visit the Selectivizr website for installation instructions should you choose to use advanced selectors now or in the future.

1. In styles.css, find the `h1` rule set:

```
h1 {
    font-weight: bold;
}
```

2. Modify the rule set to include other title elements, like so:

```
h1, h2, h3, h4 {
    font-weight: bold;
}
```

3. Save styles.css.

By separating each selector with a comma, you can apply the same styles to multiple elements.

Combinators

Combinators allow you to apply styles to elements based on their relationship with other elements. All the selectors you've learned up to now can be used with combinators to make even more specific selectors.

Descendant Combinators

A descendant selector describes the descendant of an element. Descendant combinators are great for styling elements that exist inside of other elements.

1. In styles.css, add the following:

```
blockquote p {
    font-style: italic;
}
```

2. Save styles.css.

In this example, only <p> elements that are descendants of <blockquote> will be made italic. By using a descendant combinator, you can be more specific about which elements you want to select.

Child Combinators

Sometimes, a descendant combinator isn't quite strict enough for your purposes. Follow these steps to use a child combinator:

1. In styles.css, add the following:

```
aside > h3 {
    font-weight: bold;
}
```

2. Save styles.css.

A child combinator is a greater-than character (>) placed between two selectors. With the selector you just added, the <h3> title element that is a child of the <aside> element will be made bold.

Child combinators are unsupported in Internet Explorer 6.

Sibling Combinators

CSS Level 3 introduces sibling combinators, allowing you to create selectors for elements when at the same level.

Adjacent Sibling Combinator

You can use the adjacent sibling combinator to style elements that share the same parent, providing one element immediately follows the other:

1. In styles.css, add the following:

```
p + .offer {
    color: red;
}
```

2. Save styles.css.

An adjacent sibling combinator is a plus symbol (+) placed between two selectors. With the selector you just added, the element <p class="offer"> is made red, *only* when it follows another <p> element.

Adjacent sibling combinators are unsupported in Internet Explorer 6.

General Sibling Combinator

Like the adjacent sibling combinator, elements making use of the general sibling combinator share the same parent; however, because their relationship is "general," one element doesn't have to immediately follow the other to be selected:

```
p ~ h2 {
    color: black;
}
```

Where an adjacent sibling selector selects only the element that immediately succeeds it, the general sibling selector in this example selects all <h2> elements that succeed a <p> element.

A general sibling combinator is a tilde symbol (~) placed between two selectors.

General sibling combinators are unsupported in Internet Explorer 6.

Attribute Selectors

Attribute selectors allow you to select elements by referring to their attributes, such as ID, class, title, src, and so on. "But wait," you cry! "Didn't you just cover IDs and classes?" Yes! Technically, the ID and class selectors you just learned about are their own unique types of selectors. You can also select IDs and classes using attribute selectors.

When you made the background color of the main section white, you used this rule set:

```
#main {
    background-color: white;
}
```

You can accomplish exactly the same goal using an attribute selector, like so:

```
div[id="main"] {
    background-color: white;
}
```

Which should you use? Because attribute selectors aren't supported in Internet Explorer 6, and ID and class selectors are just a little quicker to type anyway, I'd suggest using those instead.

Of course, elements can have a lot of other attributes you could use to create a selector. Here's an example:

1. In styles.css, add the following:

```
input[type="text"], input[type="email"] {
    border: none;
}
```

2. Save styles.css.

Here, you select the newsletter input fields with attribute types `text` and `email` and declare these elements should have no border.

The power of attribute selectors doesn't just end there.

Selecting Elements with an Attribute, Regardless of Its Value

This example selects an element such as `<a title="More information">`. As long as it has a title, it is selected regardless of the title's value:

```
a[title] {
    color: blue;
}
```

Selecting Elements with Multiple Attributes

The following example selects an element that has both of the attributes `type="submit"` and `class="button"`:

1. In styles.css, add the following:

```
input[type="submit"][class="button"] {
      font-size: 1em;
}
```

2. Save styles.css.

This will give the newsletter sign up button `<input class="button" id="submit-newsletter" type="submit" value="Sign Up" />` a font size of 1em. More on `font-size` in Chapter 11.

Other Attribute Selectors

Attribute selectors don't just stop there, but the remainders tend to be for more special use case situations, so they aren't covered in *CSS3 Foundations*.

These other attribute selectors allow you to select an element that:

- Contains multiple attribute values
- Has an attribute value list of hyphen-separated words
- Has an attribute value starting with, ending with, or containing a certain string of text

If the sound of those takes your fancy, I'd recommend reading more about them at `reference.sitepoint.com/css/attributeselector`.

Pseudo-Classes

Sometimes, a part of a web page isn't made up of a physical element, meaning it can't be selected through the use of a normal selector; this is where *pseudo-classes* come in use. Note that, unfortunately, pseudo-classes are not supported in all browsers and each of the following descriptions will state which browsers a pseudo-class is not be supported in. In *CSS3 Foundations*, you don't rely on too many pseudo-classes, and any issues that may occur in browsers that don't support these pseudo-classes are addressed in Chapter 16.

Dynamic Pseudo-Classes

Dynamic pseudo-classes allow you to style an element when it is interacted with—when a user hovers over it, for example.

At present, when a user hovers over links on the Cool Shoes & Socks page, the links don't change. You can use pseudo-classes to make it so:

1. In styles.css, add the following below the `body` rule set:

```
a:link {
    text-decoration: underline;
}

a:hover {
    text-decoration: none;
}
```

2. Save styles.css.

A pseudo-class consists of a colon (`:`) followed by the name of the pseudo-class. Here, you add two pseudo-class rule sets: one for an unvisited link and another for when a user is hovering over a link. An unvisited link is underlined, and when hovered over, that underline is removed. When the cursor moves away from the link (and is no longer being hovered over), the underline is applied again.

You can use other dynamic pseudo-classes along with `:hover`:

1. In styles.css, add the following:

```
a:visited {
    color: black;
}

a:active {
    color: red;
}

a:focus {
    outline-color: black;
    outline-style: dotted;
    outline-width: 1px;
}
```

2. Save styles.css.

The `:visited` styles indicate—you guessed it—a link that has been visited. When you make this link a different color, a user can quickly scan a page he is returning to should he wish to find where he navigated previously.

The `:active` style is applied when an element is activated by the user—that is, when the user presses the mouse button down on a link. Hold down your mouse button on a link, and you see its color change to red, until you release the mouse button.

Focus is when an interactive element is deemed to have the user's attention—when an input text box is clicked, for example. With the `:focus` pseudo-class, you add a very important style: the `outline` property. It may not seem that important at the moment, but it's crucial for accessibility. You take a deeper look at `outline` and its importance in Chapter 5.

It's possible for elements to have more than one pseudo-class at a time. A link can be both `:visited` and `:active`, for example. If you want to style a link such as that, you could use the following:

```
a:visited:active {
    color: orange;
}
```

Structural Pseudo-Classes

Structural pseudo-classes are used when you want to select elements based on generic criteria, such as the second <p> element amongst three.

Selecting Child Elements

The pseudo-class `:nth-child()` allows you to select a particular child element based on its position relative to its siblings within a parent element:

```
p:nth-child(1) {
    background-color: lightblue;
}
```

You should place an argument within the parentheses of `:nth-child()` to specify an element that is the *n*th element relative to its siblings; this can be either a number or the words `odd` or `even`. By selecting only odd or even elements, you can create an alternating color for rows within a table, for example (making the table easier to scan with the eye). Figure 3-3 shows this style applied to the Cool Shoes & Socks page.

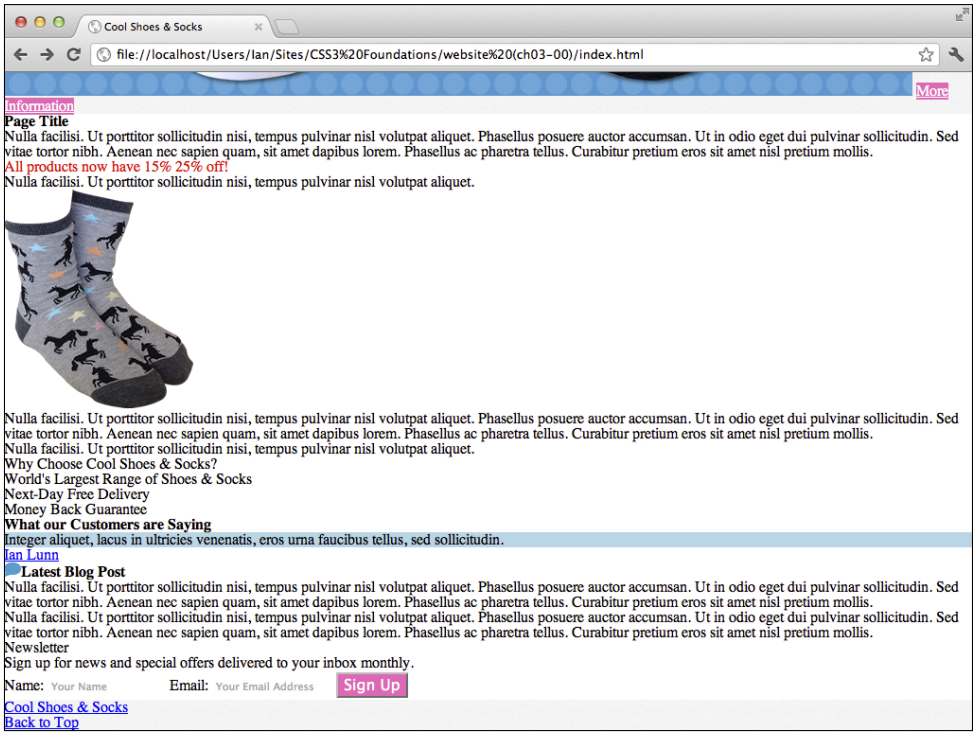


FIGURE 3-3 A light blue background color applied to p:nth-child(1).

The <p> element within the "What Our Customers Are Saying" section is given a light blue background because that is the only <p> element to be a first child among its siblings. The first child <p> elements within the content and newsletter sections are headings and not paragraphs, which is why the first paragraphs in those particular sections are not given the same background color.

Be aware that `:nth-child(1)` represents the first element, which can also be expressed as `:first-child`. Similarly, you can use `:last-child` to select the last element, or if you want to start counting from the last child, you can use `:nth-last-child()`, where `:nth-last-child(1)` is the last element and `:nth-last-child(2)` is the element next to the last.

You can also select an element if it is the only child of its parent (it has no siblings), using `:only-child`.

`nth-child()` isn't supported in Internet Explorer versions 6, 7, and 8.

Selecting Child Elements of a Particular Type

Where `:nth-child()` selects a child element based on its position among its siblings (regardless of what types of element they are), `:nth-of-type()` does the same but counts only siblings of the same type:

```
p:nth-of-type(1) {
    background-color: lightblue;
}
```

Figure 3-4 shows this style applied to the Cool Shoes & Socks page, in place of the `:nth-child(1)` rule. Now, three <p> elements are given a background color of light blue.

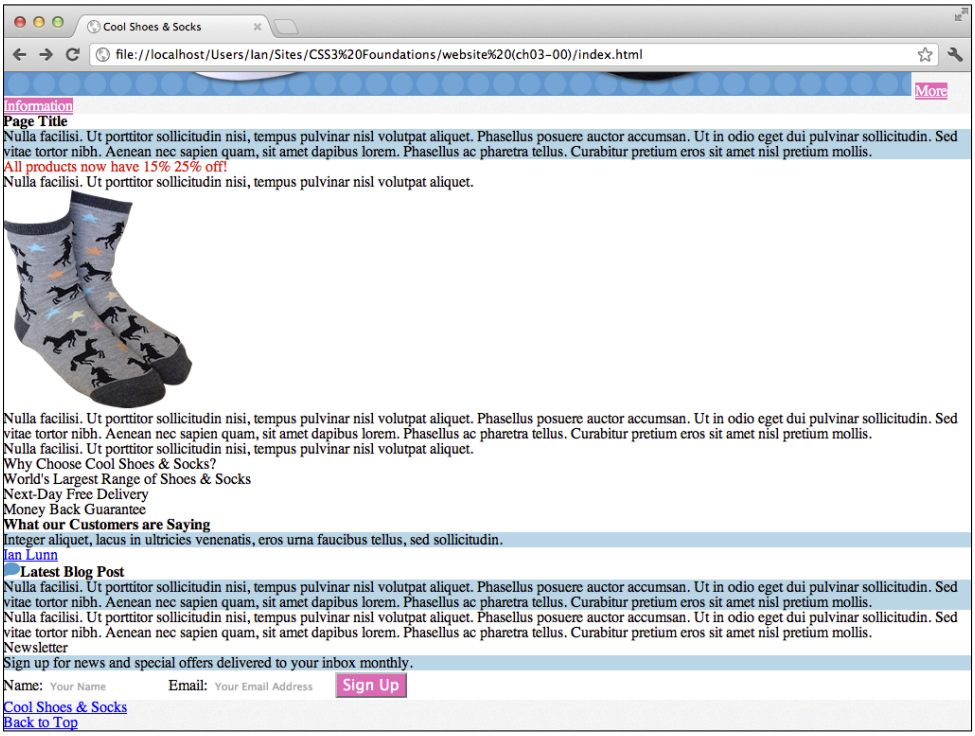


FIGURE 3-4 A light blue background color applied to p:nth-of-type(1).

As with `:nth-child()`, `:nth-of-type()` has similar variants. `:nth-last-of-type()` allows for counting elements of the same type from the last child. `:first-of-type` selects the first element of a specific type, and `:last-of-type` selects the last element of a specific type. Finally, `:only-of-type` selects an element if it is the only child of the specified type (it *can* have siblings, but only those that aren't of the same type).

`:nth-child()`, `:first-of-type()`, `:last-of-type()`, and `:only-of-type()` aren't supported in Internet Explorer versions 6, 7, and 8.

Selecting Empty Elements

Usually, you should aim to avoid having empty elements on a page (although they are sometimes used for dynamic sites where content is added after the browser or a user carries out a particular action). To select an empty element, use the `:empty` pseudo-class.

`:empty()` isn't supported in Internet Explorer versions 6, 7, and 8.

Selecting the Root Element

The root of a web page can be selected using `:root`, which most often is the `<html>` element.

`:root()` isn't supported in Internet Explorer versions 6, 7, and 8.

The Target Pseudo-Class

You are able to place an anchor link on a page that refers to a section on that same page.

At the bottom of the Cool Shoes & Socks page, there is a Back to Top anchor link, which links to the element with an ID of `header`. When the user clicks that link, the browser moves back up to the header element (at the top of the page). The header is the target of this link.

If a user clicks on an anchor link, you can style its target to make it stand out, like so:

```
 :target {
    color: red;
}
```

The UI Element States Pseudo-Classes

The `:enabled` and `:disabled` pseudo-classes are used to select elements in enabled and disabled states. For example, on a dynamic page, a form may contain input fields irrelevant to a particular user and can be disabled to prevent the user from interacting with that field.

You disable a field in the HTML, as follows:

```
<input disabled type="email" placeholder="Your Email Address" />
```

A field that isn't disabled is considered enabled. If you want to prevent user frustration when disabling an input field, it's a good idea to change its color to represent that, like so:

```
:disabled {
    background-color: lightgray;
}
```

Another element state pseudo-class is `:checked`, which again relates to input forms—this time, check boxes and radio buttons. When an element is checked, it can be styled as follows:

```
:checked {
    background-color: green;
}
```

`:enabled()`, `:disabled()` and `:checked()`, aren't supported in Internet Explorer versions 6, 7, and 8.

The Language Pseudo-Class

You can use the `:lang` pseudo-class to declare styles based on the language of an element. Assume, for example, a quote is in French, like this:

```
<blockquote>
    <p lang="fr">Integer aliquet, lacus in ultricies venenatis,
  eros urna faucibus tellus, sed sollicitudin.</p>
</blockquote>
```

The HTML specifies the quote is of a French language (`lang="fr"`). To apply a style to only elements of French language, you can use the following:

```
:lang(fr) {
    color: blue;
}
```

`:lang()` isn't supported in Internet Explorer versions 6 and 7.

The Negation Pseudo-Class

The negation pseudo-class, `:not()`, allows you to select elements that do not represent its argument. As you saw with the language pseudo-class, you can style an element if it matches a particular language, but what if you want to style all languages except French? Rather than add each language code to the language pseudo-class, you can do the following:

```
:not(:lang(fr)) {
    color: blue;
}
```

Now *all* elements that *aren't* of the French language have blue text.

`:not()` isn't supported in Internet Explorer versions 6 and 7.

Pseudo-Elements

Pseudo-elements are parts of a web page that have physical form but aren't defined by an element—the first line of a paragraph, for example.

Officially, a pseudo-element is made of two colons (`::`) followed by the name of the element. When you use *two* colons, pseudo-elements are visually distinguishable from pseudo-classes (with their *one* colon). However, using one colon is also acceptable, so the choice is yours. I use just one colon throughout *CSS3 Foundations*.

Selecting the First Line

To select the first line of text, use the pseudo-element `:first-line` (or `::first-line` if you're going with two colons), like so:

```
p:first-line {
    font-weight: bold;
}
```

Selecting the First Letter

To select the first letter, use the pseudo-element `:first-letter`, as shown in the following steps:

1. In styles.css, add the following:

   ```
   #content p:first-of-type:first-letter {
       font-size: 4em;
       font-weight: bold;
   }
   ```

2. Save styles.css.

Here, you combine quite a lot of the selectors you've learned so far. You use a descendant combinator to select only <p> elements within the `<div id="content" role="main">` element. You also combine the `:first-of-type` pseudo-class with `:first-letter`, which has selected the first element of type <p>, and then select only the `:first-letter` from that.

By using this selector, you style the first letter of the content section with a large and stylish typographic letter (as shown in Figure 3-5), much like an old book! You learn what an *em* is in the next chapter and font styles such as `font-weight` in Chapter 11.

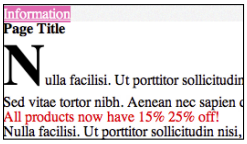


FIGURE 3-5 The first letter of the content area styled to be bigger and bolder than other text.

Generating Content Before and After an Element

Sometimes, you might want to add a style before or after an element that doesn't warrant using another element within the HTML to do that. For this, you can use `:before` and `:after` pseudo-elements.

Using `:before` and `:after`, add some quotation marks to the customer testimonials:

1. In styles.css, below the `blockquote p` rule set, add the following:

```
blockquote p:before {
    content: "\201C";
}
```

2. Save styles.css.

Here, you use another descendant combinator to select only `<p>` elements within `<blockquote>` elements and then tell the browser to create a pseudo-element before the `<p>` element and give it the content `"\201C"`. What on earth is `\201C`?! It's an American Standard Code for Information Interchange (ASCII) value, which is a scheme used by computers to understand text. The `\201C` represents an opening quotation mark (`"`). You can find more ASCII codes at `www.ascii.cl/htmlcodes.htm`.

As you see, this CSS rule adds a quotation mark before the quote. In Chapter 5 you see more uses for the `content` property, *and* if you're thinking "what place does content have in CSS?" I explain that too.

The `:after` pseudo-element works in the same way:

1. In styles.css, below the `blockquote p:before` rule set, add the following:

```
blockquote p:after {
    content: "\201D";
}
```

2. Save styles.css.

This time, `\201D` represents a closing quotation mark.

The `:before` and `:after` pseudo-elements have huge potential. Because they are non-physical elements (they're not a part of the HTML), you can do a lot with them without having to change the structure of your web page.

Sometimes you may want to use a pseudo-element to apply styles to the page but not have to put content inside the element using the `content` property. Because pseudo-elements don't work in the absence of the `content` property, you can simply use the declaration `content: "";` to trick the browser into showing the pseudo-element. You'll see an example of this in Chapter 8.

Selector Specificity and the Cascade

Now that you've learned about selectors, you probably realize that they are certainly powerful things, often with more than one way to select the same element. For example:

```
header#header > nav[role="navigation"] {
    color: blue;
}

nav {
    color: red;
}
```

These selectors both select the navigation element `<nav role="navigation">`.

The question is: Because these rules select the same element but try to apply different styles to it, which gets used?

Quite simply, the most specific selector wins. Because I've gone over the top with the first selector (I'm just showcasing my selector superpowers, but in the real world, the less your selectors consist of the better), the browser applies the first style.

This is known as *specificity,* and it is calculated in a points system, as follows:

- Count the number of ID selectors in the selector
- Count the number of class selectors, attribute selectors, and pseudo-classes in the selector
- Count the number of type selectors and pseudo-elements in the selector
- Ignore the universal selector

IDs are deemed to be very specific (because they are unique to the page) and count as 100 points. Each class, attribute, and pseudo-class selector counts as 10 points, and each type or pseudo-element selector counts as 1 point.

So, out of the two selectors in the example, `header#header > nav[role="navigation"]` gets 112 points and `nav` gets only 1!

If the first rule should be more important than the other, I could be more sensible and just make it `header nav` (giving it 2 points). This is better for performance and makes CSS more manageable in the future; plus, you can still clearly see this rule is more specific, without going to the lengths of calculating specificity!

What if both rules are as specific as each other? For example:

```
nav {
    color: red;
}

nav {
    color: blue;
}
```

In this case, the latter rule always wins.

This attribution of importance is known as the *cascade*—where the *C* in CSS comes from.

The !important Rule

If you're the type of character who doesn't play by the rules and that's how you want your CSS to be, let me introduce you to `!important`.

You've learned the rules of the cascade, but sometimes you might need to make a selector that has less specificity more important than those with higher specificity.

The `!important` rule is ideal during development when you just want to see a style applied to the page, regardless of specificity. It's arguable whether `!important` should be used in production at all because it's better for performance that you respect the cascade. Here is a good use case, though:

```
.error {
    color: red !important;
}
```

This approach tends to be used on dynamic sites. Imagine a contact form in which the user hasn't added all the required details. It's important that the color of the error message be red (above any other color potentially being applied to this element through other selectors) so that the user knows he's made a mistake and his eye is drawn to it.

The `!important` rule should start with an exclamation mark (`!`) and be placed after a property value before the closing semicolon.

Summary

Although you've seen a lot of selectors in this chapter, the key is to keep them as simple as possible. While you're learning how to create and style web pages, there is nothing wrong with just using type, class, and ID selectors until you feel comfortable enough to try others.

The reason there are so many selectors available is to allow for creating CSS that is more efficient, modular, and robust. That doesn't mean, however, that if you don't use advanced selectors your pages will be slow. A well-considered selector against a more beginner-friendly one may shave a few milliseconds off the time it takes a browser to render a page—something that only really pays off on bigger websites with heavy amounts of user traffic.

In the next chapter, you learn the different types of values that can be used with properties to style a page just the way you want!

chapter four

Creating Styles Using Property Values

SO FAR, YOU'VE used a handful of properties to add styles to the page, such as `background-color`, `background-image`, and `color`. You learn more of these properties from Chapter 5 onward. Before that, take a look at the different types of values that can be used with properties, allowing you to apply the exact styles you want to a page.

Color Keywords

You've already added some colors to the page, which all made use of color keywords, like so:

```
#main {
    background-color: white;
}
```

White is one color keyword of 16 that CSS defines:

- Black
- Silver
- Gray
- White
- Maroon
- Red

- Purple
- Fuchsia
- Green
- Lime
- Olive
- Yellow
- Navy
- Blue
- Teal
- Aqua

In CSS Level 3, another 146 color keywords have been added. I don't list them all here, but you can find them at `www.w3.org/TR/css3-color/#svg-color`.

So, as those extended color keywords were introduced in CSS Level 3, does that mean they won't work in older browsers that don't support CSS3? Not quite. This is one of those times when CSS3 was updated to match what was already being implemented by browsers. The extended colors, known as *X11 colors,* are originally from another browser technology—SVG. SVG has been supported in browsers for many years, and because of that, so too have most of the extended colors. I say "most" because you may find some browsers don't support all 146 extended colors.

Having 162 colors isn't that breathtaking, is it? The web would be a boring-looking place if you had access to only that many colors.

What if you're creating a website for somebody else and she asks you for "a color similar to hot pink, except not quite hot pink, more like a dark strawberry color, except closer to a raspberry red"? First, I sympathize with you! Second, no, that's not a color keyword, but with the possibility to use more than 16 million colors, you'll be glad to hear that there are not 16 million color keywords.

Color Values

You can also use other color values and functions to access more than 16 million colors; they are RGB and HSL.

RGB (Hexadecimal)

RGB stands for Red, Green, Blue, and in its hexadecimal (hex for short) form, it consists of a hash or pound symbol (`#`), followed by either three (RGB) or six (RRGGBB) hexadecimal characters. Hexadecimal characters are 0–9 and A–F.

Project Files Update (ch04-00): If you haven't followed the previous instructions and are comfortable working from here onward or would like to reference the project files up to this point, you can download them from `www.wiley.com/go/treehouse/css3foundations`.

I mentioned previously that you should make a background color closely match the background image. The gray color you applied earlier looks a lot darker than the gray (as shown in Figure 4-1) of the background image though, right?

FIGURE 4-1 The dark background color compared to the background image applied in Chapter 3.

You can change that using RGB:

1. In styles.css, find

```
body {
    background-color: gray;
}
```

2. Change the `gray` keyword to `#f5f5f5`.
3. Save styles.css.

Because the background image still takes precedence over the background color, you might want to comment out the background image to see the change you just made (See the sidebar Using Comments and "Commenting Out"). Remember to uncomment it again after you view the change.

Example:

```
body {
    background-color: #f5f5f5;
    /* background-image: url("../images/bg-body.jpg"); */
}
```

Using Comments and "Commenting Out"

Comments are text you can add to CSS to, well, comment! A browser ignores comments, so they aren't displayed in a web page but they do appear in the CSS file that's downloaded to a user's computer. You may like to add comments to keep notes about why you used a particular style, for example.

To let the browser know you are using a comment that it can ignore, use the following syntax:

```
/* Hi Mom. I made this web page! */
```

You should prefix a comment with a forward slash and an asterisk and suffix it with an asterisk followed by a forward slash. Comments can be used either outside or within CSS rule sets.

You can also use comments for "commenting out" styles. If you added a style to a page but want to see the page without that style for a moment, rather than delete the style, you can make it a comment, causing the browser to ignore it.

When a hex value is `#000000`, it has no color, making it black, and at the opposite end of that range, `#ffffff` is white. Your background color should now closer match the color of the background image, but how on earth do you determine `#f5f5f5` is a gray color? To get the exact color, you need to use a tool, often known as a *color picker*; see the sidebar "Color Pickers."

To make writing hexadecimal values a little quicker, when a value has repeating characters (such as `#ffffff`), you can write `#fff` instead. Three character values are expanded to six by replicating each character. For example, `#fb0` expands to `#ffbb00`.

Unfortunately, you can't shorten `#f5f5f5`.

Color Pickers

Many graphics editors come with color pickers (such as Adobe Photoshop and Fireworks). If this is something you don't already have, here are some free ones you can use:

Colors for Mac—`www.mattpatenaude.com/`.

ColorPix for Windows—`www.colorschemer.com/colorpix_info.php`.

Colorzilla Browser Add-on for Chrome and Firefox—`www.colorzilla.com/`.

RGB (Integer Range)

RGB can also be specified via an integer range from 0 – 255 or 0 – 100%. Much like hex, the bottom of the range (0) represents black, and the top (255/100%) is white.

To set the background to white, for example, you can use the following:

```
body {
    background-color: rgb(255, 255, 255);
}
```

`rgb()` is a function, and its arguments are the red, green, and blue values.

You can specify the same gray color as the background as follows: `rgb(245, 245, 245)`. Many color pickers will be able to give you these values.

RGBA

In CSS Level 3, you can specify an alpha argument along with the red, green, and blue arguments. Alpha allows an element to take on a level of transparency.

To select the main content area of the page and make its background white, with a slight transparency, follow these steps:

1. In styles.css, find the following:

   ```
   #main {
       background-color: white;
   }
   ```

2. Add another `background-color` declaration, like so:

   ```
   #main {
       background-color: white;
       background-color: rgba(255, 255, 255, 0.6);
   }
   ```

3. Save styles.css.

Here, you select the element with an ID of `main` and give it two `background-color` declarations, but why? Because RGBA is a part of the CSS Level 3 specification, it is implemented only in modern browsers. When a browser sees these declarations, if it doesn't understand `rgba()`, it renders only the first declaration, making the background color white. If it understands both declarations (in the case of modern browsers), it renders the latter (remember this is how the cascade works), setting `background-color` to `white` (255, 255, 255) with a `0.6` (60%) transparency. This is a safe way to use CSS Level 3 features, providing graceful degradation for older browsers.

HSL and HSLA

Having read about RGB, you may be thinking that specifying colors in that way is unintuitive, and you'd be right. Although you can guess at a value for a particular color, that guess will be far from accurate.

CSS Level 3 aims to make guessing color values more intuitive with the introduction of the HSL function. HSL stands for Hue, Saturation, and Lightness.

Imagine a color wheel with red at the top (0° or 360°). As you move clockwise around that color wheel, the colors fade between yellow (60°), green (120°), cyan (180°), blue (240°), magenta (300°), and then back to red. (See Figure 4-2.)

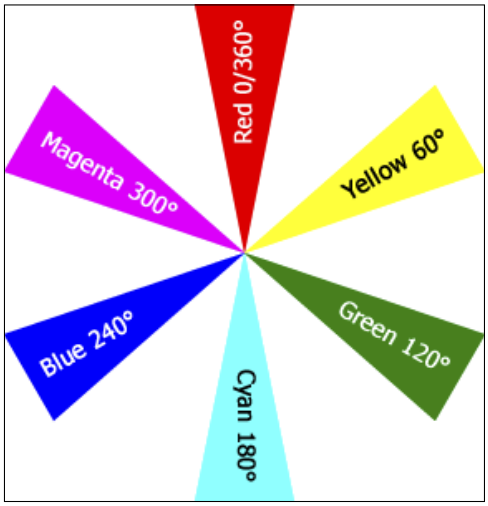


FIGURE 4-2 The position of the main colors on a color wheel.

Assuming you want a red color, you choose the hue value 0 or 360.

Next, you choose the saturation value, which is a percentage value. The higher the value, the brighter the color of red you get.

Finally, you can increase the red color's lightness using a percentage value again. Using 0% makes the color completely black, whereas 100% makes it completely white.

HSL might not sound that intuitive, but unlike RGB, it allows you to select a base color quicker and then tweak it to your needs.

Much like RGBA, an HSLA function also exists; it allows you to change the opacity of a color by using the alpha argument—a decimal value between 0 and 1, 0 being transparent, 1 being opaque.

Again, because HSL is implemented only in modern browsers, you need to add a declaration that older browsers can understand, like so:

```
body {
    background-color: #f5f5f5;
    background-color: hsl(0, 0, 0.961);
}
```

This approach is rather convoluted, though. The purpose of using HSL is to make choosing colors easier. Because older browsers require a color property they understand, you would have to go to the effort of using a color picker to determine the hexadecimal equivalent of the HSL value, defeating its purpose. Because of this, I recommend you stick with using the RGB property instead; you need to use a color picker either way!

Code Challenge: Add More Colors to the Page

In styles.css:

1. Add a rule set with the selector `#newsletter` and the declarations `color: white;` `background-color: #00ACDF;`.
2. Add a rule set with the selector `#footer` and the declarations `background-color: #ccc;` `background-image: url("../images/bg-footer.jpg");`.
3. Add a rule set with the selector `.showcase .button:hover` and the declaration `background-color: #00ACDF;`.
4. Below the rule set `input[type="submit"][class="button"]`, add a new rule set with the selector `input[type="submit"][class="button"]:hover` and the declaration `background-color: #d4326d;`.

Project files update (ch04-01): If you haven't followed the previous instructions and are comfortable working from here onward or would like to reference the project files up to this point, you can download them from `www.wiley.com/go/treehouse/css3foundations`.

Units

A *unit* is a measurement consisting of a number immediately followed by a unit identifier, such as 5cm or 20%, used for properties such as `height`, `width`, and `font-size`. Only when a unit of length is 0 can the unit identifier be omitted. There are numerous types of unit; percentages, lengths, and time to name a few.

Percentages

By using a percentage, you tell the browser how much space an element should take up, relative to its containing element. To change the width of the content area:

1. In styles.css, add the following:

```
#content {
    width: 65%;
}
```

2. Save styles.css.

At present, the content area is contained in the `div` with ID of `main`, which has a default width of 100%. This means the main area will fill the full size of the viewport (the area in which the page is rendered). Now that you've given the content area a width of 65%, try resizing your web browser to see that area expand and shrink to always be 65% of the viewport width.

Using percentages is a great way to make an element's size relative to another element's size. When making a responsive layout, you rely on percentages to allow for the changing of a web page's structure so it best fits the device on which it is being viewed.

You can apply percentages to many different types of properties: `width`, `height`, and `font-size` being just a few.

You use more percentage values and add structural CSS in Chapter 6.

Units of Length

There are two types of unit of length: absolute and relative. Absolute units consist of physical units (inches, centimeters, millimeters, points, and picas) and pixel units. Relative units consist of ems, rems, and exs.

Absolute Units

Absolute units are fixed and don't relate to one another, meaning they aren't inherited. They can be a little tricky to understand due to screens having different pixel densities (pixels are the tiny dots that make up a screen).

Physical Units

Physical units are anchored using the measurements of a device, assuming the pixel density of that device is 96dpi (96 dots per inch/pixels per inch).

Sample usage of physical units:

```
width: 10in;
width: 25.4cm; /* 1 inch = 2.54cm */
```

```
width: 254mm;
width: 720pt; /* 1 inch = 72pt */
width: 60pc; /* 1pc = 12pt */
```

If a device has 96dpi and you declare the width of an element should be 1 inch, that element is 96px wide. The problem is, not all devices are 96dpi, and therefore, physical units are not particularly useful for the web. They are, however, more useful for print stylesheets. You don't use physical units in *CSS3 Foundations*.

Pixel Units

Pixels are possibly the most common unit used for the web.

Sample usage:

```
width: 960px; /* 1px = 0.75pt */
```

Pixels are commonly relied on because they are easy to use, the downside being that you can't scale them in relation to each other.

If a user requires that the contents of a page become bigger so he can better see it, many browsers nowadays have a zoom feature that makes the whole page bigger. In this instance, the fact that pixels don't scale doesn't matter—because the page is zoomed and *everything* becomes bigger. In older browsers, this zoom feature isn't available, but in its place is a text resize feature. Users of older browsers aren't able to utilize this feature if you choose to use pixels. Furthermore, because you will want to change the size of text for mobile devices, using relative units called *ems* makes it easier in the long run, allowing you to manipulate the overall scale with just one change to the CSS.

Relative Units

Relative units are those that relate to each other, being inherited from their parent element.

Working with Relative Units

Whilst relative units are very much a part of styling the modern web, they can be a little difficult to work with due to the fact they relate to each other. When specifying a `font-size` should be 1em, what exactly does that mean? How long is a piece of string?

When specifying a relative unit or percentage, the browser will take that value and run some calculations based on other values. This is known as the *computed value*.

Say an element has a width of 500px, and you specify its child element should be 50% wide, the browser will do the calculation: 250px = 500px × 0.5.

continued

continued

When a page consists of lots of elements, these calculations can be difficult for the person styling the page to keep up with.

Using the web developer tools of a browser, you can find the calculated value of any element:

1. Open your browsers web developer tools as described in Chapter 1.
   a. The Element Inspector in all web developer tools consists of two panes: the *Document Object Model* (DOM) and the properties. The larger pane on the left (by default) is the DOM, which consists of the rendered HTML in a tree view. The smaller pane on the right (by default) is the properties, which consists of *noncomputed* and *computed styles,* and you may find your browser has other features too, which *CSS3 Foundations* doesn't cover.
   b. Noncomputed styles are those that have been taken directly from your stylesheet. Computed styles are those that the browser has calculated based on the relative and percentage units in your stylesheet. To see an example of these computed styles:
2. Navigate through the DOM by clicking the + or triangle symbols to open parent elements that contain the <h1>.
3. Once you've found it, click the <h1> element.
4. In the right-hand pane, click Computed or Computed Style (browser dependent).

Here you can see all of the computed styles applied to this element. There are quite a lot of them because this includes styles inherited from the user agent stylesheet described in Chapter 2. None of these properties are relative units or percentages because they have all been computed. Knowing the computed style often makes doing your own calculations easier.

Em

The W3C defines an *em* unit as "the computed value of the `font-size` property of the element on which it is used." Not particularly straightforward, is it? Use an em unit as follows to better understand it:

1. In styles.css, add the following to the `body` selector:

   ```
   font-size: 1em;
   ```

2. Save styles.css.

If you view the page now, you notice nothing has changed! The default font size of the `<html>` element is 16px, and because font size is inherited, the font size of the body is also 16px. By declaring a font size of `1em`, you tell the browser you want the font size to be 16px (16px = inherited value (16px) × 1em). Ems get slightly trickier when font sizes are inherited.

Now consider a list as an example to demonstrate inheritance:

```
<ul>
    <li>Item 1</li>
    <li>Item 2
        <ul>
            <li>Item 3</li>
        </ul>
    </li>
</ul>
```

Say the `<body>` element has a font size of 16px and you give a list item a font size of 2em, like so:

```
li {
    font-size: 2em;
}
```

In this case, Item 1 and Item 2 in the list have a calculated font size of 32px, but because Item 3 is a child of Item 2, it first inherits the calculated font size of 32px. Then the browser calculates 32px × 2em, giving it a font size of 64px.

To avoid this, you could just set the font size on the `<ul>`, like so:

```
ul {
    font-size: 2em;
}
```

Setting the size this way makes each list item inherit the same font size, and they all become 32px. Sometimes this result isn't avoidable, though, and you need to do the following to adjust Item 3:

```
li {
    font-size: 2em;
}
li li {
    font-size: 1em;
}
```

Here, Item 1 and Item 2 are 2 em, making them 32px (32px = inherited value (16px) × 2em). Item 3 inherits the font size of 32px; then the browser calculates 32px × 1em, giving it the same font size as its parent element.

Although ems may sound troublesome at first, they're key to creating responsive websites. They allow you to quickly scale text (and other elements) in size, which is essential for making a website accessible across many different devices. In older versions of Internet Explorer, a web page that uses ems allows users to scale text themselves via built-in browser options.

There is a nice trick you can use to make working with ems a little easier:

1. In styles.css, find the font size you applied to the `body` selector:

   ```
   font-size: 1em;
   ```

2. Change its value to `62.5%`:

   ```
   font-size: 62.5%;
   ```

3. Save styles.css.

Here, you tell the browser to make the body's font size 62.5% of its default size (inherited value (16px) × 0.625 = 10px). Now 1 em is the equivalent of 10px, 1.2em is 12px, and so on. Figure 4-3 shows this in action.

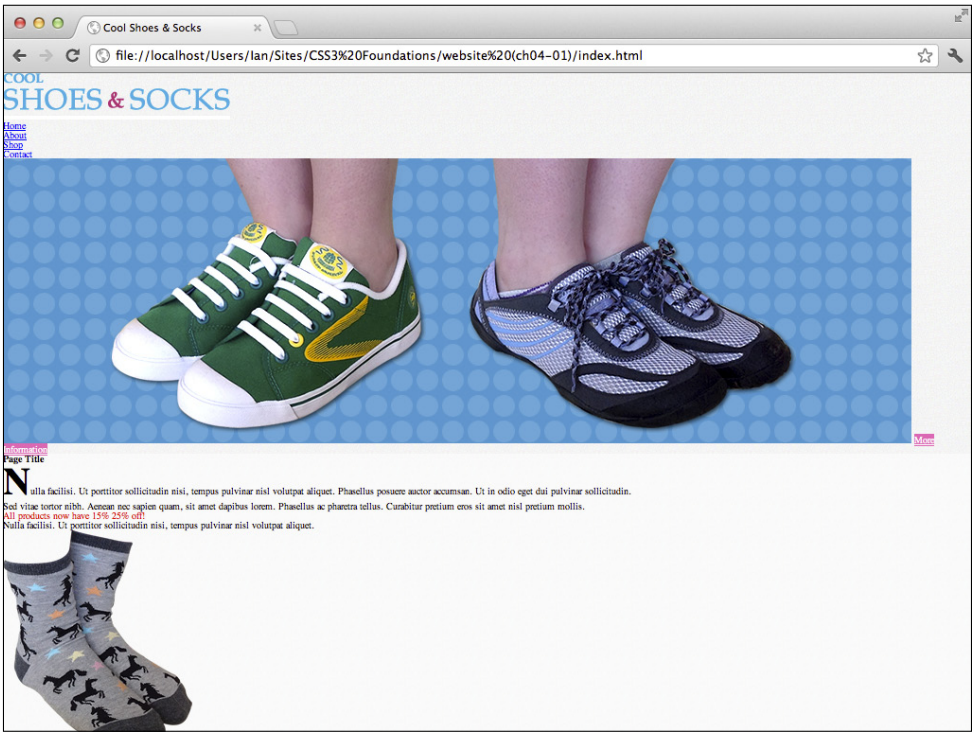


FIGURE 4-3 The Cool Shoes & Socks page with a font size of 62.5% (10px).

Yikes! As shown in Figure 4-3, the text is much smaller now! You can fix that:

1. In styles.css, below the rule set for the `body` selector, add the following:

```
#header, #main, #footer {
    font-size: 1.6em;
    /* inherited value (10px) x 1.6em = 16px */
}
```

2. Save styles.css.

Ems can be used for many more properties. You will see examples of this later in the book.

Rem

CSS Level 3 introduces a new unit called *Rem* (Root em). The W3C is aware of the difficulties when dealing with inherited values and em, so it introduced rem. Rem inherits its value only from the root element, that being the `<html>` element. So, in the em unit example, Item 3 inherited the value of its parent—causing its font size to double; when you use rem, this wouldn't happen because each element inherits only the same font size from the root.

Although this is an easier way to use ems, unfortunately, it isn't supported in all browsers yet, so for the moment, you cannot use rems to create a cross-browser-compatible website.

Ex

Ex refers to the x-height of a font (the height of the lowercase *x*) but unfortunately isn't particularly reliable. Because of this, it is less commonly relied on for creating web pages; therefore, you don't use this unit in *CSS3 Foundations*.

Other Units

The CSS3 specification lists other units that haven't been covered in this chapter. You will learn to use angle units in Chapter 12 when rotating elements, and time units in Chapter 14 when animating elements.

The specification also includes frequency and resolution units, which tend to have more of a special use-case and viewport-percentage length units that, as yet, are unsupported in browsers.

If you'd like to know more about these unit types, please see the CSS3 Values module at: `www.w3.org/TR/css3-values`.

Summary

Getting to know the different types of property values means the way in which you style a page can become more diverse and you can commit your design ideas to screen quicker, with greater accuracy.

In the next chapter, you'll learn many new presentational properties on which to use the values described in this chapter. The Cool Shoes & Socks page will really start to transform into something that closer resembles a modern day web page.

chapter five

Adding Presentational Styles

IN THIS CHAPTER, you take a deeper look at some of the properties you've already used, learn how to use shorthand properties and property values, and learn many more properties to make Cool Shoes & Socks really stand out!

Before you begin reading about presentational styles, first set up the page to give it a little more structure:

Project Files Update (ch05-00): If you haven't followed the previous instructions and are comfortable working from here onward or would like to reference the project files up to this point, you can download them from `www.wiley.com/go/treehouse/css3foundations`.

1. In styles.css, find the `body` rule set and add the following declarations:

```
margin: 0 auto;
max-width: 960px;
```

2. Save styles.css.

These properties—that center the web page as shown in Figure 5-1—are covered in Chapters 6–9. It's kind of looking more like a web page now, right?

These properties—that center the web page as shown in Figure 5-1—are covered in Chapters 6–9. It's kind of looking more like a web page now, right?

Property Definitions

In *CSS3 Foundations*, each property definition begins with key information about that property, which consists of:

- The property's initial value
- Whether the value is inherited
- Which elements the property can be applied to
- Whether the property is from the CSS2.1 or CSS3 specification
- Which browsers the property is supported in

A property's initial value is that which is applied to an element prior to your changing it via CSS. For example, many elements have an initial background color of `transparent`.

If a property is inherited, it is passed down to the child elements of the element to which it is applied. For example, when you gave the `<body>` a `font-size` of `62.5%` in Chapter 4, that `font-size` was inherited by all the elements contained within the `<body>`.

If, by default, a property is not inherited, you can give a child element a property with the value `inherit` to tell the browser to apply the same property value used for the element's parent.

Not all properties can be applied to all elements. The `table-layout` element, for example, can be applied only to table and inline-table elements such as `<table>` and `<td>`.

If a property is from the CSS2.1 specification, unless otherwise stated, it is safe to use in all browsers. Properties from the CSS3 specification may not be safe to use in all browsers—particularly Internet Explorer versions 6, 7, and 8; if this is the case, I explain how best to work around that. Furthermore, experimental properties from CSS3 may require vendor prefixes, which you learn how to use shortly.

Each property definition also includes a list of browser versions that support that property, both with and without vendor prefixes where necessary.

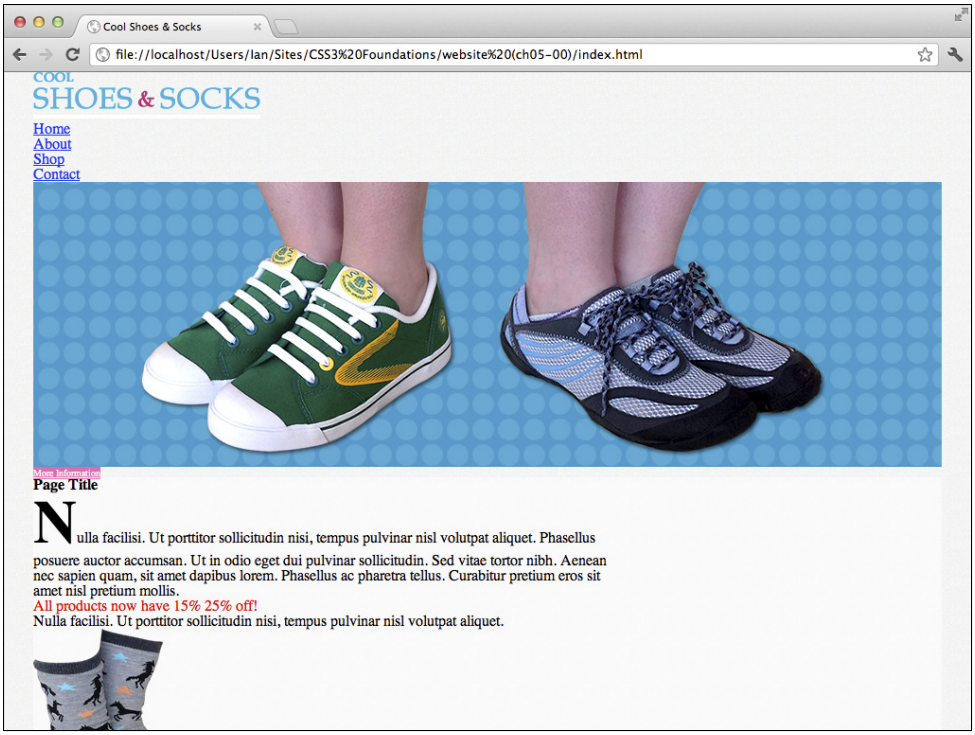


FIGURE 5-1 The web page centered and with a defined width of 960 pixels.

Using Experimental Properties Safely

In Chapter 1, I mentioned that some of the CSS3 modules are incomplete and at various stages prior to reaching the Recommendation status.

With many of CSS3's modules still being written, browser vendors are constantly updating their browsers to better support CSS3's new features. The more they can implement, the more they and their users can test and provide feedback on, so it is very important for a browser to keep up with the latest web technology.

To make implementing experimental features as safe as possible, it was decided that each browser should have its own prefix to be placed before a CSS property. That way, if two browsers do implement a feature in a different way, that feature viewed in one browser doesn't appear broken in the other; they both refer to the same feature but may have their own unique implementation of it. When a feature becomes official, the vendor-prefixed property then becomes obsolete, and the official property is relied on instead.

So, what exactly are vendor prefixes and how do you use them?

One example of an experimental property is `transition-duration`, which you will learn to use in Chapter 14. A `transition-duration` declaration may be written like so:

```
transition-duration: 2s;
```

This is the official property name that is described in the CSS3 Transitions module. However, as at the time of writing, that module is a Working Draft, for the safest implementation, `transition-duration` should be written with a vendor prefix for each browser, like so:

```
-webkit-transition-duration: 2s;
-moz-transition-duration: 2s;
-ms-transition-duration: 2s;
-o-transition-duration: 2s;
transition-duration: 2s;
```

These vendor prefixes apply to the following browsers:

- `-webkit-` — Google Chrome and Apple Safari (they use the WebKit Layout Engine)
- `-moz-` — Mozilla Firefox
- `-ms-` — Microsoft Internet Explorer
- `-o-` — Opera

The vendor-prefixed properties should be followed by the official, unprefixed property, which ensures that when this property does become official, the standardized property will be used—instead of the vendor-specific one.

In the preceding example, the same property is repeated five times. If you're anything like me, typing it that many times sounds like a hassle. It's important to use vendor prefixes, though!

You can be thankful that there are free tools that allow you to write only one vendor prefix during development, and they will fill in the rest for you. When you use vendor prefixes throughout *CSS3 Foundations*, write one only for the browser in which you've decided to create your web page (I use `-webkit-` for Google Chrome), and in Chapter 15, you use a tool that adds the other vendor prefixes to your CSS automatically.

When a property should be used with vendor prefixes, I'll let you know. However, as the web is constantly evolving, if you'd like to be certain of whether a property still requires a vendor prefix or not, I recommend visiting a website called When can I use… (`www.caniuse.com`). Not only will When can I use… tell you which properties are supported in which browsers, it'll also show whether you need to use vendor prefixes for that property.

At the time of writing, browser vendors are making a push toward removing vendor prefixes from the more established CSS3 properties and as time goes on, fewer vendor prefixes will be required, until they eventually become obsolete.

Using vendor-prefixed properties when they are no longer necessary won't cause any problems and it is better safe than sorry. A browser will simply skip over a vendor-prefixed property that it may no longer support and instead use the official property. So, providing you place the official property below the vendor-prefixed ones, your CSS will remain future proof.

Borders

Borders are graphical elements that are applied around the edge of an element.

border-color

Initial value: the value of `color` | Inherited: No | Applies to: All | CSS2.1

Browser support: IE 4+, Firefox 1+, Chrome 1+, Opera 3.5+, Safari 1+

As you would expect, `border-color` specifies the color of a border and can be given a variety of values, such as RGB, RGBA, HSL, HSLA, and color keywords.

1. In styles.css, find the `#main` rule set and add the following declaration:

```
border-color: #ccc;
```

2. Save styles.css.

This gives the border around the main content container `<div id="main" class="group">` a gray border color. *However*, for borders to work, you need to specify a `border-style` and `border-width` along with the `border-color` property. Because the initial value of `border-style` is none, you cannot see the border yet.

In the CSS2.1 specification, the initial value for `border-color` is the same as that of the `color` property. So if a `border-color` isn't specified and the `color` is red, then the `border-color` will be red too. In CSS3, the initial value is the keyword `currentColor`, which achieves exactly the same—the `border-color` will be the same as the `color`. The introduction of this keyword is just to allow you to specifically show you want the `border-color` to be the same as the `color`. If you were working in a team with multiple people working on the same stylesheet for example, the following clearly shows you intend to have the `border-color` be the same as the `color`, whereas without the `border-color` declaration, it may look like you forgot to add that style:

```
body {
    color: red;
    border-color: currentColor;
}
```

Although `currentColor` is a CSS3 keyword and may not work in older browsers, those older browsers will simply ignore it and just apply the same `color` value to the `border-color` property anyway, so it can be safely used across all browsers.

border-style

Initial value: `none` | Inherited: No | Applies to: All | CSS2.1

Browser support: IE 4+, Firefox 1+, Chrome 1+, Opera 3.5+, Safari 1+

`border-style` styles the border of an element, unless that element also has a `border-image` declaration.

1. In styles.css, underneath the `border-color` declaration you just added to the `#main` rule set, add the following:

   ```
   border-style: solid;
   ```

2. Save styles.css.

The values you can use with the `border-style` property are `none`, `hidden`, `dotted`, `dashed`, `solid`, `double`, `groove`, `ridge`, `inset`, and `outset` (see Figure 5-2).

The `none` keyword can be used with many properties and will remove a style that is inherited in the cascade or for example when you wish to have a style removed when an element is hovered over.

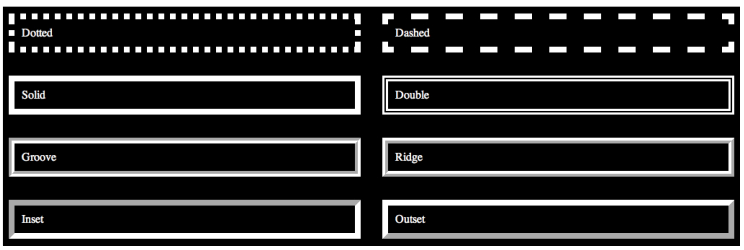


FIGURE 5-2 The dotted, dashed, solid, double, groove, ridge, inset, and outset border styles each with a border-color of white and border-width of 8px.

The border styles page can be found in the project files ch05-00, named `border-styles.html`.

You'll be able to see the border after you specify a `border-width`...I promise!

border-width

Initial value: `medium` | Inherited: No | Applies to: All | CSS2.1

Browser support: IE 4+, Firefox 1+, Chrome 1+, Opera 3.5+, Safari 1+

The `border-width` property sets the thickness of a border.

1. In styles.css, underneath the `border-style` declaration, add the following:

```
border-width: 1px;
```

2. Save styles.css.

Now finally, you can see a thin border applied around the main content area.

For `border-width`, you can specify all values of length as well as the keywords `thin`, `medium`, and `thick`.

border (Shorthand)

Browser support: IE 4+, Firefox 1+, Chrome 1+, Opera 3.5+, Safari 1+

When CSS properties share similarities, you can use one shorthand declaration in place of multiple declarations.

The `border` property is a shorthand way of writing the three properties you just added. Go ahead and change those declarations into just one declaration:

1. In styles.css, in the `#main` rule set, remove the following declarations:

```
border-color: #ccc;
border-style: solid;
border-width: 1px;
```

2. Add the shorthand `border` declaration:

```
border: #ccc solid 1px;
```

3. Save styles.css.

The `border` property requires three values (in any order), separated by spaces. By adding this shorthand declaration, you achieve the same effect using less code and less time!

Now, if you're wondering how you style the border of just one side of an element, you can do that using `border-left-width`, `border-top-style`, and so on. This means that `border-color`, `border-style`, and `border-width` are shorthand properties too; they apply styles to all four borders of an element in one go.

Try styling just one side of an element using a shorthand property:

1. In styles.css, find the `#footer` rule set, and below its other declarations, add the following:

```
border-top: #999 dotted 4px;
```

2. Save styles.css.

Here, you style the top border of the footer element with a dark gray dotted border (see Figure 5-3). You use a shorthand property, which is the equivalent of:

```
border-top-color: #999;
border-top-style: dotted;
border-top-width: 4px;
```

FIGURE 5-3 The footer area with a dotted border applied only to the top.

I like things to be as simple as possible. It's not that I'm lazy...honest! If you like the simplicity of shorthand properties, you'll be pleased to know CSS has quite a few more like these, which are covered as you make your way through *CSS3 Foundations*.

border-radius

Initial value: `0` | Inherited: No | Applies to: All, except internal table elements when `border-collapse is collapse` | CSS3

Unprefixed browser support: IE 9+, Firefox 4+, Chrome 4+, Opera 10.5+, Safari 5+

Prefixed browser support: Firefox 1+, Chrome 0.2+, Safari 3.0+

`border-radius` enables you to give elements rounded corners. It is supported in all modern browsers but not Internet Explorer 6, 7, and 8. Because it's a visual effect and nothing more, the fact that old versions of Internet Explorer don't render rounded corners isn't a problem; they just remain square.

1. In styles.css, find the `#newsletter` rule set, and below its other declarations, add the following:

```
border-radius: 8px;
```

2. Save styles.css.

Here you give the newsletter box very subtle rounded corners (see Figure 5-4). Because the CSS3 Backgrounds and Borders Module (`www.w3.org/TR/css3-background/`), which this property is a part of, is a "Candidate Recommendation," you can use `border-radius` without vendor prefixes.

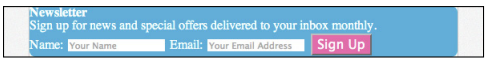


FIGURE 5-4 The newsletter box with subtle rounded corners.

Although you haven't given the newsletter box a style, `border-radius` still applies (because whether or not an element has styled borders, the border of an element still exists).

`border-radius` is shorthand for `border-top-left-radius`, `border-top-right-radius`, `border-bottom-left-radius`, and `border-bottom-right-radius`. By specifying one value of `8px` for the `border-radius`, you tell the browser you want all four corners to be `8px`. What if you want each corner to have a different value?

```
border-radius: 8px 12px 16px 20px;
```

This declaration gives the top-left corner a radius of `8px` and then works clockwise from there, giving the top-right a `12px` radius; bottom-right, `16px;` and bottom left, `20px`.

You also are able to create corners that aren't perfectly round by specifying two values for a corner, one for the horizontal radius, and the other for the vertical radius, like so:

```
border-top-right-radius: 8px 20px;
```

If this declaration is applied, the top-right corner of an element has a horizontal curve of `8px`, going into a vertical curve of `20px`.

If you want this noncircular radius to apply to all corners, you can use the shorthand `border-radius` property, first specifying the four horizontal values, followed by a slash `/` and then the four vertical values, like so:

```
border-radius: 8px 8px 8px 8px / 20px 20px 20px 20px;
```

Or, because those values are repeating, an even shorter way would be to write

```
border-radius: 8px / 20px;
```

Border Images

If none of the border styles covered so far takes your fancy, CSS Level 3 introduces border images, which allow you to use your own images in place of the basic styles described previously. Border images are unfortunately at various states between browsers at the moment. Google Chrome has good support, Firefox partial, Internet Explorer versions 6–9 have no support at all, and other modern browsers support only the shorthand property `border-image`.

Because border image is another visual style, it doesn't particularly matter that Internet Explorer doesn't show a border image. Let's look at the longhand properties for border images and then convert them to shorthand so the style works in the majority of browsers.

border-image-source

Initial value: `none` | Inherited: No | Applies to: All, except internal table elements when `border-collapse is collapse` | CSS3

Unprefixed browser support: Firefox 15+, Chrome 15+

`border-image-source` enables you to specify an image to use for the border by using a `url()` function with a path to the image (the source) as an argument. You take a closer look at the `url()` function and its arguments in the `background-image` definition shortly.

Figure 5-5 shows the custom border to be used on the Cool Shoes & Socks website. Although you apply it only to the top border of the footer section, the image to be used should still consist of four sides.

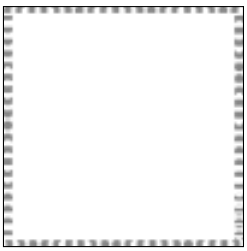


Figure 5-5 The image to be used for the border of the newsletter box.

1. In styles.css, below the other declarations in the `#footer` rule set, add the following:

```
border-image-source: url("../images/bdr-footer.png");
```

2. Save styles.css.

As you saw with the other border declarations, it takes a few of them to work together before you get a result.

border-image-slice

Initial value: `100%` | Inherited: No | Applies to: All, except internal table elements when `border-collapse is collapse` | CSS3

Unprefixed browser support: Chrome 15+

When the browser knows which image you want to use, it slices it into nine parts; four corners, four edges, and one center piece. How that slicing occurs depends on the `border-image-slice` property.

1. In styles.css, below the `border-image-source` declaration, add the following:

```
border-image-slice: 4;
```

2. Save styles.css.

The `border-image-slice` property can be given up to four number values. Here, `border-image-slice: 4;` is the same as `border-image-slice: 4 4 4 4;`, the first value being the top and then working clockwise from there.

Note that unless using percentage values, the `border-image-slice` value should not be given a unit identifier (such as px). This is because the values represent pixels when using a raster image and co-ordinates when using a vector image (a file ending with a .svg extension).

As shown in Figure 5-6, a `border-image-slice` of `4` will slice the raster image 4px from the edge of the image.

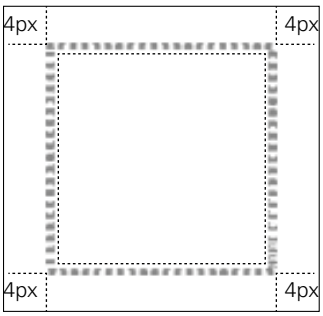


Figure 5-6 The border image is sliced with the border-image-slice property set to `4`.

border-image-width

Initial value: 1 | Inherited: No | Applies to: All, except internal table elements when `border-collapse is collapse` | CSS3

Unprefixed browser support: Firefox 13+, Chrome 15+

`border-image-width` determines the width of the image applied to the border.

In most cases, you don't need to specify a `border-image-width` because its initial value of `1` along with the use of the `border-width` property is usually adequate. The border already has a width of `4px`, which you specified on the `#footer` element earlier, like so:

```
#footer {
    background-color: #ccc;
    background-image: url("../images/bg-footer.jpg");
    border-top: #999 dotted 4px;
}
```

The `border-image-width` can be either a percentage value (or four percentage values for the top, right, bottom, and left), which causes the border's width to be a certain percentage of the element it is applied to, or a number (or four numbers for top, right, bottom, and left), which is a multiplication of the calculated `border-width`. If you specify a `border-image-width` of `4`, the `border-image-width` is 16px = 4px × 4.

As shown in Figure 5-7, the border image is now applied to the footer in Google Chrome (other browsers don't work until the shorthand `border-image` property is added). At the moment, though, the border image is stretched because the `border-image-repeat` property's default value is `stretch`. You can fix that...

FIGURE 5-7 The border image stretched across the footer.

border-image-repeat

Initial value: `stretch` | Inherited: No | Applies to: All, except internal table elements when `border-collapse is collapse` | CSS3

Unprefixed browser support: Firefox 15+, Chrome 15+

The reason the border image is currently stretched is that the initial value of `border-image-repeat` is `stretch`.

1. In styles.css, below the `border-image-slice` declaration, add the following:

```
border-image-repeat: repeat;
```

2. Save styles.css.

This tells the browser to take each of the nine slices and repeat them where necessary (see Figure 5-8). Because you're applying only this border image to the top border, the top edge slice (or tile) of the nine slices is repeated until it fills the area.

FIGURE 5-8 The border image repeating across the footer.

The Backgrounds and Borders module also describes two other values that can be used: `round` and `space`. Note that at present no browsers support these values. `round` repeats the image, but if the repeated tile does not fill the area using a whole number of tiles, it rescales the image so it does. `space` also repeats, but if the tile does not fill the area in the same way, the extra space is distributed around the tiles.

border-image-outset

Initial value: `0` | Inherited: No | Applies to: All, except internal table elements when `border-collapse is collapse` | CSS3

Unprefixed browser support: Chrome 15+

Another option for border images is the outset, which specifies how far outside an element the border should be positioned:

```
border-image-outset: 20px;
```

This example causes the border image to be 20 pixels outside the element it is applied to (see Figure 5-9). The `border-image-outset` can be any unit of length, and as with values for the other border image properties, you can specify up to four, representing the top, right, bottom, and left borders.

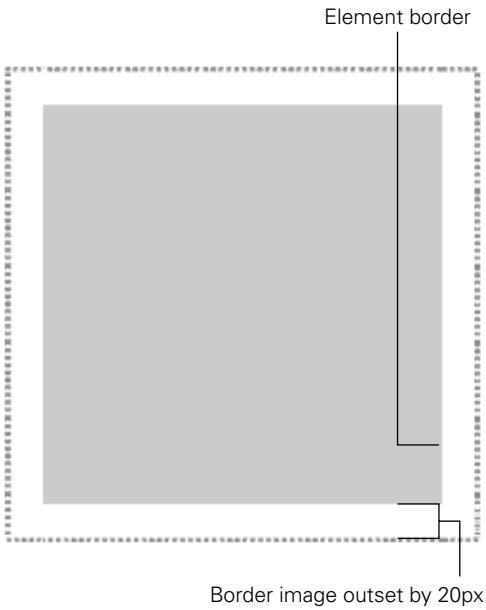


FIGURE 5-9 An element with a border-image and border-image-outset of 20px applied to it.

If you're a fan of math, you can use a multiplication number too, just like the `border-image-width` number, which is a multiplication of the computed `border-width`.

border-image (Shorthand)

Unprefixed browser support: Firefox 15+, Opera 11+

Prefixed browser support: Firefox 3.5+, Chrome 7+, Opera 10.5+, Safari 3+

As with the other border properties, border image properties have a shorthand property too and, as mentioned, at present, most browsers support only the use of the `border-image` property. Therefore, it's wise to use that property in place of the other border image properties.

The shorthand property should take the following syntax:

```
border-image: border-image-source border-image-slice border-image-
width border-image-outset border-image-repeat
```

When any of the preceding values are omitted, the properties initial value is used.

1. In styles.css, find the `#footer` rule set and remove the following declarations:

```
border-image-source: url("../images/bdr-footer.png");
border-image-slice: 4;
border-image-repeat: repeat;
```

2. Add the shorthand declaration in their place:

```
-webkit-border-image: url("../images/bdr-footer.png") 4 repeat;
```

3. Save styles.css.

Rather than use three properties here, you use one; specifying the `url()`, followed by the slice value and then the repeat value.

As mentioned earlier in this chapter, during the creation of the Cool Shoes & Socks website, I use the `-webkit-` vendor prefix because I am using Google Chrome. If you are using a different browser, use the vendor prefix for that browser.

In Chapter 15, you use a tool that automatically adds the other vendor prefixes to the CSS, saving you from having to write five different vendor-prefixed properties now.

box-shadow

Initial value: `none` | Inherited: No | Applies to: All | CSS3

Unprefixed browser support: IE 9+, Firefox 4+, Chrome 10+, Opera 10.5+, Safari 5.1+

Prefixed browser support: Firefox 3,5+, Chrome 1+, Safari 3+

The `box-shadow` property applies a drop shadow to an element.

You can add a subtle drop shadow (see Figure 5-10) to the main content area:

1. In styles.css, find the `#main` rule set and add the following below the other declarations:

```
box-shadow: 0 3px 8px 0 #ccc;
```

2. Save styles.css.

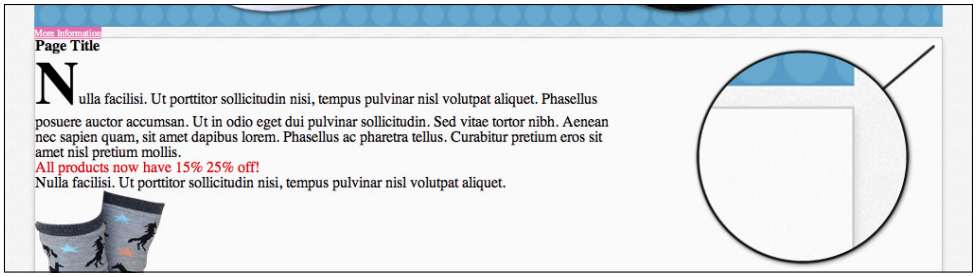


FIGURE 5-10 The main content area with a subtle box shadow.

The first four values of the `box-shadow` property can be any unit of length and modify the shadow as follows:

- The first value of a drop shadow represents its horizontal offset: how far the shadow is from the right of the box, a negative value to its left. The shadow you added doesn't have an offset.
- The second value is the vertical offset. A positive number goes below the element and negative goes above. The shadow you added is offset `3px` vertically from the main content element.
- The third value is the blur of the shadow. A value of `0` here would make the edges of the shadow sharp, whereas `8px` gives it a mild blur.
- The fourth value is the spread distance, which determines how far the shadow should spread on all sides of the element.

Following on from the shadow values is an optional color. By default, a shadow is black, but the shadow you added has a light gray color to match the border.

You also are able to make box shadows inset so the shadow appears inside the element:

1. In styles.css, find the rule set for `input[type="text"], input[type="email"]` and add the following:

```
box-shadow: -3px 3px 2px 0 #ccc inset;
```

2. Save styles.css.

Adding this declaration gives the input fields of type `text` and `email` an inset shadow. An inset shadow is often a good visual indication of an input field.

It's also possible to add more than one shadow to an element (see Figure 5-11), by separating shadows with a comma:

```
box-shadow: 0 3px 8px 0 #ccc, 0 3px 8px 0 red inset;
```

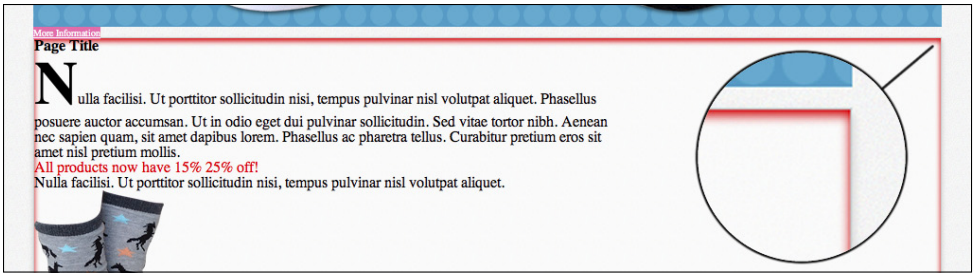


FIGURE 5-11 The main content area with a subtle gray box shadow and red inset box shadow.

Code Challenge: Add More Border and Box Shadow Styles

In styles.css, do the following:

1. Add a declaration of `border: rgba(0,0,0,0.1) solid 5px;` to the `#newsletter` rule set.
2. Add a declaration of `border-radius: 8px;` and `border: 2px solid white;` to the `.button` rule set.
3. Add a new rule set for `#header nav > ul` with the declarations `border-radius: 20px;` and `border: #ccc solid 1px;`.
4. Add a new rule set for `.showcase` with a declaration of `box-shadow: 0 -3px 8px 0 #ccc;`.

Project files update (ch05-01): If you haven't followed the previous instructions and are comfortable working from here onward or would like to reference the project files up to this point, you can download them from `www.wiley.com/go/treehouse/css3foundations`.

Backgrounds

In the preceding chapter, you applied a background color to a couple of elements, as well as a background image to the body of the page. CSS has many other properties to allow you to style a background.

background-color

Initial value: `transparent` | Inherited: No | Applies to: All | CSS2.1

Browser support: IE 4+, Firefox 1+, Chrome 1+, Opera 3.5+, Safari 1+

As you saw in the preceding chapters, you specify a background color using the `background-color` property and can give it a variety of values, such as RGB, RGBA, HSL, HSLA, and color keywords:

```
body {
    background-color: #f5f5f5;
}
```

By default, `background-color` is `transparent`, meaning it is see-through, allowing elements below it or containing it to show through.

background-image

Initial value: `none` | Inherited: No | Applies to: All | CSS2.1

Browser support: IE 4+, Firefox 1+, Chrome 1+, Opera 3.5+, Safari 1+

`background-image` accepts a `url()` function or `none` as its value. As you saw in the `border-image-source` explanation, the `url()` function takes a URI as its argument.

A URI (also referred to as a source, link, or path) within the `url()` function can optionally be wrapped with single or double quotation marks. Previously, you added this style:

```
background-image: url("../images/bg-body.jpg");
```

This style is just as acceptable written without the quotation marks and with optional whitespace inside the brackets:

```
background-image: url( ../images/bg-body.jpg );
```

The choice is yours.

You may be wondering what the two dots mean in the URI. The two types of URI you can use are relative and absolute. This type is known as relative.

An absolute URI is complete and in its entirety. For example, if you want to view the image `bg-body.jpg` directly, in your browser, you can navigate to `http://www.coolshoesandsocks.com/images/bg-body.jpg`. To make the URI of the background image apply to the body, you use the following declaration:

```
background-image: url("http://www.coolshoesandsocks.com/images/
  bg-body.jpg");
```

Avoid Hot Linking

Linking from your own web resource to another that you don't own via an absolute URI is known as *hot linking*. Say I like an image I've seen on someone else's website and I make it the background image of my web page via an absolute link. Not only may that person be upset with me because I don't have her permission to do that, but I'm also using the bandwidth of her server. It is common that web hosting is priced based on the amount of bandwidth a website uses, so potentially, by hot linking that image, I could be costing the person money!

If the owner of the image is kind enough to give you permission to use an image, you should upload that image to your own server and link to it from there, ideally via a relative URI.

Absolute URIs tend to be used when you're linking to an external resource—on another domain, for example. Although they can be used internally too, it is good practice to use relative URIs instead as they can make a web page quicker to load.

A relative URI is a path (source or link) that is relative to the document it is within. When you added the relative URI `../images/bg-body.jpg` in styles.css, you told the browser to go up one directory (denoted by `../`) into the `images` directory and then use the image bg-body.jpg from there.

Following are a few rules when using relative URIs:

- Links to resources in the same directory should consist only of the filename and file extension: `bg-body.jpg`
- Links to subdirectories consist of the subdirectories' names (without a preceding slash), followed by the filename and extension: `images/bg-body.jpg`
- To go up a directory, use two full stops (dots or periods) followed by a slash, then the subdirectory (if required), and then the filename and file extension: `../images/bg-body.jpg`

You may have noticed I refer to a ***URL*** and a ***URI.*** What's the difference? You might be sorry you asked, but I hope this explanation clears up matters! The name ***URL*** means Uniform Resource Locator and, as the CSS specification states, the `url()` function requires a URI (Uniform Resource Identifier) as its argument. A URL and URN (Uniform Resource Name) are subclasses of a URI (they can be both referred to as URIs). However, a URN has no use in CSS, so although it's technically true that the `url()` function does take a URI as its argument, only a URL is of use (hence the function being called `url()` rather than `uri()`). This distinction is not necessarily something you need to be concerned about, but you may have been curious!

background-repeat

Initial value: `repeat` | Inherited: No | Applies to: All | CSS2.1
Browser support: IE 4+, Firefox 1+, Chrome 1+, Opera 3.5+, Safari 1+

When you applied the background image to the body in Chapter 3, as you saw, that image was only 40px × 40px, but the browser repeated the image to make it fill the whole body. This is the default behavior of a background image, but by using the `background-repeat` property, you can change that.

Add a background image to the newsletter form:

1. In styles.css, find the `#newsletter` rule set and add the following:

```
background-image: url("../images/icon-newsletter.png");
```

2. Save styles.css.

Figure 5-12 shows that the little image of the envelope has repeated across the entire newsletter box. Looks a bit over the top, doesn't it?

FIGURE 5-12 The background image repeating across the newsletter box.

Because you have not yet specified a `background-repeat` declaration, the initial value of `repeat` is applied. So, what other values can you use?

- `background-repeat: repeat-x`—causes an image to repeat only horizontally.
- `background-repeat: repeat-y`—causes an image to repeat only vertically.

Although these values are useful, this image would look better if it wasn't repeating.

1. In styles.css, below the `background-image` declaration you added to the `#newsletter` rule set, add the following:

```
background-repeat: no-repeat;
```

2. Save styles.css.

Figure 5-13 shows the result of applying `no-repeat`.

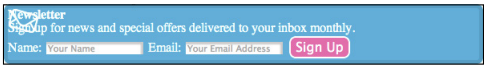


FIGURE 5-13 The background image of the newsletter box, now with no-repeat specified.

Now, the background image doesn't repeat, and you see only one image applied to the newsletter box.

background-position

Initial value: `0% 0%` | Inherited: No | Applies to: All | CSS2.1

Browser support: IE 4+, Firefox 1+, Chrome 1+, Opera 3.5+, Safari 1+

The background image is currently in its initial position of `0% 0%`. The first `0%` is the horizontal position, relative to the left of the parent element, and the second `0%` is the vertical position, relative to the top of the parent element.

When specifying a `background-position` value, you can use all the length unit types as described in Chapter 4, percentages, pixels, ems, and so on. You can also use the keywords `left`, `right`, `top`, `bottom`, and `center`.

Because the background image is currently sitting underneath the text, making the text hard to read, change its position:

1. In styles.css, below the `background-repeat` declaration you added to the `#newsletter` rule set, add the following:

```
background-position: right top;
```

2. Save styles.css.

As shown in Figure 5-14, the background image is now positioned to the far right and at the top of the newsletter box. This declaration is the equivalent of `background-position: 100% 0%`. It's just a little nicer to read and write a declaration like this when it's written using keywords!

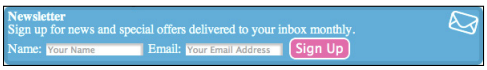


FIGURE 5-14 The background image positioned to the right and at the top of the newsletter box.

If only one of the `background-position` values is specified, the missing value is assumed to be `center`, so if you use the declaration `background-position: right`, the background image is positioned to the right of the newsletter box but vertically centered.

The position of the background image might not make a huge amount of sense at the moment, but when you give the page more structure in Chapters 6–9, it will!

In preparation, give the background image a more accurate position that will better suit the newsletter box when it has more structure:

1. In styles.css, change the `background-position` declaration to

   ```
   background-position: 91% 2%;
   ```

2. Save styles.css.

background-attachment

Initial value: `scroll` | Inherited: No | Applies to: All | CSS2.1

Browser support: IE 4+, Firefox 1+, Chrome 1+, Opera 3.5+, Safari 1+

By default, when you scroll the viewport or an element that can scroll (you learn how to make an individual element scrollable using the `overflow` property in Chapter 9), a background image moves with the scrolling of the viewport or element.

```
body {
    background-color: #f5f5f5;
    background-image: url("../images/bg-body.jpg");
    background-attachment: fixed;
    font-size: 62.5%;
}
```

For example, giving the background image applied to the `<body>` element a declaration of `background-attachment: fixed` causes the background image to remain in place when you scroll the page (as opposed to the background image moving with the scrolling).

If you want to try this example yourself, depending on your screen size, you might need to give the `body` selector a temporary declaration of `height: 2000px` to make the page tall enough to require scrolling.

Applying Multiple Background Images

Browser support: IE 9+, Firefox 3.6+, Chrome 1+, Opera 11+, Safari 1.3+

CSS Level 3 introduces multiple background images per element. Multiple background images are supported in all modern browsers, but not in Internet Explorer versions 6, 7, and 8. Because of this, it's best to use multiple background images only if they're not important, such as those used for decorative purposes.

Add another background image to the newsletter box:

1. In styles.css, change the `background-image`, `background-repeat`, and `background-position` declarations in the `#newsletter` rule set to

```
background-image: url("../images/icon-newsletter.png"),
url("../images/bg-newsletter.png");
background-repeat: no-repeat, repeat;
background-position: 91% 2%, 0;
```

2. Save styles.css.

Here, you specify two `url()` functions for the `background-image` and separate them with commas, which now apply two background images to the newsletter box. Likewise, you give the `background-repeat` and `background-position` declarations secondary values, again, by separating them with commas.

So, the secondary background image repeats (filling the whole newsletter box) and is given a position of `0` (it's set to repeat so the position doesn't matter).

Background Gradients

Prior to CSS Level 3, a gradient (an image that blends from one color to another) had to be made in a graphics application such as Adobe Photoshop or Fireworks, but now you can do it using nothing but CSS! Technically, a gradient is an image that the browser generates so it can be applied to—and in place of—any element that accepts the `url()` function.

Just like background images, provided that you specify a basic background color to fall back to, you can use gradients safely.

Gradients come in two flavors: linear and radial. A *linear* gradient is the blending of colors from one point to another, whereas a *radial* gradient starts from a center point and blends outward.

Unfortunately, gradients are in a state of disarray, with varying support and syntaxes across browsers.

Internet Explorer versions 6, 7, 8, and 9 have no support for official gradients. That said, they do actually support a different type of gradient using an unofficial `filter` property that Microsoft added before they concerned themselves with following the standardized specifications.

For more information about the unofficial `filter` property, visit `msdn.microsoft.com/en-us/library/ie/ms530752(v=vs.85).aspx`.

Linear gradients in other browsers are supported either using the official syntax or an outdated one. You'll learn both syntaxes to be able to achieve the most supported and future-proof approach to linear gradients.

Radial gradients have an even wider level of varying support. Because of this, *CSS3 Foundations* doesn't cover radials. Should you wish to learn more, I'd recommend reading the CSS3 Image Values and Replaced Content module which contains the Candidate Recommendation explanation of radial gradients, found at `www.w3.org/TR/css3-images/#radial-gradients`. The Candidate Recommendation will almost certainly be the finalized syntax, which is already supported unprefixed in Internet Explorer 10+ and Firefox 16+.

If you'd rather just create a radial gradient effect and not worry about how the code works, you can use a code generator, such as the one found at `www.colorzilla.com/gradient-editor/` (which can also generate linear gradients).

Linear Gradients

Unprefixed browser support: IE 10+, Firefox 16+, Opera 12+

Prefixed browser support: Firefox 3.6+, Chrome 10+, Opera 11+, Safari 5.1+

Start by applying a linear gradient to the buttons on the page before taking a closer look at how to write them:

1. In styles.css, find the rule set for `.showcase .button:hover` and above it, add the following rule set:

```
.showcase .button {
    padding: 20px;
    background-image:
-webkit-linear-gradient(top, #FB3876 0%, #d4326d 100%);
    background-image:
-moz-linear-gradient(to bottom, #FB3876 0%, #d4326d 100%);
    background-image:
-o-linear-gradient(to bottom, #FB3876 0%, #d4326d 100%);
    background-image:
linear-gradient(to bottom, #FB3876 0%, #d4326d 100%);
}
```

2. Save styles.css.

When using gradients, the vendor prefix is prefixed to the property value rather than the property.

First, you add `20px` of padding to the showcase button, which places space between the border of the box and the content. You learn about this property in Chapter 7. The purpose of this padding is just to make the area a little bigger so you can see that fancy gradient better!

I mentioned previously you wouldn't have to write all of the vendor-prefixed properties yourself but just to demonstrate the safest way to use gradients, you've done so in this rule set. The `-webkit-` property uses the outdated syntax for gradients, which you'll look at in a moment. The `-moz-` property applies to Firefox versions 3.6 to 15. The `-o-` property applies to Opera 11 and the official, unprefixed property applies to Internet Explorer 10+, Opera 12+, and Firefox 16+, it will eventually apply to future versions of WebKit browsers too. Note that because Internet Explorer 9 doesn't support gradients at all and Internet Explorer 10+ only supports the official unprefixed property, a prefixed `-ms-` property is not required. Should you include it, though, it won't affect anything.

So, now you know how to achieve gradients in the widest amount of browsers, what does the `linear-gradient()` function do?

As evident in Figure 5-15, `linear-gradient()` specifies the gradient line (the direction of the gradient) and two color stops, which consist of a color and the position of that color.

FIGURE 5-15 The linear gradient applied to the showcase button, starting with a hot pink, going into a darker pink at the bottom.

Gradient Line

The default argument for the gradient line is the keyword `to bottom`, so technically in this example, `to bottom` can be omitted and the same effect would be achieved. The keyword `to bottom` starts the gradient from the top and ends at the bottom. Other gradient line keywords are `to top` (going from bottom to top), `to right` (going from left to right), and `to left` (going from right to left).

Note that WebKit doesn't understand these keywords and instead opts for `top`, `bottom`, `left`, and `right` (the outdated syntax), which denote the starting point of the gradient and are the equivalents of `to bottom`, `to top`, `to right`, and `to left`.

If you would like something other than a straight gradient line, you can specify an angle in degrees. The preceding keywords each represent an angle; `to top` is `0deg`, `to right` is `90deg`, `to bottom` is `180deg`, and `to left` is `270deg`. You can also combine these

keywords to create a gradient line that goes from corner to corner; `to bottom right` goes from the top left to the bottom right, for example. You'll be pleased to know that WebKit, does understand gradient lines when expressed as degrees in the same way that Firefox does, so to make things more consistent, you could rely on degrees instead of keywords.

Color Stops

A color stop is composed of a color value (using any of the color property values described in Chapter 4), followed by an optional stop position, which can be defined by a value type such as percentages or pixels. You can specify as many color stops as you need, separating each with a comma to create a multicolored gradient. If you specify a color value but not an optional stop position, the browser assumes the stop position should be placed exactly in between the other stop positions. So, `linear-gradient(#fff, #000)` creates a gradient going from top to bottom with the top being white, fading into black as it reaches the bottom.

Figure 5-16 shows how `linear-gradient(#fff, #ff0000, #000)` creates a gradient going from top to bottom with the top being white, the center being red, and the bottom being black.

FIGURE 5-16 The linear gradient applied to the showcase button, starting with white, changing into red at the center, and then changing into black at the bottom.

When you are adding a gradient to a web page, sometimes it's better to look at a visual representation of the gradient you're trying to create, to get it exactly as you want it. Many free tools are available that make working with gradients easier. My personal recommendation is ColorZilla's gradient editor, which you can find at `www.colorzilla.com/gradient-editor/`.

background-clip

Initial value: `border-box` | Inherited: No | Applies to: All | CSS3

Browser support: IE 9+, Firefox 4+, Chrome 1+, Opera 10.5+, Safari 3+

`background-clip` determines the "painting area" of an element, the area of the element to which a background should be applied. By default, the `background-clip` value is

`border-box`, which makes the painting area start from the edges of an element, where the border starts.

Looking at Figure 5-17, you can see the background image of the newsletter box reaches the edges. Earlier in the "Add More Border and Box Shadow Styles" code challenge, you gave this element a black border with a transparency of `0.1`, but note that the border actually looks like a dark blue color. The reason is that the background image appears behind the border and shows through the partially transparent border.

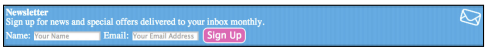


FIGURE 5-17 The newsletter box, with a background-clip of border-box.

If you want the painting area to start inside the border, you can apply the following to the `#newsletter` rule set:

```
background-clip: padding-box;
```

The "padding" of an element starts inside an element's border. In Figure 5-18, you can now see the border is black with a `0.1` transparency (making it gray because the white of the main content area shows through).

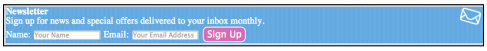


FIGURE 5-18 The newsletter box, with a background-clip of padding-box.

Padding, which is covered in Chapter 7, adds space between the edge of an element and the elements or contents it contains.

Add padding to the newsletter box:

1. In styles.css, add the following declaration to the `#newsletter` rule set:

   ```
   padding: 6%;
   ```

2. Save styles.css.

There is now a significant space between the edges of the newsletter box and its content. What if you want the painting area to be only the background of the content? In that case, you can use the final `background-clip` value, `content-box`, as shown in Figure 5-19.

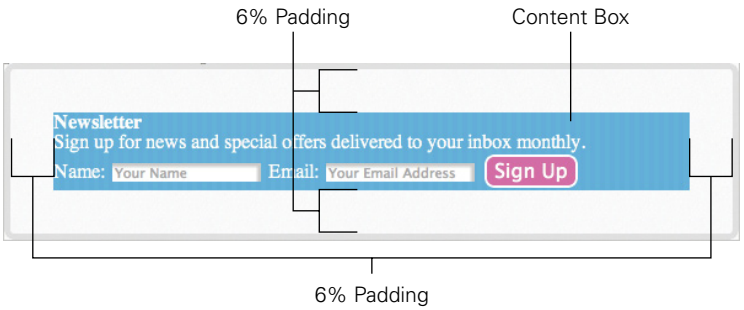


FIGURE 5-19 The newsletter box with a background-clip of content-box and 6% padding.

Note that if you want to use multiple background images, as with the other background properties, you can separate multiple `background-clip` properties with a comma.

background-origin

Initial value: `padding-box` | Inherited: No | Applies to: All | CSS3

Browser support: IE 9+, Firefox 4+, Chrome 1+, Opera 10.5+, Safari 3+

In Figure 5-19, you may have noticed the newsletter icon disappeared. With a declaration of `background-clip: content-box`, the icon was positioned outside the content area and got clipped. `background-origin` allows you to change the background positioning area.

In Figure 5-20, I changed the `background-clip` back to `border-box` and gave it a `background-position` of `92% 0`. As you can see, the newsletter icon still sits 92% to the right of the box but starts perfectly where the padding box does—the default value for `background-origin`.

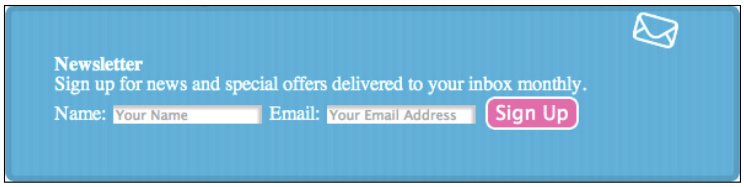


FIGURE 5-20 The newsletter icon in relation to the newsletter box with a background-clip of border-box and a background-position of 92% 0.

When you change the `background-origin` to `border-box`, the newsletter icon touches the top of the border, as shown in Figure 5-21.

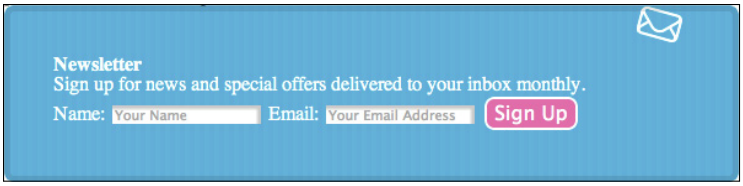


FIGURE 5-21 The newsletter icon in relation to the newsletter box with a background-origin of border-box applied to it.

Can you guess what happens when `background-origin: content-box` is applied? The position of the newsletter icon starts within the content area, as shown in Figure 5-22.

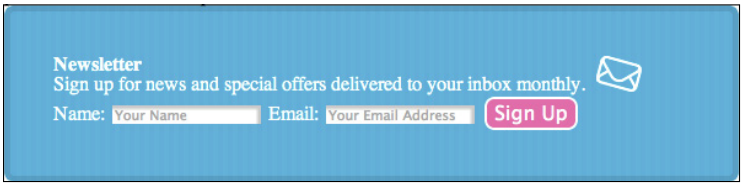


FIGURE 5-22 The newsletter icon in relation to the newsletter box with a background-origin of content-box applied to it.

background-size

Initial value: `auto` | Inherited: No | Applies to: All | CSS3

Unprefixed browser support: IE 9+
Prefixed browser support: Firefox 3.6+, Chrome 1+, Opera 9.5+, Safari 3+

The `background-size` property was introduced in CSS Level 3, which allows you to change the size of a background image. As with `background-position`, you can use all value types, percentages, pixels, ems, and so on. Percentages are relative to the background positioning area, a concept explained in the descriptions for the previous properties.

```
background-size: 50% 100%;
```

In this example, two values are specified: one for the width, one for the height. A background image with this declaration is 50% the width of the element to which it is applied and 100% its height.

If you use the declaration `background-size: 50%`, the second value is assumed to be `auto`. When `auto` is used, the browser calculates the height based on the specified width to maintain the image's aspect ratio.

You can also use two keyword values: `cover` and `contain`.

`background-size: contain` scales a background image to fit inside the background positioning area, making it as large as possible while respecting the image's aspect ratio.

`background-size: cover` scales a background image to the smallest size so that both the image's height and width cover the background positioning area while respecting the image's aspect ratio.

background (Shorthand)

Browser support: IE 4+, Firefox 1+, Chrome 1+, Opera 3.5+, Safari 1+

Much like the border properties, background properties also have a shorthand property, simply called `background`.

Rather than have up to eight declarations to style a background, you can use the background shorthand with the following syntax:

```
background: background-image background-position background-size
repeat-style attachment background-origin background-clip
background-color
```

When you use multiple background images, you should specify `background-color` only on the last layer of a background.

1. In styles.css, find the `#newsletter` rule set and remove the following declarations:

```
background-color: #00ACDF;
background-image: url("../images/icon-newsletter.png"),
url("../images/bg-newsletter.png");
background-repeat: no-repeat, repeat;
background-position: 91% 2%, 0;
```

2. Add a shorthand declaration:

```
background: url("../images/icon-newsletter.png") no-repeat
  91% 2%,
url("../images/bg-newsletter.png") repeat 0 #00ACDF;
```

3. Save styles.css.

The shorthand declaration achieves the exact same styles as was originally achieved. Because the newsletter box had two background images applied to it, a comma separates the two shorthand values. Finally, although you can have multiple background images, you can have only one background color, so you add `background-color` at the end of the last background layer.

CSS Image Replacement

Sometimes, you may want to show an image in place of text. Take the Cool Shoes & Socks link in the footer of the page for example. At the moment, it's just text, but really, it would look nicer if it was a small version of the logo.

Add a background image to the footer link:

1. In styles.css, add the following rule set:

```
.small-logo a {
    background: white url("../images/logo-small.png") no-repeat
  center;
    display: block;
    min-height: 11px;
    width: 162px;
    padding: 5px;
}
```

2. Save styles.css.

This rule set adds a small version of the Cool Shoes & Socks logo as a background image, makes the `<a>` element a block to fill a space of 11px × 162px and gives it a small amount of padding (`padding` is covered in Chapter 7 and `display` in Chapter 9).

The problem with doing this is that the text sits on top of the background image, which doesn't look right, as shown in Figure 5-23.

FIGURE 5-23 The footer link text conflicting with the background image.

You could just remove the text from the HTML and put an `<img>` element in its place. However, as the footer logo is more of a style than content, it's a best practice to make it a background image instead.

To work around the clashing of text and background image, you can use a technique commonly referred to as Image Replacement.

3. In the newly added `.small-logo a` rule set, add the following declarations:

```
font: 0/0 a;
text-shadow: none;
color: transparent;
```

4. Save styles.css.

You will learn about `text-shadow` and `font` properties in Chapter 11. For now, know that the `font` property makes the font size and line height zero. Any shadow that may be applied to the text is removed and the color is made transparent. These three properties together hide text in all browsers (see Figure 5-24) so now, although the text is still in the HTML, it can't be seen, allowing just the background image to show. If CSS is disabled in the browser, the text will be shown in place of the background image. Ideal!

FIGURE 5-24 The footer link with hidden text, allowing the background image to show through.

Code Challenge: Add More Background Properties

In styles.css, do the following:

1. Add a new rule set for `#header nav > ul > li` with the declaration `background: white;`.
2. In the `#footer` rule set, change the `background-image` and `background-color` declarations to use the `background` shorthand property.
3. In the `.showcase` rule set, add the declaration `background: #f5f5f5;`.

Project files update (ch05-02): If you haven't followed the previous instructions and are comfortable working from here onward or would like to reference the project files up to this point, you can download them from `www.wiley.com/go/treehouse/css3foundations`.

opacity

Initial value: `1` | Inherited: No | Applies to: All | CSS3

Browser support: IE 9+, Firefox 1+, Chrome 1+, Opera 9+, Safari 1.2+

opacity, introduced in CSS Level 3, gives elements a level of transparency.

```
opacity: 0.5;
```

The value for opacity is a number, ranging between 0 and 1, to one decimal place. In the preceding example, the opacity of an element is 50%.

Unlike when you use a background color with an alpha value (which is a color with a transparency), opacity applies to the element and *all* its children. So say you use the following:

```
body {
    opacity: 0.7;
}
```

Here, the entire web page has a 70% transparency, which isn't very sensible!

The opacity property works in all modern browsers except for Internet Explorer versions 6, 7, and 8. However, as with gradients, these older versions of Internet Explorer support a filter property that achieves the same effect. For more information about the unofficial filter property, please see msdn.microsoft.com/en-us/library/ie/ms530752(v=vs.85).aspx.

In Chapter 15, you learn how to use a tool to automatically write vendor prefixes you haven't included during the creation of the Cool Shoes & Socks website. It also adds the alternative filter properties to make any instance of opacity work in older versions of Internet Explorer.

visibility

Initial value: visible | Inherited: Yes | Applies to: All | CSS2.1

Browser support: IE 4+, Firefox 1+, Chrome 1+, Opera 4+, Safari 1+

Using opacity: 0 to hide an element works visually, but it's not necessarily the best way to hide an element. The visibility property can be given three values: visible, hidden and collapse. All elements, by default, are visible. visibility: hidden hides an element, but how does that differ from opacity: 0, which also hides elements?

Assume for a moment that you want to hide the "About" link on the Cool Shoes & Socks web page. When hiding the link, you use opacity, like so:

```
nav ul li:nth-child(2){
    opacity: 0;
}
```

In Figure 5-25, the "About" link is no longer visible on the page, but you can still see an empty space where that link should be. If you hover over that empty space, you can still click the "About" link, even though the element has an opacity of `0`.

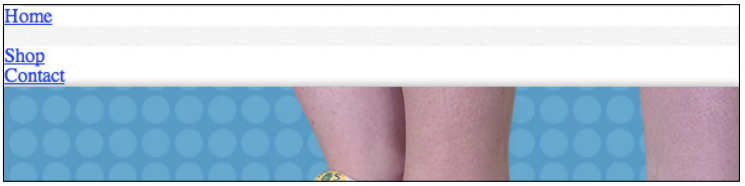


FIGURE 5-25 The blank area where the "About" link is positioned.

When `visibility: hidden` is applied:

```
nav ul li:nth-child(2){
    visibility: hidden;
}
```

The "About" link is still rendered (you can see the empty space, as in Figure 5-25); however, this time, you can't click it! So both methods of hiding an element still render that element, but `visibility: hidden` has no events—such as clicking—whereas `opacity: 0` does.

What if you want to have an element hidden and not even rendered? An element that exists in the HTML but doesn't show an empty space when it's hidden?

Using `display: none` achieves that, as shown in Figure 5-26:

```
nav ul li:nth-child(2){
    display: none;
}
```

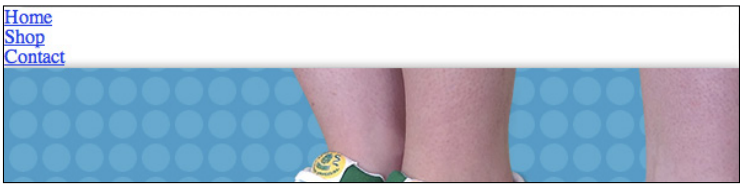


FIGURE 5-26 The "About" link is no longer rendered on the page when it is given the declaration display: none.

Because `display` relates to structure, it is covered in Chapter 9.

cursor

Initial value: `auto` | Inherited: Yes | Applies to: All | CSS2.1

Browser support: IE 4+, Firefox 1+, Chrome 1+, Opera 7+, Safari 1.2+

The `cursor` property allows you to change the type of cursor displayed when hovering over an element. The initial value of `auto` means the user agent stylesheet determines the cursor.

Changing the cursor is a great way to show users how they can interact with a particular element.

1. In styles.css, find the rule set for `input[type="submit"][class="button"]` and add the following declaration:

```
    cursor: pointer;
}
```

2. Save styles.css.

Now, when you hover over the Sign Up button in the newsletter box, the cursor changes to a pointer, letting the user know the button can be clicked.

The look of the cursors differs based on the operating system they are being viewed on. The following values can be used: `crosshair`, `default`, `pointer`, `move`, `e-resize`, `ne-resize`, `nw-resize`, `n-resize`, `se-resize`, `sw-resize`, `s-resize`, `w-resize`, `text`, `wait`, `help`, and `progress`.

If these cursors don't take your fancy, you can also use the `url()` function (as you did with the `background-image` property) to use a custom cursor.

outline (Shorthand)

Browser support: IE 8+, Firefox 1.5+, Chrome 1+, Opera 7+, Safari 1.2+

You applied styles in Chapter 3 to give every element that has focus an outline. Giving elements an outline is important for accessibility. Assume a visitor to your web page can't use a mouse (he may have a disability that prevents him from doing so or, quite simply, the batteries in his mouse ran out). To browse your web page, that user needs to use his keyboard. To move between elements of a web page using a keyboard, you can use the Tab key. With each press of the Tab key, starting from the top left of the page, elements are given an outline style, which lets the user know where on the page he is.

Because this is an important property that aids navigation, many browsers have this style built in to their default styles via the user agent stylesheet. If you want to change those default styles, you can use the `outline-color`, `outline-style`, and `outline-width` properties.

Do those property names ring a bell? That's right; they work in the exact same way as `border-color`, `border-style`, and `border-width`. You can use the shorthand `outline` in the same way as the shorthand `border` too!

Because you already added outline styles to the `a:focus` pseudo-class in Chapter 3, you don't need to add any more. However, you can change those properties into one shorthand property:

1. In styles.css, find the pseudo-class `a:focus` and remove the following declarations:

```
outline-color: black;
outline-style: dotted;
outline-width: 1px;
```

2. Add the shorthand declaration:

```
outline: black dotted 1px;
```

3. Save styles.css.

content

Initial value: `auto` | Inherited: No | Applies to: `:before` and `:after` only | CSS3
Browser support: IE 8+, Firefox 2+, Chrome 4+, Opera 9+, Safari 3.1+

In Chapter 3 you used the `content` property to give quotation marks to the customer testimonials. `content` can be applied only to `:before` and `:after` pseudo-elements.

Now try another example:

1. In styles.css, below the three `blockquote` related rule sets, add the following:

```
blockquote cite:before {
    content: "- ";
}
```

2. Save styles.css.

This adds a hyphen before the customer's name, showing that a testimonial belongs to her. This is just a visual style, and in its absence, the content of the page isn't affected. These sorts of styles are the purpose of `content`, which I hope answers the question raised in Chapter 3: What place does content have in CSS? CSS, after all, is supposed to separate style from content and structure, so why would you want to include content in your CSS?

Characters such as quotation marks and hyphens are arguably presentational styles as opposed to content. You may have a website containing 50 quotation marks, and one day decide you want to change the style of them. If those quotation marks were included in the HTML page as physical characters, you would have to go into each page and change them individually. If you generate those characters via the `content` property, changing their style is a simple case of changing one property value.

Summary

In this chapter, you learned to use many of CSS's visual styles. With the inclusion of CSS Level 3's most recent and exciting features, border and backgrounds alone are enough to give a page a unique look.

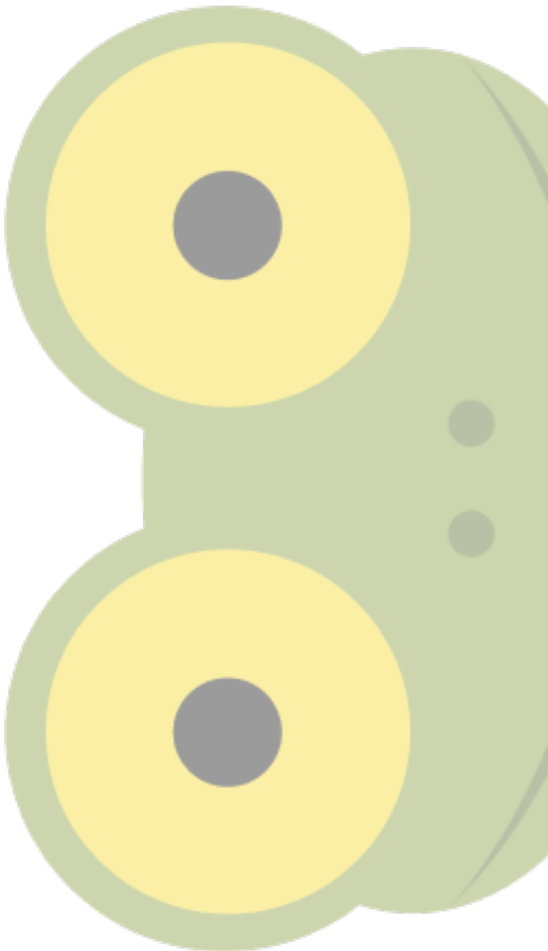
Properties such as `opacity`, `visibility`, `cursor`, and `outline` allow you to give a page certain styles when interacted with, helping a user navigate through your web site.

You've also learned how to implement experimental properties safely using vendor prefixes. As you make your way through *CSS3 Foundations*, you'll rely on vendor prefixes again, to be used with the latest CSS3 properties.

Now that you've added styles to make elements stand out and give the page its own personality, you're ready to move on to giving the page structure.

part 3

Building a Solid and Adaptable Page Structure



chapter six

Creating a Basic Page Structure

IN THE PRECEDING chapters, you added presentational styles to the Cool Shoes & Socks web page, but as yet, the page doesn't have much structure other than the basic top-to-bottom layout of elements. In this chapter, you learn about fixed and fluid page layouts, plus a hybrid of the two that will help to make Cool Shoes & Socks responsive, so a layout can respond to the device on which it is being viewed.

Structure Types

Structure types aren't necessarily something described in the CSS specification. They are a method of combining multiple CSS properties to create page structures that suit particular tasks.

In the early days of web pages, structure types tended to be fluid, meaning a web page stretched and resized as the browser did to always fill the viewport. This is where the "fluid" name comes from: It fills every area of the web page regardless of its size, just like a fluid fills its container.

As the web moved forward and design became more elaborate, web designers wanted more control over web pages. If a fluid page stretched depending on the browser window size, designers would have a difficult job designing a web page because they couldn't be sure how that page would look to the users. The majority of the web started making use of the fixed-width layout, which is still the most-used structure type today.

Although fixed-width layouts are the most used today, with new technologies such as CSS3 and HTML5, and in particular the CSS3 media queries, the web is moving toward responsive layouts that start with a fluid structure but then change that layout based on the size of a device—something that couldn't be done when fluid layouts were first in use.

In this chapter, you look at fixed and fluid layouts but actually use a hybrid of the two, and then in Chapter 16, you add media queries to make the layout responsive!

Project files update (ch06-00): If you haven't followed the previous instructions and are comfortable working from here onward or would like to reference the project files up to this point, you can download them from `www.wiley.com/go/treehouse/css3foundations`.

Fluid

A web page, by default, is fluid. The reason is that the initial value for an element's width is `auto`, which tells the browser to work out the width itself, based on the unused space around that element. Without any CSS applied to a page—other than the user agent stylesheet—`width: auto` is the equivalent of `width: 100%`.

Before you gave the `<body>` element a `max-width` in the previous chapter, the Cool Shoes & Socks web page was completely fluid.

Fluid-width websites are often used when a page has a lot of information and the designer wants to make the most of the space available.

Fixed

Fixed-width layouts are made by giving the element that contains the rest of the page's elements—in the case of Cool Shoes & Socks, the `<body>` element—a `width` property. The `width` property is usually a pixel value such as 960px, which makes the `<body>` element always 960px, regardless of the size of the browser.

Why a width of 960px? When the web was accessed by fewer types of devices (mainly the only consideration was differing desktop display sizes), it was wise to use a width of 960px because the most-used screen resolution was 1,024 × 768, meaning that 960px would fit nicely into a 1,024px display (as shown in Figure 6-1), as well as anything bigger than that. If a web page is 1,260px wide and is viewed on a screen 1,024px wide, the browser has to show horizontal scroll bars to allow the user to move across to see the full page. Likewise, if viewing a web page that is 960px wide on a screen or in a browser window less than 960px, horizontal scroll bars appear. Although you can't control the size of the browser window, you can make a web page a size that fits in the majority of screen sizes. The choice of the browser window size and whether a horizontal scroll bar appears is then left to the user.

So, what's wrong with horizontal scroll bars? Because users expect a web page to be navigated vertically, the addition of a horizontal scroll bar can make navigation around a web

page feel awkward. For the sake of good usability, you should avoid horizontal scroll bars where possible.

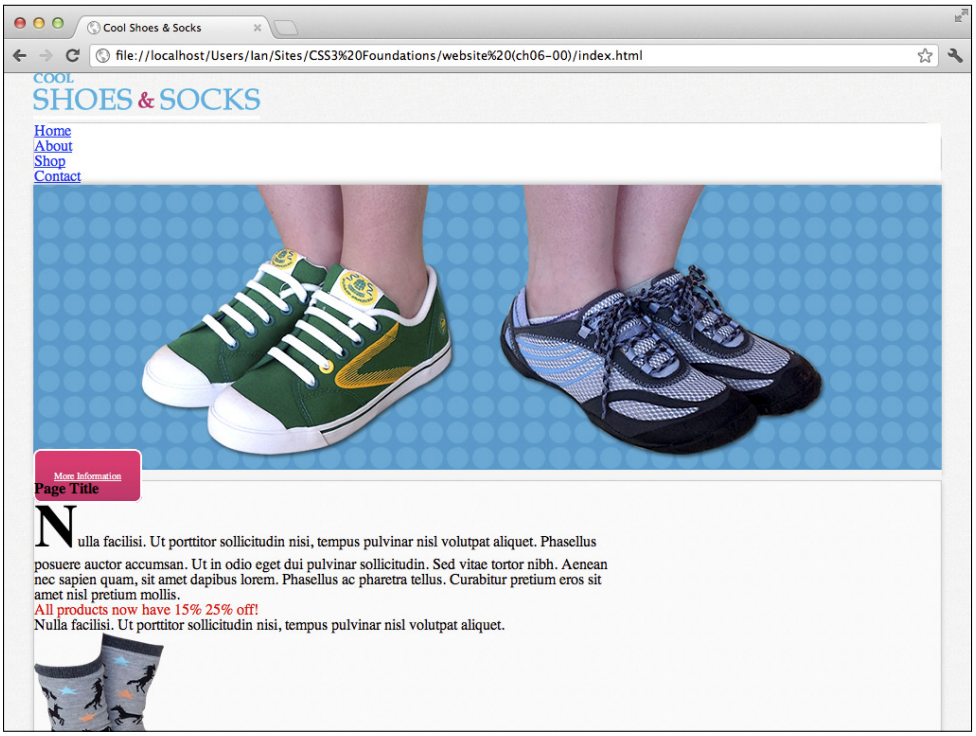


FIGURE 6-1 The Cool Shoes & Socks web page with a fixed width of 960px, viewed on a 1,024 × 768 resolution.

Further to this, 960 is a nice round figure that can be divided into round values when using columns, making a web page of 960px easier to lay out and more pleasing to the eye.

A page being centered using the following declaration on the element that contains all other elements is another common characteristic of fixed-width layouts:

```
margin: 0 auto;
```

If a centered, fixed-width layout is viewed on a screen with a width greater than the page width, the page is still centered. This is another technique often used in fixed-width layouts.

That's fixed-width layouts in a nutshell. Note that when referring to fixed, fluid, and hybrid layouts, I am referring only to the width of the layout. Although you can have lots of control over the width of a layout, because content can be resized—either by the device viewing that page or the user—the height of a layout is always an unknown, and as a best practice, you should avoid declaring specific heights on any element of a page.

Hybrid Layout for Responsive Design

In the previous chapter, you made the page a hybrid layout, by adding the following rule set to styles.css:

```
body {
    ...
    margin: 0 auto;
    max-width: 960px;
}
```

Let's start with the obvious: `max-width` is the maximum width an element should be. Here, you specified the `<body>` element should get no bigger than `960px`. Because a `width` isn't specified, the default width of the `<body>` is `100%` unless your browser is narrower than `960px`, in which case the `<body>` element is the same width as your browser.

This layout is a hybrid. Although you fixed the maximum width of the page to 960px, it is still fluid below those 960 pixels.

Technically, a layout doesn't have to be a hybrid (a mixture of fixed and fluid) to be responsive; it just needs to be fluid. The reason you make Cool Shoes & Socks a hybrid—fluid below 960px and fixed above—is to make sure the site scales down to size on mobile devices but doesn't grow so big that it looks messy on displays with a large resolution. Currently, the biggest screen resolution available is 2,560px wide. If the page is allowed to stretch 100% across a screen of that size, sentences become very long and there is a lot of white space between elements, neither of which is great for readability and usability. Of course, a 960px wide site on a 2,560px wide screen leaves a lot of empty space on either side of the site, so you might want to make the `max-width` wider to better accommodate. I leave that choice to you, but for *CSS3 Foundations*, I stick with 960px.

You also added the declaration `margin: 0 auto`, so what does that do? The `margin` property is similar to `padding`, but rather than adding space between the borders of an element and its child elements, `margin` adds space between the borders of an element and its parent element. The layout of elements in this way is known as the *box model,* which is covered in Chapter 7. The margin you added to the `body` rule set specifies two values: the first being the top and bottom values; and the second, the left and right. So, the `<body>` element has no margin on the top and bottom, but the left and right are half of the remaining space between the `<body>` and `<html>` element—in other words, centered!

Now that the page has a hybrid layout, if you reduce the size of the browser, the page shrinks to fit. However, the smaller the browser becomes, the less room elements have and everything starts to look crammed. In Chapter 16, you learn to use media queries to change the style of elements based on the size of the browser window or device on which the page is being viewed, making the page responsive.

Fluid Images

If you're keen of eye, you may have noticed that although the Cool Shoes & Socks page is now a hybrid layout, while most of the webpage shrinks to fit the browser window below 960px, the showcase image doesn't.

As you can see in Figure 6-2, when a browser is shrunk below 960px, a horizontal scroll bar is added to the page so the user can move across to see the overflow.

FIGURE 6-2 Cool Shoes & Socks viewed in Google Chrome at a width of 500px.

When scrolling horizontally to the right, you can see the page doesn't look too great; in fact, it looks broken, as shown in Figure 6-3. Although the container of the page is 500px—the same size as the browser window—the showcase image sticks out like a sore thumb. The reason is that the natural size of the showcase image is 960px × 301px, and the browser has respected those dimensions so the image doesn't look stretched or shrunken.

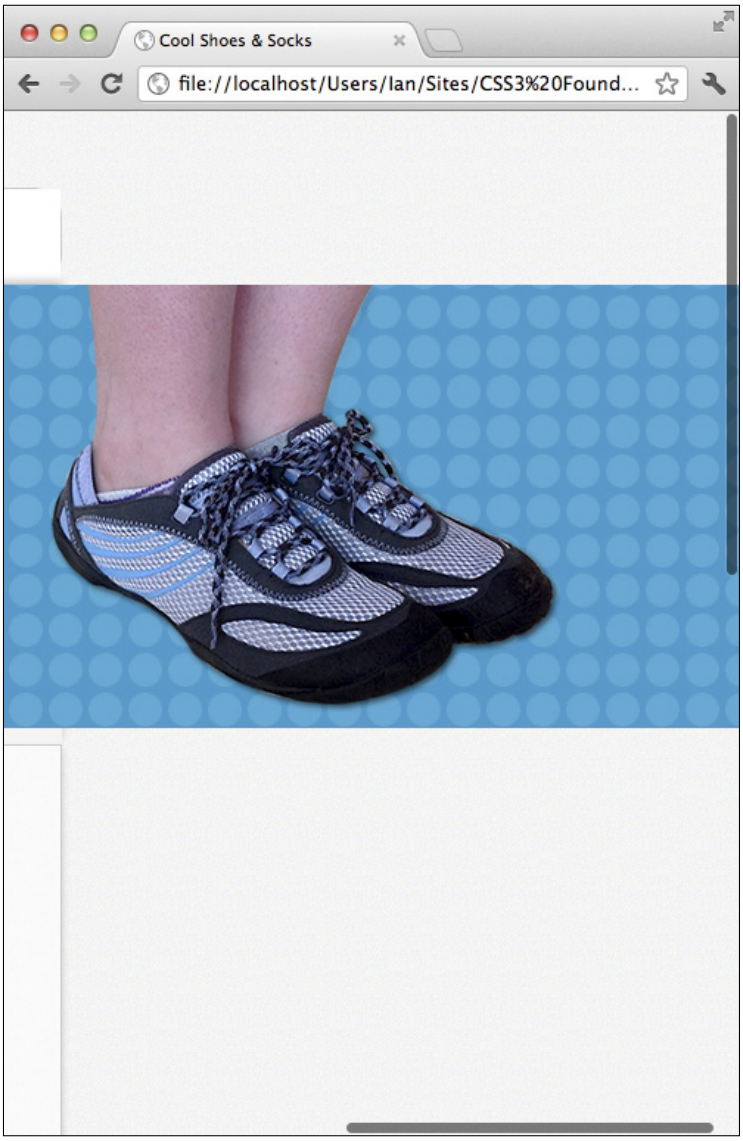


Figure 6-3 The overflow of the Cool Shoes & Socks page when viewed in Google Chrome at a width of 500px.

Go ahead and fix that:

1. In styles.css, underneath the `body` rule set, add the following rule set:

```
img {
    max-width: 100%;
}
```

2. Save styles.css.

This is a rule set very common to responsive design and doesn't just fix the showcase image, but makes all images on the page scale based on the width of the element they are contained within.

With this rule set, the showcase image can now only be the same size as the `<div class="showcase">` element it is contained within. Now, no matter what the width of the browser window, the image is fluid, just like the rest of the page, as shown in Figure 6-4.

Adaptive Design

If *responsive design* is something you've heard of before, you may have heard about *adaptive design,* too. What's the difference?

Where a responsive design uses a fluid layout and media queries to make a web page change its layout for every different width, an adaptive design uses a fixed layout and media queries to change the layout only when specific width conditions are met. A responsive design changes its layout for every different pixel width, whereas an adaptive design may change its layout several times: one for mobile, one for tablets, and one for desktop/laptops, for example.

FIGURE 6-4 The image is now rescaled to fit perfectly in the browser window with a width of 500px.

Mobile First Design

While on the subject of responsive and adaptive design, let's cover another buzz phrase you may have heard of: mobile first design.

The purpose of mobile first design is to begin creating a website with mobile devices in mind first.

This mobile first mindset forces you to think about what is the most important content to display in the space available. You can then slowly build up the less relevant content as and when you have more space to play with. This approach has performance benefits too—for mobile devices that typically have less processing power and generally less bandwidth than other devices.

Because a responsive site unavoidably has some styles on a web page that need to be overwritten to change the layout appropriate for the device on which it is being viewed, it is better to begin with a mobile layout. This way, the mobile device doesn't have to process styles or download images that it doesn't need. Instead, you can leave the overwriting of styles to devices with more processing power and bandwidth, where speed and efficiency aren't quite as critical.

So why isn't this type of design practiced in *CSS3 Foundations*? Well, for a few reasons. First, not every website needs to be responsive, so it is better to learn how to create a web page as you are doing now—by building the desktop version first. In the future, you may find that if you are creating web pages for clients or other people, they request only a desktop version of a website but then later ask you to modify it to be more efficient and accessible for mobile devices. Although "mobile first" is *best* practice, "desktop first" is a real-world practice, too, especially during this time when the web is transitioning from fixed to responsive layouts. You can apply the method of adding media queries, which you do in Chapter 16, in the same way if you decide to try a "mobile first" approach in the future. So by learning the "desktop first" approach, you don't miss out on any techniques, and learning is more efficient.

Summary

In this chapter, you learned about the various layouts that are commonly used for web pages; you also learned what a few of the web industry's most recent buzz phrases mean. The Cool Shoes & Socks web page now has a hybrid layout, and the contents within that layout are scaling relatively to it. With this new overall page structure applied to the page, in the next chapter, you learn about the box model and `box-sizing`, concepts which are used to structure the elements within the main page.

chapter seven

Creating Space and Understanding the Box Model

THE BOX MODEL describes the layout of elements on a web page. In this chapter, you learn how `margin`, `padding`, and `border` affect the layout of the box model, making structuring a page easier.

Project files update (ch07-00): If you haven't followed the previous instructions and are comfortable working from here onward or would like to reference the project files up to this point, you can download them from `www.wiley.com/go/treehouse/css3foundations`.

The Box Model

In Chapter 5, you used `padding` to create some space between the content and border of the newsletter box and changed the `border` properties. By making these changes, you affected the box model of the main newsletter element `<form id="newsletter" action="#">`.

Now add some margin and padding to the footer to better understand these properties:

1. In styles.css, below the other declarations in the `#footer` rule set, add the following:

```
margin-top: 2.5em;
padding: 1.4em 0;
```

2. Save styles.css.

Here, you change the box model of the `<footer id="footer" role="contentinfo">` element, as shown in Figure 7-1.

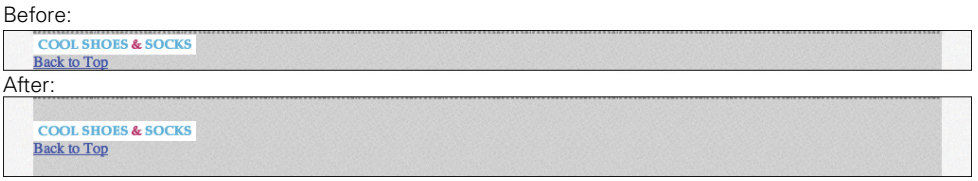


FIGURE 7-1 The footer element before and after adding the margin and padding properties.

Because the box model is invisible—it changes the visual layout of an element, but you can't see the boundaries that make up that element—Figure 7-2 shows exactly how the `margin-top` and `padding` property change the box model for the footer.

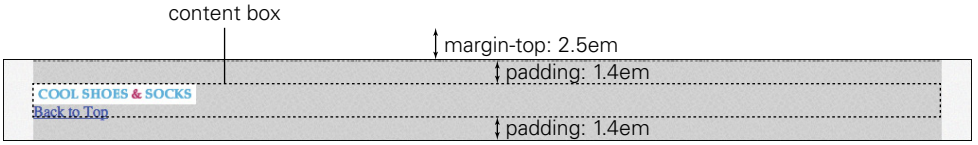


FIGURE 7-2 The way the box model of the footer element is laid out.

Look familiar? Figure 5-19 in Chapter 5 showed the content area and padding area to demonstrate `background-clip` and `background-origin`. In Figure 7-2, the margin area is also shown, which adds space outside the footer element.

Figure 7-3 shows how the content, padding, border, and margin areas relate.

- The content area is that which contains the content.
- The padding area, which is optional and `0` by default, is the area between the content area and the inner edge of the border.
- The border, which is optional and `none` by default, is the visual edge around an element (in Figure 7-3 I made the border 27px so it can be clearly seen).
- The margin area, which is optional and `0` by default, is the area outside the border.

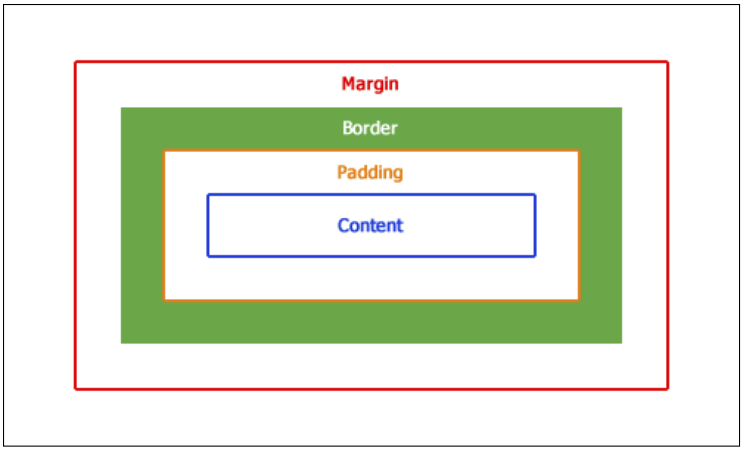


FIGURE 7-3 A simplified version of the box model.

Using Web Developer Tools to Better Understand the Box Model

Sometimes working with the box model can be troublesome. Although manipulating the box model affects elements visually, you can never be sure where the boundaries of the content, border, padding, and margin areas are.

A handy little trick is to apply a temporary border around an element to see exactly how it is laid out, by using the following declaration on the element you are working on:

```
border: red solid 1px;
```

Of course, because this is only a temporary style for your use during the creation of a web page, you can make the border style whatever you want. Sometimes I find myself using temporary border styles each with a different color on multiple elements.

Although seeing the border of an element is helpful, it doesn't show the margin area, and you can only guess at where the content and padding areas are.

Many web developer tools make working with the box model easier, by showing the areas that make up the box model.

In Figure 7-4, I right-clicked on the footer element and selected Inspect Element to use Google Chrome's web developer tools.

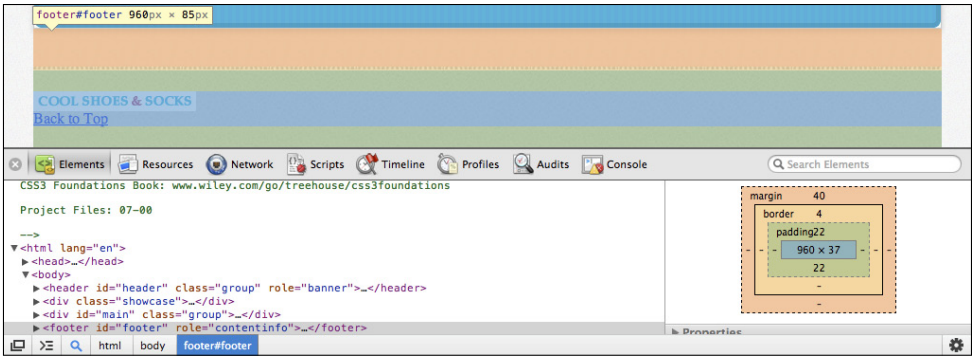


FIGURE 7-4 The footer element being inspected in Google Chrome's web developer tools with the simplified box model in the smaller pane to the right.

Chrome, Firefox (when using Firebug), Safari, and Opera show a visual representation of the box model directly on the page. All browsers (including Internet Explorer 9+) show a visual representation along with measurements in the side panel under "Layout" (or "Metrics" in Chrome and Safari).

In Chrome and Safari 6+, each particular area that makes up the box model is color-coded. The blue area is the content area; green, the padding; yellow, the border; and orange, the margin.

Opera and older versions of Safari don't color-code the areas. Just remember that the content area is the innermost, followed by the padding, and then margin areas.

When using Firebug for Firefox, you must right-click an element and select Inspect Element. Firebug doesn't show the box model until you hover over the element in the DOM window, though. It colors the content area blue, the padding purple, and the margin green.

Internet Explorer 9's equivalent of the Inspect Element feature is called Select element by click, which you can find in the F12 Developer Tools, represented as a white cursor. After you select that, you can then click on the element you want to inspect.

margin

Initial value: 0 | Inherited: No | Applies to: All except elements with table display types other than table-caption, table, and inline-table | CSS2.1

Browser Support: IE 3+, Firefox 1+, Chrome 1+, Opera 3.5+, Safari 1+

As you saw in the box model explanation, `margin` applies space outside an element. A margin can be a value of any length type, such as pixels, percentages, and ems. You also can use

the auto keyword to make the browser determine the margin of an element based on the unused space around that element.

Using margin is a great way of adding *white space*—also known as *negative space*—between elements on a web page. White space is an empty area and doesn't necessarily mean a space is white in color. It is used to give web pages design aesthetics such as rhythm and balance, making for a better user experience.

You may agree at the moment that the Cool Shoes & Socks page doesn't have much rhythm or balance; all the elements are squashed together because they have margins of 0. You can change that:

1. In styles.css, find the following declaration in the body rule set:

   ```
   margin: 0 auto;
   ```

   This is the declaration you added in Chapter 5 to make the page centered.

   The margin property is actually shorthand for the properties margin-top, margin-right, margin-bottom, and margin-left. By specifying two values for the shorthand margin property, you tell the browser the top and bottom of the <body> element should have no margin and the left and right should have a margin of auto.

2. Change the margin value of the body rule set to the following:

   ```
   margin: 2.5em auto 0 auto;
   ```

3. Save styles.css.

Now the <body> has a margin-top of 2.5em, a margin-right of auto, a margin-bottom of 0, and a margin-left of auto. The <body> is still centered but now with some white space above it so the logo no longer touches the top of the window.

You also are able to achieve the same effect using just three values, like so: margin: 2.5em auto 0. In this case, the margin-top is 2.5em, the margin-left *and* margin-right are auto, and the margin-bottom is 0.

Code Challenge: Add More Margins to Elements

In styles.css, do the following:

1. Add a new rule set for #header with the declaration margin: 0 0 2em 0;.
2. Add a new rule set for #header .logo with the declaration margin: 2.4em 0;.
3. Add a new rule set for #header nav with the declaration margin-top: 2em;.

4. Add the declaration `margin: 0 auto;` to the `#content` rule set.
5. Add a new rule set for `.container` with the declarations `max-width: 920px;` and `margin: 0 auto;`.
6. Add the declaration `margin: 2em auto 1em;` to the `input[type="submit"][class="button"]` rule set.
7. Add a new rule set for `aside` with the declaration `margin: 0 0 2em 0;`.
8. Add the declaration `margin: 0 0 1.6em 0;` to the rule set `aside > h3`.
9. Add a new rule set for `#content h1, #content h2, #content h3, #content h4` with the declaration `margin: 0 0 1.6em 0;`.
10. Add a new rule set for `#content p, #content ul` with the declaration `margin: 0 0 1.25em 0;`.
11. Add a new rule set for `blockquote cite` with the declaration `margin: 0.625em 0 0 0;`.
12. Add a new rule set for `#newsletter h3` with the declaration `margin: 0 0 1em 0;`.
13. Add a new rule set for `#newsletter label` with the declaration `margin: 1.5em 0 0 0;`.
14. Add a new rule set for `.aside` with the declaration `margin: 0 0 2em 0;`.
15. Add a new rule set for `.post h3` with the declaration `margin: 0 0 1em 0;`.
16. Add a new rule set for `.post img` with the declaration `margin: 0 .5em 0 0;`.

Project files update (ch07-01): If you haven't followed the previous instructions and are comfortable working from here onward or would like to reference the project files up to this point, you can download them from `www.wiley.com/go/treehouse/css3foundations`.

padding

Initial value: 0 | Inherited: No | Applies to: All except table-row-group, table-header-group, table-footer-group, table-row, table-column-group and table-column | CSS2.1

Browser Support: IE 4+, Firefox 1+, Chrome 1+, Opera 3.5+, Safari 1+

`padding` works in a similar way to `margin` but instead applies to the inside of an element prior to the border, rather than the outside of the element after the border.

The value of `padding` can be a percentage or any unit of length, but with it, unlike `margin`, you can't use the `auto` keyword.

In Chapter 5, you added padding to the newsletter box, like so:

```
#newsletter {
    ...
    padding: 6%;
}
```

This value gives each side of `<form id="newsletter" action="#">` a `6%` padding, which is a percentage relative to the overall width of the newsletter box.

Now, because you haven't defined a width for the newsletter box, its width by default is `auto`. This `auto` setting, as explained previously, tells the browser to calculate an element's width based on the unused space. The width is calculated after the padding, so the calculated width of the newsletter box is `88%`. Why so? Because each side of the newsletter box has a `6%` padding, meaning in total the left and right have 12% padding, leaving that figure 88%. The way in which the box model is calculated seems—to me at least—more difficult than it needs to be, and I expand on this issue shortly so it doesn't catch you out.

As with `margin`, you can specify one to four values for `padding`:

- One applies to all four sides of an element.
- The first value of two applies to top and bottom and the second left and right.
- The first value of three applies to the top, the second to left *and* right, and the third to the bottom.
- Four values apply to top, right, bottom, and left, respectively.

Code Challenge: Add More Padding to Elements

In styles.css, do the following:

1. Add the declaration `padding: 2.5em;` to the `#main` rule set.
2. Add the declaration `padding: 1em;` to the `.button` rule set.
3. Remove the declaration of `padding: 20px;` from the `.showcase .button` rule set.
4. Add the declaration `padding: .8em;` to the `input[type="submit"][class="button"]` rule set.
5. Add a new rule set for `.testimonials` with the declaration `padding: 1em 0 0 0;`.
6. Add the declaration `padding: .4em 1em;` to the `input[type="text"], input[type="email"]` rule set.

Project files update (ch07-02): If you haven't followed the previous instructions and are comfortable working from here onward or would like to reference the project files up to this point, you can download them from `www.wiley.com/go/treehouse/css3foundations`.

The Pitfall of the Box Model and Working Around It

In the definition of the `padding` property, I mentioned the box model can sometimes be harder to work with than it needs to be. Let's take a deeper look at this issue so you can avoid any confusion in the future, or if you are in the fortunate position of not needing to support anything but modern browsers, a way around these difficulties altogether.

When creating a page, you may decide to declare that an element should have a width of 100%. Do that now for the newsletter box:

1. In styles.css, find the `#newsletter` rule set and add the following:

   ```
   width: 100%;
   ```

2. Save styles.css.

As shown in Figure 7-5, the newsletter box has a width greater than 100%, even though you told it to be 100%!

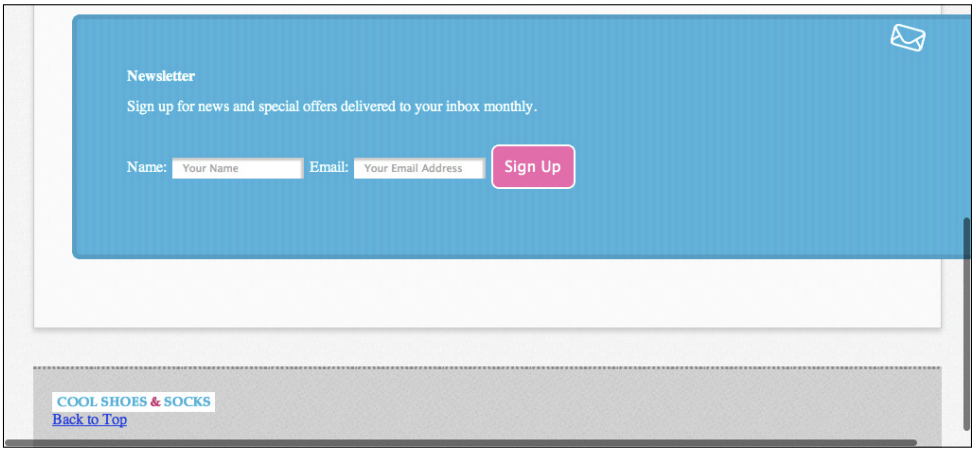


FIGURE 7-5 The newsletter extends beyond the boundaries of the page.

The reason is that, by default, the width of an element is applied only to the content area. The `#newsletter` rule set declares that the newsletter box should have a border width of 5px, a padding of 6%, and a width of 100%:

```
#newsletter {
    color: white;
    border-radius: 8px;
    border: rgba(0,0,0,0.1) solid 5px;
    background:
```

```
        url("../images/icon-newsletter.png") no-repeat 91% 2%,
        url("../images/bg-newsletter.png") repeat 0 #00ACDF;
padding: 6%;
    width: 100%;
}
```

The width of the newsletter box is now

- 10px (5px for the left border and 5px for the right) +
- 12% (for the left and right padding) +
- 100% (for the content area)

Although the browser can easily do these calculations, it makes your job styling elements a little harder because you have to do the math, too. It's not a very human-friendly way of working with the box model, is it? You'll be glad to hear that the W3C is aware of this issue, and CSS Level 3 introduces the `box-sizing` property, which allows you to change how the box model works. Unfortunately, because it's a CSS Level 3 feature, it isn't supported in Internet Explorer versions 6 and 7. If you have the privilege of not needing to support these older browsers, I cover `box-sizing` in a moment, but because the Cool Shoes & Socks scenario requires Internet Explorer 7 support, you need to fix the newsletter box so it *does* fit into the page properly:

1. In styles.css, change the width declaration of the `#newsletter` rule set:

   ```
   width: 88%;
   ```

2. Save styles.css.

This solution is still not completely perfect. You've changed the `width` of the newsletter box to account for the 12% of padding, but the calculated width is still 88% + 12% + 10px, meaning the newsletter box is still 10px bigger than 100%. Because the border is specifically 5 pixels on each side, you can't adjust the width again to account for the border, as you can't use percentages with the `border-width` property.

As you can see in Figure 7-6, the newsletter box isn't quite centered. The left of the newsletter box has a space of 40px, and the right is 32px. This 8px difference is due to the 10px border on the newsletter box (minus 2px for the border on `#main`).

FIGURE 7-6 The amended newsletter box that is no longer centered with a width of 88%.

To fix this, do the following:

1. Add a `margin` declaration to the `#newsletter` rule set:

```
margin-left: -4px;
```

2. Save styles.css.

By giving the newsletter box a left margin of `-4px`, you forcefully pull it over to the left to balance out the space, making each side 36px. You may feel a little strange doing that. It kind of feels as though you are forcefully yanking the newsletter box back over to the left—fighting against the browser. As you gain experience with CSS, though, you'll find that from time to time these little "hacks" give you the control you need to make everything look perfect.

box-sizing

Initial value: `content-box` | Inherited: No | Applies to: All elements that accept width and height | CSS3

Unprefixed browser support: IE 8+, Chrome 9+, Opera 7+, Safari 5.1+

Prefixed browser support: Firefox 1+, Chrome 1+, Safari 3+

The `box-sizing` property, which was introduced in CSS Level 3, gives web page creators the choice of how they want the box model to work. The initial value for `box-sizing` is `content-box`, which has been used throughout *CSS3 Foundations* up to this point and is the described box model in this chapter.

In a nutshell, with `box-sizing` set to `content-box`, when you are specifying a width and height, those properties are applied to the content area of an element, and then the padding and border are applied outside those dimensions, as shown in Figure 7-7.

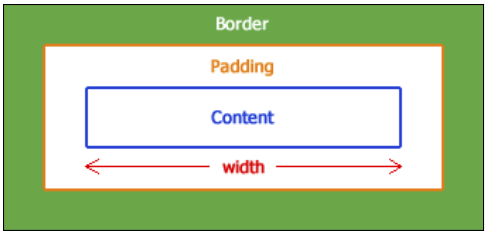


FIGURE 7-7 The way the areas of the box model are laid out when the box-sizing is content-box.

By changing the `box-sizing` to `border-box`, you, like me, may find the box model easier to work with. Before I explain, remember that `box-sizing` works in all modern browsers, as well as Internet Explorer 8, but not versions 6 and 7. For this reason, you use `box-sizing: border-box;` for only a few elements that don't need to be perfect in older versions of Internet Explorer.

`border-box` applies padding and borders inside an element with a specified width and height, as shown in Figure 7-8.

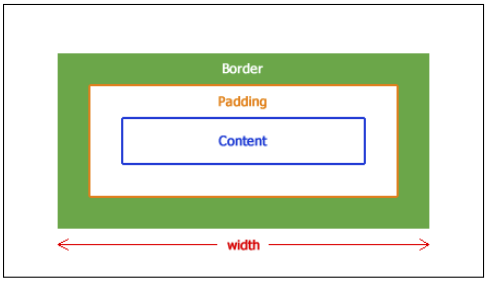


FIGURE 7-8 The way the areas of the box model are laid out when the box-sizing is border-box.

So, if the newsletter box has a specified width of 100%, you don't have to work around the centering issues you saw. These styles caused the newsletter box to appear bigger than the width of the page:

```
#newsletter {
    color: white;
    border-radius: 8px;
    border: rgba(0,0,0,0.1) solid 5px;
    background:
        url("../images/icon-newsletter.png") no-repeat 91% 2%,
        url("../images/bg-newsletter.png") repeat 0 #00ACDF;
    padding: 6%;
    width: 100%;
}
```

If you were to give the `#newsletter` rule set the declaration `box-sizing: border-box;`, the browser would make the newsletter box 100% and then apply the padding and border inside that, meaning the newsletter box wouldn't get any bigger than the defined 100%. Woohoo! Such a simple property makes working with the box model so much easier. No more math!

Again, `box-sizing` isn't supported in Internet Explorer 6 and 7, so you should use it only if you're not supporting those browsers or use it on noncritical styles. Now add a noncritical style to the page and make use of `box-sizing: border-box` in doing so:

1. In styles.css, find the rule set for `input[type="text"], input[type="email"]` and add the following:

```
width: 100%;
```

2. Save styles.css.

The input fields with type `text` and `email` (which are in the newsletter box) now have a width of 100%, *but* because those elements also have padding, the same issue occurs as with the newsletter box itself.

In Figure 7-9, you can see that the input fields have a width of 100% + 2em because of the declaration `padding: .4em 1em;`.

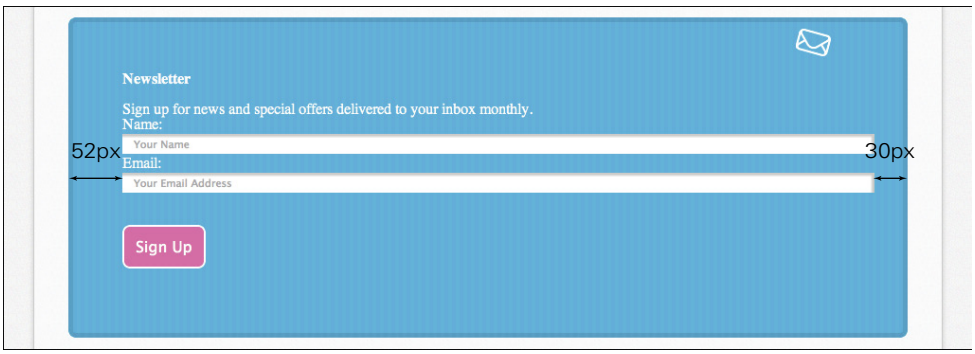


FIGURE 7-9 The input fields are not centered.

Rather than adjust values and use that negative `margin-left` hack as you did before, use the `box-sizing` property to change how the box model is laid out instead:

1. In styles.css, add the following to the rule set `input[type="text"], input[type="email"]`:

```
box-sizing: border-box;
```

2. Save styles.css.

As yet, Firefox does not support the unprefixed `box-sizing` property. If you are using Firefox as your development browser, please use `-moz-box-sizing: border-box;` instead. If using a different browser, you will provide support for Firefox in Chapter 16.

And now, the input fields are exactly 100% wide and not a single pixel more, as you can see in Figure 7-10. You don't even need to do any math. Freedom!

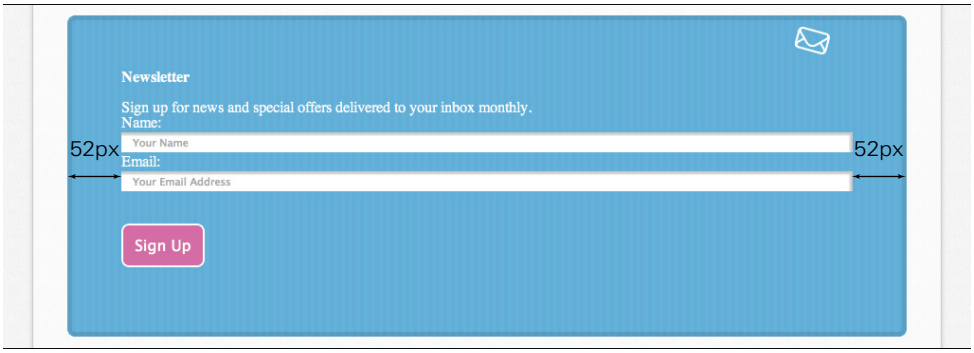


FIGURE 7-10 The input fields are centered now the box-sizing has been changed to border-box.

You may remember `content-box` and `border-box` from the `background-clip` and `background-origin` definitions in Chapter 5. You may also remember a third value those properties accepted, called `padding-box`, which at present is included in the CSS3 User Interface Module, where `box-sizing` is described (`www.w3.org/TR/css3-ui/`).

`padding-box` works in a similar way to `border-box`. Where `border-box` draws a border and padding inside the specified width and height of an element, `padding-box` draws the padding only on the inside and the border remains on the outside. I'm not sure why you would choose this property over the original box model `content-box` (because it's just as unintuitive to start with, if not more so) or the newer and easier to use `border-box`. This is obviously an opinion shared by browser vendors and the W3C, too, because not all browser vendors have implemented `padding-box`, and the W3C has suggested `padding-box` might be removed from the specification in the future.

Summary

In this chapter, you learned about the mysterious box model—the way in which browsers determine the layout of elements. The box model is an important concept to understand if you really want to achieve well-laid-out pages that can not only change in size based on the device a page is being viewed on but will also remain robust and modular as you continue building and styling a page. As you saw in the code challenges, a page will often have many

padding and margin properties applied to it. Spending a little extra time carefully considering padding and margin during the layout stage can often save you a lot of time in the future.

With this knowledge under your belt, you're close to completing the layout of the Cool Shoes & Socks page. In the next chapter, you learn to make elements "float" next to each other, creating a multicolumn layout.

chapter eight

Creating a Multicolumn Layout

ONE OF THE most common layouts in web design is a multicolumn layout. By giving a web page columns, you can separate content into relevant sections, making better use of the space available.

Multicolumn layouts are most commonly achieved using the `float` property, which technically was never added to the CSS specification for the purpose of creating robust, multicolumn layouts. Its true purpose is actually quite a simple one: to allow one element to flow next to another. Creating multiple columns via the use of `float`, `clear`, and other properties is really a "hack"—you're making use of something in the way it wasn't intended—but it's a hack that is needed because CSS, as yet, offers very little in terms of enabling the creation of multicolumn layouts.

CSS Level 3 sees the introduction of flexbox (`www.w3.org/TR/css3-flexbox`) and regions (`www.w3.org/TR/css3-regions`), modules that aim to improve how columns and content areas are created on a web page. Unfortunately, both of these modules are in working draft and have little to no support in modern browsers, so they can't be used on a working website.

Currently, each element on the Cool Shoes & Socks web page vertically follows the next. In this chapter, you learn the basics of the `float` and `clear` properties and then combine these properties along with others to place elements side by side to make better use of the horizontal space available.

Project files update (ch08-00): If you haven't followed the previous instructions and are comfortable working from here onward or would like to reference the project files up to this point, you can download them from `www.wiley.com/go/treehouse/css3foundations`.

float

Initial value: `none` | Inherited: No | Applies to: All unless `display` in `none` or `position` is `absolute` | CSS2.1

Browser Support: IE 4+, Firefox 1+, Chrome 1+, Opera 7+, Safari 1+

At present, the product image of those irresistible horse socks—positioned in the main content of the Cool Shoes & Socks page—breaks the flow of text and takes up more space than necessary. By floating that image, you can make the text flow alongside it:

1. In styles.css, below the `#main` rule set, add a new rule set:

```
.product {
    float: left;
}
```

2. Save styles.css.

As shown in Figure 8-1, the product image now floats to the left of the containing element, and following elements that were vertically positioned below that element now flow to its right. By floating the elements this way, you utilize some of that unused space, and the image looks as though it relates to the text now.

Of course, you can float elements to the right of the containing element too, by declaring `float: right;`, which makes elements flow to the left of that floated element.

`float` also accepts the value none, which means an element should not float.

When using `float`, you can style a floating element just as you would when it's not floating. Although having the text flow to the side of the product image is desired, at the moment there's not much space between the image and text. You can change that:

1. In styles.css, add the following to the `.product` rule set:

```
margin: 0 1em 1em 0;
```

2. Save styles.css.

The image now has a small amount of white space to its right as well as below it.

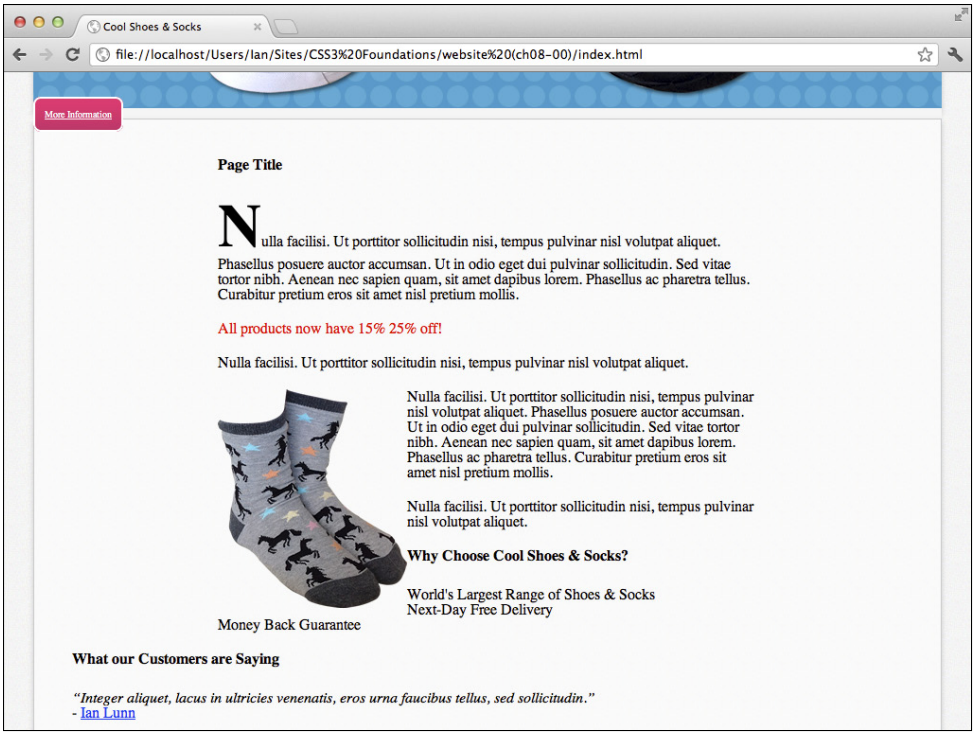


FIGURE 8-1 The main text flowing to the right of the floating image.

clear

Initial value: `none` | Inherited: No | Applies to: Block-level elements (covered in Chapter 9) | CSS2.1

Browser support: IE4+, Firefox 1+, Chrome 1+, Opera 3.5+, Safari 1+

You may notice that now the product image is floating and the text flows to its right, so too does the "Why Choose Cool Shoes & Socks" title and list items below it (contained with the `<div class="benefits">` element). Assume the "Why Choose Cool Shoes & Socks" content is to appear below the product image, it can be pushed down by clearing the float.

`clear` works in tandem with the `float` property and accepts the values `left`, `right`, `both`, and `none`.

1. In styles.css, add a new rule set for `.benefits`, like so:

```
.benefits {
    clear: left;
}
```

2. Save styles.css.

By clearing the left side of the `<div class="benefits">` element, you tell the browser this element shouldn't have a floating element to its left. When you declare `clear: left;`, the browser positions a cleared element to make sure its top border is below the bottom border of the floating element, pushing that element down, as shown in Figure 8-2.

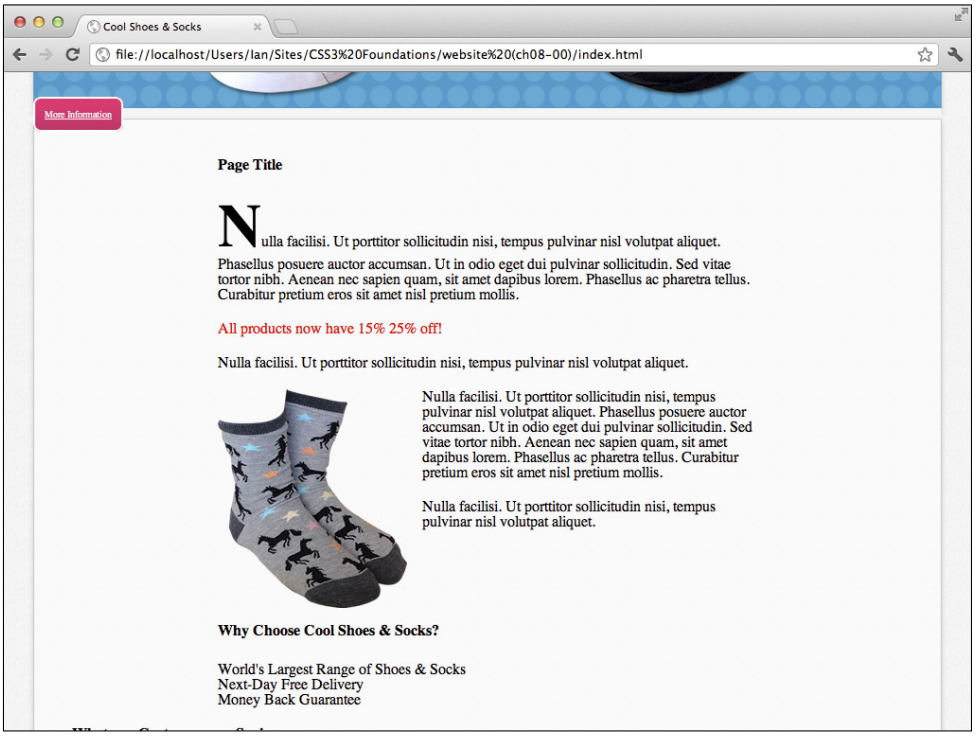


FIGURE 8-2 The element clearing any floating elements to its left.

Likewise, if you need to clear the right side, you can declare `clear: right;`, or to clear both sides, you can declare `clear: both;`.

If you give the `.benefits` rule set the declaration `clear: both;`, because no elements are floating to the right, the same visual effect is achieved.

Floating Multicolumns

The `float` and `clear` properties work together to allow for placing elements next to each other. When `float` and `clear` were introduced to CSS, it became apparent that they could be used beyond their intended purpose for creating multiple columns that can sit aside each other.

The sidebar was intended to sit to the right of the main content on the Cool Shoes & Socks web page, so set that up now:

1. In styles.css, find the `#content` rule set and add the following:

   ```
   float: left;
   ```

2. Save styles.css.

   Note that the `<div id="content" role="main">` element was originally centered via the declaration `margin: 0 auto;`, but this no longer applies because the `float` declaration takes precedence, and as shown in Figure 8-3, the content box now floats to the left of its containing element `<div id="main" class="group">`.

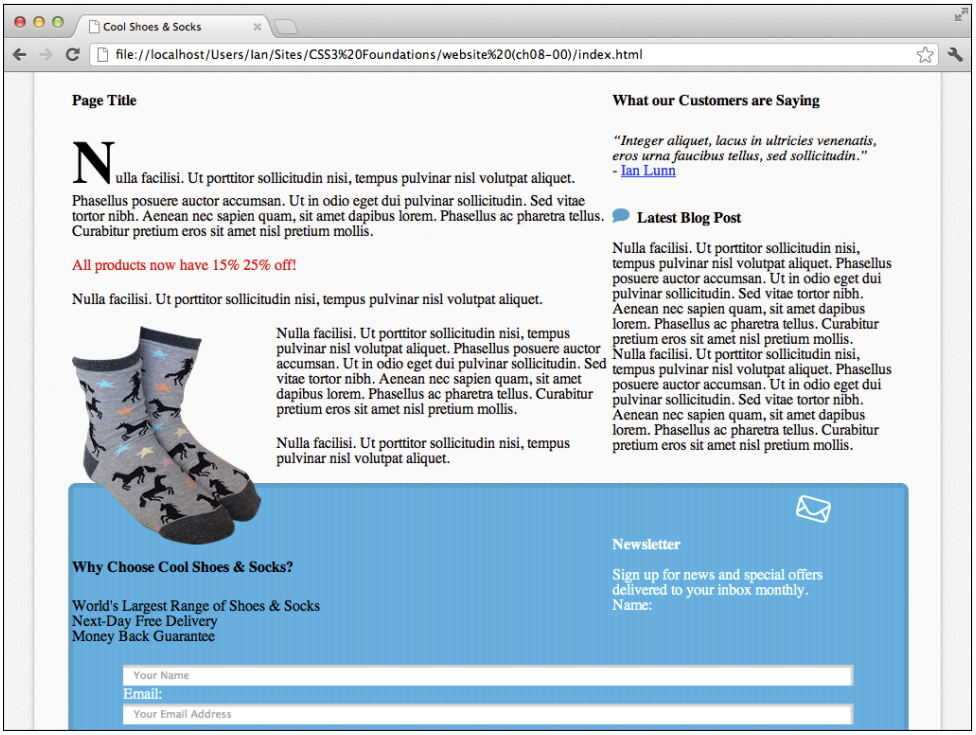


FIGURE 8-3 The content box floating to the left of its containing element.

3. In styles.css, add a new rule set:

```
.sidebar {
    float: right;
    width: 25%;
}
```

4. Save styles.css.

As you can see in Figure 8-4, although you created two columns, the layout isn't quite right. The reason is that now all the elements within the main element are floating; there's nothing clearing the elements below it. Consequently, those elements are trying to flow to the sides of the columns.

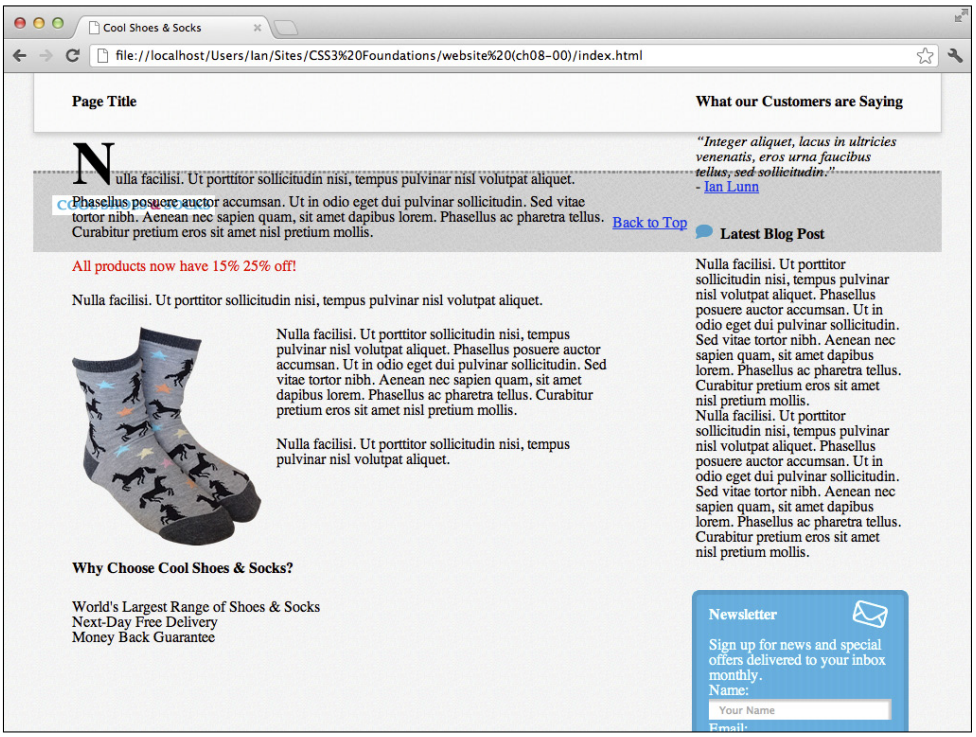


FIGURE 8-4 The two columns floating side by side, but elements below that aren't being cleared.

This is the "hacky" nature of creating columns with floats, and you can find many ways around this issue.

You may think that clearing the footer `<footer id="footer" role="contentinfo">` element would seem to work because that is the element that follows on from the floating elements.

To do that, you could add the following to the `#footer` rule set:

```
clear: both;
```

Although this technique does as expected—the footer clears below the floating columns—an unexpected behavior occurs, too: The main container doesn't appear to be aware of the size of the columns and is no longer wrapped around them, as shown in Figure 8-5.

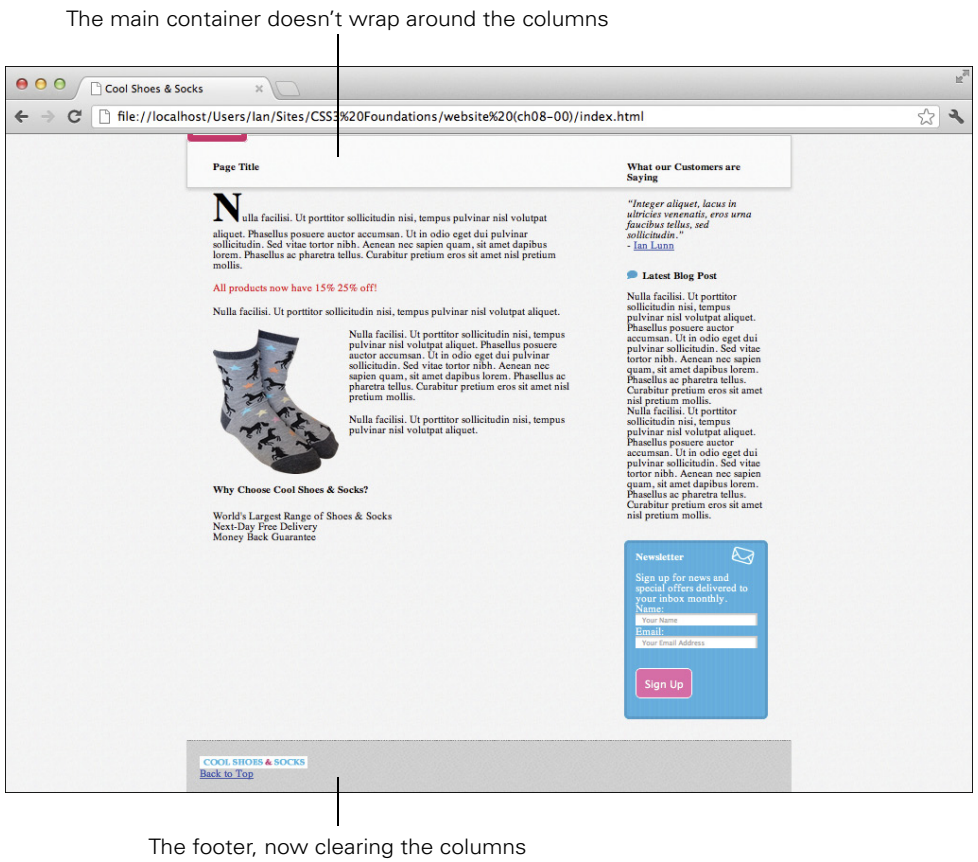


FIGURE 8-5 The browser window zoomed out to show the footer is now cleared, but the height of the main container isn't right.

You might be tempted to specify the height of the main container to make it *appear* to contain the columns, but doing so is a bad practice because you can't guarantee the main container or its contents will always be the same height. This solution is a no-go.

You could add an empty HTML element below the two floated elements:

```
<div id="main" class="group">
    <div id="content" role="main">
        ...
    </div>
    <div class="sidebar" role="complementary">
        ...
    </div>
    <div class="clear"></div>
</div>
```

Then use CSS to select the empty element to clear it:

```
.clear {
    clear: both;
}
```

Figure 8-6 shows this solution works perfectly in all browsers, but it means adding an empty element to the page, which goes against best practice. So is there a better technique?

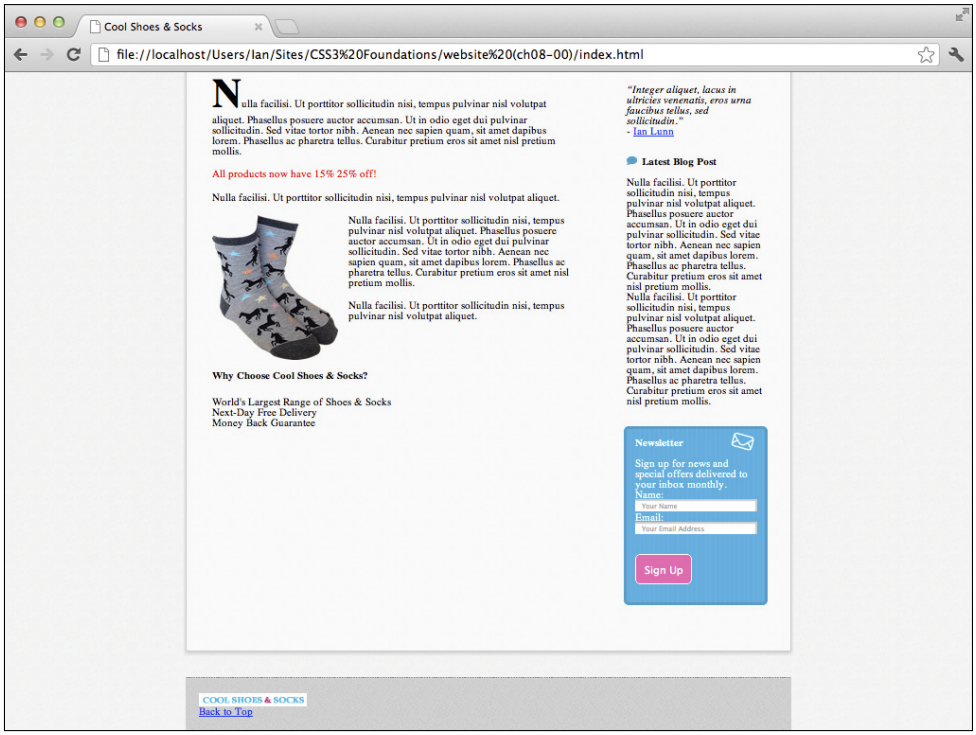


FIGURE 8-6 The browser window zoomed out to show the columns cleared by an empty HTML tag.

I could show you many different ways to work around this situation, and if you were to ask a few web designers and developers, they all would probably give you a different answer as to which technique they use. Remember that `float` and `clear` weren't created for the purpose of columns, so there's no explanation in the CSS specification of how to achieve this effect. Ideally, the best technique is the easiest to implement while not breaking any rules of good practice. With this in mind, I personally use and recommend a technique that has evolved over time and eventually been given the name *micro clearfix hack* by Nicolas Gallagher, who has worked to make this particular technique as small and efficient as possible. You can read about his work on it at `http://nicolasgallagher.com/micro-clearfix-hack`.

The micro clearfix hack in all of its glory:

```
.group:before, .group:after {
    content: "";
    display: table;
}
.group:after {
    clear: both;
}
.group {
    zoom: 1;
}
```

You may see this code used elsewhere on the web with the class name "clearfix" or "cf" for short. I personally prefer to call the class "group" to show that it contains a group of elements (acting as columns).

By using this CSS, you can give an element containing floating elements the class named `group` to fix the issue. The main container already has the class `group`, so add this fix to the CSS:

1. In styles.css, below the `body` rule set, add the following:

```
.group:before, .group:after {
    content: "";
    display: table;
}

.group:after {
    clear: both;
}

.group {
    zoom: 1;
}
```

2. Save styles.css.

The page now looks as intended, as it did in Figure 8-6 but with a more robust clearfix solution applied to it, but what exactly is that CSS doing?

It's using some properties yet to be covered:

```
.group:before, .group:after
```

First, pseudo-elements are placed before and after the `.group` element.

```
content: "";
display: table;
```

The pseudo-elements are given an empty content declaration, which, as explained in Chapter 3, is required to get pseudo-elements working.

These pseudo-elements are told to display as if they were tables. The `display` property is covered in the Chapter 9. When you make them act as tables, margin-collapse is prevented, which is to say any margins on elements before and after these elements are still respected.

```
.group:after {
    clear: both;
}
```

The pseudo-element that is placed after the `.group` element is then cleared on both sides, which makes the main container the correct height—so it actually contains the floating elements—and pushes the footer down below the main container.

These rule sets alone create a clearing effect that fixes the issue seen in Figure 8-6. However, as is often the case, this fix doesn't work in Internet Explorer versions 6 and 7 without one final tweak:

```
.group {
    zoom: 1;
}
```

As explained with the `filter` property in Chapter 5, Microsoft used to be in the habit of adding unofficial properties to its browsers. `zoom` is another of those properties, but here you can use it to get Internet Explorer versions 6 and 7 to clear the floating elements correctly. By using `zoom`, you can work around a bug in these browsers that make the clear fix work as expected.

Now this CSS is in place, whenever an element contains floating elements, you can give that containing element a class of `group` to make the container clear them properly.

With the sidebar now floating to the right of the page and with a width of 25%, you can see that all those styles you added to the newsletter box finally make sense. No longer is the newsletter box stretched right across the page with lots of padding; instead, it's a nice-looking box that draws the users' eyes in an attempt to persuade them to sign up for the newsletter.

You may have noticed the header also has a class of `group`; the reason is that the logo and navigation float side by side, just like the content and sidebar.

1. In styles.css, find the `#header .logo` rule set and add the following:

```
float: left;
width: 240px;
```

2. Find the `#header nav` rule set and add the following declaration:

```
float: right;
```

3. Save styles.css.

By floating the logo to the left and navigation to the right, as shown in Figure 8-7, you make better use of the horizontal space available. At the moment, the navigation still looks a little messy, but in Chapter 9, you learn about `display` properties to better align those navigation links.

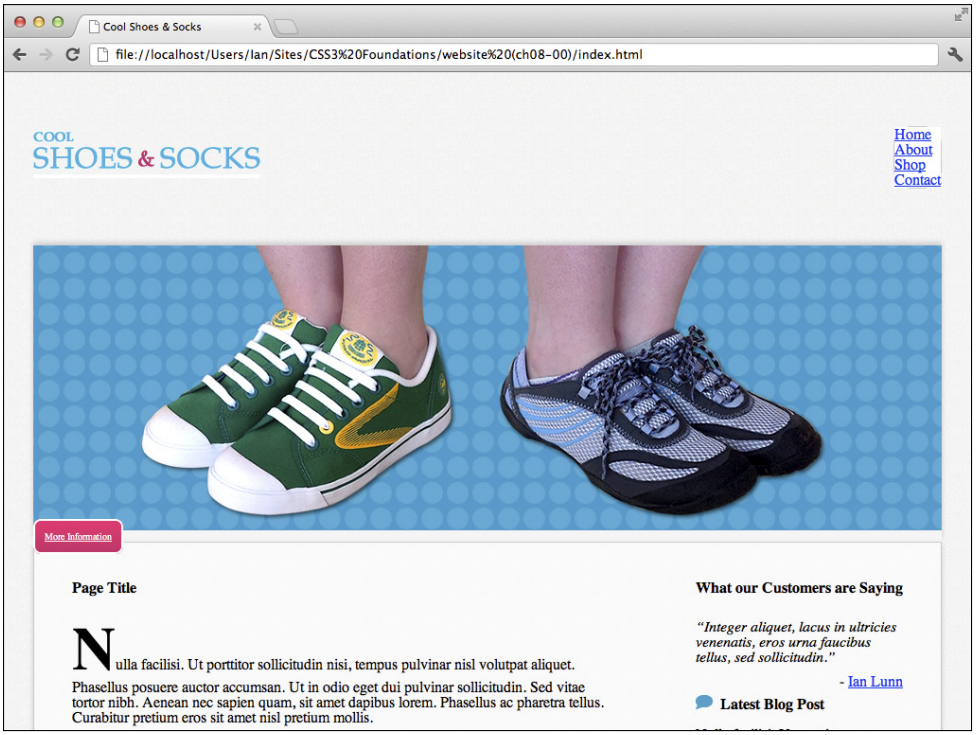


FIGURE 8-7 The logo and navigation links now float side by side.

Code Challenge: Make the Footer Elements Float Side by Side

In styles.css, do the following:

1. Add a new rule set for `#footer li:nth-child(1)` with the declaration `float: left;`.
2. Add a new rule set for `#footer li:nth-child(2)` with the declaration `float: right;`.
3. Add the declaration `float: right;` to the `blockquote cite` rule set.

Summary

After you make the footer elements float, you may notice that the footer isn't being cleared. In the next chapter, you use a different method to change how the footer is displayed, so clearing it isn't necessary.

In this chapter, you've given the Cool Shoes & Socks page a multicolumn layout by floating elements, then clearing any elements that should sit below those columns.

In the next chapter, you complete the main layout of Cool Shoes & Socks. You change the layout of the navigation links using the `display` property and learn about manipulating the position of elements in the flow of the document.

chapter nine

Understanding Display, Position, and Document Flow

IN THIS CHAPTER, you complete the main layout of Cool Shoes & Socks, making use of the `display` and `position` properties. Plus, you learn about the flow of a document. Later in the chapter, you use some of the more advanced CSS selectors that were covered in Chapter 3, to create a drop-down menu that will appear when users hover over the "Shop" link.

Project files update (ch09-00): If you haven't followed the previous instructions and are comfortable working from here onward or would like to reference the project files up to this point, you can download them from `www.wiley.com/go/treehouse/css3foundations`.

Document Flow

A *document* is the fancy name for a web page. A web page has a flow, which determines the layout of elements. Prior to the preceding chapter, in which you made elements float, the entirety of the Cool Shoes & Socks page had a "normal flow."

All elements on a web page, by default, have a normal flow, referred to as being "in flow." This means the position of one element affects the next. Figure 9-1 shows how the Cool Shoes & Socks page looked at the end of Chapter 7, where all elements were in flow—each element following the next.

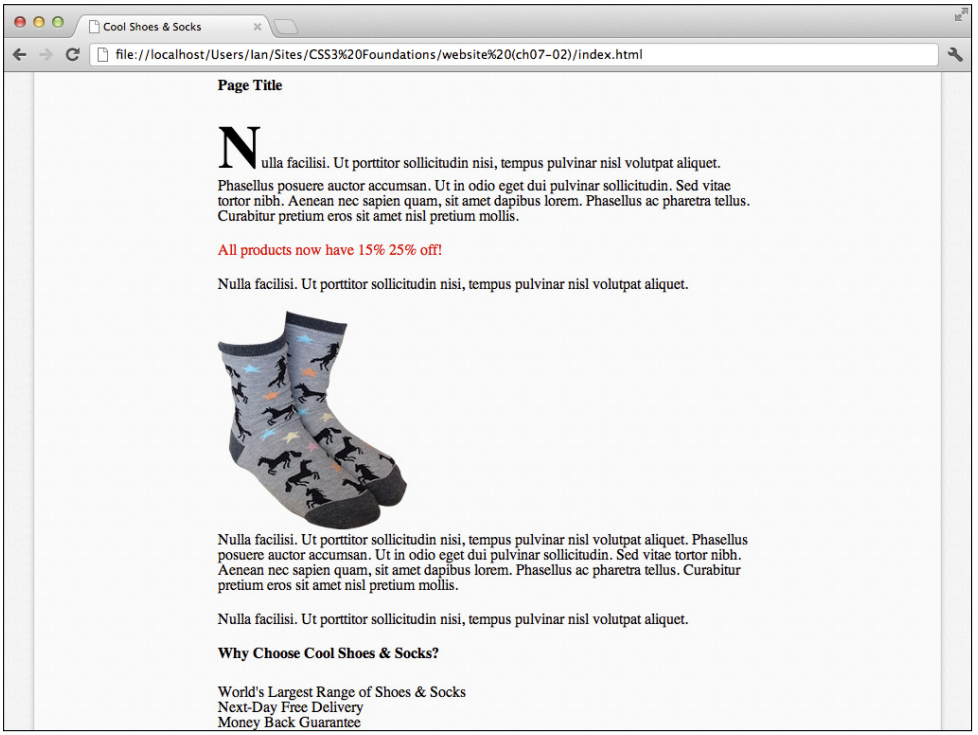


FIGURE 9-1 The Cool Shoes & Socks page as it was at the end of Chapter 7.

Assume you have a page with two paragraphs of content (whether they are on a web page or a word processing document, which also has a normal flow). When you add a new paragraph in between the other two, the second paragraph is pushed down to accommodate the new paragraph. This happens because the document has a normal flow and elements are positioned relative to each other.

Using CSS, you are able to take an element "out of flow," using the `position: absolute;` declaration. When you do this, an element no longer affects or is affected by the elements around it. This capability has an endless number of uses, and when an element doesn't need to be in flow, it allows for greater control over the position of that element.

The beauty of this capability is that a containing element can be in flow but its child elements can be out of flow—useful for web page features such as drop-down menus (shown in Figure 9-2), which you add to the page later in this chapter.

The normal flow and absolute position are types of positioning schemes that affect the document's flow. *Float* is another form of positioning scheme, which essentially is a hybrid of normal flow and absolute positioning. Floats are first laid out according to the normal flow (affecting the elements relative to it) and then taken out of the flow of the document, being absolutely positioned, typically to the left or right of its containing element.

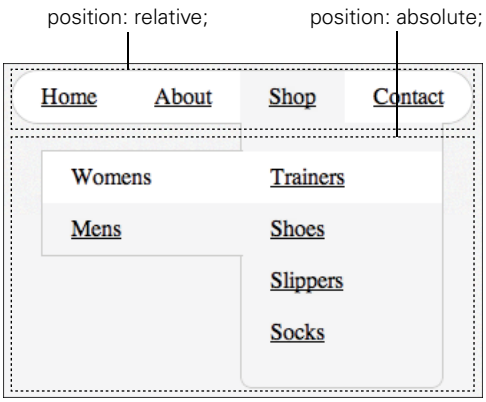


FIGURE 9-2 A menu positioned relative and "in flow," with a drop-down submenu positioned absolute, "out of flow."

You already used the float positioning scheme in the preceding chapter, so now take a look at the normal flow and how you can manipulate elements within that scheme.

display

Initial value: `inline` | Inherited: No | Applies to: All | CSS2.1

Browser support: IE 4+, Firefox 1+, Chrome 1+, Opera 7+, Safari 1+

The `display` property determines the type of box an element generates and has values such as `block`, `inline`, `inline-block`, `list-item`, `none`, and quite a few different values relating to the display of tables.

block

Most elements are given the declaration `display: block;` by a browser's default stylesheet. An element that is a block—referred to as being block-level—doesn't have other block-level elements to the side of it, which, as shown in Figure 9-1, is why elements only appear one after the other vertically rather than side by side.

By default, block-level elements usually have some white space around them, but this is something you removed when adding the CSS Reset in Chapter 2 for better consistency across browsers.

Figure 9-3 shows two `<div>` elements, which are block-level and 100% wide by default.

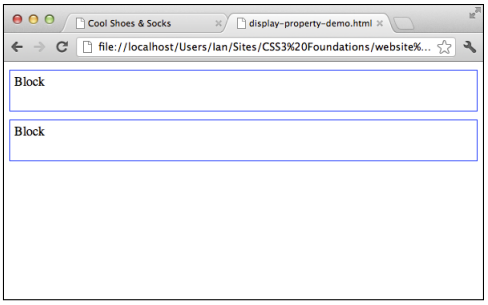


FIGURE 9-3 Two block elements with a default width of 100% and specified height of 40px.

This display properties demonstration page can be found in the project files ch09-00, named display-property-demo.html

On the Cool Shoes & Socks page at present, the Sign Up button in the newsletter box is an inline-block element. While the margin top value of `2em` and margin bottom value of `1em` of the Sign Up button are being rendered, the margin left and right value of `auto` are not. The reason is that inline and inline-block elements don't have a defined width; they are simply the width of their contents—more on this shortly. So, when the browser sees that margin left and right value of `auto`, it can't determine the unused space around that element because it doesn't know how wide that element is, and as such, the button doesn't get centered. To work around that problem, make the Sign Up button block-level instead:

1. In styles.css, find the rule set for `input[type="submit"][class="button"]` and add the following declaration:

   ```
   display: block;
   ```

2. Save styles.css.

Now the Sign Up button is block-level, the margin declaration `margin: 2em auto 1em;` takes full effect, and the button is horizontally centered relative to the newsletter box.

Code Challenge: Make the Newsletter Labels Block-level

In styles.css, do the following:

Add a new rule set for `label` below the `#newsletter label` rule set with the declaration `display: block;`.

inline

An element with the declaration `display: inline;` is only the size of the content it contains and positions itself inline, meaning it doesn't attempt to push itself away from, or below other elements.

In Figure 9-4, I placed two inline-level elements in between two block elements. Because the element above the inline-level elements is a block, it can't have elements to its side, so the inline elements are placed below it. The inline elements don't form new blocks, so they sit side by side and are only as wide as the contents they contain. Because of this, properties that affect the layout of an element can't be used on inline elements, such as `width`, `height`, and `margin`. In Figure 9-4, each element has a height of 40px, but this is ignored for the inline-level elements.

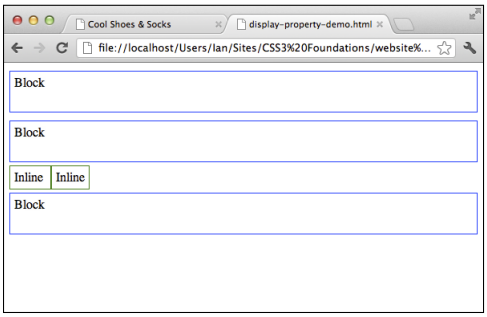


FIGURE 9-4 Two inline elements among two block elements.

By default, elements such as anchors `<a>`, images `<img>`, and quotes `<q>` are inline-level.

List of Inline Elements

The following elements are inline by default:

- `<b>, <big>, <i>, <small>, <tt>`
- `<abbr>, <acronym>, <cite>, <code>, <dfn>, <em>, <kbd>, <strong>, <samp>, <var>`
- `<a>, <bdo>, <br>, <img>, <map>, <object>, <q>, <script>, <span>, <sub>, <sup>`
- `<button>, <input>, <label>, <select>, <textarea>`

Prior to giving the Cool Shoes & Socks page a `max-width` declaration in Chapter 5, you may have noticed some inline-level elements.

Figure 9-5 shows that the image and More Information button in the product showcase are both inline-level, positioning them side by side (the More Information button breaks onto a new line because the page isn't wide enough for both the image and link). The fact that these elements are inline is not as obvious when you give the page a maximum width because there isn't enough room for the elements to sit beside each other.

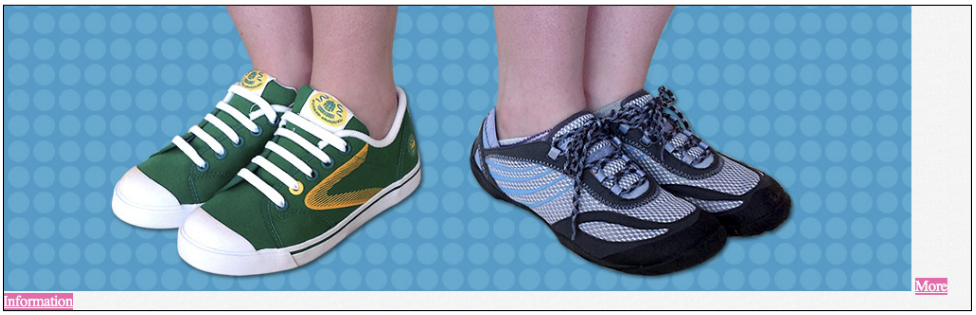


FIGURE 9-5 Some of the inline-level elements prior to Chapter 5.

Later in this chapter, you use the `position` property to position the More Information button over the top of the showcase image.

Some elements that display well as inline-level are the navigation links, which are currently `display: list-item;`. Change that as follows:

1. In styles.css, find the `#header nav > ul > li` rule set and add the following declaration:

   ```
   display: inline;
   ```

2. Save styles.css.

When you make each navigation list item inline-level, those links line up horizontally, as shown in Figure 9-6. This arrangement makes better use of the horizontal space available.

FIGURE 9-6 The navigation links positioned inline.

inline-block

Sometimes you might want an element to be inline but still give it dimensional values such as height and margin. These types of elements can be given the declaration `display: inline-block;`.

In Figure 9-7, I added two inline-block elements to a page. These properties take on attributes of both inline-level and block-level elements. First, they take on an inline-level, being positioned next to each other. Unlike inline-level elements, though, they are then formatted as a block-level, allowing them to be given dimensional properties. As you can see, these inline-block elements have taken on the height declaration of 40px, just like the block-level elements have.

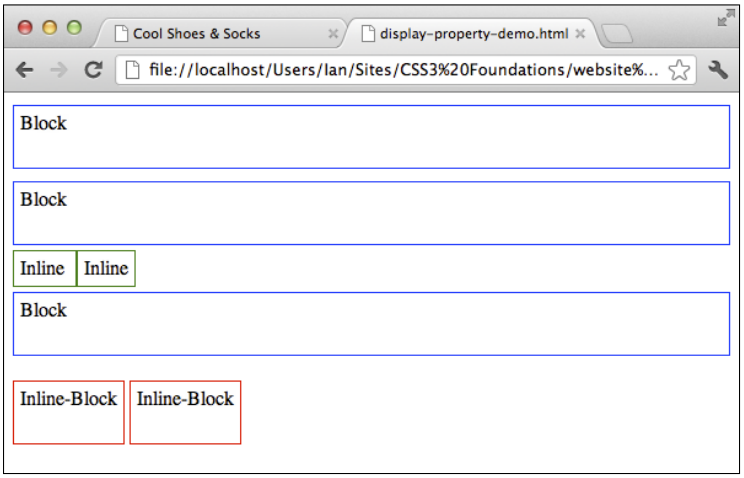


FIGURE 9-7 Inline-block elements in relation to block and inline elements.

Aside from the height and margin of the inline-block elements, can you see what else is different about them when compared to the inline-level elements? Inline-block elements are horizontally separated by 4 pixels of white space. This is an intended behavior although, admittedly, an annoying one! I show you how to fix this behavior in a moment.

Although you made the navigation links inline, you will shortly add `margin` properties to them, so instead make them inline-block:

1. In styles.css, change the display declaration of `#header nav > ul > li` to

   ```
   display: inline-block;
   ```

2. Save styles.css.

As you can see in Figure 9-8, `inline-block` elements are separated by 4 pixels, and although the width of the navigation links isn't critical (there's plenty of room for the navigation links to stretch out), it's often easier to get rid of these additional 4 pixels, especially when you know some elements are going to have quite a few different styles applied to them.

FIGURE 9-8 The navigation links now being displayed as inline-block.

These 4 pixels aren't actually caused by the CSS but instead the white space in the HTML (the spaces that separate the `<li>` elements).

1. In index.html, find the unordered list in the header:

```
<nav role="navigation">
    <ul>
        ...
    </ul>
</nav>
```

2. Remove any white space between the top level closing `</li>` and opening `<li>` elements, so

```
<li><a href="#" title="Return to the front page">Home</a></li>
<li><a href="#" title="About Company Name">About</a></li>
```

   becomes

```
<li><a href="#" title="Return to the front
page">Home</a></li><li><a href="#" title="About Company
  Name">About</a></li>
```

   and so on. You should only need to remove three amounts of white space between four list items.

3. Save index.html.

When you do this, the HTML that makes up the navigation links is a little more difficult to read but it at least removes the 4 px of white space, as shown in Figure 9-9.

FIGURE 9-9 The navigation links with no space between them.

Note that Internet Explorer versions 6 and 7 don't support `display: inline-block;`. However, they fortunately treat `display: inline;` in the same way that other browsers—that better support the CSS specification—treat `display: inline-block;`. So, when an element is required to be inline-block, you can simply specify the declaration `display: inline;` only for Internet Explorer 6 and 7. You learn how to create a stylesheet specifically for the use of older versions of Internet Explorer in Chapter 15.

list-item

List items—the `<li>` elements that are placed within either ordered `<ol>` or unordered `<ul>` lists—have the declaration `display: list-item;` by default. When an element is a list-item, it generates a block box *and* a marker box—if that element has a visual representation of being a list item in the form of a bullet point or other graphical element. Styling lists are covered in Chapter 11.

Displaying Tables

When a page contains a table, using the HTML element `<table>`, it, by default, has the declaration `display: table;`. Aside from the hack you saw in the preceding chapter, where `display: table;` was used to prevent the margins of an element from collapsing (a trait of tables), and one other hack to achieve vertical alignment—which you see later in this chapter—`display: table;` tends to be used only for tables, suprisingly! Because the `<table>` element already has this display value written into user agent stylesheets, you most likely don't need to use it in your own CSS. This, too, goes for the other table display values; `inline-table`, `table-row`, `table-row-group`, `table-header-group`, `table-footer-group`, `table-column`, `table-group-column`, `table-cell`, and `table-caption`.

If you would like to find out more about these table values, see the CSS Table Model at `www.w3.org/TR/CSS21/tables.html#value-def-table`.

none

As you saw in Chapter 5, giving an element the declaration `display: none;` doesn't just hide that element; it tells the browser not to even render it so that no box is generated at all and the layout is treated as though this element doesn't exist—even though it's still present in the HTML.

If you would like to hide an element but still have it affect layout, see the `opacity` and `visibility` properties, which were covered in Chapter 5.

position, top, right, bottom, and left

Initial value: `static` | Inherited: No | Applies to: All | CSS2.1

Browser support: IE 4+, Firefox 1+, Chrome 1+, Opera 4+, Safari 1+

`position` is another property that affects the flow of a document. The values that can be used are `static` and `relative`, which affect the normal flow, and `fixed` and `absolute`, which take elements out of flow.

The CSS3 Positioned layout module found at `www.w3.org/TR/css3-positioning/` introduces two new values, `center` and `page`, but unfortunately they are yet to be implemented into any browsers, so it is too early to cover them.

You can use the `position` declaration in conjunction with the properties `top`, `right`, `bottom`, and `left` (except for when using `position: static;`), which allow you to change the position of an element. How these properties relate to the position of an element is determined based on the value of the `position` declaration.

You can give `top`, `right`, `bottom`, and `left` any type of value such as percentages, pixels, or ems.

static

Elements are `position: static;` by default. A static element is a normal box laid out in the normal flow. Static elements aren't affected by the `top`, `right`, `bottom`, and `left` properties.

relative

When you make an element `position: relative;`, it too is laid out in the normal flow of a document—known as the "normal position." From its normal position, you can then offset the position of an element relative to its normal position using the `top`, `right`, `bottom`, and `left` properties. The elements following a relative element are affected based on the normal position of that element; its relative position doesn't affect layout.

Assume you would like to change the position of the `<h1>` page title on the Cool Shoes & Socks page. In Figure 9-10, you can see the page title is static and in flow, meaning the `<p>` elements and `<img>` below it are placed below it.

If you give `<h1>` the declaration `position: relative;`, the page appears exactly the same as when that element is `position: static;`. However, a relative element can be manipulated with `top`, `right`, `bottom`, and `left`.

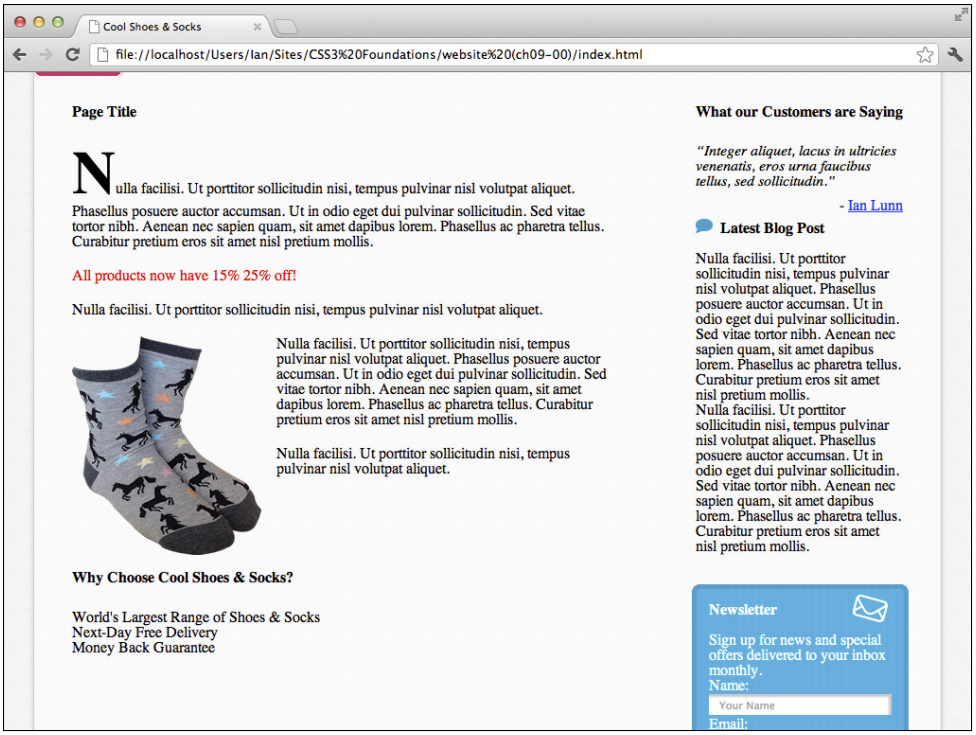


FIGURE 9-10 The page title in its default static position.

In Figure 9-11, I gave the `<h1>` a `top` value of `30px`. This value moved the page title 30px, relative to the top of its normal position.

Note the only difference between Figure 9-10 and Figure 9-11 is the position of the text—the area taken up by the `<h1>` element doesn't change. The flow of the document isn't affected because the page title's normal position remains in flow and in the same position, so although the text is offset, the contents below aren't affected in any way.

When an element is `position: relative;`, `top`, `left`, `right`, and `bottom` offset that element relative to the top, left, right, and bottom of the normal position, respectively.

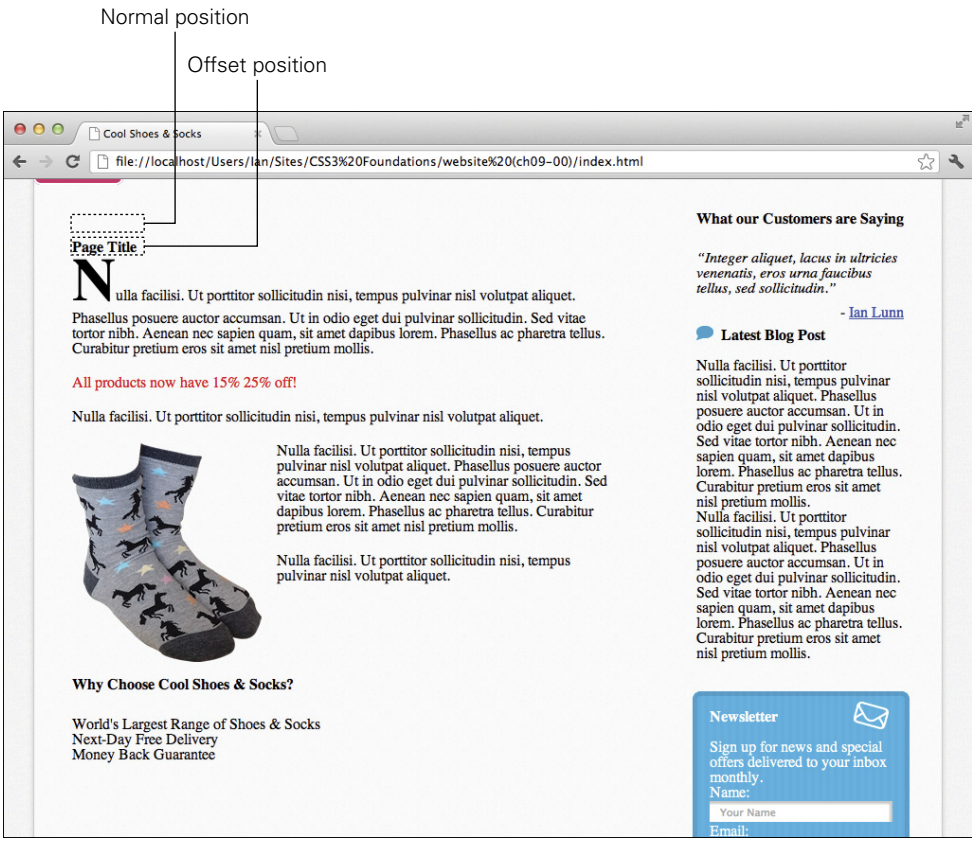


FIGURE 9-11 The page title with the declarations position: relative; top: 30px;.

absolute

When you make an element `position: absolute;`, it is taken out of the document's flow and ignores the layout of elements surrounding it.

Now use `position: absolute;` to finally move that pesky More Information button:

1. In styles.css, add the following declarations to the `.showcase .button` rule set:

```
position: absolute;
top: 0;
```

2. Save styles.css.

   When you make the button absolute, it is taken out of flow (see Figure 9-12), and the main container finally touches the bottom of the product showcase. This effect was always intended, but because the button was in flow and sat between the two, they were separated.

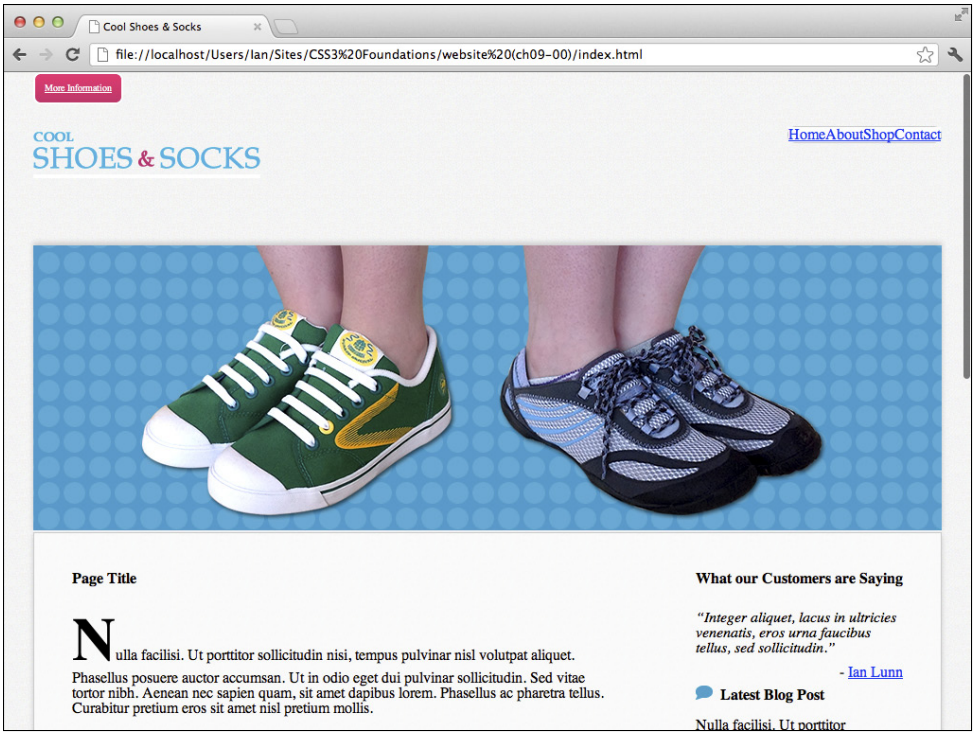


FIGURE 9-12 The More Information button no longer affects the flow of the page.

Note that after this change, the button touches the top of the page, which isn't the intended final position. This happens because the position of an absolute element is relative to the next containing element with a defined position other than `static`. Because you haven't declared any other element as being `absolute`, `relative`, or `fixed` yet, the browser treats the `<html>` element as the containing element, and as such, the button is positioned relative to the overall page. So, as you can see, `top: 0;` makes the button touch the top of the page.

3. In styles.css, add the following declaration to the `.showcase` rule set:

```
position: relative;
```

4. Save styles.css.

Now that you've made the `<div class="showcase">` element relative, the button contained within it is positioned relative to it, as shown in Figure 9-13. It touches the top of the showcase, but it would look better if it were placed over to the right and vertically centered.

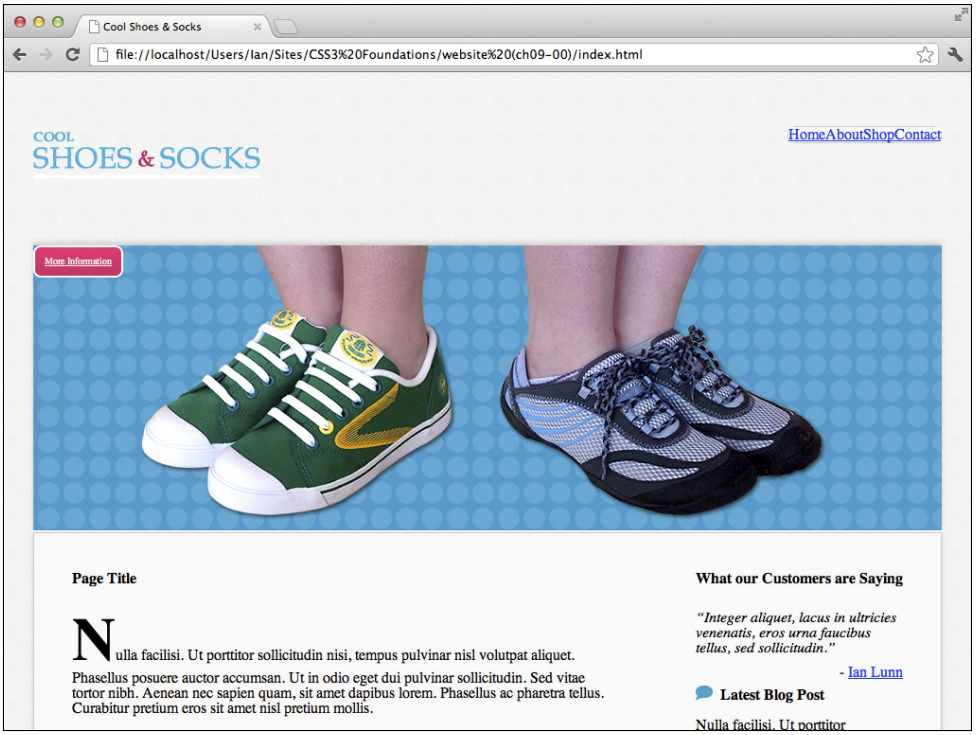


FIGURE 9-13 The More Information button is now positioned relative to its containing the element.

Unfortunately, vertically centering an element via CSS isn't as straightforward as it could be. There *is* a `vertical-align` property, but it doesn't work in a way you may initially expect—it caught me out to begin with. Continue with the `position` property for now and come back to the `vertical-align` property later in this chapter along with a hack to make the showcase button vertically centered.

Make the footer stretch across the entire page (regardless of the page size and disrespecting the containing <body> element):

1. In styles.css, add the following declarations to the `#footer` rule set:

```
left: 0;
position: absolute;
width: 100%;
```

2. Save styles.css.

Although the footer is contained with the <body> element, because the footer has no containing element with a position declaration, it is positioned relative to the <html> element.

fixed

The final value for the position property is fixed. As with absolute elements, a fixed element is taken out of a document's flow and does not affect the layout of elements around it. Unlike absolute, however, a fixed element is positioned relative to the viewport (the browser window) and does not move when the page is scrolled.

For this example, add a banner image to the bottom of the page:

1. In index.html, below the closing </footer> and above the closing </body> elements, add the following HTML:

```
<img class="banner-ad" src="images/banner-25percent.png"
  alt="25% Off All Purchases" />
```

2. Save index.html
3. In styles.css, add the following declaration:

```
.banner-ad {
    bottom: 0;
    position: fixed;
    right: 0;
}
```

4. Save styles.css.

As shown in Figure 9-14, the 25% off banner remains fixed to the bottom right of the browser window. No matter whether or not the user scrolls, that element remains fixed in that position, so it's always on display—sort of like you had the power to glue an element to the inside of the user's screen!

Unfortunately, Internet Explorer 6 doesn't support position: fixed;, and furthermore, very few browsers on mobile devices support position: fixed; due to technical performance issues. Fixed elements tend to be used for noncritical aspects of a page, such as notices and adverts, so for browsers that don't support fixed elements, they can be hidden or modified to be displayed in a different way using styles specific to those nonsupporting browsers; this topic is covered in Chapter 15. Because the 25% banner used for Cool Shoes & Socks is an image—that can only be square in proportion—the transparent section of the image prevents a user from clicking anything below that invisible area, so for smaller screen sizes where that area may cover any part of the page that may be interacted with, it is necessary to hide that banner anyway.

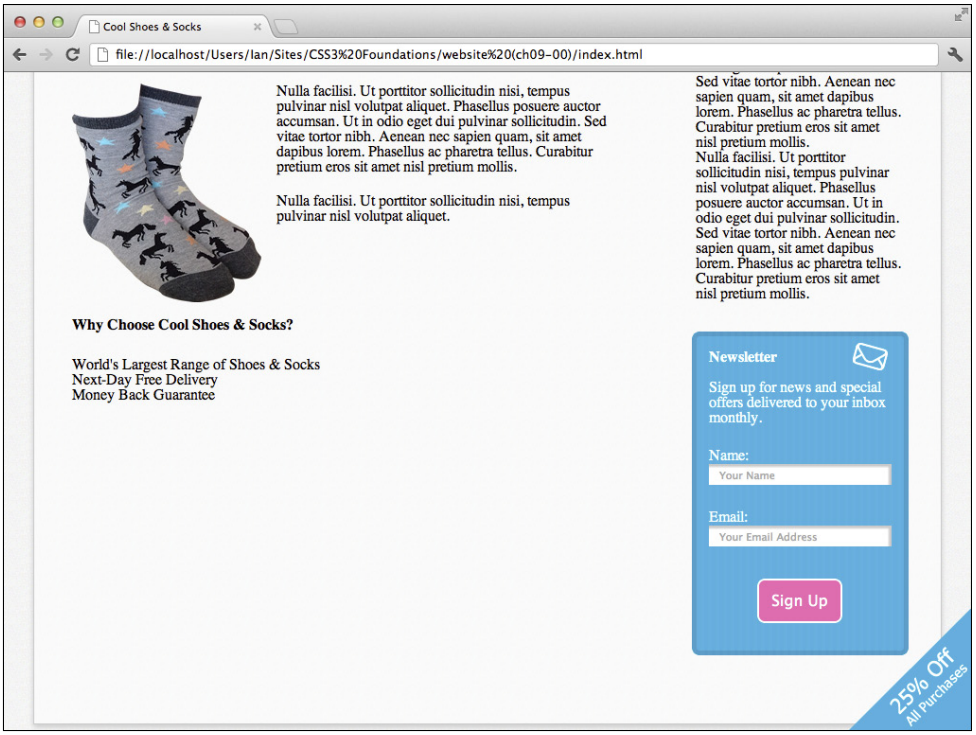


FIGURE 9-14 The banner ad fixed to the bottom and right of the viewport.

Code Challenge: Change the Position of the Quotes Around the Customer Testimonials Without Affecting the Flow

In styles.css, do the following:

1. Add a new rule set for `blockquote` with the declaration `position: relative;`.
2. Add the declarations `position: absolute; left: -20px; top: -10px;` to the `blockquote p:before` rule set.
3. Add the declarations `position: absolute; right: 0; bottom: -20px;` to the `blockquote p:after` rule set.

Project files update (ch09-01): If you haven't followed the previous instructions and are comfortable working from here onward or would like to reference the project files up to this point, you can download them from `www.wiley.com/go/treehouse/css3foundations`.

Using display, position, and z-index to Create a Drop-Down Menu

Now that you have the power of `display` and `property` tucked into your CSS utility belt, you can flex this knowledge and create a drop-down menu.

1. In index.html, below this HTML:

   ```
   <a href="#" title="Visit our shop">Shop</a>
   ```

   Add the following HTML:

   ```
   <ul>
       <li><a href="#" title="">Trainers</a>
           <ul>
               <li><a href="#" title="">Womens</a></li>
               <li><a href="#" title="">Mens</a></li>
           </ul>
       </li>
       <li><a href="#" title="">Shoes</a>
           <ul>
               <li><a href="#" title="">Womens</a></li>
               <li><a href="#" title="">Mens</a></li>
           </ul>
       </li>
       <li><a href="#" title="">Slippers</a></li>
       <li><a href="#" title="">Socks</a></li>
   </ul>
   ```

2. Save index.html

   This HTML will add a submenu to the Shop link. Now to style it:

3. In styles.css, below the `#header nav > ul > li` rule set, add the following rule sets:

   ```
   #header nav ul > li:hover {
       background: #f5f5f5;
   }
   #header nav ul ul > li:hover {
       background: white;
   }
   ```

   These rules make each navigation link of the top-level list a gray color and any link from second-level lists onward white whenever hovered over.

4. Below the `#header nav ul ul > li:hover` rule set, add the following rule set:

```
#header nav ul li a {
    color: black;
    display: block;
    padding: 10px 20px;
}
```

Here, you make the links within the list items `display: block;`. When you make the `<a>` elements block (remembering they're inline by default), they can now take on dimensional properties and, as such, become bigger than if they were left as inline.

Take a breather here for a minute. As shown in Figure 9-15, you now have a partially styled submenu but it's more of a drop-up list at the moment! This is caused because the child lists of the top-level lists are in flow and affecting the elements around them. The answer is to take them out of flow.

FIGURE 9-15 The second-level unordered lists, which need to be positioned differently.

5. Add the following rule set below the `#header nav ul li a` rule set:

```
#header nav ul ul {
    /*style all lists below the top-level list*/
    background: #f5f5f5;
    border: 1px solid #ccc;
    border-bottom-left-radius: 8px;
    border-bottom-right-radius: 8px;
    border-top: none;
    padding: 20px 0;
    position: absolute;
    min-width: 140px;
}
```

This rule set styles any unordered list below the top-level list, and because the drop-down menu consists of quite a few rule sets, a comment is added in this rule set to

remind you or anyone else in the future what this rule set selects—a good practice, particularly when selectors start to get complex.

Along with the borders, padding, and a minimum width, you also give the drop-down menus an absolute position, moving them out of flow so they no longer affect the elements around them. However, as shown in Figure 9-16, although the drop-down menu kind of looks as though it's in the right position, it's hidden behind the product showcase.

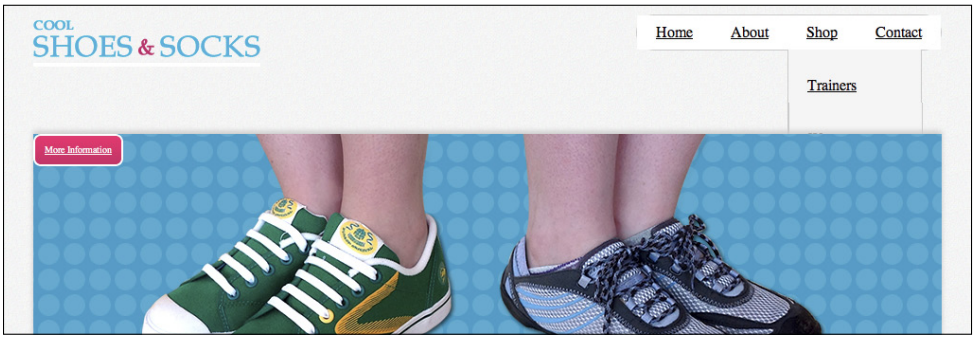


FIGURE 9-16 The drop-down menu out of flow but hidden behind the product showcase.

For now, fix this and then come back to the explanation after the drop-down menu is working.

6. In the same rule set, add the following declaration:

```
z-index: 10;
```

The drop-down menu now appears above the product showcase, although the third level of links isn't quite styled correctly.

7. Below the `#header nav ul ul` rule set, add a new rule set:

```
#header nav ul ul ul {
    /*style all lists below the second-level list*/
    border: 1px solid #ccc;
    border-right: none;
    border-bottom-left-radius: 0;
    border-bottom-right-radius: 0;
    padding: 0;
}
```

The links Trainers and Shoes both have their own drop-down menus (a third-level list of links). As shown in Figure 9-17, these menus appear over their parent menu at present; ideally, it would be good to have those menus appear to the left of their respective parent links.

FIGURE 9-17 The third-level menus appear over their second-level parents.

8. Add the following declarations to the `#header nav ul ul ul` rule set:

```
left: -100%;
top: -1px;
```

Now the menus appear to the left perfectly, *but* each menu sits in the same position at the top. These third-level lists inherit the declaration `position: absolute;` from their parent lists. Remember that the position of an absolute element is relative to its parent, so at the moment, the declaration `top: -1px;` places *all* third-level lists 1 pixel above the place where the second-level list is positioned. Having each third-level list start in the same vertical position as the second-level link it belongs to would make more sense.

9. Below the rule set `#header nav ul ul`, add a new rule set:

```
#header nav ul ul li {
    position: relative;
}
```

Now the third-level menus are positioned relative to their respective second-level link instead of the overall second-level list. The `top` value of `-1px` accounts for the 1 pixel border.

This wouldn't be much of a drop-down menu if it constantly sat there with every link showing; after all, the point of a drop-down menu is to make better use of space, hiding links until the user needs to use them.

10. Add a declaration to the `#header nav ul ul` rule set:

```
visibility: hidden;
```

11. Below the rule set `#header nav ul ul ul`, add a new rule set:

```
#header nav li:hover > ul{
    /*show the menu of the list item being hovered over*/
    visibility: visible;
}
```

Here, you hide any list of links below the top level and allow them to become visible only when the user hovers over their respective parent link.

Finally, for the drop-down menu, make the corners of the top-level list rounded.

12. Below the rule set `#header nav > ul > li`, add two new rule sets:

```
#header nav > ul > li:first-child {
    border-top-left-radius: 20px;
    border-bottom-left-radius: 20px;
}

#header nav > ul > li:last-child {
    border-top-right-radius: 20px;
    border-bottom-right-radius: 20px;
}
```

13. Save styles.css.

Congratulations! The drop-down menu is now fully functional, as was shown in Figure 9-2. Before you move on, though, you need to understand what that `z-index` declaration does.

z-index

Initial value: `auto` | Inherited: No | Applies to: Positioned elements | CSS2.1

Browser support: IE 4+, Firefox 1+, Chrome 1+, Opera 4+, Safari 1+

When giving elements a `position` declaration, you might find that sometimes elements overlap one another. The problem you saw in Figure 9-16, where the drop-down menu appears behind the product showcase, is that both of these elements have a defined position and neither one is in flow, making them visually clash—meaning one sits behind the other. When elements are taken out of flow, a page becomes three-dimensional, and by using `z-index`, you can control the stacking order of those elements.

The `z-index`, by default, is `auto`, meaning the stacking order is `0`. By giving the `z-index` property a number as a value, you determine how high up the stack an element is.

Before you gave the drop-down menu a `z-index`, it and the product showcase had a default stack order of `0`. If two elements are both at the same position in the stack order, their positioned is determined by the order of the HTML; elements that appear later in the HTML are given precedence. By giving the drop-down menu a `z-index` of `10`, you move it above the product showcase.

Why 10? The navigation is the most important part of the website that has a position declaration, so at no point should it be hidden below another element. There aren't 10 positioned elements on the page, so 10 can be assumed as being the highest level. If you want to be extra safe though, you can give it a `z-index` of `10000` or even `1000000`. The CSS specification doesn't determine a maximum number for `z-index` but some very dedicated people have done tests to determine this number for each browser. You'll be pleased to know that every browser in use today can support stack orders higher than 2 billion (for more information, visit `www.softwareas.com/whats-the-maximum-z-index`). I hope that's enough for you! Of course, if you want to make your CSS maintainable now and in the future, it's best to be sensible with `z-index` values.

Code Challenge: Apply z-index to Other Elements

In styles.css, do the following:

1. Add the declaration `z-index: 5;` to the `.showcase .button` rule set.
2. Add the declaration `z-index: 9;` to the `.banner-ad` rule set.

Project files update (ch09-02): If you haven't followed the previous instructions and are comfortable working from here onward or would like to reference the project files up to this point, you can download them from `www.wiley.com/go/treehouse/css3foundations`.

vertical-align and Vertical Centering Techniques

As explained early in this chapter, achieving vertical alignment via CSS is unfortunately not as straightforward as you might expect. Although the `vertical-align` property seems like a perfect fit for the job, it works only for inline-level and table-cell elements. Let's look at the `vertical-align` property and then use some "hacks" to make block-level elements vertically center.

vertical-align

Initial value: `baseline` | Inherited: No | Applies to: inline-level and table-cell elements | CSS2.1

Browser support: IE 4+, Firefox 1+, Chrome 1+, Opera 4+, Safari 1+

The small speech bubble image that is included in the "Latest Blog Post" title is inline-level (remember images are inline by default):

```
<h3><img src="images/bg-blog.png" alt="Latest Blog Post" />Latest
  Blog Post</h3>
```

The initial value for `vertical-align` is `baseline`, meaning the bottom of the speech bubble image vertically aligns with the baseline of the title, as shown in Figure 9-18.

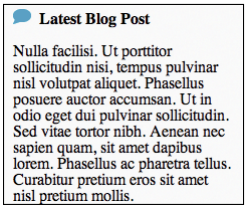


FIGURE 9-18 The speech bubble vertically aligned to the baseline of the text.

`vertical-align` also accepts the following values, as demonstrated in Figure 9-19:

- `middle`—Aligns the vertical midpoint of the inline element with the baseline and half of the x-height (the height of the lowercase *x* character). In Figure 9-19, note that because the speech bubble image is 15 px tall, the browser doesn't quite perfectly center the image.
- `sub`—Lowers the baseline of the inline element to meet the position for subscripts `<sub>`.

- `super`—Raises the baseline of the inline element to meet the position for superscripts `<sup>`.
- `text-top`—Aligns the top of the inline element with the top of the text.
- `text-bottom`—Aligns the bottom of the inline element with the bottom of the text.

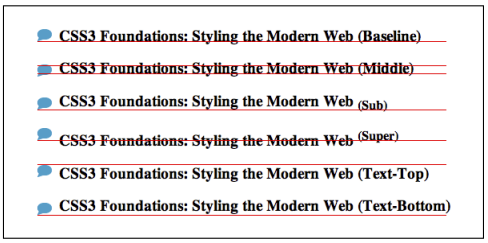


FIGURE 9-19 The speech bubble when vertically aligned to the baseline, subscript, superscript, text-top, and text-bottom positions.

This vertical alignment demonstration page can be found in the project files ch09-02, named vertical-align-property-demo.html

You can give two other values to `vertical-align`: `top` and `bottom`. Where `text-top` and `text-bottom` align an inline-level element relative to text, `top` and `bottom` align an inline-level element with the top or bottom of the parent's content area.

In Figure 9-20, I demonstrated this technique by adding an inline image of those awesome horse socks with a `vertical-alignment` of `text-bottom` (aligning its bottom with the bottom of the text). In the first example, I gave the speech bubble image a vertical align of `text-top`, which aligns its top with the top of the text. However, because the image of the socks is much taller than the rest of the contents, the parent's content area is bigger too. If you want to position the speech bubble relative to the top of the content area instead of the text, you can specify a vertical alignment of `top`.

If none of those values take your fancy and you want a little more control, you can also specify a unit of length or a percentage where `0` represents the same as `baseline`. A positive number raises the inline-level or table-cell element above the baseline, and a negative number lowers it.

FIGURE 9-20 The difference between vertically aligning the speech bubble to the text-top and top.

Vertical Centering Techniques

It's both disappointing and initially misleading that the `vertical-align` property can be used only on inline-level and table-cell elements. The `position: center;` declaration introduced in the CSS Level 3 Positioned layout module found at `www.w3.org/TR/css3-positioning/` will, I hope, see the light of day and make vertical alignment easier in the future. For now, though, let's go back to positioning the showcase button and cover a few different techniques for making it vertically centered.

The following techniques all have their own pros and cons, and unfortunately, there isn't a perfect solution. Therefore, when you are vertically aligning elements in the future, use your best judgment.

To give you an understanding of each technique, steps are provided that ask you to make changes to the project files, then undo those changes to try the next technique. If you want to follow along, please do. However, if you just want to understand the technique that the Cool Shoes & Socks page will use, please follow the instructions for the third technique only.

The Fake Table Cells Technique

If you can't go to the party, you can bring the party to you. You can use `vertical-align` only on inline-level and table-cell elements, so cheat and use `display: table-cell;` as follows:

1. In styles.css, add the declaration `display: table` to the `.showcase` rule set:

```
display: table;
```

2. Remove the following declarations from the `.showcase .button` rule set:

```
position: absolute;
top: 0;
```

3. Add a `width` declaration to the `.showcase` img rule set:

```
width: 100%;
```

4. Add a new rule set below the `.showcase .button` rule set:

```
.button-cell {
    display: table-cell;
    vertical-align: middle;
    width: 148px;
}
```

5. Save styles.css.

   When you use `display: table-cell;` on an element, that element must be contained in a table (or an element mimicking a table with the declaration `display: table;`). Because the showcase `<div class="showcase">` contains the showcase button, *that* is given the declaration `display: table;`. If the `.showcase .button` rule set were to be given the declaration `display: table-cell;`, the button would become 100% tall because cells stretch to fill the entirety of the containing table. This is the downfall to using fake table cells. Although this approach emulates vertical alignment beautifully, it requires an additional element to be wrapped around the one you want to vertically align to prevent that element from becoming 100% tall.

6. In index.html, modify:

```
<a class="button purchase" href="#" title="Purchase product">
    More Information
</a>
```

   Make sure it is contained in a `<div>` with the class `button-cell`:

```
<div class="button-cell">
    <a class="button purchase" href="#" title="Purchase
  product">
        More Information
</a>
</div>
```

7. Save index.html.

The other problem, as shown in Figure 9-21, is that elements within a table can only sit aside each other, meaning the button isn't visually over the top of the showcase images.

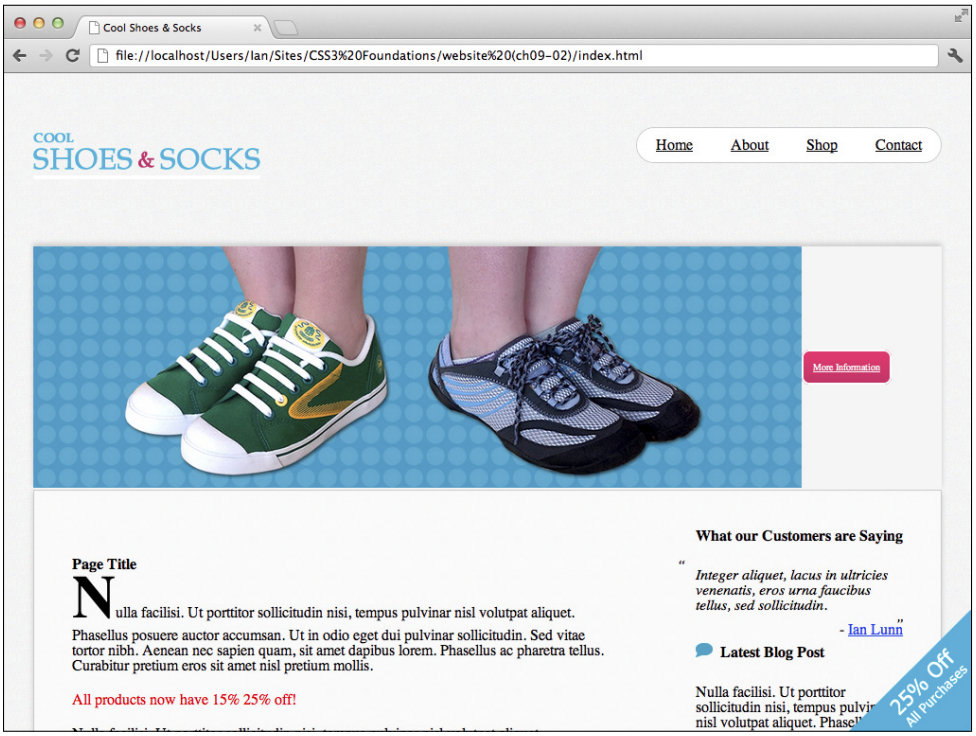


FIGURE 9-21 The showcase button vertically centered.

Using fake table cells offers the most robust vertical alignment. When you are increasing text size in browsers such as Mozilla Firefox, the element stays vertically centered, but it means using more HTML and, in the showcase example, prevents elements from appearing on top of each other. Furthermore, `display: table-cell;` doesn't work in Internet Explorer versions 6 and 7.

The Stretched Element Technique

Stretching an element positions the element absolutely in relation to its containing parent and allows for both vertical and horizontal alignment.

1. Undo the changes made for the Fake Table Cells technique both in index.html and styles.css.
2. In styles.css, add the following declarations to the `.showcase .button` rule set:

```
bottom: 0;
left: 0;
```

```
right: 0;
height: 16px;
margin: auto;
width: 148px;
```

As shown in Figure 9-22, this technique is much easier than the Fake Table Cells technique and works in a way more desirable for the product showcase.

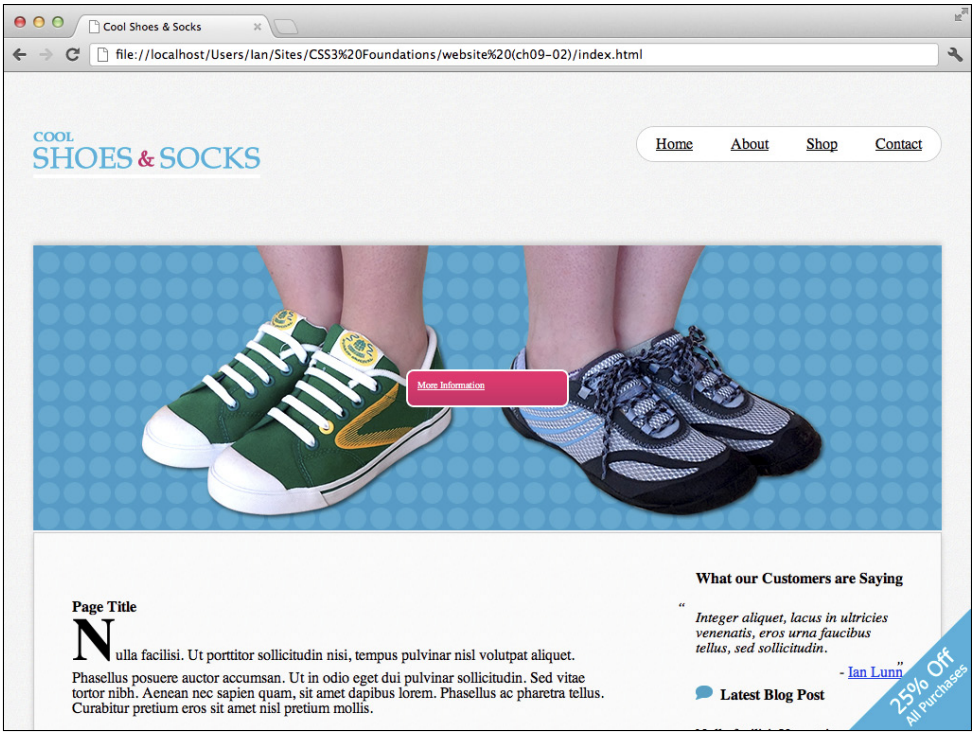


FIGURE 9-22 The showcase button both vertically and horizontally aligned.

3. Because the button should be positioned over to the right, you can remove the `left` declaration and modify the `right` declaration:

```
right: 2.1875em;
```

4. Save styles.css.

The Stretched Element technique is another useful method, particularly because it's so easy and can horizontally center an element, too. However, when you are increasing the size of text in a browser such as Firefox, as shown in Figure 9-23, although the button always stays vertically centered, the text breaks out of the button. Because the purpose of resizing text is to make a page more readable, the user may be annoyed to find this happens to the text at a larger size.

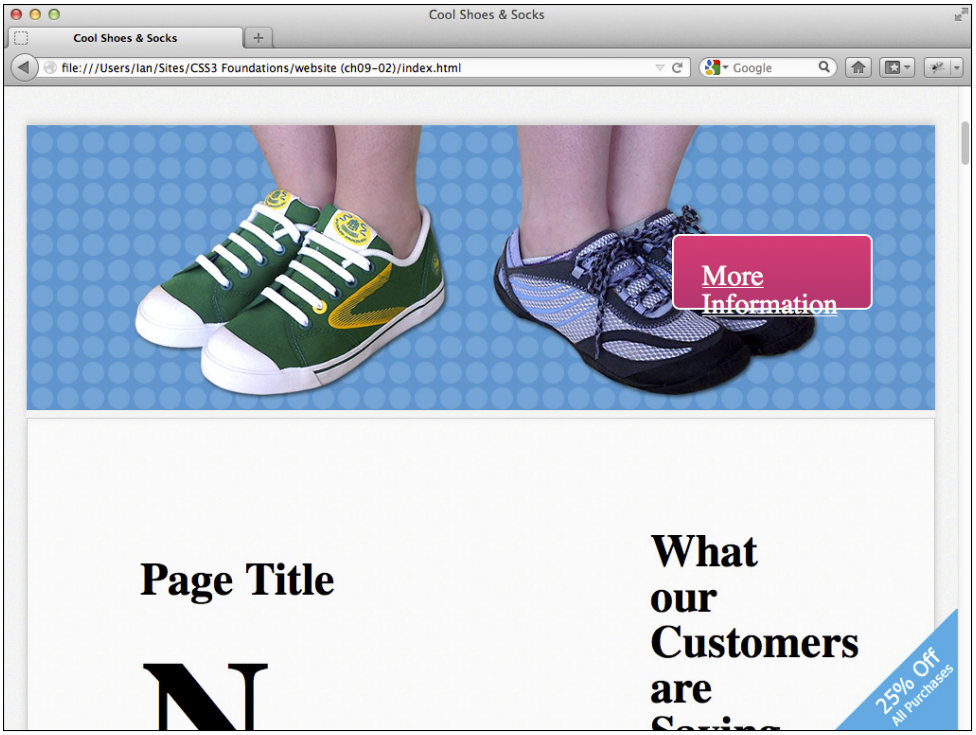


FIGURE 9-23 The showcase button vertically centered but with a larger font size in Mozilla Firefox.

The 50% Top Minus Half the Elements Height Technique

Another solution, and the final solution used for the Cool Shoes & Socks page, is to set the top position of the showcase button to 50% and then pull up the button by half of its own height to make it vertically centered:

1. Undo the changes made for the Stretched Element technique.
2. In styles.css, change the top declaration in the `.showcase .button` rule set:

```
top: 50%;
```

3. In the same rule set, add the following declarations:

```
font-size: 1.6em;
right: 2.1875em;
margin-top: -26px;
```

4. Save styles.css.

As shown in Figure 9-24, the button is now vertically aligned. When you give the button a top position of 50%, the top of the button is placed in the vertical center of the product showcase. However, this isn't a true vertical alignment. For it to be perfectly centered, the vertical center of the button needs to be in the vertical center of the showcase. To achieve

this result, you specify a `margin-top` of `-26px`, half of the element's overall height, which pulls up the button by 50%.

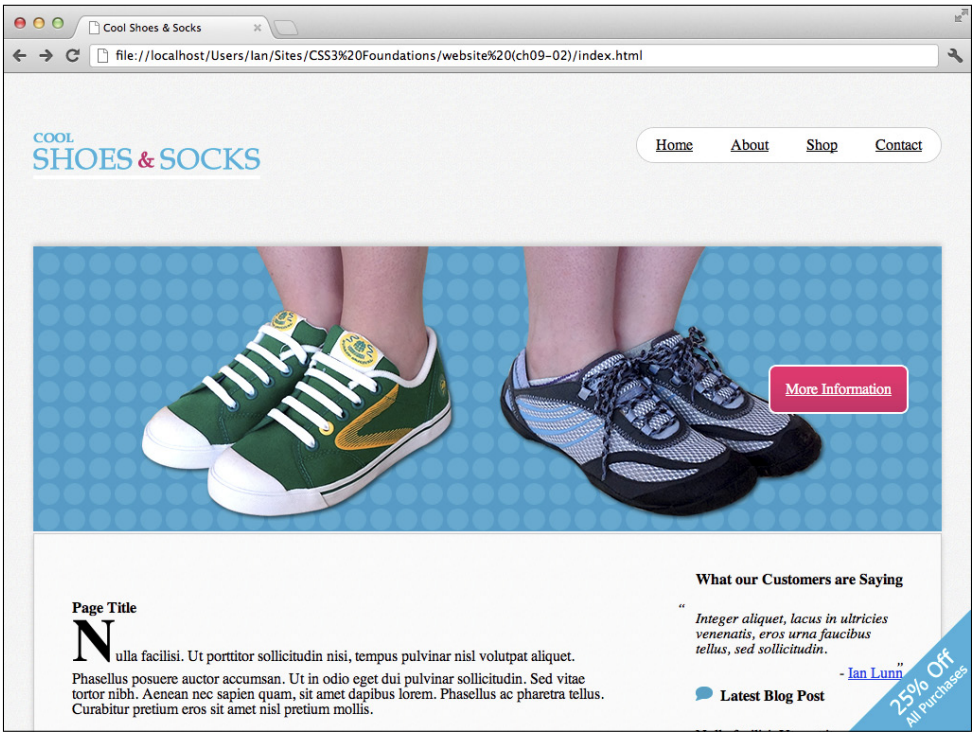


FIGURE 9-24 The showcase button is now vertically aligned.

Yes, this breaks the rule I've mentioned several times already: You should try not to declare or expect an element to be a specific height because in some browsers, users can change the font size to their liking.

Because the showcase button doesn't have a specified height, though, it stretches with the size of the text. So with this technique, unlike the Stretched Element technique, the text doesn't break out of its containing box.

This does mean, however, that the button loses its vertical alignment if the text *is* resized, as shown in Figure 9-25, but it means the text is more readable—better serving the purpose of a browser's text resizing feature. Putting readability before aesthetics is a good priority.

Although the Cool Shoes & Socks page uses the 50% Top Minus Half the Elements Height technique, all three vertical alignment techniques have their uses. Which technique you use really depends on the scenario. On the web, you can find other vertical alignment techniques, such as at `www.css-tricks.com/centering-in-the-unknown/`, which also have their own pros and cons.

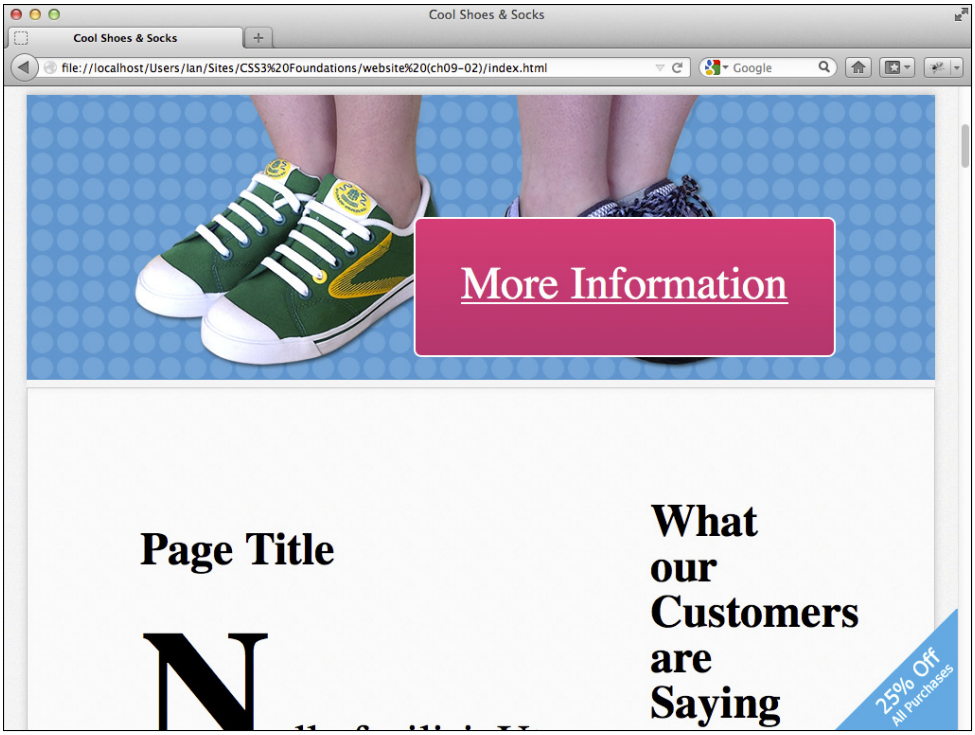


FIGURE 9-25 The web page in Firefox with the default text size increased.

overflow

Initial value: `visible` | Inherited: No | Applies to: block elements | CSS2.1

Browser support: IE4+, Firefox 1+, Chrome 1+, Opera 7+, Safari 1+

When creating a web page, sometimes you find that content you want to place in a containing element is bigger than its container. Throughout *CSS3 Foundations*, I have mentioned that you shouldn't restrict an element to a specific height because in some browsers, a user can control the size of text and content may grow bigger than the height of that element. If you do have content bigger than its containing element or you absolutely have to give an element a specific height (clients and bosses can be pushy!), you can use the `overflow` property to better control how that overflowing content is displayed—or not displayed, as the case may be.

Go ahead and restrict the height of the "Latest Blog Post" element:

1. In styles.css, add a new rule set for the `.post` element, under the `.testimonials` rule set:

```
.post {
    height: 200px;
}
```

In Figure 9-26, I also gave the `<aside class="post">` element a red border along with its height of 200px to demonstrate the boundaries of that element. As you can see, with no specified `overflow` declaration, the overflowing content is visible. Note that the containing element affects the document flow, but the overflowing content does not. It all looks a bit messy and unreadable.

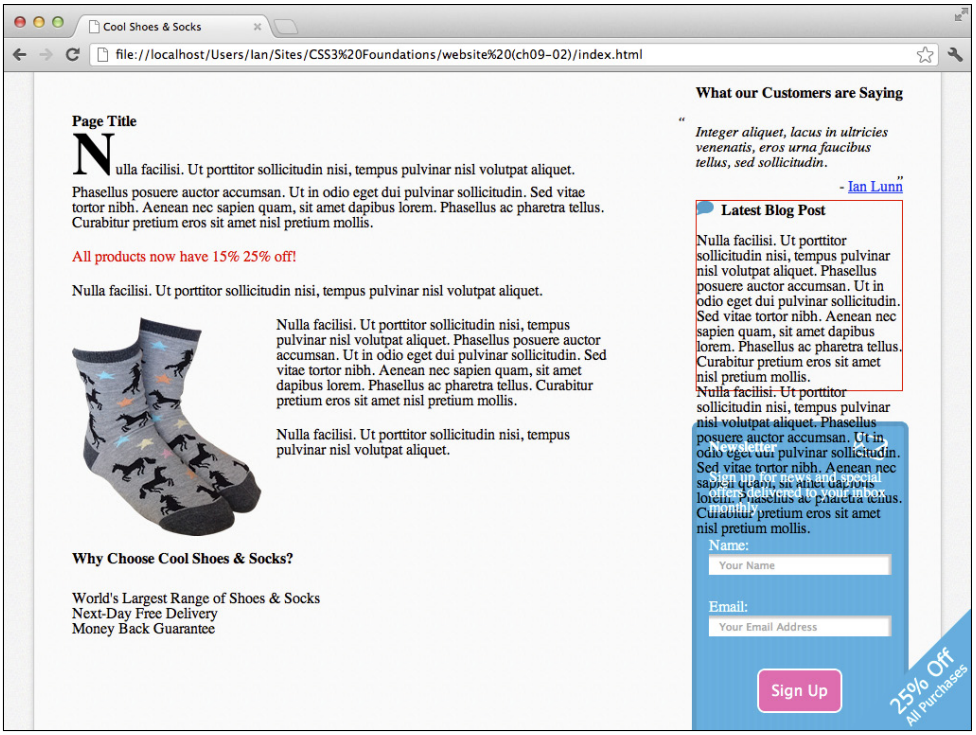


FIGURE 9-26 The "Latest Blog Post" box with overflowing content.

If you feel that the content isn't very important, you can simply hide the overflow by giving it an overflow declaration of `hidden`.

2. Add an `overflow` declaration:

```
overflow: hidden;
```

As shown in Figure 9-27, the overflowing content is now hidden, *but* the line of text at the bottom is cut off. Because it's text being cut off and not something less forgiving,

such as an image, hiding text in this way isn't ideal. It may lead the users to becoming frustrated because they know they are missing out on some content they can't get to.

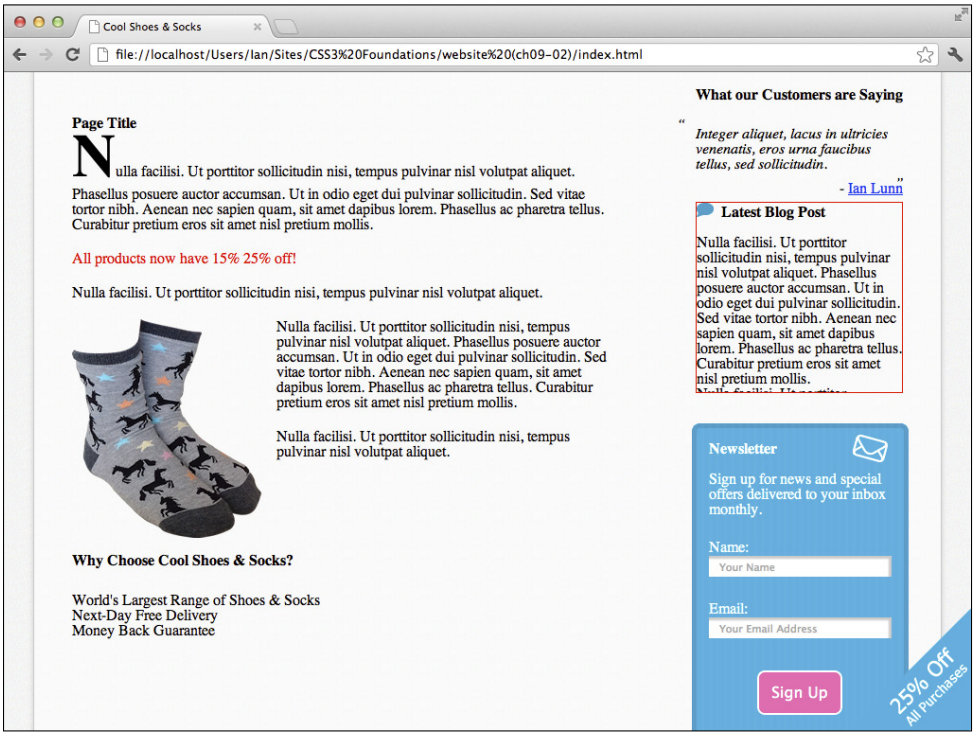


FIGURE 9-27 The "Latest Blog Post" with its overflowing content hidden.

This is where `scroll` value for `overflow` comes in handy.

3. Change the hidden value of the `overflow` declaration to `scroll`:

```
overflow: scroll;
```

Figure 9-28 shows how the "Latest Blog Post" box is displayed in Mozilla Firefox on an Apple Mac and a Windows PC; now the box has a scroll bar of its own. Firefox, Opera, and Internet Explorer show both vertical and horizontal scroll bars for an `overflow: scroll;` element, even if the content of that element overflows only on one axis. Chrome and Safari are a little more intelligent and show a scroll bar only when it is needed. To work around this and make the experience more consistent across browsers, you can use the properties `overflow-x` and `overflow-y`.

Because the `.post` element has a specified height with the default width `auto`, it never overflows horizontally, only vertically. So, to make Firefox, Opera, and Internet Explorer show only a vertical scroll bar, you can move on to step 4.

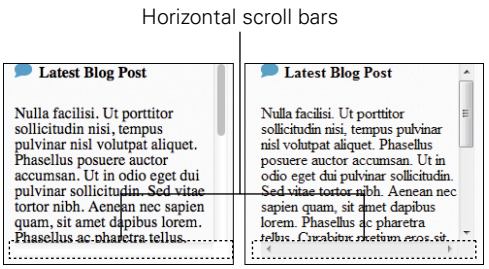


FIGURE 9-28 The "Latest Blog Post" with a scroll bar to allow the user to see overflowing content. Viewed in Firefox on an Apple Mac to the left and Firefox on a Windows PC to the right.

4. Change the property `overflow` to `overflow-y`:

```
overflow-y: scroll;
```

5. Save styles.css.

Now, as you can see in Figure 9-29, all browsers show only the vertical scroll bar.

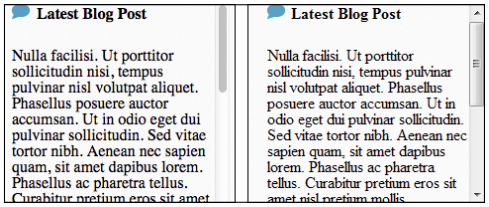


FIGURE 9-29 The "Latest Blog Post" with a vertical scroll bar only. Viewed in Firefox on an Apple Mac to the left and Firefox on a Windows PC to the right.

Summary

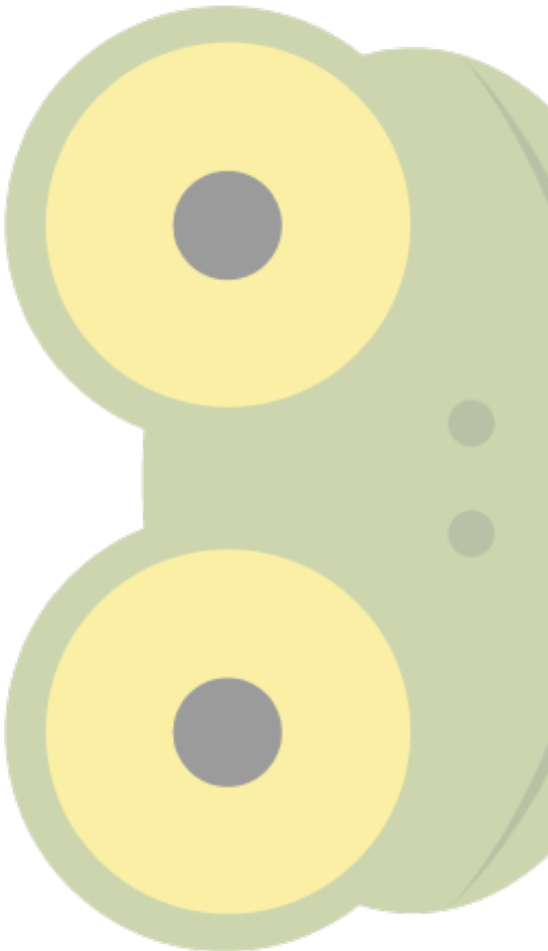
You've finished the main layout of the Cool Shoes & Socks page. Looks pretty cool, don't you think? Of course, it functions well, too. The page now has quite a lot of navigation links, but they're all tucked away nicely in a drop-down menu, which means users can find where they want to navigate quickly without being overwhelmed by choice.

If you view the page in a browser other than the one you're using to create Cool Shoes & Socks, almost certainly you will find inconsistencies and broken bits. Breathe a sigh of relief—that's to be expected. In Chapters 15 and 16, you learn to make Cool Shoes & Socks cross browser compatible and modify its layout for when it is viewed on mobile devices.

In the next chapter, you learn to change the font and font styles, characteristics that can really make Cool Shoes & Socks unique.

part 4

Typography



chapter ten

Changing the Font

FONTS ARE AN important part of the design aesthetic. Although text—whether on the web or in print—conveys a particular message, the choice of font can greatly reinforce that message, representing voice, authority, friendliness, competence, and so on.

Until recently, the range of fonts available for web design was rather lacking. Although a quick search on the web shows hundreds of thousands of fonts are available to download and use, in reality, not every device has the font you want to use for your web page. In fact, only a handful of fonts that you can use are shared across the majority of devices. These fonts are known as "web safe" because you can use them safe in the knowledge that almost everyone can see them.

When features of CSS3 were first implemented into browsers, however, the technology became available to allow for displaying a font that wasn't installed on a computer. This means you can now choose from a much larger selection of fonts for your web page.

In this chapter, you learn how to apply different fonts to the Cool Shoes & Socks page, using third-party services such as Google Web Fonts to access a wider range of fonts.

Project files update (ch10-00): If you haven't followed the previous instructions and are comfortable working from here onward or would like to reference the project files up to this point, you can download them from `www.wiley.com/go/treehouse/css3foundations`.

Choosing a Web Safe Font Using font-family and Font Stacks

Web safe fonts are those fonts that are likely to be installed on the majority of devices accessing the web. Between computers with varying operating systems, you can find around 50 web safe fonts, but with mobile devices now accessing the web too, the number of web safe fonts drops to just a handful. Thankfully, to work around this number, you can specify which font you would ideally like a user to see a web page presented in and then add more fonts after that in a comma-separated list—known as a font stack—to act as fallbacks in case your chosen font is not present.

If a device has no fonts whatsoever—which is unlikely unless somebody decided to just delete them all—the CSS specification defines five generic font names to act as fallbacks: serif, sans-serif, cursive, fantasy, and monospace, as shown in Figure 10-1. These fallbacks don't necessarily represent a specific font but should express particular characteristics.

- **Serif**—Characters tend to have finishing strokes, flared or tapering ends, or actual serifed endings.
- **Sans-serif**—Characters tend to have stroke endings that are plain.
- **Cursive**—Characters have either joining strokes or other cursive characteristics, resulting in a handwritten look.
- **Fantasy**—Characters are primarily decorative but still contain representations of characters.
- **Monospace**—All characters have the same fixed width, similar to a manual typewriter and often used to set samples of computer code.

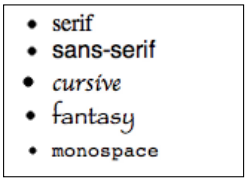


FIGURE 10-1 The five generic fonts when viewed in Google Chrome.

This generic font demonstration page can be found in the project files ch10-00, named generic-fonts-demo.html.

font-family

Initial value: browser dependent | Inherited: Yes | Applies to: All | CSS2.1
Browser support: IE 3+, Firefox 1+, Chrome 1+, Opera 3.5+, Safari 1+

The `font-family` property enables you to specify a font or list of fonts to be applied to an element.

1. In styles.css, find the rule set for `body` and add the following:

   ```
   font-family: serif;
   ```

   The initial value for `font-family` is dependent on the browser, but for all desktop browsers, each has chosen serif as its default. As a result, you don't yet see a difference to the page, but nonetheless you have specified a fallback font for the entire page, which is always wise. Wise...but kind of boring, right? So you can change this font declaration into a font stack.

2. Change the `font-family` declaration to the following:

   ```
   font-family: Georgia, serif;
   ```

3. Save styles.css.

As shown in Figure 10-2, the Cool Shoes & Socks page is now entirely set in Georgia. When a browser comes to render this declaration, it first tries to set the font as Georgia, but if the device doesn't have it, the browser then falls back to serif. This is known as a *font stack*. A font stack can consist of a long list of fonts, starting with your most desired, working down through fonts that are closer to being web safe, finally reaching one of the generic fonts that best represents the characteristics of your desired font. Georgia, however, is web safe, so it is almost certainly installed on all devices.

Of course, web safe fonts have been used time and time again, so try using the CSS3 `@font-face` rule to use a font that can be displayed without users initially having it on their device.

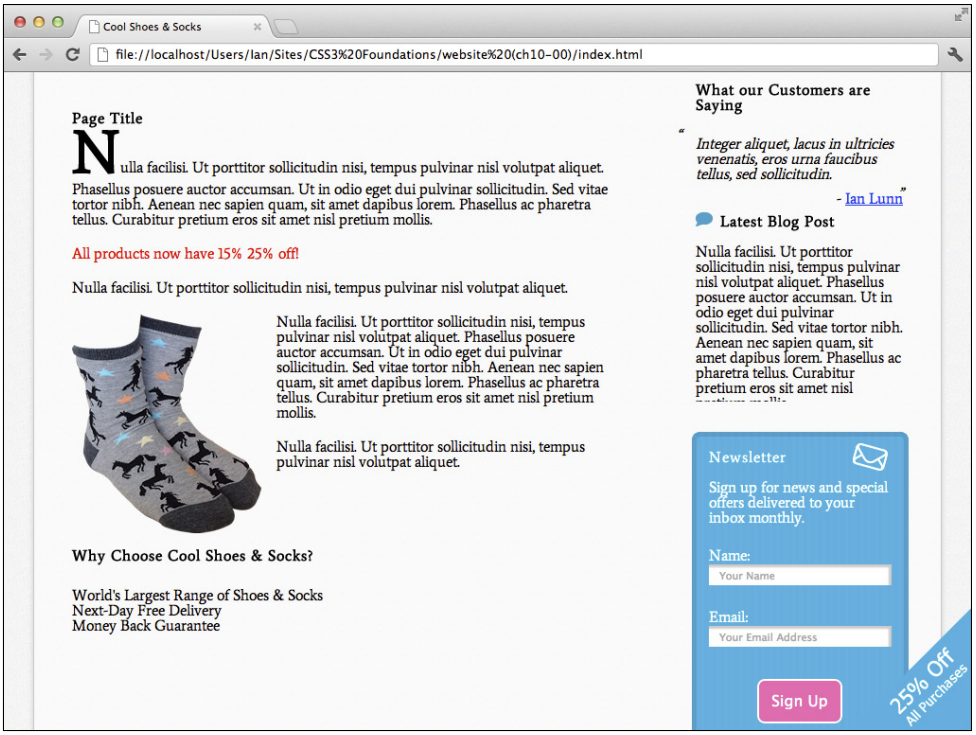


FIGURE 10-2 The Cool Shoes & Socks page with a font-family of Georgia.

Applying Fonts Using @font-face

Browser support: IE 4+, Firefox 3.5+, Chrome 4+, Opera 10+, Safari 3.1+

`@font-face` is a rule set of its own rather than a property, which allows you to specify a font to be downloaded from a particular source. Although `@font-face` appears only in the CSS3 Fonts module (`www.w3.org/TR/css3-fonts/`), it was actually a feature first implemented by Microsoft in Internet Explorer 4—yes, another one of those situations in which Microsoft did as it pleased. This is good news, though; it means every browser in use today supports `@font-face`, *but* it isn't without its caveats.

Fonts come in the following formats:

- .ttf (TrueType Font)
- .otf (OpenType Font)
- .eot (Embedded OpenType)
- .woff (Web Open Font Format)
- .svg (Scalable Vector Graphics)

When Internet Explorer 4 implemented `@font-face`, it supported only the .eot format, and the situation remained that way up to Internet Explorer 8. Because the .eot format is proprietary, belonging to Microsoft, no other browser supports it. The only format to be supported by all other browsers (including Internet Explorer 9) is .woff. So, when using `@font-face`, you should aim to have .eot and .woff formats of the same font.

If you have only one font type, numerous tools available online take that one format and convert it into the rest for you. My personal recommendation is Font Squirrel's @font-face Generator, found at `www.fontsquirrel.com/fontface/generator`.

> Converting fonts to a different format may be against the license for a font, so you should always be certain a font license allows for this behavior before going ahead. More on font licenses in a moment.

The Cool Shoes & Socks project files contain an .eot and .woff font called Average. Tell the browser to download this font for later use:

1. In styles.css, above the `body` rule set, add the following:

```
@font-face {
    font-family: Average;
    src: url("../fonts/Average-Regular.eot");
    src: local("Average"),
        url("../fonts/Average-Regular.woff");
}
```

   Because this rule set doesn't have a selector, it only lets the browser know the name of the font and where it is located, via the `src` declaration. Due to Internet Explorer 6, 7, and 8 supporting only .eot font formats; you specify two `src` properties. Internet Explorer 6, 7, and 8 read and apply the first property, but because they don't understand the syntax of the second, they ignore it.

   The second `src` property consists of two functions: `local()` and `url()`. By placing the `local()` function first, you tell the browser (those which aren't Internet Explorer versions below 9) to check to see whether the device already has the font saved locally, and if not, to download it from the source specified in the `url()`.

   At a minimum, the `@font-face` rule must include `font-family` and `src` declarations; otherwise, it is ignored.

   Now to use this font, do the following:

2. In the `body` rule set, modify the `font-family` declaration:

```
font-family: Average, Georgia, serif;
```

3. Save styles.css.

When a browser reads the `Average` font as the first font in the list, it renders the text in that font. However, if that font isn't available (you've made a good job of making sure it is via the `@font-face` rule, so it should be), the browser moves onto the next font `Georgia` and tries to render the text using that font.

Along with the `font-family` and `src` that are included in a `@font-face` rule set, you can also give the font default styles using properties such as `font-size` and `font-weight`, which are covered in the next chapter.

To specify multiple fonts per stylesheet, you simply add another `@font-face` rule. The Cool Shoes & Socks page uses two different fonts using the `@font-face` rule, but instead of getting the browser to download these fonts from the server, you actually use a third-party font service instead.

Font Licenses and Third-Party Font Services

Although `@font-face` offers a way to use fonts that may not be installed on a user's computer, that doesn't mean you can start using every font you have on your own computer. When you use `@font-face`, a user's computer has to download a copy of that font, meaning *you* are distributing that font. Why does that distinction matter? Often fonts aren't licensed to allow for distribution. By distributing a font, you are essentially giving it away free. You should be cautious when using `@font-face` and always make sure a font's license allows for distribution.

You can find the license information for the Average and Belgrano fonts in text files distributed with the ***CSS3 Foundations*** project files.

A growing number of third-party services offer fonts to be used on web pages, some free and some paid. Some of these services use the `@font-face` technique, whereas others use JavaScript to display a font in a way that means a user doesn't have to download it—working around the licensing issue somewhat.

The Average font was downloaded free from Google Web Fonts (`www.google.com/webfonts`), which is becoming one of the web's most popular font services. What's more, all its fonts are completely free and licensed for web page usage, meaning it's one of if not *the* easiest and most carefree font delivery services.

Google Web Fonts

Although Google Web Fonts makes all its fonts available to download free, you're best to use the stylesheet it provides. This stylesheet references the font on its web server, meaning you don't need to save the font on your own server. In doing this, you save yourself some bandwidth usage, and fonts tend to load faster when hosted by Google. Go ahead and remove the `@font-face` rule from the Cool Shoes & Socks stylesheet so Google can host the font instead:

1. In styles.css, find the `@font-face` rule set and delete it (keep the `font-family` declaration in the `body` rule set, however).
2. In your web browser, navigate to Google Web Fonts (`www.google.com/webfonts`) and search for "Average." When the Average font is displayed, as in Figure 10-3, click Add to Collection.

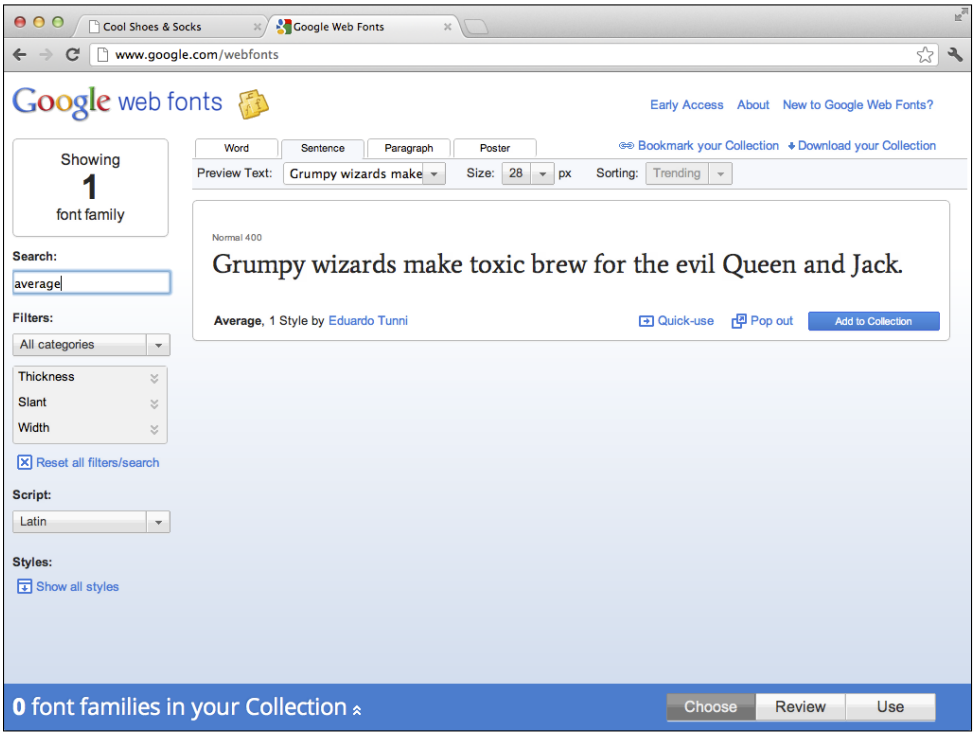


FIGURE 10-3 The Average font selected on the Google Web Fonts page.

The blue box at the bottom of the page shows the fonts you have in your collection. Google Web Fonts allows you to add as many fonts to your page as you like, so find another font, too.

3. Search for the font "Belgrano," and when it appears, click Add to Collection.
4. Now you have two fonts, click Use in the blue collection box.
    - Step 1 of this page shows that these fonts have a "Normal 400" style. This means the fonts come in only a normal style, nonbold, nonitalic, and so on. However, when learning to change the styles of fonts in the next chapter, you see that the browser can change a font's style without there being a specific variant of that font. In this step, you also see a speed dial that shows these fonts will be quick to load. Great! If a font comes with more styles and you know you're never going to use them, you can uncheck those styles to speed up the delivery of fonts.

- Step 2 shows the character set to be used. More often that not, you will use only the Latin set, so you don't need to change anything in this step.
- Step 3 provides the code you must use to have your chosen fonts working on your web page. You're presented with three different methods of adding the fonts to a page. For this example, use the Standard method.

5. Copy the HTML under the Standard tab:

```
<link
    href='http://fonts.googleapis.com/
  css?family=Average|Belgrano'
    rel='stylesheet' type='text/css'>
```

6. In index.html, paste this HTML above the link to the styles.css stylesheet, like so:

```
<link
    href='http://fonts.googleapis.com/
  css?family=Average|Belgrano'
    rel='stylesheet' type='text/css'>
<link rel="stylesheet" href="css/styles.css" type="text/css" />
```

7. Save index.html.

- If you're curious about what this step is doing, you can actually navigate to the external stylesheet by visiting `fonts.googleapis.com/css?family=Average|Belgrano`, and as you will see, Google uses the `@font-face` rule to tell the browser you will be using the Average and Belgrano fonts:

```
@font-face {
    font-family: 'Belgrano';
    font-style: normal;
    font-weight: 400;
    src: local('Belgrano'), local('Belgrano-Regular'),
    url('http://themes.googleusercontent.com/static/fonts/
  belgrano/v3/9nICvxZmkDv7_ninPVYjoXYhjbSpvc47ee6xR_80Hnw.
  woff')
    format('woff');
}

@font-face {
    font-family: 'Average';
    font-style: normal;
    font-weight: 400;
    src: local('Average'), local('Average-Regular'),
    url('http://themes.googleusercontent.com/static/fonts/
  average/v1/4iG3r29DvHyol7Yxf3Wz2wLUuEpTyoUstqEm5AMlJo4.woff')
    format('woff');
}
```

- Note that Google Web Fonts doesn't use two `src` declarations as you did to get the font working in versions of Internet Explorer below version 9. Does this matter? Because the fonts Average and Belgrano aren't displayed for these older versions of Internet Explorer (unless users have them installed on their device), the browser moves down the font stack you created to apply the Georgia font instead. Because the font is such a close match, it doesn't particularly matter.

The final step explains how to make use of these fonts by adding a `font-family` declaration to the stylesheet. You gave the `body` rule set a `font-family` declaration earlier, so the Average font is already applied. Now apply the Belgrano font.

8. In styles.css, find the `h1, h2, h3, h4` rule set and add the following:

```
font-family: Belgrano, Georgia, serif;
```

9. Save styles.css.

Now, as illustrated in Figure 10-4, all the text on the Cool Shoes & Socks page is styled in the Average font, except for the titles that are styled in Belgrano—all generously hosted and provided free by Google Web Fonts.

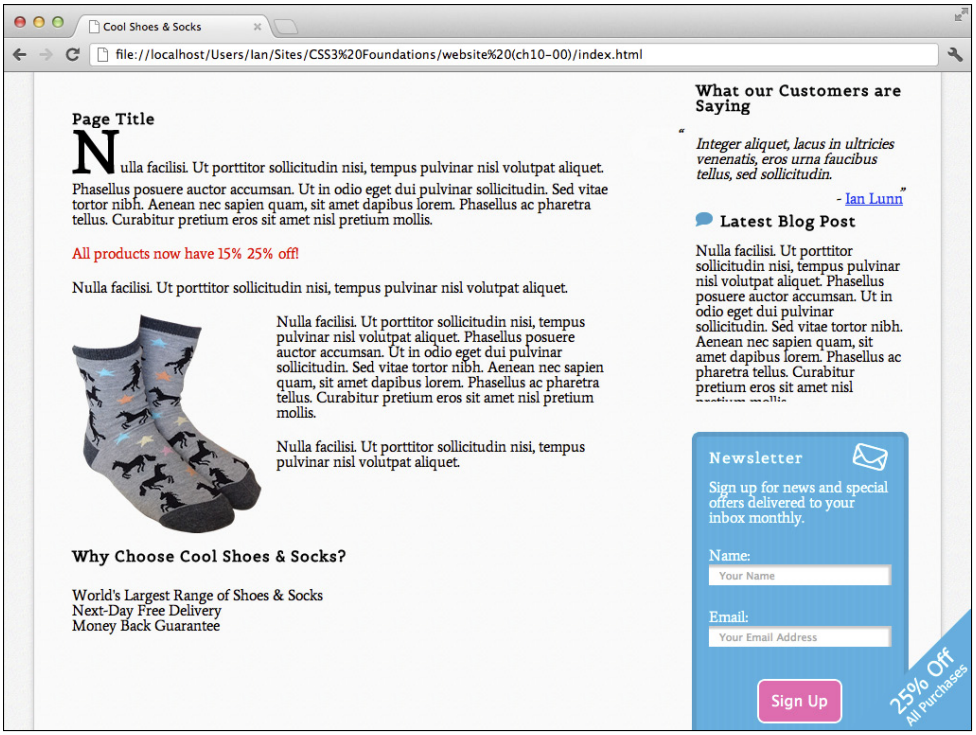


FIGURE 10-4 The Cool Shoes & Socks page styled with the Average and Belgrano fonts.

Other Font Services

Of course, other font services are available, too.

- `www.fontdeck.com`—Fontdeck has a wide selection of fonts and a great pricing structure that means you pay only for the exact font you need. Prices per font start at $2.50 per year, which includes 1 million page views a month. You can also try every font free for as long as you like.
- `www.typekit.com`—Typekit also has a great selection of fonts. Pricing is based on subscription plans that offer varying features. The free plan offers 25,000 page views per month with access to fonts in the Trial Library. The personal plan offers 50,000 page views per month with access to the Personal Library at a cost of $24.99 a year. A portfolio plan offers 100,000 page views with access to the entire library of fonts, at a cost of $49.99 a year. Typekit also offers performance and business plans if you need more than 1 million page views per month.

You may also like to try the following:

- **Font Squirrel**—`www.fontsquirrel.com/`
- **Webtype**—www.webtype.com/
- **FontsLive**—www.fontslive.com/
- **TypeFront**—www.typefront.com/

Summary

In this chapter, you added fonts to the Cool Shoes & Socks page that users may not necessarily have installed on their device. If you use the `@font-face` rule along with the Google Web Font service, the user can still see those fonts.

Now that you've added some fonts to the page, in the next chapter, you learn to style those fonts to give certain text weight and emphasis.

chapter eleven

Styling Fonts and Text

IN THE PRECEDING chapter you gave the Cool Shoes & Socks page some unique fonts via Google Web Fonts. With a knowledge of `@font-face` and third-party font services, you have access to tens of, if not hundreds of thousands of, fonts to give web pages a unique style of their own. Why stop there, though? CSS offers many ways to style fonts and text to make a page even more useful and unique.

Project files update (ch11-00): If you haven't followed the previous instructions and are comfortable working from here onward or would like to reference the project files up to this point, you can download them from `www.wiley.com/go/treehouse/css3foundations`.

Styling Fonts

Changing the appearance of text doesn't end at changing the font face. If all text on a page were the same style, no matter how fancy the font, the page would look dull and wouldn't direct the users eye particularly well. By changing attributes relating to the font, such as size, variant, weight, and so on, you can help the user determine which text is more important than other text, allowing them to scan the page quicker.

font-style

Initial value: `normal` | Inherited: Yes | Applies to: All | CSS2.1
Browser support: IE 4+, Firefox 1+, Chrome 1+, Opera 7+, Safari 1+

By using the `font-style` property, you can change the style of a font between `normal`, `italic`, and `oblique`. In the preceding chapter, I mentioned that the two fonts chosen from Google Web Fonts are only available in a normal style. Fonts often have various styles, such as oblique and italic. Both of these styles are slanted; oblique is a distorted version of the normal style, and italic is a style of its own, generally cursive in nature. The default `font-style` is `normal`, so at present all fonts on the Cool Shoes & Socks page have a normal style.

Because the fonts you used from Google Web Fonts come only in a normal style, does that mean you can't use italic and oblique styles? No. Assuming you declare a `font-style` of `italic`, a browser looks to see whether an italic version of a font is available. If not, it looks for an oblique version, and if that doesn't exist either, the browser renders the normal style of font with a sloping transformation applied—making an oblique style.

1. In styles.css, in the `p + .offer` rule set, add the following:

   ```
   font-style: italic;
   ```

2. Save styles.css.

Here, you declare an italic `font-style` for the offer text "All products now have 15% 25% off!" (later in this chapter you will learn how to put a strike through 15%, showing the discount has increased), but because the font doesn't have an italic or oblique variant to fall back to, the browser renders an oblique style of its own, as shown in Figure 11-1.

FIGURE 11-1 The offer text set in oblique.

font-variant

Initial value: `normal` | Inherited: Yes | Applies to: All | CSS2.1
Browser support: IE 4+, Firefox 1+, Chrome 1+, Opera 3.5+, Safari 1+

The `font-variant` property simply takes the values `normal` or `small-caps`. By default, the `font-variant` is `normal`. When you set the value to `small-caps`, the font is changed to a variant that uses `small-caps`, meaning the lowercase letters look similar to the uppercase letters but with smaller proportions.

font-weight

Initial value: `normal` | Inherited: Yes | Applies to: All | CSS2.1
Browser support: IE 3+, Firefox 1+, Chrome 2+, Opera 3.5+, Safari 1.3+

You use `font-weight` to change the weight of characters in a font, their degree of blackness, or stroke thickness. You briefly used this property in Chapter 3 by giving some elements the declaration `font-weight: bold;`.

You can use these four keyword values: `normal`, `bold`, `bolder`, and `lighter`. You can also use the values `100` to `900` in 100-point increments that commonly correspond to these weight names:

- `100`—Thin
- `200`—Extra Light (Ultra Light)
- `300`—Light
- `400`—Normal (equivalent of the `normal` keyword)
- `500`—Medium
- `600`—Semi Bold (Demi Bold)
- `700`—Bold (equivalent of the `bold` keyword)
- `800`—Extra Bold (Ultra Bold)
- `900`—Black (Heavy)

By using the `bolder` and `lighter` keywords, you declare the weight of the font should be bolder or lighter than the inherited value.

You will find that very few fonts support the complete range of weights; in this case, the browser uses the nearest weight to the one you specified.

Because both of the Google Web Fonts used for Cool Shoes & Socks are available only in "Normal 400," much like how a browser can render an oblique font style using the normal font, a browser can also render bolder and lighter fonts if an alternative weight of font is not present.

1. In styles.css, add a declaration to the `.button` rule set to make its text bold:

   ```
   font-weight: bolder;
   ```

2. Save styles.css.

The More Information button `<a class="button purchase" href="#" title="Purchase product">` and Sign Up button `<input class="button" id="submit-newsletter" type="submit" value="Sign Up" />` inherit a default `font-weight` of `normal`. By changing the `font-weight` to `bolder`, you tell the browser to make the font of those buttons heavier than `normal`. The "Average" font has no weight variants, so the browser takes the normal font and renders it as bold (or 700).

3. Add a declaration to the `label` rule set to change the weight of the input labels in the Newsletter box:

```
font-weight: 900;
```

4. Save styles.css.

Although this declaration for the `label` rule set is different from that of the `.button` rule set, the same effect is achieved because the "Average" font doesn't have a 900 variant. The browser again takes its own action and renders the font bold.

Obviously, it's better to be consistent with `font-weight`. Although the "Average" font doesn't have any styles other than "Normal 400," fallback fonts (the ones specified in the font stack) may have, in which case `font-weight: bolder;` and `font-weight: 900;` *may* be rendered differently. If you like, change the previously added declarations to `font-weight: bold;` or `font-weight: 700;` to keep them consistent.

font-size

Initial value: `medium` | Inherited: Yes | Applies to: All | CSS2.1

Browser support: IE 5.5+, Firefox 1+, Chrome 1+, Opera 7+, Safari 1+

The `font-size` property controls the layout of the text on a web page. You can change `font-size` in many ways, each with advantages and disadvantages. As already established, Cool Shoes & Socks uses the em unit for not just font sizes but also other properties such as `padding` and `margin`.

Let's come back to using the em unit for `font-size` in a moment, but first see what other methods you can use to change the font size.

Keywords

Similar to `font-weight`, `font-size` can be changed using a series of keywords: `xx-small`, `x-small`, `small`, `medium`, `large`, `x-large`, and `xx-large`. Likewise, you can use the keywords `larger` and `smaller` to increase or decrease the font size relative to an element's inherited `font-size`.

Using these keywords makes changing font sizes easy; however, you may feel slightly limited by them. What's more, the CSS specification provides a guideline to browser vendors that suggests what size of font these keywords should relate to. Although browsers do a good job of adhering to the guidelines, you may find inconsistencies between browsers.

Percentages

When you use a percentage value, the font size is relative to the parent element's font size—that which is inherited. Assuming the inherited `font-size` of an element is 16px, if you declare that element's `font-size` should be `50%`, the calculated `font-size` is 8px.

Using percentages offers a wide range of control over the size of a font and also allows a font to be resized easily. However, because percentages relate to different factors depending on the property they are used on, they don't allow for a consistent workflow during the creation of a website. More on this topic when comparing percentages and ems in a moment.

Absolute Length Units

Absolute length units are those such as pixels (px), centimeters (cm), inches (in), and so on. Pixels are the most common absolute length unit and allow for very specific control over the size of a font. Absolute length units are fixed, however, and should be used only when the output environment is known. Because many types of devices access the web nowadays, it is wise to avoid absolute length units because although a font size of 16px on a desktop may look perfectly readable, on a much smaller device, it may be too small. By using absolute length units, you also prevent users from being able to increase the size of text via their browser's built-in functionality.

Relative Units

Relative units are those such as em and ex. On the modern web, ems are the suggested unit for `font-size`. Much like percentages, font sizes that use relative units are calculated relative to the parent element's font size and can easily be scaled where necessary.

To help you better understand why relative units are important—and in the long run save you time—Figure 11-2 shows Cool Shoes & Socks in Opera's Mobile Emulator. On a small device, notice how small the font is? Although the Cool Shoes & Socks page is sized well for reading on a desktop computer, it's difficult to read on a mobile device.

In Figure 11-3, I modified the `body` rule set to increase the `font-size` from `62.5%` to `1.2em`. Because all the fonts relate to that `font-size`, they are all scaled up, too, making the entire page much more readable on smaller devices.

FIGURE 11-2 The current Cool Shoes & Socks page viewed in Opera Mobile Emulator.

I'm jumping ahead a little though. You work on improving the readability on mobile devices in Chapter 15.

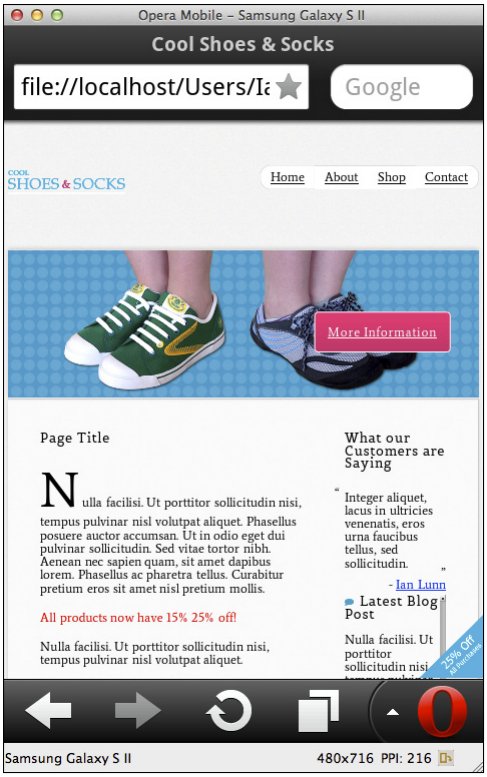


FIGURE 11-3 A version of the Cool Shoes & Socks page viewed in Opera Mobile Emulator with the body font increased to 1.2 em.

Percentages Versus Ems

If relative units, such as ems, work in the same way as percentages, why choose ems?

When you specify a `font-size` using percentages or ems, 1 em is the equivalent of 100%, 0.5 em is the equivalent of 50%, and so on. Ems and percentages work in the exact same way when applied to `font-size`. However, when used on other properties such as `padding` and `margin`, they differ.

Ems are always relative to an element's `font-size`, whereas percentages are relative to another value, such as the `height` and `width` dimensions of an element. Consider the following CSS as an example:

```
div {
    font-size: 16px;
    padding-left: 1em;
    width: 200px;
}
```

The calculated `padding-left` is 16 px because 1 em is relative to the `font-size` multiplied by one, which is 16 px. A `padding-left` of 2 em creates a calculated `padding-left` of 32 px and so on.

```
div {
    font-size: 16px;
    padding-left: 100%;
    width: 200px;
}
```

In this example, the `padding-left` is changed to a percentage, so it relates to the `width` of the element instead, and the calculated `padding-left` is therefore 200px.

Because properties such as `padding` and `margin` better relate to the size of text, it is best to use ems for `font-size`. When I scaled up the font size between Figure 11-2 and Figure 11-3, the padding and margins scaled also, aiding the readability of the page.

When working with relative units and calculated values, rather than doing all of the calculations yourself, you can use your browser's web developer tools to find the calculated value of any element. Please refer to the sidebar "Working with Relative Units" in Chapter 4.

You may also like to refer to Chapter 4's description of the Rem unit, which always inherits `font-size` from the `<html>` element, making dealing with inheritance easier. Unfortunately, at the time of writing, Rem doesn't have enough browser support to make it useful in a real-world website. It will almost certainly be the unit used for `font-size` in the future though.

line-height

Initial value: `normal` | Inherited: Yes | Applies to: All Elements | CSS2.1

Browser support: IE 4+, Firefox 1+, Chrome 1+, Opera 7+, Safari 1+

`line-height` specifies the minimal height of a line of text within an element. The initial value of `normal` means the `line-height` is the same value as the `font-size`. You also can specify a unit of length, percentage, or number. A number value is a multiplication of an element's font size.

1. In styles.css, find the `#content p, #content ul` rule set and add the following declaration:

   ```
   line-height: 1.4em;
   ```

2. Save styles.css.

As shown in Figure 11-4, the main content now has an increased `line-height` (the space between lines of text has increased). You can achieve the same effect by using the declarations `line-height: 140%;` or `line-height: 1.4;`.

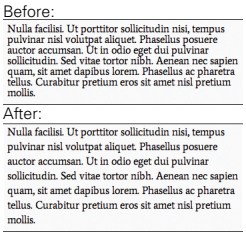


FIGURE 11-4 The line-height of the main content is increased.

font (Shorthand)

Browser support: IE 4+, Firefox 1+, Chrome 1+, Opera 3.5+, Safari 1+

As you've seen, styling and changing the font consist of quite a lot of properties. To make life easier, you can use the `font` property, which is a shorthand consisting of the properties `font-style`, `font-variant`, `font-weight`, `font-size`, `line-height`, and `font-family`. Use the following syntax when using the `font` property:

```
font: font-style font-variant font-weight font-size/line-height
font-family;
```

At a minimum, `font-size` and `font-family` must be present. When specifying `font-size` and `line-height`, you separate the two with a forward slash (`/`).

1. In styles.css, find the `body` rule set and delete the following declarations:

   ```
   font-family: Average, Georgia, serif;
   font-size: 62.5%;
   ```

2. In their place, add this shorthand `font` declaration:

   ```
   font: 62.5% Average, Georgia, serif;
   ```

3. Save styles.css.

When you use the shorthand `font` declaration, all font-related properties are first reset to their initial values, so in the declaration you just added, `font-style`, `font-variant`, and `font-weight` are absent and will be set to their initial value of `normal`. `font-size` will be set to `62.5%`, `line-height` is also absent so will be set to `normal`, and finally, `font-family` will be set to `Average, Georgia, serif`.

Now, you might want to go through the rest of the font-related properties in the Cool Shoes & Socks stylesheet to make them shorthand, but doing so wouldn't be efficient. Consider the following rule set:

```
h1, h2, h3, h4 {
    font-weight: bold;
    font-family: Belgrano, serif;
}
```

Because the `font` declaration resets all font-related properties to their initial value, when you use the `font` property in this rule set, the `font-size` is reset to `medium`, overriding the more desirable font size that is inherited. Of course, you could specify the `font-size` as well, like so:

```
h1, h2, h3, h4 {
    font: bold 1em Belgrano, serif;
}
```

By doing this though, you're making a little extra work for yourself because these elements will inherit a `font-size` anyway. Using shorthand properties where possible is a good idea, but in this case—and with all other font-related declarations in the Cool Shoes & Socks stylesheet—it's more efficient to use the longhand properties instead, allowing inheritance to work its magic.

Code Challenge: Change the Style of More Fonts

In styles.css, do the following:

1. In the rule set `#header nav ul li a`, add the declaration `font-weight: bold;`.
2. In the rule set `input[type="submit"][class="button"]`, add the declaration `font-size: 1em;`.
3. In the rule sets `blockquote p:before` and `blockquote p:after`, add the declaration `font-size: 2em;`.
4. In the rule set `label` and `input[type="text"], input[type="email"]`, add the declaration `font-size: .8em;`.

Project files update (ch11-01): If you haven't followed the previous instructions and are comfortable working from here onward or would like to reference the project files up to this point, you can download them from `www.wiley.com/treehouse/css3foundations`.

Styling Text

The previously described properties manipulate the font face but properties of the text can also be changed, such as color, decoration, alignment, and so on.

color

Initial value: browser dependent | Inherited: Yes | Applies to: All | CSS2.1

Browser support: IE 3+, Firefox 1+, Chrome 1+, Opera 3.5+, Safari 1+

The initial value for the color of text is browser dependent although all browsers in use today use black text, except for links, which are blue when unvisited, purple when visited, and red when active.

1. In styles.css, below the `.benefits` rule set, add a new rule set:

```
.benefits ul {
    color: #38AEDB;
}
```

2. Save styles.css.

The text under "Why Choose Cool Shoes & Socks?" is now blue, helping draw the users eye to it. Shortly you'll add images to these points to make them stand out even more.

Color values such as keywords, hex values, and `rgb()` can be used to apply color to text, as explained in Chapter 3.

text-decoration

Initial value: `none` | Inherited: No | Applies to: All | CSS2.1

Browser support: IE 3+, Firefox 1+, Chrome 1+, Opera 3.5+, Safari 1+

The `text-decoration` property allows you to change the line decoration applied to text. You can use the values `underline`, `overline`, and `line-through`. The CSS2.1 specification also describes the value `blink`, but due to the accessibility concerns blinking text raises, the CSS3 Text module no longer lists this value and browsers do not support it.

The CSS3 Text module (`www.w3.org/TR/css3-text/`) shows that the `text-decoration` property will eventually become a shorthand for the properties `text-decoration-line`, which takes the same values as the current `text-decoration` (`underline`, `overline`, and `line-through`); `text-decoration-color`, which lets you change the color of the line; and `text-decoration-style`, which lets you change the style of the line to either `solid`, `double`, `dotted`, `dashed`, or `wavy`. However, only Mozilla Firefox version 6 and onward support these properties by adding the `–moz` prefix to each property. Until these properties get better support in browsers, you can only safely use `text-decoration`, which decorates text with a solid line the same color as the text.

The offer text that you made italic, shows that the price of the products is now marked down by a percentage, but there are two percentage values. To strike out one of those values, do the following:

1. In styles.css, below the `p + .offer` rule set, add a new rule set:

```
.strike {
    text-decoration: line-through;
}
```

2. Save styles.css.

The 15% is wrapped with span tags with a class of `strike`. As shown in Figure 11-5, the 15% discount now has a line through it so the user can see the discount has been increased to 25%. Maybe this company should have been called *Cheap Shoes & Socks*!

All products now have ~~15%~~ 25% off!

FIGURE 11-5 The 15% discount with a line through it.

If you would like an element to have multiple text decorations, you can specify multiple values, each separated by a space, such as `text-decoration: line-through underline overline;`.

text-transform

Initial value: `none` | Inherited: No | Applies to: All | CSS2.1

Browser support: IE 4+, Firefox 1+, Chrome 1+, Opera 7+, Safari 1+

Sometimes you might want to change text case. Rather than go in to the HTML and make those changes, you can tell the browser to automatically change that case for you, using the `text-transform` property.

`text-transform` takes the following values:

- `none`—Do not transform the text.
- `uppercase`—Change all the text to uppercase (capital letters).
- `lowercase`—Change all the text to lowercase.
- `capitalize`—Change the first letter of each word to uppercase.

Because the discount in the body text of the Cool Shoes & Socks page may persuade a user to make a purchase, you can make that text stand out a little more by making it uppercase:

1. In styles.css, add the following declaration to the `p + .offer` rule set:

   ```
   text-transform: uppercase;
   ```

2. Save styles.css.

text-shadow

Initial value: `none` | Inherited: Yes | Applies to: All | CSS3
Browser support: IE 10+, Firefox 3.5+, Chrome 2.1+, Opera 9.5+, Safari 1.1+

Much like how you give a box a shadow using the `box-shadow` property, which was covered in Chapter 5, you can also give text a shadow using the `text-shadow` property (see Figure 11-6). `text-shadow` works in all browsers without a prefix except for Internet Explorer versions 6, 7, 8, and 9. Because a text shadow is only a visual style, the fact it isn't shown in these versions of Internet Explorer doesn't matter.

1. In styles.css, add the following declaration to the `p + .offer` rule set:

   ```
   text-shadow: 2px 2px 1px rgba(0, 0, 0, 0.2);
   ```

2. Save styles.css.

FIGURE 11-6 The text-shadow applied to the offer text.

The `text-shadow` syntax works in a similar way to `box-shadow` but takes a maximum of three values, rather than four, which modify the shadow as follows:

- The first value represents the horizontal offset—how far the shadow is to the right of the text, a negative value to its left. The shadow you added is offset 2 pixels to the right.

- The second value represents the vertical offset—how far the shadow is to the bottom of the text, a negative value to the top. Again, you gave the offer text shadow a 2-pixel offset to the bottom.
- The third value is the blur of the shadow. This value is optional, and in its absence, the shadow would have no blur. When you declare a blur of 1 px, the shadow applied to the offer text has a very mild blur.

Following from the shadow values is an optional color. By default, a shadow is black, but the shadow you added uses the `rgba()` color function to apply black with an alpha opacity of `0.2`, making it more of a gray color.

As with `box-shadow`, you can use multiple shadows on text by separating each shadow with a comma:

1. In styles.css, modify the `text-shadow` declaration to include two text shadows on the offer text:

```
text-shadow: 2px 2px 1px rgba(0, 0, 0, 0.2),
             4px 4px 2px rgba(0, 0, 0, 0.1);
```

2. Save styles.css.

A shadow that follows another is placed behind it. The second text shadow applied is even subtler than the first, but both work together to help the text stand out a little more.

letter-spacing

Initial value: `normal` | Inherited: Yes | Applies to: All Elements | CSS2.1
Browser support: IE 4+, Firefox 1+, Chrome 1+, Opera 3.5+, Safari 1+

The `letter-spacing` property allows you to control the amount of space between each individual character with the initial value of `normal` being the default amount.

By giving this property a length value, you can increase the space between characters in addition to the default space.

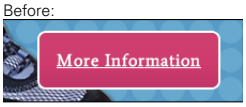
1. In styles.css, find the `.showcase .button` rule set and add the following declaration:

```
letter-spacing: 2px;
```

2. Save styles.css.

As you can see in Figure 11-7, when you give the More Information button a `letter-spacing` of `2px`, the space between characters is slightly increased.

Before:



After:

FIGURE 11-7 The difference between the "More Information" button before and after having its letter-spacing modified.

word-spacing

Initial value: `normal` | Inherited: Yes | Applies to: All Elements | CSS2.1
Browser support: IE 6+, Firefox 1+, Chrome 1+, Opera 3.5+, Safari 1+

Just like `letter-spacing`, the `word-spacing` property enables you to change the spacing between words. The initial value of `normal` represents the default space between words, and you can specify a length value to adjust this space as you see fit.

direction

Initial value: `ltr` | Inherited: Yes | Applies to: All Elements | CSS2.1
Browser support: IE 5.5+, Firefox 1+, Chrome 2+, Opera 9.2+, Safari 1.3+

The `direction` property determines the direction of text. The initial value `ltr` means left to right. The value `rtl` (right to left) can be used for Hebrew and Arabic text. Note however, that specifying the direction of text is normally done in the HTML using the `dir` attribute.

text-align

Initial value: `start` | Inherited: Yes | Applies to: Block Containers | CSS2.1
Browser support: IE 3+, Firefox 1+, Chrome 1+, Opera 3.5+, Safari 1+

By using `text-align`, you can align content to the left, right, or center of its parent; in addition, you can make that content justified, meaning flush to both sides of the parent. The CSS2.1 specification describes the values `left`, `right`, `center`, and `justify` to be used with `text-align` and an initial "nameless" value that acts as `left` if the direction of text is left to right or `right` if the text direction is right to left (for languages such as Arabic and Hebrew).

For the sake of completeness CSS Level 3 introduces two new values, `start` and `end`, with `start` replacing the "nameless" initial value of `text-align`. `start` and `end` achieve the exact same results as `left` and `right` but make more sense when used in conjunction with differing directions of text, `start` being whatever side the text starts from and `end` being the opposite. However, `start` and `end` are currently supported only in Firefox, Chrome, and Safari. Until support improves, it's best just to work with the CSS2.1 values: `left`, `right`, `center`, and `justify`.

1. In styles.css, add a declaration to the `p + .offer` rule set to make the text centered:

   ```
   text-align: center;
   ```

2. Add a declaration to the `blockquote` rule set to make the text justified:

   ```
   text-align: justify;
   ```

3. Save styles.css.

As shown in Figure 11-8, the offer text is now centered relative to its containing parent element, and the customer testimonial is justified, spaced out to touch the edges of its containing parent.

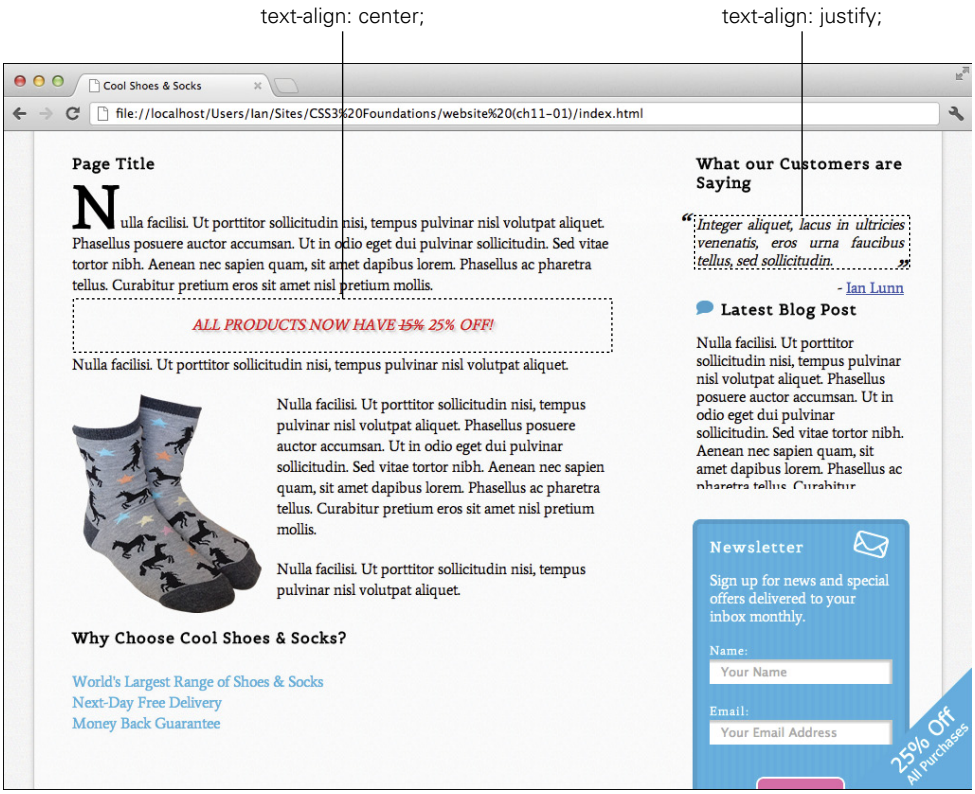


FIGURE 11-8 Text centered and justified.

text-indent

Initial value: 0 | Inherited: Yes | Applies to: Block Containers | CSS2.1
Browser support: IE 3+, Firefox 1+, Chrome 1+, Opera 3.5+, Safari 1+

You can use `text-indent` to apply an indentation to the first line of text in a block container, using either a unit of length such as pixels and ems, or a percentage that is relative to the width of the containing block.

If, like me, you were taught at school to always indent a new paragraph—which seems to be old-fashioned now—you could use the following rule set to indent all <p> elements:

```
p {
    text-indent: 1.25em;
}
```

white-space

Initial value: `normal` | Inherited: Yes | Applies to: All Elements | CSS2.1
Browser support: IE 5.5+, Firefox 1+, Chrome 1+, Opera 4+, Safari 1+

The `white-space` property determines how white space inside an element is handled. Consider the first paragraph of the Cool Shoes & Socks page as an example:

```
<p>Nulla facilisi. Ut porttitor sollicitudin nisi, tempus pulvinar
nisl volutpat aliquet. Phasellus posuere auctor accumsan. Ut in odio
eget dui pulvinar sollicitudin. Sed vitae tortor nibh. Aenean nec
sapien quam, sit amet dapibus lorem. Phasellus ac pharetra tellus.
Curabitur pretium eros sit amet nisl pretium mollis.</p>
```

With `white-space` set to its initial value of `normal`, white space is collapsed (meaning any space within this HTML does not take effect), and lines are wrapped (broken and pushed onto a new line) where necessary to fill the containing element. So the following achieves the same visual representation as the preceding example:

```
<p>Nulla facilisi. Ut porttitor sollicitudin nisi, tempus pulvinar
nisl volutpat aliquet. Phasellus posuere auctor accumsan.

Ut in odio eget dui pulvinar    sollicitudin. Sed vitae tortor nibh.
Aenean nec sapien quam, sit amet dapibus lorem. Phasellus ac pharetra
tellus. Curabitur pretium eros sit amet nisl pretium mollis.</p>
```

You also can use the following values with `white-space`:

- `pre`—White space isn't collapsed, and lines are broken only when they are broken in the HTML.
- `nowrap`—White space is collapsed as per `normal`, but lines of text are never wrapped.
- `pre-wrap`—White space isn't collapsed, but lines are wrapped where necessary or when broken in the HTML.
- `pre-line`—White space is collapsed, and lines are wrapped where necessary or when broken in the HTML

overflow-wrap and word-wrap

Initial value: `normal` | Inherited: Yes | Applies to: All Elements | CSS3

Browser support: IE 5.5+, Firefox 3.5+, Chrome 1+, Opera 10.5+, Safari 1+

`word-wrap` is another of those properties that was introduced by Microsoft and was later added to the CSS3 specification. It has undergone a little bit of a makeover in the CSS3 Text module (`www.w3.org/TR/css3-text/#overflow-wrap`) and now has the new name `overflow-wrap`. However, because only CSS3 supporting browsers recognize `overflow-wrap`, you should use `word-wrap`, which both modern browsers and Internet Explorer versions 6, 7, and 8 support.

The initial value of `normal` means lines of text may break only between words. Usually, this is fine, but if a word is longer than the element it is contained within, it will overflow rather than break, as shown in Figure 11-9. Where this effect is undesirable, you can use the value `break-word` to force a word to break and wrap onto a new line instead, also shown in Figure 11.9.

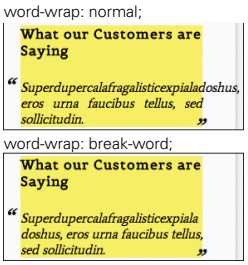


FIGURE 11-9 The difference between word-wrap: normal; and word-wrap: break-word.

Code Challenge: Change the Style of Various Text Elements

In styles.css, do the following:

1. Add a new rule set for `a.button:link` below the `.button` rule set with the declarations `text-transform: uppercase; text-decoration: none;`.
2. Add the declarations `text-transform: uppercase; text-decoration: none;` to the `#header nav ul li a` rule set.

Project files update (ch11-02): If you haven't followed the previous instructions and are comfortable working from here onward or would like to reference the project files up to this point, you can download them from `www.wiley.com/go/treehouse/css3foundations`.

Styling Lists

A list, such as the unordered list of benefits under "Why Choose Cool Shoes & Socks?", is made up of text as well as a marker such as a bullet point. Using CSS, you can change the type of marker used and change its position.

list-style-type

Initial value: `disc` | Inherited: Yes | Applies to: Elements with `display: list-item` | CSS2.1

Browser support: IE 4+, Firefox 1+, Chrome 1+, Opera 3.5+, Safari 1+

The `list-style-type` property specifies the appearance of list item markers. When you declare none, a list has no markers.

You can use the following values to mark list items with glyphs:

- `disc`
- `circle`
- `square`

The following values can be used to mark list items with numbers:

- `decimal`—Decimal numbers, beginning with 1
- `decimal-leading-zero`—Decimal numbers padded by initial zeros

- `lower-roman`—Lowercase Roman numerals
- `upper-roman`—Uppercase Roman numerals
- `georgian`—Traditional Georgian numbering
- `armenian`—Traditional uppercase Armenian numbering

The following values can be used to mark list items with alphabetic systems:

- `lower-latin` or `lower-alpha`—Lowercase ASCII letters
- `upper-latin` or `upper-alpha`—Uppercase ASCII letters
- `lower-greek`—Lowercase classical Greek alphabet

Figure 11-10 shows each `list-style-type`.

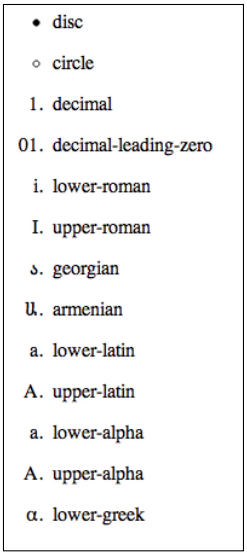


FIGURE 11-10 Examples of each list-style-type.

This demonstration page can be found in the project files ch11-02, named list-style-demo.html.

Note that although the initial value for `list-style-type` is `disc`, the CSS Reset you applied in Chapter 2 changed all list items to have a `list-style-type` of `none`. Now give the "Why Choose Cool Shoes & Socks" disc markers:

1. In styles.css, find the `.benefits ul`, rule set, and add the following:

```
list-style-type: disc;
```

2. Save styles.css.

list-style-image

Initial value: `none` | Inherited: Yes | Applies to: Elements with `display: list-item` | CSS2.1
Browser support: IE 4+, Firefox 1+, Chrome 1+, Opera 7+, Safari 1+

If none of the `list-style-type` markers takes your fancy, you can use your own image as a marker using the `list-style-image` property and declaring a `url()` function as its value:

1. In the `.benefits ul` rule set, add this new declaration:

```
list-style-image: url("../images/check.png");
```

2. Save styles.css.

Although a marker image takes the place of a basic marker determined by the `list-style-type`, it's wise to specify a `list-style-type` along with a `list-style-image` just in case the browser can't get the image.

list-style-position

Initial value: `outside` | Inherited: Yes | Applies to: Elements with `display: list-item` | CSS2.1
Browser support: IE 4+, Chrome 1+, Opera 3.5+, Safari 1+

As shown in Figure 11-11, the list of benefits now uses images in place of the browser-rendered markers. Note that the markers sit outside the principal block of content. `outside` is the initial value for `list-style-position`, a property that changes the position of list markers.

FIGURE 11-11 The "Why Choose Cool Shoes & Socks" list now with images used for the markers.

1. In the `.benefits ul` rule set, add the following declaration:

```
list-style-position: inside;
```

2. Save styles.css.

As you can see in Figure 11-12, by declaring a `list-style-position` of `inside`, you can position list markers inside the principal block.

FIGURE 11-12 The list markers are now positioned inside the principal block.

list-style (Shorthand)

You are able to combine `list-style-type`, `list-style-image`, and `list-style-position` into one handy shorthand declaration, `list-style` with the following syntax:

```
list-style: list-style-type list-style-position list-style-image
```

1. Remove the three declarations from the `.benefits ul` rule set and replace them with the `list-style` shorthand declaration:

   ```
   list-style: disc inside url("../images/check.png");
   ```

2. Save styles.css.

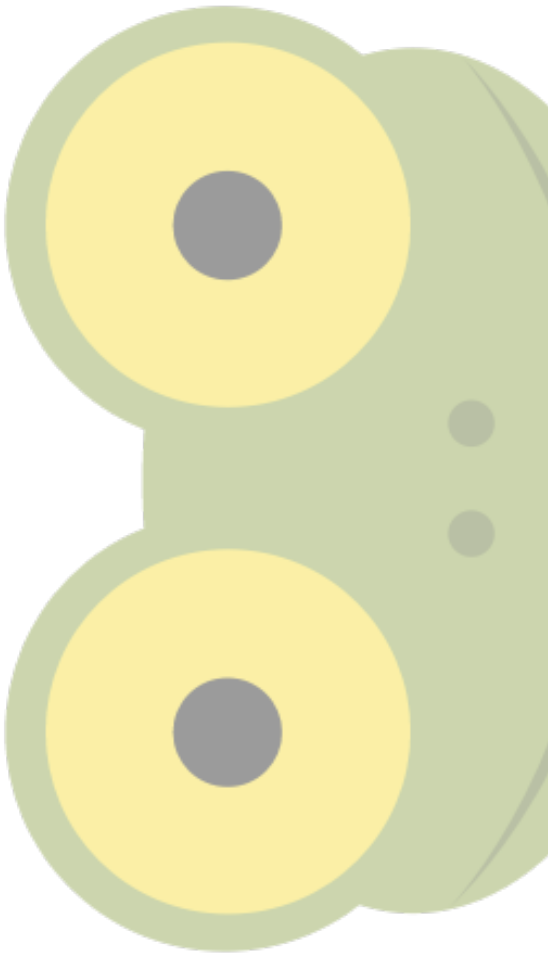
Summary

In this chapter, you added the finishing touches to the fonts and text on the Cool Shoes & Socks page. Any text that needs to stand out and grab the users' eyes does so, and the main text is well laid out and easy to read.

In the next chapter, you jump deep into the modern web, learning some of CSS3's most unique features: 2D transforms!

part 5

Taking It to the Next Level with Transforms and Animations



chapter twelve

Adding 2D Transforms

IN THIS CHAPTER, you leave behind older browsers and make use of one of CSS3's most exciting features, transforms, which let you move, rotate, and scale elements, among other manipulations.

Project files update (ch12-00): If you haven't followed the previous instructions and are comfortable working from here onward or would like to reference the project files up to this point, you can download them from `www.wiley.com/go/treehouse/css3foundations`.

Safely Using Experimental CSS3 Properties

Because all the CSS features explained in this chapter are from the CSS3 specification, they are unsupported in Internet Explorer 6, 7, and 8. Likewise, 3D transforms, transitions, and animations described in Chapters 13 and 14 are also unsupported in these browsers, as well as Internet Explorer 9. They do however have good support in Internet Explorer 10 and other major browsers.

Although these CSS3 properties are "experimental" and a part of Working Draft modules, they have been implemented in modern browsers for some time, and are actually close to being safe enough to use without prefixes. At the time of writing though, these properties should be used with prefixes, so you'll learn to do just that—the most future proof way to use the latest CSS3 features.

Because I am creating the Cool Shoes & Socks page in Google Chrome, I use the `-webkit-` prefix. If you are using a different browser, replace the `-webkit-` prefix with your browser's prefix:

- `-webkit-` for Google Chrome and Apple Safari
- `-moz-` for Mozilla Firefox
- `-o-` for Opera
- `-ms-` for Microsoft Internet Explorer

In Chapter 15, you use a tool that adds the missing prefixes to your stylesheet so that the Cool Shoes & Socks page works across all browsers.

transform and 2D Transform Functions

Initial value: `none` | Inherited: No | Applies to: All elements with a `display` declaration relating to block, inline or table | CSS3

Unprefixed browser support: IE 10+, Firefox 16+, Opera 12.5+

Prefixed browser support: IE 9+, Firefox 3.5+, Chrome 12+, Opera 10.5+, Safari 3.1+

The `transform` property allows you to manipulate the position, size, rotation, and skew of an element. When prefixed, the property itself looks like this:

```
-webkit-transform
-moz-transform
-ms-transform
-o-transform
```

The values to be used with `transform` are all functions: `translate()`, `translateX()`, `translateY()`, `scale()`, `scaleX()`, `scaleY()`, `rotate()`, `skewX()`, and `skewY()`. They all affect an element in 2D. The next chapter covers 3D functions to be used with the `transform` property.

translate(), translateX(), and translateY()

The translate functions allow you to move elements around on a web page. That capability may not seem very exciting because you've already seen this happen lots with properties such as `position`, `top`, `left`, `right`, and `bottom`, but `translate()` is an easier way to move elements around without affecting the rest of the document. Ideally, you use translate functions when you want to animate the position of elements, creating movement.

Remember that "25% Off" banner image you fixed to the bottom of the Cool Shoes & Socks page in Chapter 9? I mentioned that the image is square and has a transparent section, which may prevent users from interacting with elements below it. By using the `transform` property, you can make that banner without using an image and work around that problem of the image taking up an area it doesn't need to.

1. In index.html, find the following HTML and delete it:

```
<img class="banner-ad" src="images/banner-25percent.png"
  alt="25% Off All Purchases" />
```

2. In its place, add the following:

```
<div class="banner-ad"><span>25% Off</span>All Purchases</div>
```

3. Save index.html.

   Now that you've replaced the image, add some CSS to make the `<div>` look like the image.

4. In styles.css, add the following declarations to the `.banner-ad` rule set below the existing declarations:

```
background: #00B0DC;
color: white;
text-align: center;
font-family: Tahoma, Verdana, sans-serif;
font-size: 1.1em;
font-weight: bold;
min-height: 50px;
width: 200px;
```

5. Below the `.banner-ad` rule set, add a new rule set:

```
.banner-ad span {
    clear: both;
    display: block;
    font-size: 2.2em;
}
```

6. Save styles.css.

   As shown in Figure 12-1, the banner advertisement image is replaced with text and CSS. At the moment, the banner sits square on the page, but you can change that position using the `transform` property.

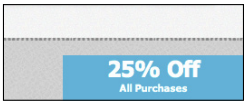


FIGURE 12-1 The banner advertisement now made using CSS rather than an image.

7. In the `.banner-ad` rule set, add the following declaration:

```
-webkit-transform: translate(50px, -25px);
```

8. Save styles.css.

The banner now sits partially off-screen, outside the viewport, as shown in Figure 12-2. This change is all part of the master plan and puts the banner in a place where you can rotate it in a moment.

FIGURE 12-2 The banner advertisement moved 50px on the X axis and –25px on the Y axis.

By giving the `translate()` function two values (the X and Y positions), you tell the browser to move the banner 50px horizontally to the right and move it vertically upward 25px on the Y axis. Negative values on the X axis move an element horizontally to the left and vertically upward on the Y axis.

Putting the banner in this position may seem strange, but it makes more sense when you consider the origin of the element. Each element has an origin, which is the position used to calculate transforms. As shown in Figure 12-3, an element's origin (represented by a cross) is its center by default, which you can change via the `transform-origin` property that is covered shortly.

By using the `translate()` function to position the center of the banner here, you can then rotate it to be perfectly positioned in the way the image was.

FIGURE 12-3 The origin of the banner advertisement and how it is used to calculate the banner's new position.

You can give the `translate()` function any unit of length or a percentage as its values. If the second value, representing the Y axis, is absent, it is assumed to be `0`.

`translate()` is the shorthand function for `translateX()` and `translateY()`, which change the X and Y axis, respectively.

rotate()

If you're itching to get to it, you can rotate that banner straight away and then take a closer look at the `rotate()` function:

1. In styles.css, change the `transform` declaration in the `.banner-ad` rule set to include a `rotate()` function:

   ```
   -webkit-transform: translate(50px, -25px) rotate(-45deg);
   ```

2. Save styles.css.

As shown in Figure 12-4, the banner is now rotated and appears in the same way that the image did. Why go to that effort? Although the element is still square after you rotate it, the diagonal top of the banner is now the top of the element, so there is no invisible space, as there was with the image. What's more, in Chapter 14, you take this process further and animate the banner when it's hovered over.

FIGURE 12-4 The banner advertisement rotated to sit at –45 degrees in the corner of the viewport.

Now look at that code you added:

```
-webkit-transform: translate(50px, -25px) rotate(-45deg);
```

Because the banner is to be both translated and rotated, you add two functions to the `transform` declaration, separated by a space. You can give a `transform` declaration as many functions as required.

You can give the rotate function a value either in degrees, gradians, radians, or turns:

- **Degrees (`deg`)**—There are 360 degrees in a full circle.
- **Gradians (`grad`)**—There are 400 gradians in a full circle.
- **Radians (`rad`)**—There are 2π radians in a full circle (meaning 3.1415 is the equivalent of 180 degrees).
- **Turn (`turn`)**—There is 1 turn in a full circle.

When you specify a `rotate()` value of `-45deg`, the banner is rotated counterclockwise by 45 degrees. A positive number rotates an element clockwise.

scale(), scaleX(), and scaleY()

The `scale()` function increases or decreases an element in size. As with `translate()`, `scale()` takes two arguments: the X and Y axes. If the second value is absent in `scale()`, however, unlike `translate()`, that one value is applied to both the X and Y axes. If you want to transform only one axis, you can use the functions `scaleX()` and `scaleY()`.

1. In styles.css, change the `transform` declaration in the `.banner-ad` rule set to include a `scale()` function:

   ```
   -webkit-transform: translate(50px, -25px) rotate(-45deg)
     scale(1.1);
   ```

2. Save styles.css.

When you specify a `scale()` value of `1.1`, the banner element and its contents are scaled by 110% of its original size. You can give `scale()` only a number, which is a multiplication of its calculated size.

skewX() and skewY()

Skewing an element makes that element appear slanted or twisted.

You may be wondering why there's no `skew()` function to go with `skewX()` and `skewY()`. An early version of the CSS3 Transforms module (`www.w3.org/TR/css3-transforms/`) *did* include `skew()`, and you may find browsers still render the function, but it has been removed from the specification. This was due to concerns about its name being misleading. For this reason, you should only use `skewX()` and `skewY()`.

`skewX()` and `skewY()` skew an element around its X and Y axis, respectively.

I'll use the More Information button to demonstrate skew transforms but not actually keep the effect:

```
-webkit-transform: skewX(10deg);
```

As shown in Figure 12-5, when you give the rule set `.showcase .button` a `transform` of `skewX(10deg)`, the More Information button is skewed 10 degrees counterclockwise from the Y axis.

FIGURE 12-5 The More Information button with a 10-degree skew applied to the X axis.

When `skewX(10deg)` is replaced with `skewY(10deg)`, like so:

```
-webkit-transform: skewY(10deg);
```

The button is skewed 10 degrees clockwise from the X axis, as shown in Figure 12-6.

FIGURE 12-6 The More Information link with a 10-degree skew applied to the Y axis.

If you want to skew both axes, rather than use the no-longer standard `skew()` function, simply use both `skewX()` and `skewY()` separated by a space:

```
-webkit-transform: skewX(10deg) skewY(10deg);
```

Skew functions accept degrees, radians, or gradians as arguments.

matrix()

The `transform` property also accepts the `matrix()` function, which allows for complex transformations. Because this is an advanced feature, it is not demonstrated in *CSS3 Foundations*, but if you want to learn more, I recommend reading Opera's article, "Understanding the CSS Transforms Matrix," available at `http://dev.opera.com/articles/view/understanding-the-css-transforms-matrix/`.

transform-origin

Initial value: `50% 50%` | Inherited: No | Applies to: All elements with a `display` declaration relating to block, inline or table | CSS3

Unprefixed browser support: IE 10+, Firefox 16+, Opera 12.5+

Prefixed browser support: IE 9+, Firefox 3.5+, Chrome 12+, Opera 10.5+, Safari 3.1+

When you are transforming an element, that element has a transform origin, which, by default, is `50% 50%` (on both the X and Y axes)—the center of the element.

As shown in Figure 12-7, the top element has the default origin of `50% 50%`, which places the origin in the center (represented by a cross). When you apply a rotate transform of 45 degrees, that element is rotated around its origin.

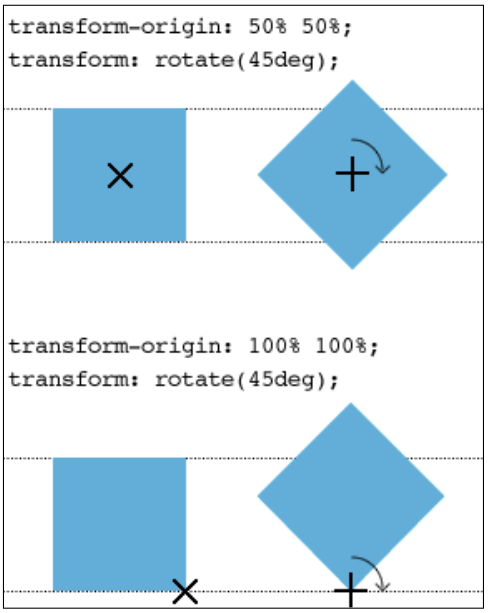


FIGURE 12-7 An element being rotated 45 degrees around an origin of 50% 50% and 100% 100%.

In Figure 12-7, the element at the bottom has an origin that is placed 100% on the X axis and 100% on the Y axis (the bottom-right corner). When you rotate the element 45 degrees around *this* origin, the transformation creates a different effect.

Let's go back to the "25% Off" banner advertisement to make use of `transform-origin`.

Although the banner is in the desired position, as shown in Figure 12-8, when the text size is scaled in a browser such as Firefox, the banner goes a bit crooked, and not all the text can be seen! This happens because the Y axis in the `translate()` function is `-25px`, which is supposed to be half the height of the element. When text is resized, the height of the element grows, and that `-25px` value may no longer be half the height of the banner, causing its position to go off center.

FIGURE 12-8 The banner advertisement goes crooked when text is resized.

You can fix this problem by changing the origin of the banner:

1. In styles.css, add the following declaration to the `.banner-ad` rule set:

```
-webkit-transform-origin: 50% 100%;
```

2. Modify the `transform` declaration, as follows:

```
-webkit-transform: translate(75px, -25px) rotate(-45deg)
  scale(1.1);
```

3. Save styles.css.

As shown in Figure 12-9, the origin is still placed on the center of the X axis but at the bottom of the Y axis. The `translate()` value for the X axis is `75px` = 100 px (half of the banner's width) – 25px (to place the X origin 25px inside the viewport), and the Y axis is `-25px` to also place it 25 px inside the viewport. Because the banner has a specified width of `200px`, the X origin is always a constant, and now that the origin on the Y axis is changed to the bottom of the element, it too is a constant, allowing the banner to grow as tall as it needs and always remain anchored 25px inside the viewport.

FIGURE 12-9 The origin of the banner element now placed on the center of the X axis and the bottom of the Y axis.

Summary

In this chapter, you scratched the surface of 2D transforms. Making the banner advertisement out of text and CSS instead of an image doesn't prevent users from interacting with elements below it like the image did. Although not all browsers support 2D transforms, those that don't still see the banner, just without the transforms applied to it (as it was in Figure 12-1). In Chapter 14, you learn to animate that banner to make it move in and out of the page when interacted with.

In the next chapter, you make some elements jump off the page using 3D transforms!

chapter thirteen

Going Beyond with 3D Transforms

IN THE PRECEDING chapter, you used the `transform` property to create 2D visual effects, but you can also use it for 3D effects. As with the 2D effects, the 3D effects are great for adding visual flair to a site, especially when used in conjunction with the transitions and animations covered in the next chapter.

At the time of writing, Internet Explorer 10 and Firefox 16 do not require vendor prefixes for properties relating to 3D transforms. Like 2D transforms though, for the safest possible implementation, use vendor prefixed properties followed by the official unprefixed property.

In this chapter, you make the Cool Shoes & Socks sidebar rotate in a 3D space.

Project files update (ch13-00): If you haven't followed the previous instructions and are comfortable working from here onward or would like to reference the project files up to this point, you can download them from `www.wiley.com/go/treehouse/css3foundations`.

perspective

Initial value: `none` | Inherited: No | Applies to: All elements with a `display` declaration relating to block, inline or table | CSS3

Unprefixed browser support: IE 10+, Firefox 16+

Prefixed browser support: Firefox 10+, Chrome 12+, Safari 4+

Before you can begin making elements 3D, you need to create a 3D space using the `perspective` property.

1. In styles.css, find the `#main` rule set and add the following declaration:

   ```
   -webkit-perspective: 1000px;
   ```

2. Save styles.css.

This declaration doesn't create any visual effect on the page; it just tells the browser to make the main content container `<div id="main" class="group">` 3D.

Think of the perspective as being how far the viewer is from an object.

Figure 13-1 shows that if `perspective` is `50px` and an element is moved toward the viewer by 25px, the element is 50% closer to the viewer. If `perspective` is `500px` and the element is again moved toward the viewer by 25px, the element is only 5% closer. A lower number of pixels applied to the `perspective` creates a greater visual impact because the viewer appears to be closer to the elements onscreen.

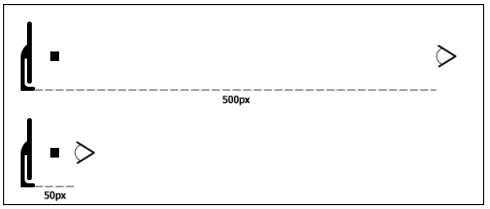


FIGURE 13-1 A low number applied to perspective has a greater visual impact.

You can give `perspective` any unit of length as a value.

perspective-origin

Initial value: `50% 50%` | Inherited: No | Applies to: All elements with a `display` declaration relating to block, inline or table | CSS3

Unprefixed browser support: IE 10+, Firefox 16+

Prefixed browser support: Firefox 10+, Chrome 12+, Safari 4+

Whereas `perspective` determines how far the viewer appears to be looking at a 3D element, `perspective-origin` determines the appearance of the viewer's position. The initial value of `50% 50%` means the viewer will see a 3D element as if that viewer is looking at it straight on. The X and Y position will be in the middle.

Again, `perspective-origin` just sets up the 3D space for the elements within it, so alone it doesn't create any visual representation. I demonstrate how it does affect elements in a moment when using the `transform` function, `translateZ()`.

`perspective-origin` takes two values, the X and Y positions, and can be given any length value; a percentage; or the keywords `left`, `center`, `right`, `top`, and `bottom`. A percentage value of `0% 0%` is the equivalent of `left top`, `100% 100%` is `right bottom`, and so on. If only one value is given, the second is assumed to be `center`.

transform and 3D Transform Functions

Initial value: `none` | Inherited: No | Applies to: All elements with a `display` declaration relating to block, inline or table | CSS3

Unprefixed browser support: IE 10+, Firefox 16+

Prefixed browser support: Firefox 10+, Chrome 12+, Safari 4+

In the preceding chapter, you used 2D transform functions such as `translateX()`, `translateY()`, `scaleX()`, `scaleY()`, and so on. When creating 3D transforms, you can use functions that control the Z axis along with the X and Y axes.

translateZ() and translate3d()

Whereas the functions `translateX()` and `translateY()` move an element left and right and up and down, the `translateZ()` function moves forward and back (toward and away from the viewer).

The following instructions are temporary, used to demonstrate `translateZ()`. If you'd like to follow them to experiment in your own browser, note that the additional HTML and CSS will be removed afterward.

Place an empty `<div class="temp"></div>` element immediately inside the main container `<div id="main" class="group">` and give it the following rule set:

```
.temp {
    border: black solid 5px;
    box-sizing: border-box;
    position: absolute;
    height: 100%;
    width: 100%;
}
```

Temporarily remove the `padding` declarations from the `#main` rule set and make it `position: relative;` so that the empty `<div class="temp">` becomes the same width and height, as shown in Figure 13-2.

FIGURE 13-2 Google Chrome zoomed out to show the empty div with a black border around it.

With the main content container `<div id="main" class="group">` set to have a perspective of `1000px`, its direct child elements can be moved in that 3D space, which is what you can do with the empty `<div class="temp">`:

```
-webkit-transform: translateZ(-500px);
```

After you transform the empty `<div class="temp">` and move it back 500 pixels on the Z axis, it appears to move away from the viewer, as shown in Figure 13-3.

FIGURE 13-3 The empty div, moved back 500px in the 3D space of the main container.

Now look at how `perspective-origin` can affect this 3D effect by applying the following declaration to the `#main` rule set:

```
-webkit-perspective-origin: 30% 30%;
```

As Figure 13-4 shows, when you change the `perspective-origin`, the element is no longer viewed straight on. The viewer now sees the element as if looking at it from an angle.

FIGURE 13-4 The empty div, now viewed from a different perspective origin.

Of course, this is just a visual demonstration to better show how 3D space works on a 2D screen.

To add a real-world 3D effect, remove the previous modifications if you chose to follow them, then go ahead and apply `translateZ()`, like so:

1. In styles.css, below the `.sidebar` rule set, add a new rule set:

```
.sidebar:hover {
    -webkit-transform: translateZ(100px);
}
```

2. Save styles.css.

Now, when the sidebar is hovered over, it appears to move toward the viewer by 100px, as shown in Figure 13-5. You may think this effect seems excessive, and I agree! At the moment, the entire sidebar moves forward, which may get annoying for the user. Shortly, you change this effect so it applies only to the newsletter box, and I explain why you can't do that just yet.

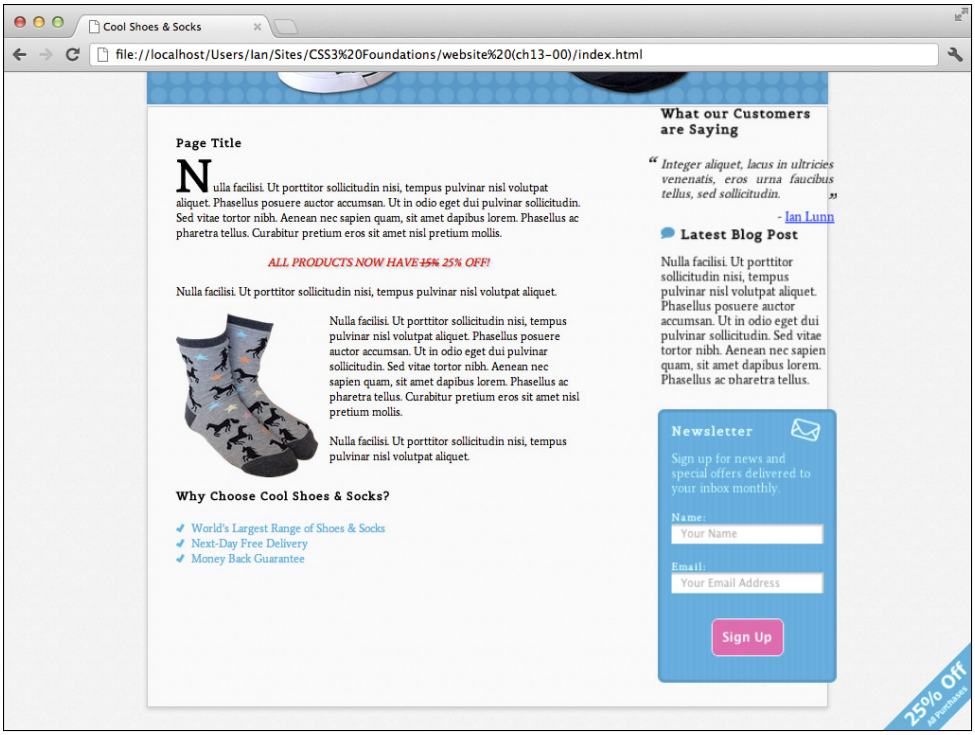


FIGURE 13-5 The Cool Shoes & Socks page zoomed out to show the sidebar moves toward the viewer when hovered over.

When creating 3D effects, if you are declaring more than one translate function, instead of specifying `translateX()`, `translateY()`, and `translateZ()`, you can use the shorthand function `translate3d()`, which takes three arguments: the X axis, Y axis, and Z axis, in that order.

rotateX(), rotateY(), rotateZ(), and rotate3d()

In the preceding chapter where you manipulated elements in two dimensions, you rotated the "25% Off" banner using the `rotate()` function.

When rotating in 2D, elements can be rotated only clockwise or counterclockwise. Now that the main content container is a 3D space, any elements within it can be rotated not just in 2D but in 3D too, either rotated on the X, Y, or Z axis, or a combination.

Now rotate the sidebar of the Cool Shoes & Socks page.

1. In styles.css, find the `.sidebar` rule set and add the following declaration:

```
-webkit-transform: rotateX(10deg);
```

2. Save styles.css.

   `rotateX()` rotates an element on the X axis. Rotated on a positive angle, the `.sidebar` is rotated away from the viewer at the top and toward the viewer at the bottom, as shown in Figure 13-6; it kind of looks like the *Star Wars* opening crawl text! A 180-degree rotation on the X axis would flip an element vertically.

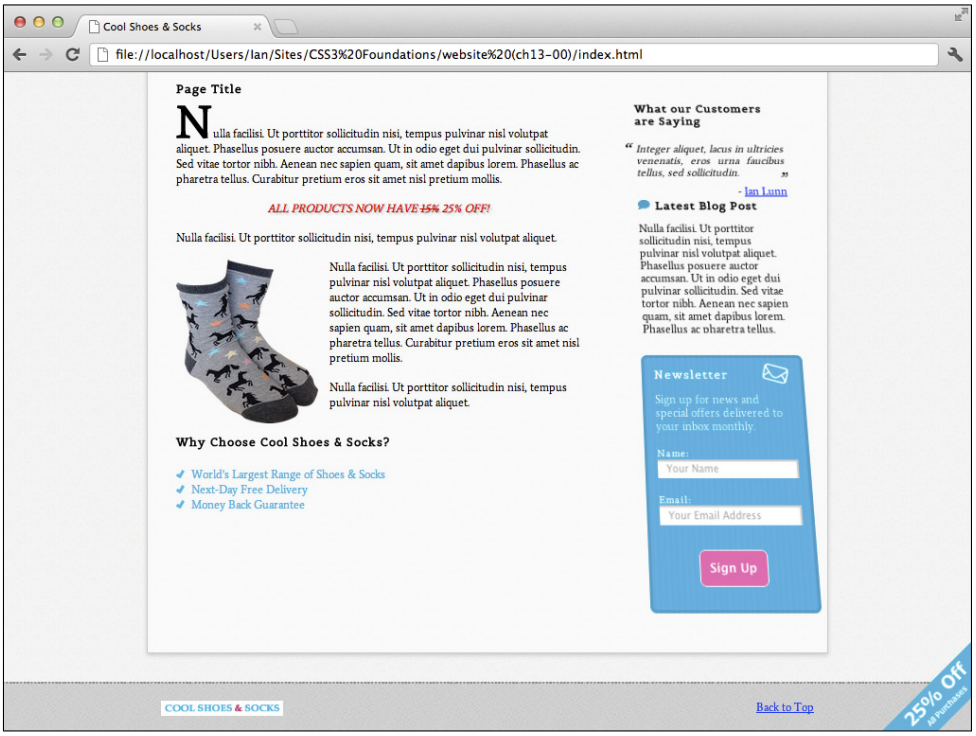


FIGURE 13-6 The Cool Shoes & Socks page zoomed out to show the sidebar rotated on the X axis by 10 degrees.

   Because the X axis rotation is unsuitable for the page, change it to a Y axis rotation instead.

3. In the `.sidebar` rule set, modify the `transform` declaration as follows:

```
-webkit-transform: rotateY(-40deg);
```

4. Save styles.css.

Rotating on the Y axis by a positive angle rotates an element's left side toward the viewer and right side away. The opposite occurs when rotating by a negative angle, which is what you do by modifying the `transform` declaration, as shown in Figure 13-7. A 180-degree rotation on the Y axis would flip an element horizontally.

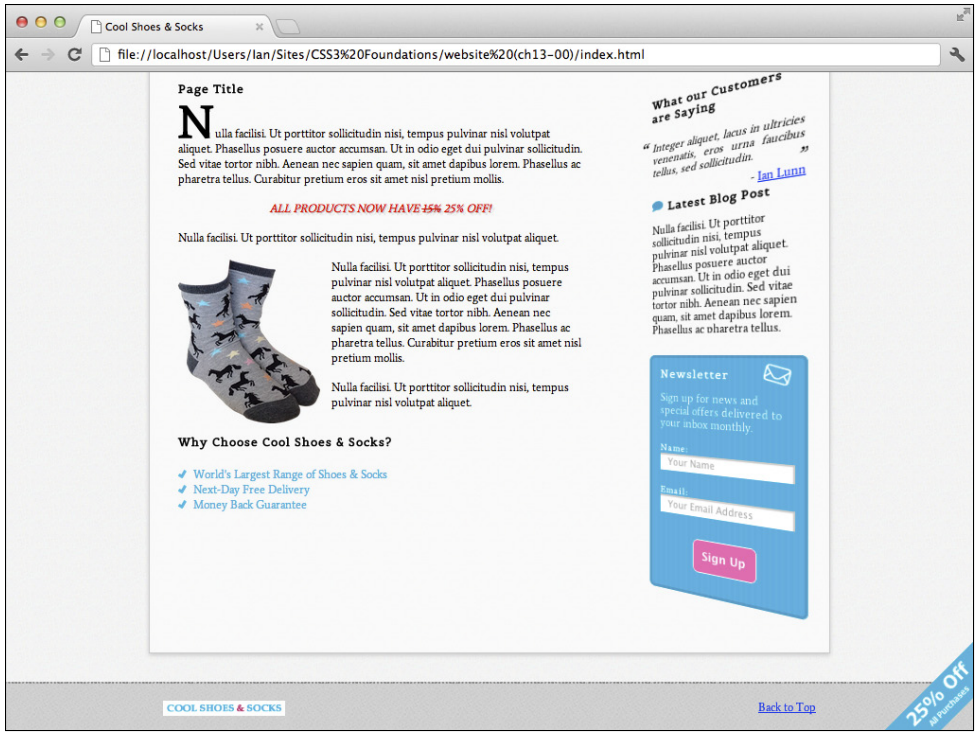


FIGURE 13-7 The Cool Shoes & Socks page zoomed out to show the sidebar rotated on the Y axis by –40 degrees.

Finally, using `rotateZ()` to rotate on the Z axis achieves the same effect as the `rotate()` function in a 2D space. A positive angle rotates an element clockwise; and a negative angle, counterclockwise.

If rotating more than one axis, you may consider using the shorter `rotate3d()` function, which should be given the X, Y, and Z values in that order.

scaleZ() and scale3d()

If you can translate and rotate in 3D, you won't be surprised to read you can scale, too. A quick recap: `scaleX()` scales an element horizontally, and `scaleY()` scales an element vertically. So what about `scaleZ()`?

`scaleZ()` controls depth but, of course, not the depth of the element because an element doesn't have a thickness; it can only be flat. `scaleZ()` works in conjunction with `translateZ()` to scale an element along the Z axis, affecting its visual size (although not its actual dimensions) as if it were moved toward or away from the viewer.

At the moment, when the sidebar is hovered over, it's moved forward 100px by this declaration:

```
-webkit-transform: translateZ(100px);
```

You can increase this visual effect by scaling on the Z axis:

1. In styles.css, find the `.sidebar:hover` rule set and modify the `transform` declaration:

   ```
   -webkit-transform: scaleZ(2) translateZ(100px);
   ```

2. Save styles.css.

This modification scales the sidebar on the Z axis by 2, meaning the size of the element is the same size as if it were sitting on the Z axis at 200px, as shown in Figure 13-8. Again, this effect is over the top, but you can make it a little more sophisticated as you go on.

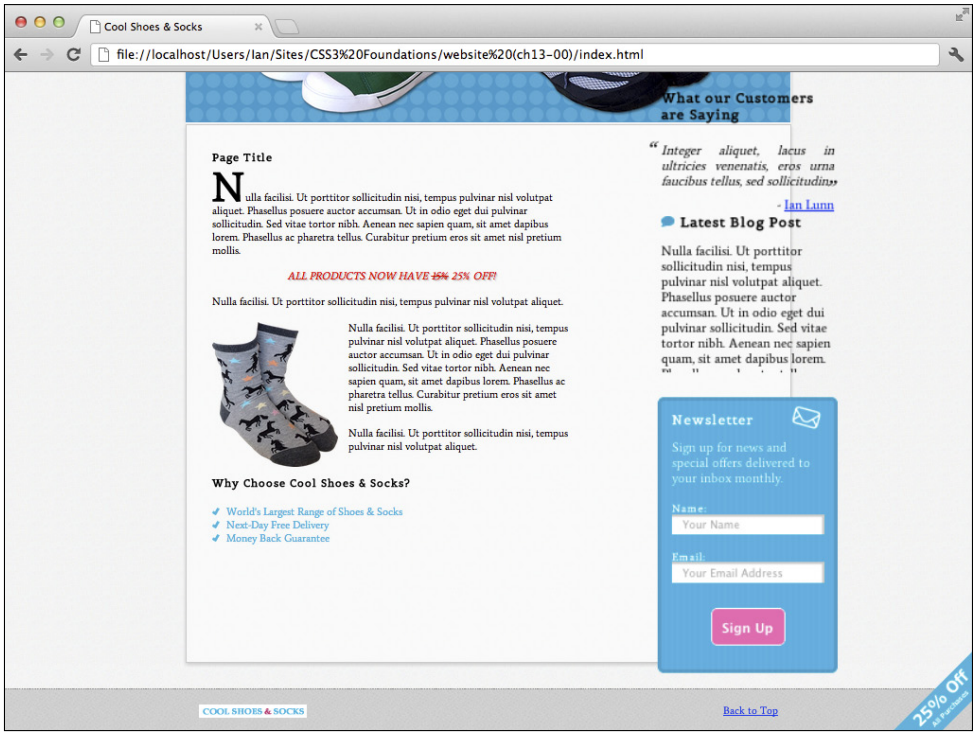


FIGURE 13-8 The sidebar scaled on the Z axis by 2.

A transform such as

```
transform: scaleZ(2) translateZ(100px);
```

creates the same visual effect as

```
transform: translateZ(200px);
```

`scaleZ()` can be given only a number as its argument, which is a multiplication of an element's position on the Z axis.

If you need to scale all three axes, rather than specify three separate functions, `scaleX()`, `scaleY()`, and `scaleZ()`, you can use the function `scale3d()`, which takes three arguments: the X, Y, and Z values.

Multiple 3D Transform Functions

As with 2D `transform` functions, you can use as many as necessary by separating functions with a space, such as

```
transform: rotateY(-40deg) scaleZ(2) translateZ(100px);
```

Note that when you list `transform` functions in this way, a browser treats each as if it is specified separately in the order listed. This effect is particularly important when you're using `ScaleZ()`. The Z axis must be scaled prior to an element being translated on that axis. Therefore,

```
transform: rotateY(-40deg) scaleZ(2) translateZ(100px);
```

scales on the Z axis, whereas

```
transform: rotateY(-40deg) translateZ(100px) scaleZ(2);
```

doesn't scale on the Z axis.

transform-style

Initial value: `flat` | Inherited: No | Applies to: All elements with a `display` declaration relating to block, inline or table | CSS3

Unprefixed browser support: IE 10+, Firefox 16+

Prefixed browser support: Firefox 10+, Chrome 12+, Safari 4+

When an element is given a `perspective` declaration—making it a 3D space—only its direct child elements can be manipulated in 3D. The sidebar, which is currently rotated in 3D, is the direct child of the main content container `<div id="main" class="group">`.

Child elements of that sidebar are treated as having a `transform-style` of `flat`. This is the reason you couldn't apply the `translateZ()` and `scaleZ()` functions only to the newsletter box previously. With a `transform-style` of `flat`, the newsletter box wouldn't have been affected by those transforms because it doesn't exist in a 3D space yet.

Now tell the browser that the sidebar should pass down its 3D space to its child elements (each `<aside>` element):

1. In the `.sidebar` rule set, add the following declaration:

   ```
   -webkit-transform-style: preserve-3d;
   ```

2. Save styles.css.

   By doing this, you can now manipulate the child elements of the sidebar in the same 3D space, something you couldn't do before, and this is the reason you applied the 3D effects to the whole sidebar instead of just the newsletter box.

   Make it so that the "What Our Customers Are Saying" and "Latest Blog Post" sections rotate only when hovered over and the newsletter box moves toward the viewer.

3. Change the `.sidebar:hover` selector to `.sidebar aside:nth-child(3):hover`.

4. Modify the `transform` declaration in the `aside:nth-child(3):hover` rule set to include `rotateY()` and `translateX()` functions:

   ```
   -webkit-transform: rotateY(40deg) translateX(-50px) scaleZ(2)
     translateZ(100px);
   ```

5. Below the `.sidebar` rule set, add a new rule set:

   ```
   .sidebar aside:hover {
       -webkit-transform: rotateY(40deg);
   }
   ```

6. Save styles.css.

Now, when the individual sections of the sidebar are hovered over, they rotate 40 degrees, back to a flat position so the user can read the contents in 2D. The newsletter box also rotates, but it moves toward the user in a 3D space. By making the newsletter box "pop" off the screen, you can hope that it will persuade the user to consider signing up for the Cool Shoes & Socks newsletter.

In the next chapter, you apply smooth animations to these transforms.

backface-visibility

Initial value: `visible` | Inherited: No | Applies to: All elements with a `display` declaration relating to block, inline or table | CSS3

Unprefixed browser support: IE 10+, Firefox 16+

Prefixed browser support: Firefox 10+, Chrome 12+, Safari 4+

As mentioned previously, a 180-degree rotation on the X and Y axes would "flip" an element—as if somebody picked it up and turned it over. Try this on the `.sidebar` rule set, for example:

```
-webkit-transform: rotateY(180deg);
```

You can see the sidebar from the back, as shown in Figure 13-9.

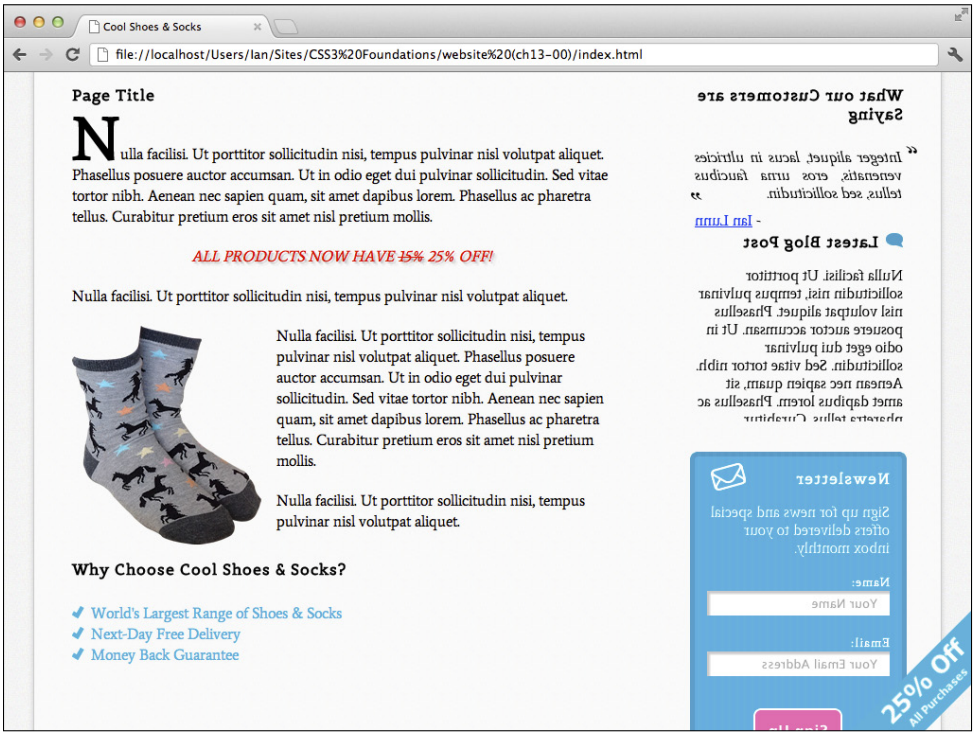


FIGURE 13-9 The sidebar "flipped" 180 degrees, so it displays in reverse.

You have control over whether the contents of an element are shown when they're viewed from the back like this, via the `backface-visibility` property. By default, all transformable elements have a visible backface; however, if you want to hide an element when it is showing its backface, you can use the following:

```
-webkit-backface-visibility: hidden;
```

Summary

In this chapter, you learned how to go beyond the traditional 2D layout of a web page and move elements in a 3D space. The `transform` property has a huge amount of potential with all the 2D and 3D functions it offers. Because this is a reasonably new CSS feature, the web is only beginning to see what creative possibilities can be made with the power of 3D transforms.

Not all browsers support transforms, but that's not a problem; the visual effects you added in this chapter simply don't show in nonsupporting browsers, and the content still appears perfectly readable and accessible, as it did prior to adding these transforms.

The fun with transforms doesn't stop here. Possibly the best thing about them is that they can be animated, too! You do just that in the next chapter.

chapter fourteen

Bringing Your Website to Life with Transitions and Animations

POSSIBLY CSS3'S MOST exciting features are transitions and animations. Transitions allow the values of properties to change smoothly over a specific duration, meaning the color of an element can fade from red to blue, slide from one area of the page to another, shrink and grow when hovered over, and so on. The possibilities are endless.

Whereas transitions change an element from a start to end state, animations allow you to have more control over those states, specifying keyframes, the exact moments when a property should change.

Project files update (ch14-00): If you haven't followed the previous instructions and are comfortable working from here onward or would like to reference the project files up to this point, you can download them from `www.wiley.com/go/treehouse/css3foundations`.

Animating Elements from A to B Using Transitions

To demonstrate transitions, make the "25% Off" banner transition in size when it's hovered over. Before you add transitions to the element though, first set up the start and end states of the transition:

1. In styles.css, find the `.banner-ad` rule set and modify the `transform` declaration to reduce the `scale()` function, as follows:

```
-webkit-transform: translate(75px, -25px) rotate(-45deg)
  scale(.8);
```

2. Below the `.banner-ad` rule set, add a new rule set:

```
.banner-ad:hover {
    cursor: default;
    -webkit-transform: translate(75px, -25px) rotate(-45deg)
  scale(1.1);
}
```

3. Save styles.css.

As yet, these changes haven't added a transition to the banner. In the first step, you make the banner scale down to 80%. Then by adding the `.banner-ad:hover` rule set, you make it so that the banner will scale up to 110% when hovered over. Think of the first rule set as the start state and the `:hover` rule set as the end state of a transition. You are telling the browser how the banner should look at the start and end. Now you need to tell it how to transition between those two states using several `transition` properties.

transition-property

Initial value: `all` | Inherited: No | Applies to: all elements | CSS3

Unprefixed browser support: IE 10+, Firefox 16+, Opera 12.5+

Prefixed browser support: Firefox 4+, Chrome 4+, Opera 10.5+, Safari 3.1+

Because an element can have many different CSS properties applied to it, when making it transition, you might need to be specific about which property is transitioned. For example, say an element has the following properties:

```
color: blue;
height: 100px;
```

The initial value for `transition-property` is `all`, meaning all animatable properties are animated. You might want only the `color` property to transition, though. The `transition-property` allows you to specify that.

1. In styles.css, in the `.banner-ad:hover` rule set, add the following declaration:

```
-webkit-transition-property: -webkit-transform;
```

2. Save styles.css.

The `-webkit-transform` value refers to the property in the same rule set, which you added a moment ago:

```
-webkit-transform: translate(75px, -25px) rotate(-45deg)
  scale(1.1);
```

Although you've now told the browser you only want the `transform` declaration to animate, it won't animate until a duration is also specified, which you'll do in a moment.

The only other property in this rule set is `cursor: default;`, which is a property that can't be animated anyway. Technically, you could get away with not including this declaration in the rule set because the initial value of `all` selects only the `transform` property anyway. Of course, for maintainable CSS, it's better practice to specify exactly which properties you want to animate in case you add more properties in the future that you may not want to have animated.

If you want to specify more than one property, separate each with a comma, like so:

```
transition-property: transform, color;
```

For a list of properties that can be animated, see the list of animatable properties at `www.w3.org/TR/css3-transitions/#animatable-properties`.

transition-duration

Initial value: `0s` | Inherited: No | Applies to: all elements | CSS3
Unprefixed browser support: IE 10+, Firefox 16+, Opera 12.5+
Prefixed browser support: Firefox 4+, Chrome 1+, Opera 11.6+, Safari 3+

Now that the browser knows you want to have the `transform` property transition, you need to tell it how long that transition should take—the time between going from the start to end states.

1. In the `.banner-ad:hover` rule set, add the following declaration:

   ```
   -webkit-transition-duration: 1s;
   ```

2. Save styles.css.

Here, you specify a one-second duration between the start and end states. Hover over the banner in your browser, and you see this happen (see Figure 14-1). Cool, right? The banner now transitions from an 80% scale to 110% over a one-second period.

Before:

After:

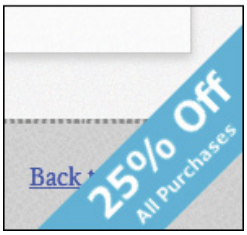


FIGURE 14-1 The start and end states of the banner.

The `transition-duration` is specified in seconds although if you want a millisecond value, you can place a decimal point before the value; that is, `.1s` is the equivalent of 100 milliseconds.

transition-timing-function

Initial value: `ease` | Inherited: No | Applies to: all elements | CSS3
Unprefixed browser support: IE 10+, Firefox 16+, Opera 12.5+
Prefixed browser support: Firefox 4+, Chrome 1+, Opera 11.6+, Safari 3+

Although you've now achieved the transition on the banner, you can use a few more properties to have even more control over the effect. The `transition-timing-function` property describes how the intermediate values used during a transition are calculated—in other words, how the bit between the start and end acts.

The initial value for `transition-timing-function` is `ease`, and there are a lot of other keyword values you can use with it, too:

- `ease`—This timing function starts gently (not quite slow), speeds up in the intermediate, and then ends slow.
- `linear`—Start, intermediate, and end are all at the same constant speed.
- `ease-in`— This timing function starts slow, speeds up in the intermediate, and continues until end.

- `ease-out`—Start and intermediate are the same speed, then it slows down at the end.
- `ease-in-out`— This timing function starts slow, speeds up in the intermediate. and slows down at the end.
- `step-start`— This timing function jumps immediately to the end and stays there.
- `step-end`— This timing function stays in its start state until the end and then jumps immediately to the end state.

There are also some advanced functions you can use to *really* have complete control over the transition: `steps()` and `cubic-bezier()`.

These functions aren't covered in *CSS3 Foundations* because they are a little too advanced, but if you want to know more, visit `www.w3.org/TR/css3-transitions/#transition-timing-function-property`. If you want to experiment with `cubic-bezier()`, which essentially allows you to create the keyword values and everything in between, I recommend visiting `http://cubic-bezier.com/`, which allows you to easily create your own timing functions.

transition-delay

Initial value: `0s` | Inherited: No | Applies to: all elements | CSS3
Unprefixed browser support: IE 10+, Firefox 16+, Opera 12.5+
Prefixed browser support: Firefox 4+, Chrome 1+, Opera 11.6+, Safari 3+

The final `transition` property is `transition-delay`, which allows you to specify a delay that occurs prior to the transition starting. Its initial value of `0s` means a transition will start immediately. A `transition-delay` of `1s`, for example, causes the browser to wait one second prior to starting a transition. As with `transition-duration`, if you would like to use an amount less than one second, you can place a decimal point before the value to specify milliseconds.

With `transition-delay`, you also are able to specify a negative value. Assuming you give the banner a `transition-delay` of `-.5s`, the transition does not occur for half a second and then appears to begin partway through—from half a second onward.

transition (Shorthand)

Unprefixed browser support: IE 10+, Firefox 16+, Opera 12.5+
Prefixed browser support: Firefox 4+, Chrome 1+, Opera 11.6+, Safari 3+

To make working with transitions easier, you can combine the transition-related properties into one easy-to-use declaration using the `transition` property.

1. In styles.css, in the `.banner-ad:hover` rule set, remove the following declarations:

```
-webkit-transition-property: -webkit-transform;
-webkit-transition-duration: 1s;
```

2. In their place, add the shorthand declaration:

```
-webkit-transition: -webkit-transform 1s;
```

3. Save styles.css.

The shorthand property should take the following syntax:

```
transition: transition-property transition-duration transition-
  timing-function transition-delay;
```

Because you don't use `transition-timing-function` and `transition-delay` properties here, you can omit them from the shorthand declaration. Note that because `transition-duration` and `transition-delay` can be given the same type of value, the order in which they are specified is important. The first value is treated as the duration; and the second, the delay.

When applying the transition to more than one property, you should separate values of the transition shorthand property with a comma. For example,

```
transition: transform 1s ease 0s, color 2s linear 0s;
```

This declaration causes the `transform` property in a rule set to transition for one second with an ease and the `color` property to transition for two seconds with a linear timing function. Neither property has a delay.

Making the Banner Transition Back to Its Normal State

Now that the banner is transitioning just the way you like when it's hovered over, you may notice it snaps back to its start state when the cursor is moved away from it. How to make it transition back? All you need to do is apply the same `transition` declaration to the `.banner-ad` rule set:

1. In the `.banner-ad` rule set, add the following declaration:

```
-webkit-transition: -webkit-transform 1s;
```

2. Save styles.css.

Code Challenge: Make the Sidebar Sections Transition

In styles.css, do the following:

1. In the `.sidebar aside:hover` rule set, add the declaration `-webkit-transition: -webkit-transform .5s;`.
2. Add a new rule set for `.sidebar aside` below the `.sidebar` rule set with the declaration `-webkit-transition: -webkit-transform .5s;`.

Project files update (ch14-01): If you haven't followed the previous instructions and are comfortable working from here onward or would like to reference the project files up to this point, you can download them from `www.wiley.com/go/treehouse/css3foundations`.

Animating Elements from A to Z Using Keyframes

The transitions you just added to the page make an element smoothly change from one property value to another over a specific period of time. These transitions occur when an event is triggered, such as a user hovering over an element.

Animations also make an element smoothly change between different property values, but unlike transitions, they don't have to just consist of a start and end state. Using keyframes, you can create points that an animation should reach before changing again, allowing much more control over the effect onscreen. If an animation is 10 seconds long, you can specify exactly what happens over that period of time, maybe changing an element's color to blue at 2 seconds, green at 6 seconds, and black at 10 seconds. Animations can also be triggered by events or can be started when the page is loaded, with options to allow you to decide how many times an animation should repeat.

To demonstrate animations, you turn the product showcase (the section with the image of some trainers and a More Information button) into a set of images that cycle from one to the other.

Set up the basics first:

1. In index.html, find the following code:

```
<div class="showcase">
    <img src="images/banner-trainers.jpg"
alt="Some of our cool trainers" />
    <a class="button purchase" href="#" title="Purchase
 product">
      More Information
```

```
    </a>
</div>
```

2. Replace it with the following:

```
<div class="showcase">
    <ul>
        <li>
            <img src="images/banner-trainers.jpg"
alt="Some of our cool trainers" />
        </li>
        <li>
             <img src="images/banner-shoes.jpg"
alt="Some of our cool shoes" />
        </li>
    </ul>
    <a class="button purchase" href="#" title="Purchase
  product">
        More Information
    </a>
</div>
```

3. Save index.html.

   This HTML adds two product images to the product showcase (you add a third later), each in its own list item `<li>`, as shown in Figure 14-2. Using CSS, you place one image over the other so only one can be seen at a time. You then use CSS3 animation properties to make the images cycle between one another.

4. In styles.css, find the `.showcase .button` rule set and increase the `z-index` from 5 to 9.

5. Find the `.showcase .button:hover` rule set, and below it, add these rule sets:

```
.showcase ul {
    background: #009CC8;
    position: relative;
    z-index: 9;
}

.showcase li {
    position: absolute;
    top: 0;
    width: 100%;
}
.showcase li:first-child {
    z-index: 3;
    position: relative;
}
```

```
.showcase li:nth-child(2) {
    z-index: 2;
}
```

6. Save styles.css.

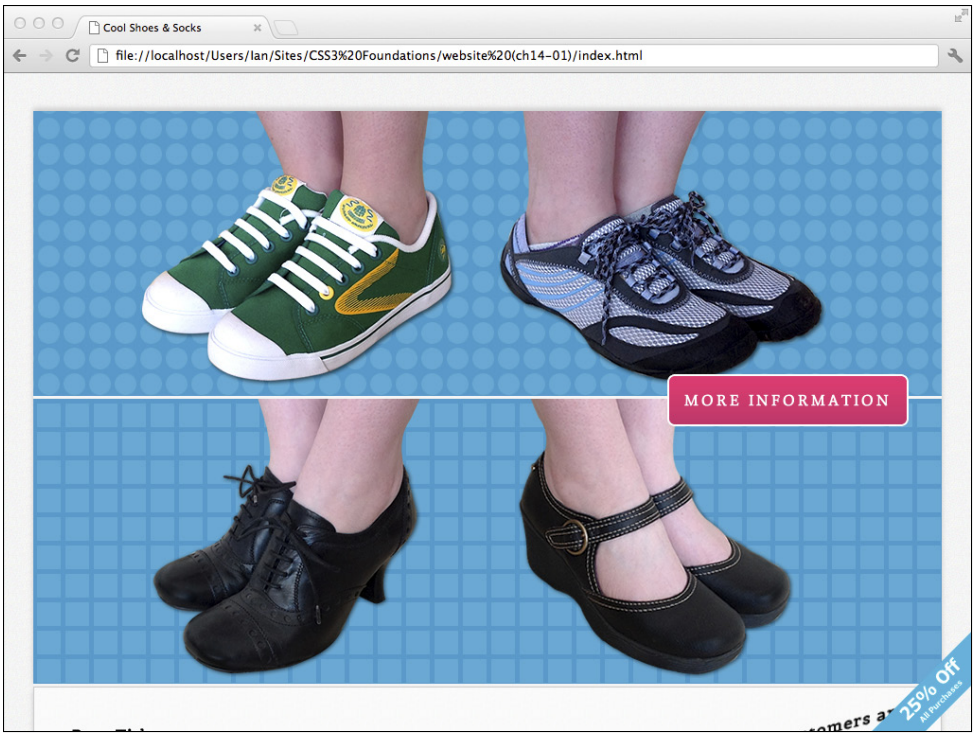


FIGURE 14-2 The Cool Shoes & Socks page zoomed out to show two product images in the showcase, which will be eventually be positioned one over the other.

When you add this CSS, the Cool Shoes & Socks page appears the way it did prior to your adding the new showcase HTML. However, the second product image now sits behind the first. When you make `.showcase ul` a `relative` position, its child elements (the list items `<li>`) can be positioned absolutely. The `z-index` of 9 makes it appear above all positioned elements, apart from the drop-down menu.

Each list item `<li>` is positioned `absolute` so that it doesn't affect the flow of the document and is placed at the top of the showcase (making all list items added to the showcase stack on top of each other).

By using the pseudo-selector `:first-child`, you select the first list item that contains a product image. Because this is the first product image to be displayed, it's given a `z-index` of 3 (you eventually make the product image consist of three images). In the preceding paragraph,

I said that each list item is positioned `absolute` so it doesn't affect the document flow and the items can be stacked on top of each other. If the list items were all left like this, the showcase would disappear because it has no contents affecting it, and its height would become 0 px. To stop this from happening, you position the first list item only as `relative`. This way, it still affects the document's flow, and because it's the first image in the showcase anyway, it is always at the top. Another way to work around this situation is make the position of all list items `absolute` and then specify a height for `.showcase ul`. If you do this, though, the product showcase is always a set height and doesn't shrink with the images when the browser window is resized.

Finally, you use another pseudo-selector, `.showcase li:nth-child(2)`, to select the second list item and place that below the first, using `z-index`.

With this code added, you're good to go! Now to animate the showcase images...

@keyframes

Unprefixed browser support: IE 10+, Firefox 16+, Opera 12.5+

Prefixed browser support: Firefox 5+, Chrome 1+, Opera 12+, Safari 4+

When creating CSS animations, you must declare the states of an animation in a `@keyframes` rule. Because animations are still experimental, you should prefix the `@keyframes` rule, such as `@-webkit-keyframes`.

1. In styles.css, below the `.showcase .button:hover` rule set, add a `@keyframes` rule:

```
@-webkit-keyframes fadeOut {

}
```

   This rule consists of the prefixed `@-webkit-keyframes` keyword and an identifier `fadeOut`, which is the name of the animation you will create; you're free to use whatever identifier you like. The name of the `@keyframes` rule is referenced using the `animation-name` property, which links the animation you're going to create to the element you want it to apply to.

   Within this `@keyframes` rule, you can now place rule sets that act as the states of the animation:

2. In the `@-webkit-keyframes fadeOut` rule, add the following rule sets:

```
from {
    opacity: 1;
}

to {
```

```
        opacity: 0;
    }
```

3. Save styles.css.

The first rule set, `from`, tells the browser an element should begin with an `opacity` of `1` (opaque) and then animate `to` an `opacity` of `0` (transparent). Because the `fadeOut` animation is not yet linked to an element, no elements fade yet.

The `from` keyword represents 0% of an animation (the start), and the `to` keyword represents 100% (the end). An animation that goes from 0% to 100% works in the same way as a transition (changing from one style to another), but within animations, unlike transitions, you can add keyframes anywhere, using percentage values. Later, you use this rule to complete the cycling product showcase:

```
@-webkit-keyframes fadeOut {
    from {
        opacity: 1;
        z-index: 4;
    }

    16.666% {
        opacity: 0;
        z-index: 4;
    }

    50% {
        opacity: 0;
        z-index: -1;
    }

    51%{
        opacity: 1;
    }
}
```

This animation makes an element start with a `z-index` of `4`, and then 16.666% (I explain this percentage later) into the animation, change its `opacity` to `0` (fading out the element). At 50% in, the `z-index` changes to `-1` (moving the element behind other elements in the product showcase), and then at 51% in, the `opacity` changes back to `1` to make the element visible again, ready for the animation to be cycled.

Don't worry if this description goes over your head. For now, stay with the simple `from - to` animation to learn how to apply it to an element and then come back to this more complex animation later.

animation-name

Initial value: `none` | Inherited: No | Applies to: all elements | CSS3
Unprefixed browser support: IE 10+, Firefox 16+, Opera 12.5+
Prefixed browser support: Firefox 5+, Chrome 1+, Opera 12+, Safari 4+

The `animation-name` property links an element to a `@keyframes` rule. The animation you added to styles.css is called `fadeOut`, so reference that now:

1. In styles.css, in the `.showcase li:first-child` rule set, add the following declaration:

   ```
   -webkit-animation-name: fadeOut;
   ```

2. Save styles.css.

That's all there is to it. The first list item now has the `fadeOut` animation applied to it, but the element doesn't animate just yet because it doesn't know how long to animate.

You can also give the `animation-name` the value `none`, in case you need to stop an element from inheriting an animation.

animation-duration

Initial value: `0s` | Inherited: No | Applies to: all elements | CSS3
Unprefixed browser support: IE 10+, Firefox 16+, Opera 12.5+
Prefixed browser support: Firefox 5+, Chrome 3+, Opera 12+, Safari 4+

`animation-duration` works in the exact same way as `transition-duration`. You give it a value in seconds, like so:

1. In `.showcase li:first-child` add the following declaration:

   ```
   -webkit-animation-duration: 5s;
   ```

2. Save styles.css.

With the `animation-duration` added to the first list item, when the page loads, the animation runs, causing the image to go from an opacity of `1` to `0` over a five-second period. As the first image fades out, the second that sits behind it is shown, as shown in Figure 14-3.

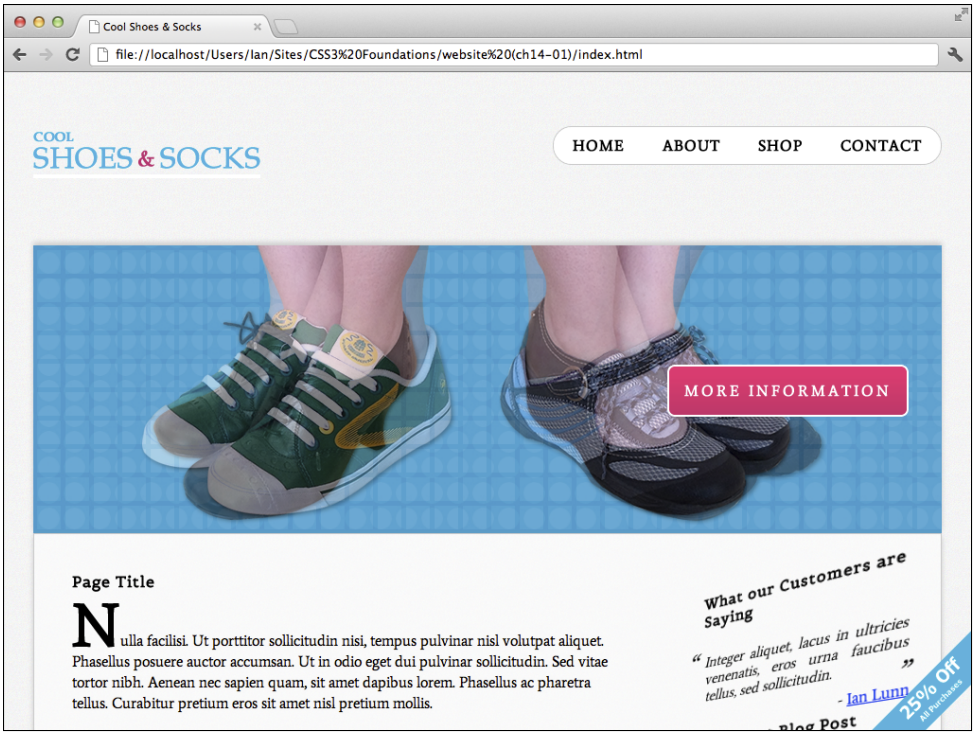


FIGURE 14-3 The first product image midway through fading out, revealing the second product image.

animation-timing-function

Initial value: `ease` | Inherited: No | Applies to: all elements | CSS3

Unprefixed browser support: IE 10+, Firefox 16+, Opera 12.5+

Prefixed browser support: Firefox 5+, Chrome 3+, Opera 12+, Safari 4+

`animation-timing-function` does exactly the same as the `transition-timing-function` property. It describes how the intermediate values used during a transition are calculated. Because animations can repeat multiple times, the `animation-timing-function` describes these intermediate values over each cycle of an animation.

1. In `.showcase li:first-child`, add the following declaration:

   ```
   -webkit-animation-timing-function: ease-in-out;
   ```

2. Save styles.css.

For the full list of timing functions you can use, see the `transition-timing-function` property description earlier in this chapter.

animation-delay

Initial value: `0s` | Inherited: No | Applies to: all elements | CSS3
Unprefixed browser support: IE 10+, Firefox 16+, Opera 12.5+
Prefixed browser support: Firefox 5+, Chrome 3+, Opera 12+, Safari 4+

Just like transitions, animations can also be given a delay, which works in the exact same way.

1. In `.showcase li:first-child`, add the following declaration:

   ```
   -webkit-animation-delay: 5s;
   ```

2. Save styles.css.

Now, when the page is first loaded, the first product image doesn't start fading out until after a five-second delay.

As with `transition-delay`, you can also specify a negative delay that causes an animation to wait for *x* number of seconds and then start *x* number of seconds into that animation.

animation-iteration-count

Initial value: `1` | Inherited: No | Applies to: all elements | CSS3
Unprefixed browser support: IE 10+, Firefox 16+, Opera 12.5+
Prefixed browser support: Firefox 5+, Chrome 3+, Opera 12+, Safari 4+

`animation-iteration-count` determines how many times an animation should occur. You can give it a number or the keyword `infinite` to make an animation cycle with no end. By default, `animation-iteration-count` is `1`.

1. In `.showcase li:first-child`, add the following declaration:

   ```
   -webkit-animation-iteration-count: infinite;
   ```

2. Save styles.css.

When you make the iteration count `infinite`, the animation constantly cycles with no end.

animation-direction

Initial value: `normal` | Inherited: No | Applies to: all elements | CSS3
Unprefixed browser support: IE 10+, Firefox 16+, Opera 12.5+
Prefixed browser support: Firefox 5+, Chrome 3+, Opera 12+, Safari 4+

At the moment, the opacity of the product image is animated to make it fade out, but when the animation cycles back to its start, the image instantly snaps back to opaque, which doesn't look quite right. Using the animation-direction property, you can make the animation reverse when it reaches its end.

1. In .showcase li:first-child add the following declaration:

   ```
   -webkit-animation-direction: alternate;
   ```

2. Save styles.css.

When you alternate the animation direction, odd iterations of the animation cause the product image to fade out, and each even iteration reverses the animation, making the product image fade in, achieving a much more subtle effect.

You can use the following keywords with animation-direction:

- normal (initial value)—All iterations of the animation are played as specified.
- reverse—All iterations of the animation are played in the reverse direction from the way they were specified.
- alternate—The animation cycle iterations that are odd counts are played in the normal direction, and the animation cycle iterations that are even counts are played in reverse.
- alternate-reverse—The animation cycle iterations that are odd counts are played in reverse, and the animation cycle iterations that are even counts are played in a normal direction.

The reverse and alternate-reverse keywords are currently only supported in Internet Explorer 10+, Firefox 16+, and Chrome 19+.

Note that when an animation is reversed, the timing functions are also reversed. So, when played in reverse, an ease-in animation appears to be an ease-out animation.

animation-play-state

Initial value: running | Inherited: No | Applies to: all elements | CSS3

Unprefixed browser support: IE 10+, Firefox 16+, Opera 12.5+

Prefixed browser support: Firefox 5+, Chrome 3+, Opera 12+, Safari 4+

The animation-play-state property determines whether or not an animation is paused. By default, all animations have the value running, but you can also use paused to pause an animation.

Dealing with a WebKit Bug

Due to a bug in WebKit browsers (Google Chrome and Apple Safari), it is recommended that you comment out or do not use the `animation-play-state` property because it doesn't work as expected when used with `animation-delay`. The purpose of vendor prefixes shows their worth here; although the WebKit property doesn't work, you can still use the property for browsers that do, like so:

Example:

```
.showcase ul:hover li {
    /* -webkit-animation-play-state: paused;
  */
    -moz-animation-play-state: paused;
    -ms-animation-play-state: paused;
    -o-animation-play-state: paused;
    animation-play-state: paused;
}
```

1. In styles.css, below the `.showcase li` rule set, add the following rule set:

```
.showcase ul:hover li {
    -webkit-animation-play-state: paused;
}
```

2. Save styles.css.

With this rule set, whenever the product showcase `.showcase ul` is hovered over, each list item that contains an image is paused.

animation-fill-mode

Initial value: `none` | Inherited: No | Applies to: all elements | CSS3
Unprefixed browser support: IE 10+, Firefox 16+, Opera 12.5+
Prefixed browser support: Firefox 5+, Chrome 3+, Opera 12+, Safari 4+

The last property to be used with animations is `animation-fill-mode`, which defines what values are applied by the animation outside the time it is executing.

The initial value for `animation-fill-mode` is `none`, meaning properties defined in a `@keyframes` rule apply only to an element while it is being animated.

Before you added the `animation-direction` property with a value of `alternate`, the animation just snapped back to the beginning when it reached its end, which looked a little messy. Sometimes you just want an animation to run on an element and then stay that way after the animation finishes. You can achieve this effect by using the `animation-fill-mode` value `forwards`. This value tells the browser that after an animation ends, the final styles applied to the element in that animation should continue forward, outside the animation's execution time.

Likewise, you can use an `animation-fill-mode` value of `backwards` to have the first values of an animation applied to an element before the animation begins, or a value of `both`, which combines the `forwards` and `backwards` values.

Because the `fadeOut` animation you currently have set up is infinite (it has no end), changing the `animation-fill-mode` doesn't make a difference, but you use it later to make the cycling image showcase better.

animation (Shorthand)

Unprefixed browser support: IE 10+, Firefox 16+, Opera 12.5+

Prefixed browser support: Firefox 5+, Chrome 3+, Opera 12+, Safari 4+

In total, eight properties relate to animations, and for this reason, they can be combined into the shorthand property `animation`—all except `animation-play-state` because, as you saw, that tends to get added to a different rule set, such as `:hover`.

The shorthand property should take the following syntax:

```
animation: animation-name animation-duration animation-timing-
  function animation-delay animation-iteration-count
  animation-direction animation-fill-mode
```

Just like the `transition` shorthand, because some values can be the same (such as `animation-duration` and `animation-delay`), the order in which the `animation` property is formed is important.

1. In styles.css, remove the six `animation` properties from `.showcase li:first-child` and replace with the following shorthand `animation` declaration:

   ```
   -webkit-animation: fadeOut 5s ease-in-out 5s infinite
     alternate;
   ```

2. Save styles.css.

Using this shorthand doesn't change the page visually in any way; it's just a nicer and more maintainable way to write animations.

Project files update (ch14-02): If you haven't followed the previous instructions and are comfortable working from here onward or would like to reference the project files up to this point, you can download them from `www.wiley.com/go/treehouse/css3foundations`.

Creating a Cycling Image Showcase

You've learned how to use the animation properties, but the cycling image showcase consists of only two images and doesn't particularly look that great yet. Now take a look at a slightly more advanced animation technique.

Start by adding a third image to the product showcase:

1. In index.html, look for the following HTML:

```
<li>
    <img src="images/banner-shoes.jpg"
alt="Some of our cool shoes" />
</li>
```

2. Below that HTML, add another list item:

```
<li>
    <img src="images/banner-socks.jpg"
alt="Some of our cool socks" />
</li>
```

3. Save index.html.

Because the current `fadeOut` animation only hides and shows the top image over a five-second period, this newly added third image won't ever be shown with the animation as it is.

1. In styles.css, below the `.showcase li:nth-child(2)` rule set, add a new rule set for the third product:

```
.showcase li:nth-child(3) {
    z-index: 1;
}
```

2. Remove the following declaration from `.showcase li:first-child`:

```
-webkit-animation: fadeOut 5s ease-in-out 5s infinite
  alternate;
```

   Here, you remove the animation from the first product so that you can apply an animation to all the images; that way, they each fade out.

3. In the `.showcase li` rule set, add the following declaration:

```
-webkit-animation: fadeOut 15s ease-in-out 5s infinite
  forwards;
```

4. Save styles.css.

   This animation is modified somewhat. It still refers to the `fadeOut` `@keyframes` animation but now spans over a 15-second period and no longer alternates (goes in reverse when it reaches the end). The animation also has a 5-second delay. Because this animation applies to all the products in the showcase, they all fade out at the same time. Now make it so that they fade out at different times.

5. In the `.showcase li:nth-child(2)` rule set, add the following declaration:

   ```
   -webkit-animation-delay: 10s;
   ```

6. In the `.showcase li:nth-child(3)` rule set, add the following declaration:

   ```
   -webkit-animation-delay: 15s;
   ```

7. Save styles.css.

   A useful feature of CSS is that you can apply the shorthand `animation` property to multiple elements but then overwrite individual values of `animation`, as you do here—overwriting the `animation-delay`. The second product image now has a 10-second delay, and the third has a 15-second delay.

   Finally, you need to adjust the animation itself to get the product images fading out one at a time.

8. In the `@-webkit-keyframes fadeOut` rule, replace the `from` and `to` rule sets with the following:

   ```
   from {
       opacity: 1;
       z-index: 4;
   }

   16.666% { /*2.5s*/
       opacity: 0;
       z-index: 4;
   }

   50% { /*7.5s*/
       opacity: 0;
       z-index: -1;
   }

   51%{
       opacity: 1;
   }
   ```

9. Save styles.css.

With this animation now added to the stylesheet, each product image is visible for 5 seconds and then takes 2.5 seconds to fade out (revealing the next image).

I added comments to the `16.666%` and `50%` rule sets to show that they are active at 2.5 seconds and 7.5 seconds into the animation, but because there's quite a lot going on here, let's take a closer look at what's happening (see Figure 14-4).

After the initial 5-second delay on the first product (which occurs only once, when the page loads), the `from` rule set is activated, and the first product is placed on top of the others and made opaque. From the start of the animation, the first product image begins to fade until it is completely transparent at 16.666% into the animation (16.666% of 15 seconds is roughly 2.5 seconds).

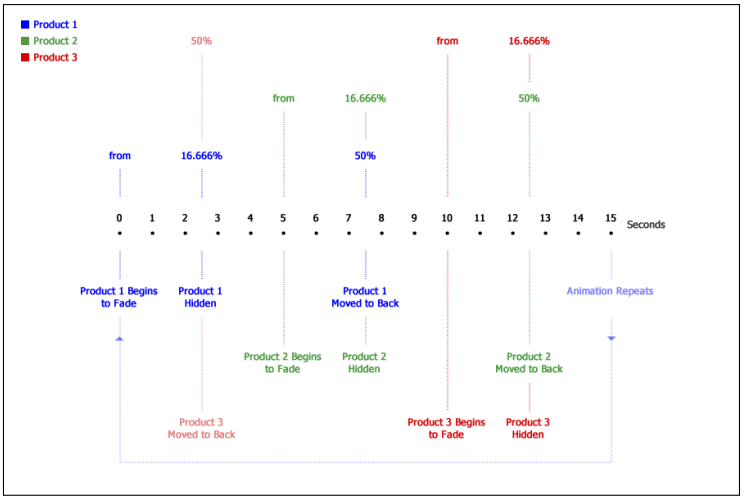


FIGURE 14-4 The events of the animation over a 15-second period.

So, at 2.5 seconds into the animation, the first product is now transparent, which means the second is showing. For another 2.5 seconds, the second image is displayed until it too begins to fade.

At 50% into the animation (7.5 seconds), the third image is displayed because the second product image is transparent, and the first image is moved behind them both.

Just after the first image is moved to the back, at 51%, is it then made opaque, ready and waiting for the third image to fade and reveal it, completing one cycle of the animation.

Summary

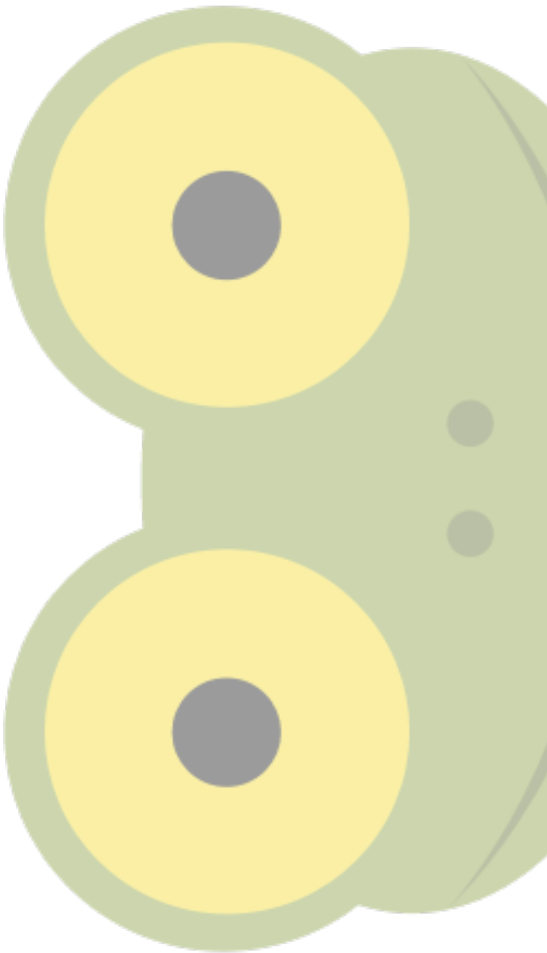
CSS3's transitions and animations are great additions to the technology you have for creating web pages. The most exciting aspect is that because they're so new, there are many cool things to be done with them yet to be discovered; you just need some imagination!

Unfortunately, transitions and animations aren't supported in all browsers, but because they only add an extra edge to the page, the fact that some browsers don't understand them doesn't matter. You can safely use them in the knowledge that in their absence they don't cause a page to look broken.

In the next chapter, you look at the Cool Shoes & Socks page you created using other web browsers and begin fixing their inconsistencies to make the page cross browser compatible.

part 6

Preparing for Multiple Browsers and Devices



chapter fifteen

Testing Across Multiple Browsers

THE FIRST THING to do to make your website cross browser compatible is to get yourself a glass of water (to drink, not to throw over the computer) and take a deep breath. You've spent countless hours trying to make your website perfect. You've used best practices and been extra careful to use the latest CSS3 features safely. But when you open a browser you haven't developed in—especially older browsers such as Internet Explorer versions 6 and 7—you're going to see broken bits, sometimes just a few broken bits, other times lots. Fear not, this happens to everyone and is just the nature of creating web pages. Usually, a broken page looks worse than it actually is, and you can fix it in no time.

Obviously, every web page is different, so how you go about achieving cross browser compatibility is unique to each page. The most universal approach is to follow best practices—keeping your code clean and semantic, which you've done throughout *CSS3 Foundations*.

In this chapter, you start by making sure the Cool Shoes & Socks page works in modern browsers, add the CSS3 properties that need to be prefixed for other browsers, and then move on to add fixes for Internet Explorer versions 6, 7, and 8.

Project files update (ch15-00): If you haven't followed the previous instructions and are comfortable working from here onward or want to reference the project files up to this point, you can download them from `www.wiley.com/go/treehouse/css3foundations`.

Vendor Prefixing the Easy Way

Until now, you've used vendor prefixes only on CSS3's experimental properties to save having to write out the same property multiple times.

I developed the Cool Shoes & Socks page in Google Chrome, and now that it's complete, I want to take a look at it in Mozilla Firefox. If you used Firefox to create Cool Shoes & Socks, take a look at it in another modern browser, such as Chrome, Safari, or Opera.

Pleasingly, Cool Shoes & Socks doesn't look too bad at all in Firefox. Figure 15-1 shows that the page has a few issues, but they all relate to the fact that the stylesheet uses experimental CSS3 properties, which is yet to include those properties with a `-moz-` prefix—or any other vendor prefix for that matter. Without the full set of vendor prefixes, the showcase doesn't animate, the input fields in the newsletter box are too wide (because of no prefixed `box-sizing` property), and the "25% Off" banner isn't rotated.

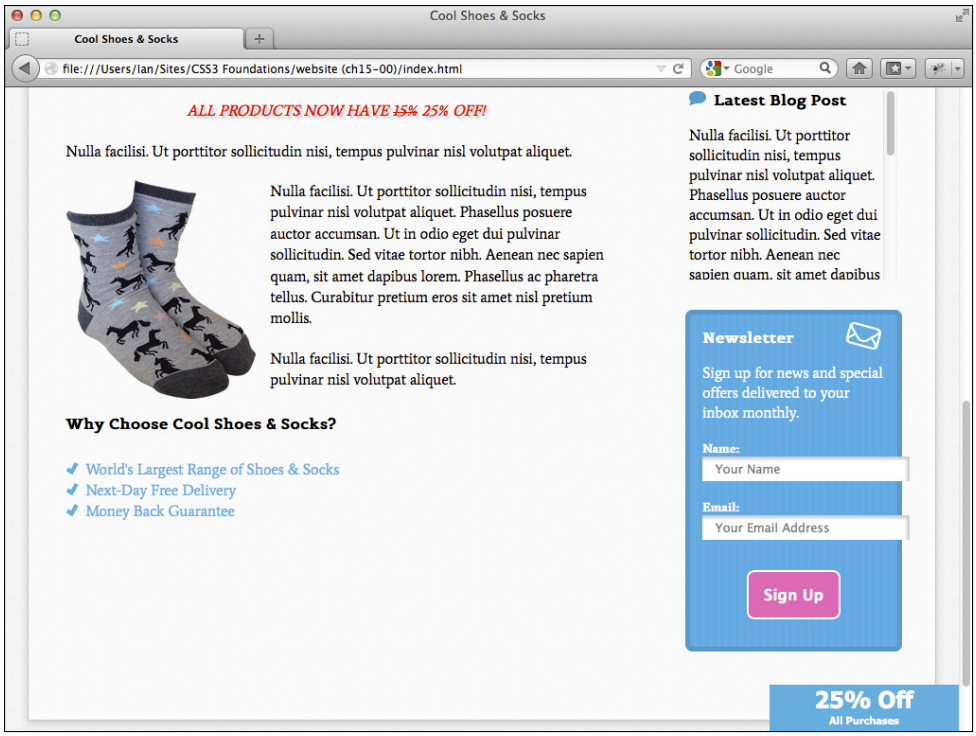


FIGURE 15-1 The Cool Shoes & Socks page viewed in Mozilla Firefox.

In Chapter 3, I mentioned the free tools available to speed up writing prefixed properties. My personal recommendation is `www.prefixr.com/`.

Using Prefixr, you can have the prefixed properties you wrote for the Cool Shoes & Socks page converted into those that you purposely did not write to save yourself some time.

1. In styles.css, highlight and copy all the text from the `body` rule set onward.
2. Visit `www.prefixr.com/` in your browser.
3. Click the text area on the Prefixr website and delete any content within it.
4. Paste your copied stylesheet into the empty text area, as shown in Figure 15-2.

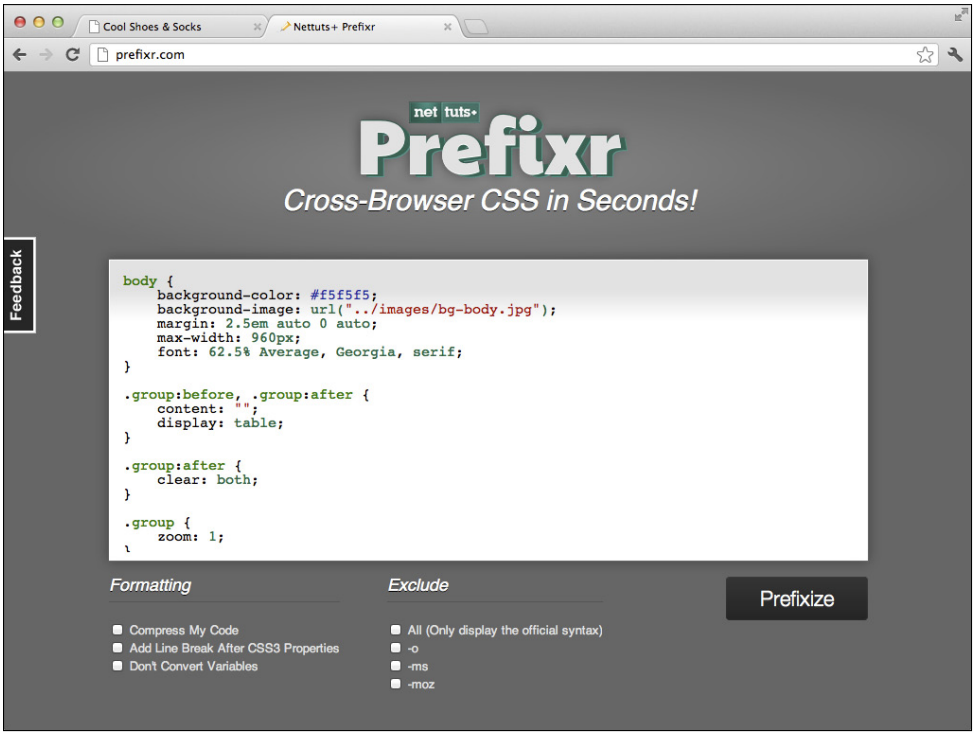


FIGURE 15-2 The #main rule set copied into the prefixr.com text area.

5. Uncheck the Add Line Breaks After CSS3 Properties option (all options should be unchecked).
6. Click the Prefixize button.

   Prefixr then returns the stylesheet with any necessary prefixed properties included.
7. Highlight and copy this modified stylesheet from the Prefixr text area.
8. Paste the modified stylesheet in place of the existing one.
9. Save styles.css.

Prefixr—or any other automatic CSS prefixer tool for that matter—doesn't quite do 100% of the job, so you need to make a few changes manually. Making these manual changes, however, is much less work than having to write all the vendor prefixes yourself. Let's look at what Prefixr *does* and *doesn't* do:

1. In styles.css, find the `#main` rule set:

```
#main {
    background: white;
    background: rgba(255, 255, 255, 0.6);
    border: #ccc solid 1px;
    -webkit-box-shadow: 0 3px 8px 0 #ccc;
    box-shadow: 0 3px 8px 0 #ccc;
    padding: 2.5em;
    -webkit-perspective: 1000px;
}
```

   In Chapter 5, you added the `box-shadow` declaration to this `#main` rule set. As you can see, Prefixr automatically adds the prefixed `-webkit-box-shadow` declaration for you. You can use the `box-shadow` property in all modern browsers without a prefix, but some mobile browsers that use the WebKit layout engine still require the prefixed property, so Prefixr adds it.

   Prefixr unfortunately doesn't automatically prefix the `perspective` property, so you can do that manually.

2. After the `-webkit-perspective: 1000px;` declaration, add the following:

```
-moz-perspective: 1000px;
-ms-perspective: 1000px;
-o-perspective: 1000px;
perspective: 1000px;
```

3. Save styles.css.

   Note that just as Prefixr does, you should always place the official, nonprefixed property below the prefixed versions; that way, the official property will be used over the unofficial properties as soon as a browser supports it.

   The next rule set that Prefixr works its magic on is `.showcase .button`. As you can see, this tool takes the work out of having to write the gradient applied to the showcase button:

```
background: -webkit-linear-gradient(top, #FB3876 0%, #d4326d
  100%);
background: -moz-linear-gradient(top, #FB3876 0%, #d4326d
  100%);
background: -o-linear-gradient(top, #FB3876 0%, #d4326d 100%);
background: -ms-linear-gradient(top, #FB3876 0%, #d4326d 100%);
background: linear-gradient(top, #FB3876 0%, #d4326d 100%);
```

No action is required here, but I'm sure you'll agree, not having to write out all these properties with the heavy syntax of `linear-gradient()` is a timesaver.

Just below the `.showcase .button` rule set, you see the `@keyframes` rules. Prefixr takes the `@-webkit-keyframes` rule and makes one for each browser, including a nonprefixed `@keyframes` rule. Unlike other rule sets, the keyframes rules are only referenced, so their position within a stylesheet isn't essential. Prefixr chooses to place the official `@keyframes` rule above the prefixed versions, which isn't a problem.

As mentioned in Chapter 5, when you use the `opacity` property—which isn't supported by Internet Explorer versions 6, 7, and 8—Prefixr automatically adds the `filter` property, which emulates `opacity`. You used `opacity` in the `@keyframes` rules, but because old versions of Internet Explorer don't understand `@keyframes` anyway, the `filter` property can safely be removed. The `filter` properties don't actually cause any harm if they remain in the stylesheet, so this step is optional, but always trying to keep your stylesheets as clean as possible is good practice.

4. Remove any instance of the `filter` and `-ms-filter` properties from styles.css.

   As you make your way through the stylesheet, you see that Prefixr prefixes many more properties too:

   - The `.showcase li` animations from the preceding chapter
   - The `box-sizing` property in `input[type="text"],input[type="email"]`
   - The `transform` property added to several of the `.sidebar` elements
   - The `transform`, `transform-origin`, and `transition` properties added to the `.banner-ad` rule set

   None of these modifications require your action. However, Prefixr slightly tampers with the content declarations in the `blockquote p:before` and `blockquote p:after` rule sets. The `content` properties originally have the values `"\201C"` and `"\201D"`, but Prefixr strips out the backslashes, so you need to add them back in.

5. In `blockquote p:before`, add a backslash inside the quotation marks:

   ```
   content: "\201C";
   ```

6. In `blockquote p:after`, add a backslash inside the quotation marks:

   ```
   content: "\201D";
   ```

   Finally, because Prefixr also strips out some of the prefixed `border-image` properties in the `#footer` rule set, add those back in.

7. In the `#footer` rule set, below the `-moz-border-image` declaration, add the following:

   ```
   -ms-border-image: url("../images/bdr-footer.png") 4 repeat;
   -o-border-image: url("../images/bdr-footer.png") 4 repeat;
   ```

8. Save styles.css.

All in all, when you run a stylesheet through Prefixr, you need to do a little work to tidy things up afterward, but overall you save some time and, most importantly, your sanity. Writing those prefixed properties isn't fun!

Testing Modern Browsers

Now that you've prefixed all the necessary CSS3 properties, how do those changes help with the cross browser compatibility of Cool Shoes & Socks?

When testing cross browser compatibility, look for visual issues, but also test the functionality of the page. Hover over the drop-down menus and other elements with `:hover` rules, make sure the showcase animation runs correctly, and also resize your browser to make sure everything scales correctly.

Firefox 13 and Safari 5

Now the stylesheet contains the necessary prefixed properties, and due to modern browsers all closely supporting the CSS specifications, achieving browser consistency isn't particularly hard. In Figure 15-3, you can see that Firefox now displays the page just as Chrome does. Safari 5 also renders the page just as Chrome and Firefox do.

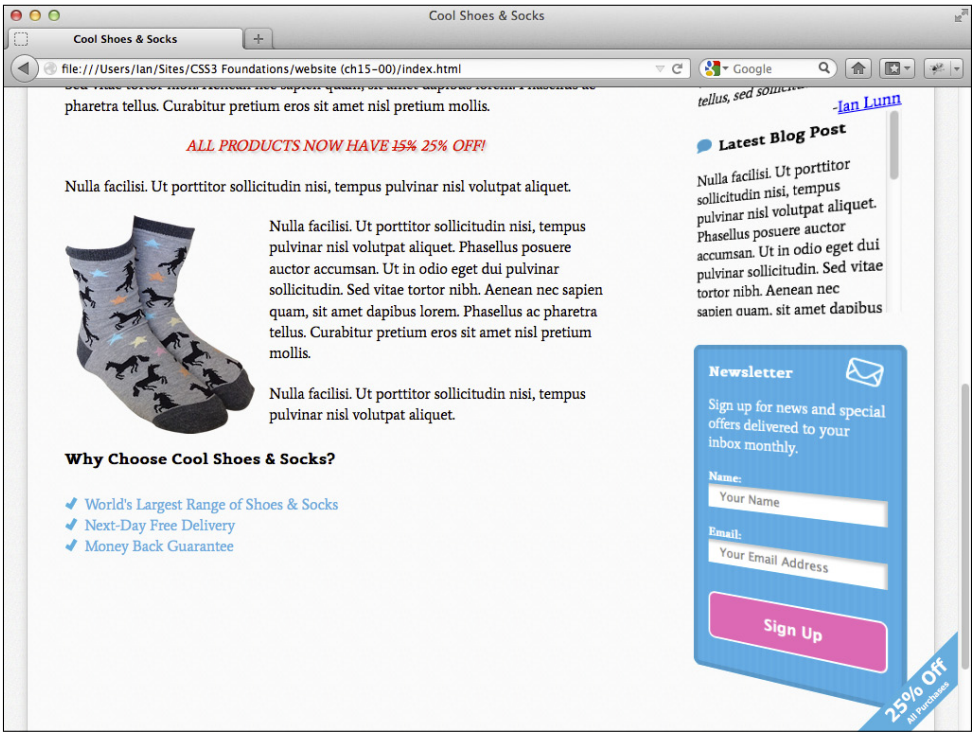


FIGURE 15-3 The Cool Shoes & Socks web page viewed in Firefox 13.

Opera 11 and 12

When viewed in Opera 11 and 12, Cool Shoes & Socks doesn't provide quite the same experience as the other browsers, due to Opera 11 and 12 not supporting `@keyframes` or 3D transforms, as shown in Figure 15-4. This is not a problem, though; Opera still displays the web page perfectly well, and the code exists in the stylesheet for the time when Opera *does* decide to support 3D transforms, making the page future proof.

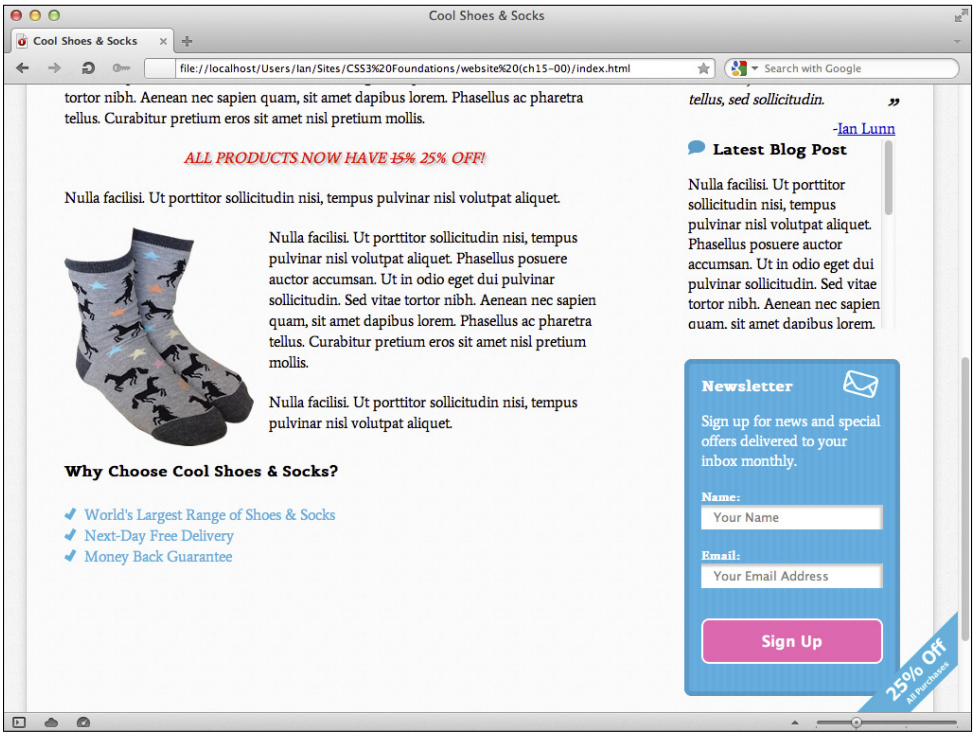


FIGURE 15-4 The Cool Shoes & Socks web page viewed in Opera 12.

Internet Explorer 10

At the time of writing, Microsoft Internet Explorer 10 has not been officially released but has been made available for testing. Visually, Internet Explorer 10 has no issues, but there is an issue with the 3D transform applied to the sidebar when hovered over, as shown in Figure 15-5.

Internet Explorer 10 unfortunately doesn't support `transform-style: preserve-3d` yet, which, as shown in Figure 15-5, is why the newsletter box appears to be positioned strangely when hovered over, causing horizontal scroll bars to appear—something you've tried to avoid up until now.

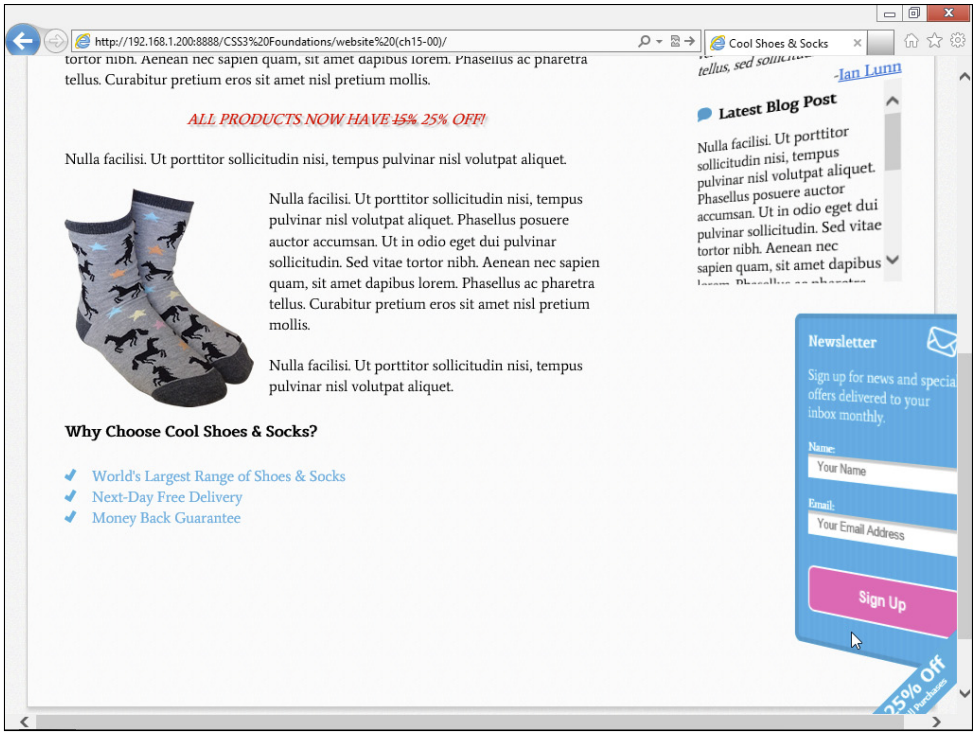


FIGURE 15-5 The Cool Shoes & Socks web page viewed in Internet Explorer 10.

You can find the CSS that affects the sidebar within the `.sidebar` rule set:

```
.sidebar {
    float: right;
    width: 25%;
    -webkit-transform: rotateY(-40deg);
    -moz-transform: rotateY(-40deg);
    -ms-transform: rotateY(-40deg);
    -o-transform: rotateY(-40deg);
    transform: rotateY(-40deg);
    -webkit-transform-style: preserve-3d;
    -moz-transform-style: preserve-3d;
    -ms-transform-style: preserve-3d;
    -o-transform-style: preserve-3d;
    transform-style: preserve-3d;
}
```

Unfortunately, Microsoft chose to support the nonprefixed `transform` property despite not yet supporting `transform-style: preserve-3d`. This means that you can't simply remove the prefixed `-ms-transform` property to fall back to a 2D sidebar because the

nonprefixed `transform` property is still applied. You could remove the `transform` property too, but then all other browsers that *do* support `transform-style: preserve-3d` would also lose out on the 3D sidebar.

This situation is somewhat frustrating. Microsoft shouldn't really have decided to support the nonprefixed `transform` property until the prefixed `-ms-transform` property worked fully. You can work around this issue, however:

1. In styles.css, find the `.sidebar` rule set and delete the following declaration:

   ```
   -ms-transform: rotateY(-40deg);
   ```

2. In the same rule set, below the `transform: rotateY(-40deg);` declaration, add the following:

   ```
   -ms-transform: none;
   ```

3. In the `.sidebar .aside:hover` rule set, delete the following declaration:

   ```
   -ms-transform: rotateY(-40deg);
   ```

4. In the same rule set, below the `transform: rotateY(-40deg);` declaration, add the following:

   ```
   -ms-transform: none;
   ```

5. In the `.sidebar .aside:nth-child(3):hover` rule set, delete the following declaration:

   ```
   -ms-transform: rotateY(40deg) translateX(-50px) scaleZ(2)
     translateZ(100px);
   ```

6. In the same rule set, below the `transform: rotateY(40deg) translateX(-50px) scaleZ(2) translateZ(100px);` declaration, add the following:

   ```
   -ms-transform: none;
   ```

7. Save styles.css.

The `.sidebar` rule set now looks like this:

```
.sidebar {
    float: right;
    width: 25%;
    -webkit-transform: rotateY(-40deg);
    -moz-transform: rotateY(-40deg);
    -o-transform: rotateY(-40deg);
    transform: rotateY(-40deg);
    -ms-transform: none;
    -webkit-transform-style: preserve-3d;
    -moz-transform-style: preserve-3d;
    -ms-transform-style: preserve-3d;
```

```
    -o-transform-style: preserve-3d;
    transform-style: preserve-3d;
}
```

What you do here is break the rules a little—by placing a prefixed property below the nonprefixed property—but only to fix the issue caused by Microsoft breaking the rules—supporting the nonprefixed `transform` property before it fully supported the nonprefixed `-ms-transform` property.

Because you place the `-ms-transform: none;` declaration below the `transform: rotateY(-40deg);` declaration, Internet Explorer 10 does not have a transform applied to it. The sidebar just falls back to 2D. This is the safest way to deal with Microsoft's shortcomings. If—and it's always a big *if* with Internet Explorer—Microsoft does follow the advice set out in the CSS specifications, it will eventually drop support for prefixed properties such as `-ms-transform`, so your declaration of `-ms-transform: none;` will be ignored and the `transform: rotateY(-40deg);` declaration will be used in its place. This result is what is desired for Cool Shoes & Socks anyway; you just have to take these measures temporarily while Internet Explorer 10 doesn't fully support 3D transforms.

Internet Explorer 9

Internet Explorer 9 offers a similar experience to Opera 12. `@keyframes` and 3D transforms aren't supported. Neither are transitions, so when you hover over the "25% banner," the banner just snaps between sizes. The rounded corners on elements are also unsupported. Again, this lack of support isn't a problem. The users will never know anything is wrong because the page still appears and functions perfectly for them; they just miss out on a few visual treats.

Firefox 3.6

In Chapter 1, I mentioned that Firefox 3.6 is still in use today—despite the latest Firefox being version 13 at the time of writing! Firefox 3.6 provides a similar experience to Internet Explorer 9, and thanks to the best practices you've adhered to throughout *CSS3 Foundations*, it doesn't require any additional attention; it displays and functions to the level the browser is capable of.

Testing Older Versions of Internet Explorer

So far, the browser-testing ride has been pretty smooth. Cool Shoes & Socks works in the latest versions of modern browsers used today. It's a certainty that older versions of Internet Explorer will require some extra attention though.

Internet Explorer 8

From the outset, Cool Shoes & Socks doesn't look too bad in Internet Explorer 8, although a few issues arise as you move down the page, as shown in Figure 15-6.

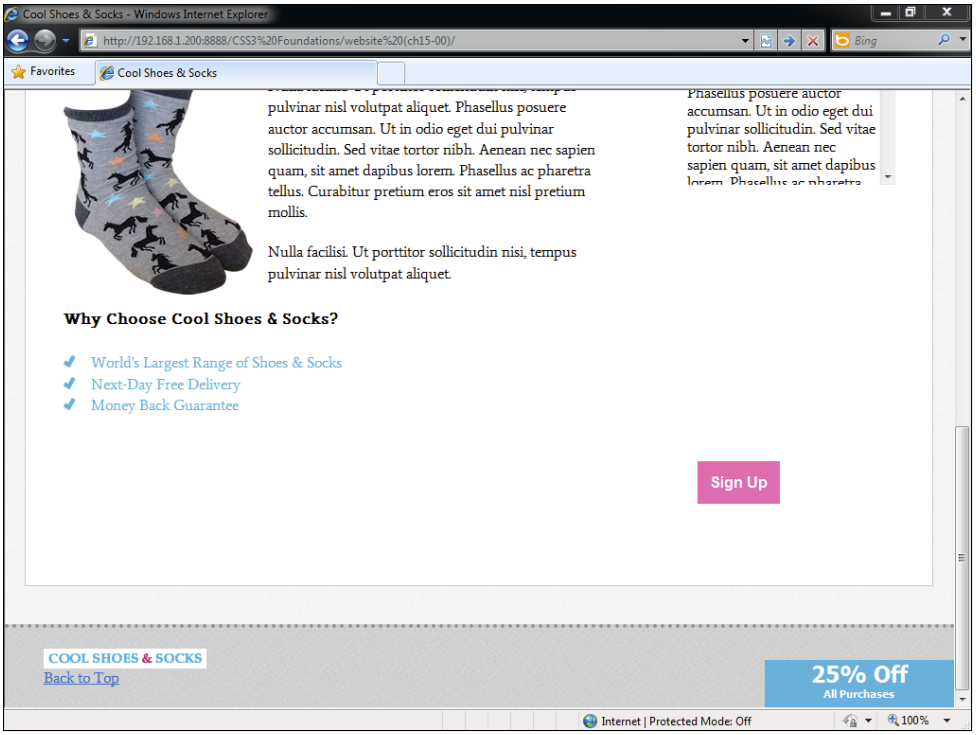


FIGURE 15-6 The Cool Shoes & Socks web page viewed in Internet Explorer 8.

First of all, the newsletter box disappeared! This isn't actually a browser bug, though. When adding multiple backgrounds to the newsletter box in Chapter 5, I didn't include an instruction to provide a fallback for older browsers that don't support multiple backgrounds. If you forget to add fallbacks—we all do from time to time—you can pick up these omissions during browser testing and fix them:

1. In styles.css, find the `#newsletter` rule set:

```
#newsletter {
    color: white;
    border-radius: 8px;
    border: rgba(0,0,0,0.1) solid 5px;
    background: url("../images/icon-newsletter.png") no-repeat
  91% 2%, url("../images/bg-newsletter.png") repeat 0 #00ACDF;
    padding: 6%;
    margin-left: -4px;
    width: 88%;
}
```

2. Above the `background` declaration, add another `background` declaration:

```
background: url("../images/bg-newsletter.png") repeat 0 #00ACDF;
```

3. Save styles.css.

   With two `background` declarations, browsers that don't understand the second multiple image declaration simply ignore it and apply the first.

   Now that the background image is in place, it also becomes apparent that the `border` declaration isn't being applied either. The reason is that Internet Explorer 9 doesn't understand the `rgba()` function used to style the border. Add another fallback property.

4. In the `#newsletter` rule set, above the `border` declaration, add the following:

```
border: #009CC8 solid 5px;
```

5. Save styles.css.

This procedure provides browsers that don't support the `rgba()` function a value that they do understand.

With the border fixed, you may notice the Sign Up button in the newsletter box isn't centered as it is in other browsers. This is a bug in Internet Explorer 8, and to work around it, you need to give the Sign Up button a `width`.

1. In styles.css, find the `input[type="submit"][class="button"]` rule set and add the following declaration:

```
width: 100%;
```

2. Save styles.css.

Finally, for Internet Explorer 8, the Cool Shoes & Socks logo and Back to Top link in the footer aren't floated to the left and right of the page. The reason is that the rule sets for these elements use the `:nth-child()` selector, which isn't supported in versions 6, 7, and 8 of Internet Explorer.

The Cool Shoes & Socks page uses the `:nth-child()` selector five times, but three of those are to apply transforms and animations to the product showcase and sidebar, which older versions of Internet Explorer don't support anyway. Because only two of the `:nth-child()` selectors affect Internet Explorer 6, 7, and 8, the simplest solution is just to give the elements a class name and change the rule set to use a class selector instead of `:nth-child()`.

1. In index.html, find the two `<li>` elements within `<div class="nav container">`.
2. Change the first `<li>` to give it the class `small-logo`, like so:

```
<li class="small-logo">
```

3. Change the second `<li>` to give it the class `back-to-top`, like so:

```
<li class="back-to-top">
```

4. Save index.html.

5. In styles.css, find the rule set `#footer li:nth-child(1)` and change the selector to `#footer .small-logo`.
6. In styles.css, find the rule set `#footer li:nth-child(2)` and change the selector to `#footer .back-to-top`.
7. Save styles.css.

Although this is a sensible option for the Cool Shoes & Socks page, you might not always want to have to change all your advanced selectors to simpler ones that older browsers can understand; otherwise, there would be no point in using them. One solution for this problem is to use JavaScript to help older browsers understand advanced selectors. Rather than write your own JavaScript to give them this capability, though, you can use a free utility to do this job for you. Called Selectivizr, it is located at `www.selectivizr.com/`.

When you download and place a `<script>` reference in the `<head>` of the page—just as you did with live.js in Chapter 2—along with a JavaScript library such as jQuery, many advanced CSS selectors work in older browsers. Visit the Selectivizr website at `www.selectivizr.com/` for complete instructions on how to add this reference if you need to.

Conditional Comments for Internet Explorer 6, 7, and 8

Internet Explorer versions 6 and 7 never made much of an effort to adhere to standards—with Internet Explorer 8 doing a little better but still not great—and because of that, whenever you check a web page in them, you certainly find issues, in particular in Internet Explorer 6. Sometimes you need to make fundamental changes to a page to get it working in these older versions of Internet Explorer, but that is at the detriment to the other browsers that *do* work. Thankfully, you can use conditional comments to apply styles to specific versions of Internet Explorer.

Let's look at some of the issues in Internet Explorer 7 and then use conditional comments to add fixes.

As shown in Figure 15-7, the first issue with Internet Explorer 7 is the layout of the navigation links; they appear vertically one after the other rather than side by side.

In Chapter 9, I mentioned that Internet Explorer 6 and 7 don't support `display: inline-block;`, which is what is applied to these navigation links. Luckily, these browsers treat `display: inline;` as if it were `display: inline-block;` anyway.

If you go into styles.css and change the `#header .nav > ul > li` rule set so that the `display` property has a value of `inline`, this change fixes Internet Explorer 6 and 7 but breaks modern browsers. Instead, create a conditional stylesheet:

FIGURE 15-7 The Cool Shoes & Socks web page viewed in Internet Explorer 7.

1. In index.html, below the `<link>` reference to `css/styles.css`, add the following:

```
<!--[if lt IE 8]><link rel="stylesheet" href="css/styles-ie7.
  css" type="text/css" /><![endif]-->
```

2. Save index.html.

This code adds a stylesheet to the page that is to be applied only when a specific condition is met. The condition `if lt IE 8` means "if less than Internet Explorer 8." So, when the Cool Shoes & Socks page is viewed in any version of Internet Explorer below version 8, the styles-ie7.css stylesheet is applied.

Following are a few examples of conditional comments you might like to use:

- `if IE`—If any version of Internet Explorer
- `if IE 6`—If the browser is specifically Internet Explorer version 6
- `if gt IE 8`—If the browser is greater than version 8 of Internet Explorer
- `if gte IE 8`—If the browser is greater than or equal to version 8 of Internet Explorer

- `if lt IE 9`—If the browser is less than version 9 of Internet Explorer
- `if lte IE 8`—If the browser is less than or equal to version 8 of Internet Explorer

You can also use an exclamation mark (`!`) to apply to anything that does not match a condition; for example, `if !IE` applies to anything that isn't Internet Explorer. For more information about conditional comments, see `www.quirksmode.org/css/condcom.html`.

Now that you have a conditional comment set up, add a new stylesheet to be applied to Internet Explorer 7.

3. Create a new file called styles-ie7.css and save it in the css folder.
4. In styles-ie7.css, add the following rule set:

```
#header .nav > ul > li {
    display: inline;
    vertical-align: top;
}
```

5. Save styles-ie7.css.

Because you add the conditional stylesheet below the main stylesheet, any styles added to it overwrite the main styles. This rule set overwrites the navigation links' `display: inline-block;` declaration from the main stylesheet and makes them `display: inline;` instead. You also include `vertical-align: top;` to make each link align to the top.

Also shown in Figure 15-7 is that the page has a horizontal scroll bar. Sometimes, older versions of Internet Explorer have inexplicable bugs such as this, and the best way to find a fix is through trial and error. By commenting out sections of HTML in index.html, I found that it was either the sidebar `<div class="sidebar" role="complementary">` or an element within the sidebar that was causing the page to be wider than necessary. By then commenting out sections within the sidebar, I found that it was the `<cite>` tag within the `<blockquote>`. With this knowledge, I then went into styles.css and commented out each declaration in the `blockquote cite` rule set until I found the culprit: `float: right;`.

The `<cite>` element appears to be acting bigger than it is (causing the page to become wider), so my natural reaction was to specify a `width`. As with the Send button issue you saw in Internet Explorer 8, this issue appears to be similar. To fix it, add an Internet Explorer 7–specific style:

1. In styles-ie7.css, add the following rule set:

```
blockquote cite {
    width: 100%;
```

```
    text-align: right;
}
```

2. Save styles-ie7.css

Just one visual issue is left in Internet Explorer 7. The input fields in the newsletter box are too wide for the box. The reason is that you used `box-sizing: border-box;` on these elements, which isn't supported in Internet Explorer 6 or 7. Internet Explorer 6 and 7 only treat `box-sizing` as the default `padding-box`. For more information on this issue, see Chapter 7.

1. In styles-ie7.css, add the following rule set:

```
input[type="text"],input[type="email"] {
    width: 88%;
}
```

2. Save styles-ie7.css.

Because the newsletter box has a `padding` of `6%` both on the left and right sides, when you make the width of the input fields `88%`, they perfectly fit in the box again. Note that this effect is true only when the width of the browser is 960px or greater—which leads to the next Internet Explorer fix.

In the next chapter, you use media queries to make the Cool Shoes & Socks page adapt based on the width of the page. Internet Explorer 6, 7, and 8 don't support media queries, though, so you can add a conditional style sheet for them that prevents the web page shrinking below 960px.

1. In index.html, above the previously added conditional stylesheet, add the following:

```
<!--[if lt IE 9]><link rel="stylesheet" href="css/styles-ie.
  css" type="text/css" /><![endif]-->
```

2. Save index.html.
3. Create a new file called styles-ie.css and save it in the css folder.
4. In styles-ie.css, add the following declaration:

```
body {
    width: 960px;
}
```

When you add this conditional stylesheet that applies to all versions of Internet Explorer below version 9, Cool Shoes & Socks viewed in those browsers isn't able to shrink below a width of 960px. This workaround ensures the layout doesn't shrink down to the point where content becomes unreadable.

Universal Internet Explorer 6 Stylesheet

Now comes the gut-wrenching moment...when viewing Cool Shoes & Socks in Internet Explorer 6, it isn't going to be pretty. Thankfully, the days of supporting Internet Explorer 6 are close to over—if not already over.

If you absolutely have to support Internet Explorer 6 and make it appear close to how other browsers do, you can add fixes in the same way as with Internet Explorer 7 and 8: Create a conditional stylesheet that applies styles to Internet Explorer 6 only. Cool Shoes & Socks takes more of a radical approach, though.

As shown in Figure 15-8, Cool Shoes & Socks viewed in Internet Explorer 6 isn't pretty. Rather than spend *x* number of hours fixing it with a conditional stylesheet, you can simply remove all styles, leaving just the bare content—all that's really important on the page.

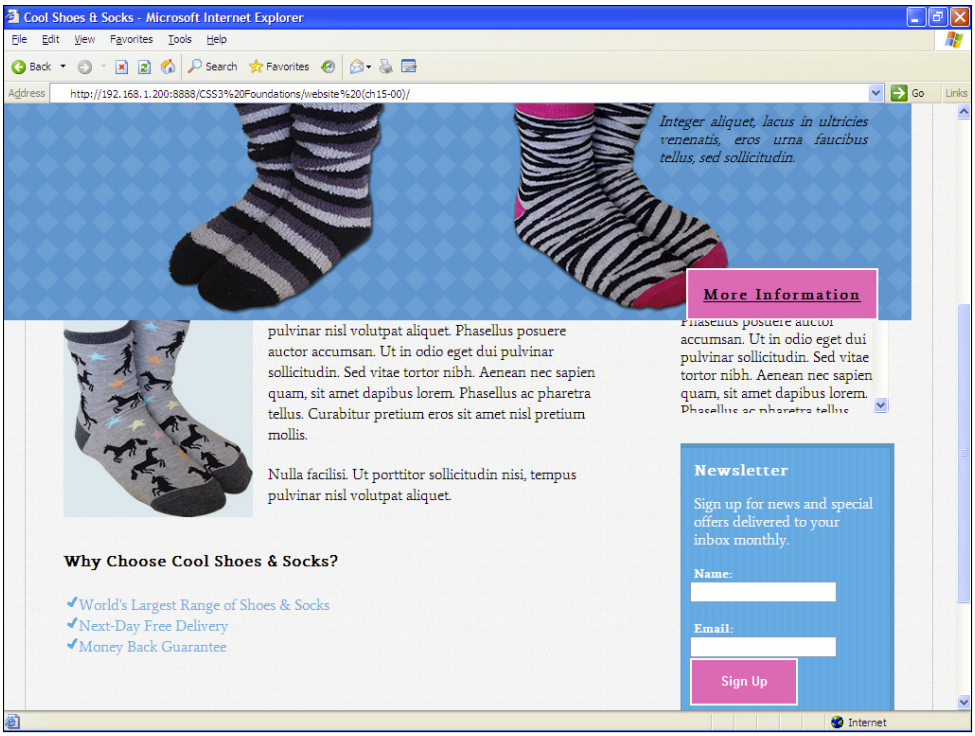


FIGURE 15-8 The Cool Shoes & Socks web page viewed in Internet Explorer 6.

Using Andy Clarke's (`www.stuffandnonsense.co.uk/`) Universal Internet Explorer 6 Stylesheet `code.google.com/p/universal-ie6-css/`, you can achieve this result easily.

1. In index.html, modify the `<link>` reference to styles.css so it is wrapped in a conditional comment, like so:

```
<!--[if ! lte IE 6]><!--><link rel="stylesheet" href="css/
  styles.css" type="text/css" /><![endif]-->
```

This lines hides the main stylesheet from Internet Explorer versions 6 and below but still allows it to apply to everything else.

2. Below the `<link>` reference to styles-ie7.css, add the following line:

```
<!--[if lte IE 6]><link rel="stylesheet" href="http://
  universal-ie6-css.googlecode.com/files/ie6.1.1.css"
  media="screen, projection"><![endif]-->
```

3. Save index.html.

With this last line you add, versions of Internet Explorer less than or equal to version 6 are given a basic stylesheet that removes all but the necessary styles from Cool Shoes & Socks and any other page you care to add it to in the future.

Summary

In this chapter, you made the Cool Shoes & Socks web page cross browser compatible for desktop browsers—not the easiest of tasks but one that is essential. Each browser now displays Cool Shoes & Socks without issues and at a level it is capable of.

The steps you took to provide a basic experience for Internet Explorer 6 may seem radical, but with Internet Explorer 6 having a very small market share now—and not even Microsoft supports it anymore—that experience still provides any remaining users of Internet Explorer 6 with all the content the page has to offer. They just don't get the bells and whistles.

In the next chapter, you use media queries to make the page change based on the attributes of the device it is being viewed on, making mobile experiences in particular richer.

chapter sixteen

Making Your Website Look Great Across Multiple Devices

WITH SO MANY different types of devices able to access the web today, how a web page is displayed on those devices needs to change to best suit the device it is being viewed on. At the moment, the Cool Shoes & Socks page is suited for desktop browsers, but cramming all of that page into the small screen of a mobile device, for example, makes the content very small and difficult to read.

As you saw in Chapter 2, CSS2 offers media types (`www.w3.org/TR/CSS2/media.html`), which allow you to apply styles to specific conditions such as screen, print, and handheld. Of course, these media types were rather short-sighted. What is handheld? A mobile phone? A tablet device, such as an iPad? A Nintendo DS? All these devices could be considered as "handheld," but they still differ greatly—in size, in features, and the environment in which they are used.

The CSS3 media queries module (`www.w3.org/TR/css3-mediaqueries/`) adds many new ways for you to query attributes of a device on which a page is being viewed and to set up styles specific to those conditions.

In this chapter, you use the free Opera Mobile Emulator (`www.opera.com/developer/tools/mobile/`) to build and test a Cool Shoes & Socks page that is capable of adapting to varying device conditions.

Project files update (ch16-00): If you haven't followed the previous instructions and are comfortable working from here onward or would like to reference the project files up to this point, you can download them from `www.wiley.com/go/treehouse/css3foundations`.

Using Opera Mobile Emulator

Before you begin adding media queries to the Cool Shoes & Socks stylesheet, first take a look at the page in Opera Mobile Emulator.

1. Download and install Opera Mobile Emulator from `www.opera.com/developer/tools/mobile/`.
2. After it is installed, open Opera Mobile Emulator.

   When you open Opera Mobile Emulator, you are presented with the options for the device you want to emulate, as shown in Figure 16-1. By changing these settings, you can set up an environment similar to many Android devices such as Samsung Galaxy S, HTC Desire, Motorola Xoom, and so on.

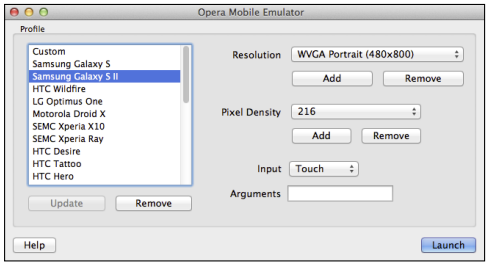


FIGURE 16-1 The initial window of Opera Mobile Emulator.

   Opera Mobile Emulator conveniently has a list of profiles for many devices, which saves you from having to change options such as Resolution and Pixel Density.

   Ideally, you should test in as many devices as possible, but as you did with desktop browsers, choose one profile to develop with, and when you're done, test in other profiles.

3. From the Profile list, select Samsung Galaxy S II and click Launch.

   When a profile launches, you are presented with a window that has the same dimensions as the Samsung Galaxy S II, and you can start using the Opera Mobile browser as if you were using it on that device.

4. With the Cool Shoes & Socks page open in your desktop browser, copy the URL of the page from the address bar.

5. Return to Opera Mobile Emulator, click the address bar, and paste the URL. Then press Return on your keyboard or click the Go button in the bottom right.

You now see the Cool Shoes & Socks page in the same way it would be viewed on a Samsung Galaxy S II, as shown in Figure 16-2. Cool stuff, right? You can view your pages in a whole range of mobile devices.

FIGURE 16-2 The Cool Shoes & Socks page viewed in a Samsung Galaxy S II profile using Opera Mobile Emulator.

In the bottom-right corner of the Opera Mobile Emulator window is a small rotate symbol, as shown in Figure 16-3. Click this symbol to rotate the device, as if it is in your hands and you are changing it from portrait to landscape view and vice versa.

As you can see, Cool Shoes & Socks is difficult to read on such a small device. If this were the real device in your hand, you would probably zoom in to read the content. Using media queries, you can make it so that users don't need to zoom in and swipe around the page to be able to read it, while generally improving their experience of Cool Shoes & Socks when viewing it on a mobile device.

FIGURE 16-3 The rotate button in Opera Mobile Emulator that allows you to change the emulated device between portrait and landscape views.

Scaling the Viewport on Mobile Devices

When you view a web page on a mobile device, many mobile browsers scale the viewport so the page fits on the screen without the need for horizontal scroll bars. The reason Cool Shoes & Socks is so small when viewed in Opera Mobile Emulator—and you'll find the same happens across many devices—is that the page is scaled down to fit.

Because media queries are a new technology, many websites don't use them. Therefore, a mobile browser scaling down the viewport in this way provides a solution that makes the majority of web pages display reasonably well on a smaller screen. Text can often be hard to

read at this size, but users can get a quick overview of the page and zoom in to the sections they want to see or read—a better approach than being able to show only a small part of a web page at a higher zoom level.

Because you're working on the Cool Shoes & Socks page now, though—with the media queries module in a Recommendation status—you get the privilege of choosing how your web page looks on a mobile device. Letting a mobile browser scale the viewport is a quick fix but not a perfect one, which is where media queries step in.

As yet, there isn't a way to change the scaling of the viewport via CSS, although the feature is on its way in the CSS Device Adaption module (`dev.w3.org/csswg/css-device-adapt/#the-viewport-rule`). However, you can scale the viewport using HTML.

1. In index.html, below `<meta charset="utf-8" />`, add the following:

```
<meta name="viewport"
content="width=device-width, initial-scale=1, maximum-scale=1">
```

2. Save index.html.

When you specify the `width` as `device-width`, the viewport is set to the same width as the device. The `initial-scale` is the zoom applied to the page when it first loads, `1` being the equivalent of 100%. When `maximum-scale` is set to `1`, the page can't be zoomed in more than 100%. The CSS Device Adaption module explains that the `initial-scale` property will actually be called `zoom`; likewise, similar properties such as `minimum-scale` and `maximum-scale` (which determine how far in and out the user is allowed to zoom) will be translated to `min-zoom` and `max-zoom`.

Because the CSS Device Adaption module is an Editor's draft at the time of writing, *CSS3 Foundations* doesn't cover that module. If you would like more information about `<meta name="viewport">`, see the Mobile Web Application Best Practices at `www.w3.org/TR/mwabp/#bp-viewport`.

So what does this change do to Cool Shoes & Socks? As shown in Figure 16-4, the page now zooms right in. Everything is bigger but crammed into a smaller place, so whether the page is more readable is arguable. Likewise, some elements look a little broken and/or could do with being positioned differently. Using media queries, you can address all these issues and only for the mobile experience—the desktop experience isn't changed.

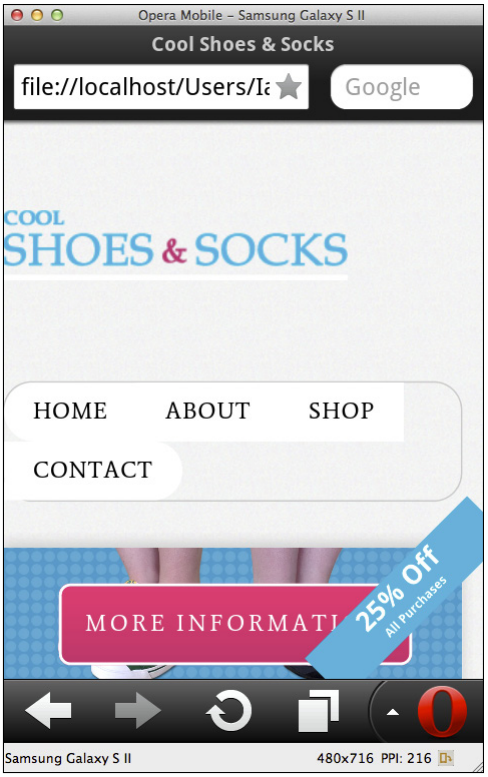


FIGURE 16-4 The Cool Shoes & Socks page viewed in Opera Mobile Emulator with an `initial-scale` of 1 and the viewport width set to be the same as the device's width.

Using Media Queries

Browser support: IE 9+, Firefox 3.5+, Chrome 4+, Opera 9.5+, Safari 4+

Although the media queries module is a part of the CSS3 specification, these queries are in a Recommendation status, meaning they are no longer experimental and can be used safely. Unfortunately, Internet Explorer 6, 7, and 8 don't support media queries, but you already took steps in the preceding chapter to work around that issue. What's more, mobile devices and tablets don't use these older browsers, so media queries and older versions of Internet Explorer don't pose much of a problem.

As you saw in Chapter 2, you add media types to a stylesheet using the `@media` rule, for example:

```
@media screen {
    /* styles to be applied to screen devices here */
}
```

Any rule sets placed with this `@media` rule apply only to the screen. Media queries use the same syntax and build on media types. Whereas the previous media type applied to the screen, the following media query applies to a *color* screen:

```
@media screen and (color){
    /* styles to be applied to color screen devices here */
}
```

You can use the same query with the `media` attribute in a `<link>` element, like so:

```
<link rel="stylesheet" media="screen and (color)"
href="styles.css" />
```

You place media queries in parentheses `()`; they are logical expressions that can either be true or false. Here, if the media is a color screen, the query is true and the necessary styles are applied; otherwise, the query is false and the styles are ignored.

Color is just one of the many media features that you can use to compose a media query, which you look at after learning how to use logical operators.

Using Logical Operators

Logical operators allow you to create complex `@media` rules—stringing media queries together.

And

The `and` keyword enables you to combine features of a query. For example:

```
@media screen and (color)
```

In this query, the media must be a screen *and* a color one at that. If the media is a screen but can display only black and white, the `@media` rule is false and the styles don't apply.

Or

To specify a logical OR expression, you can list queries, separated by a comma. For example:

```
@media screen and (color), projection and (color)
```

This query applies to color screens *or* color projectors. Only one of the queries in the list needs to be true for the `@media` rule to be true.

Not

When you use the `not` keyword, the result of a media query is negated. The following `@media` rule is true on a color screen:

```
@media screen and (color)
```

The following is false for a color screen but true for everything else:

```
@media not screen and (color)
```

This means anything that isn't a color screen has the specified styles applied to it.

Only

Because older browsers can read `@media` rules but not media queries, older browsers may ignore the query and still apply a stylesheet regardless of whether the query is true or false. To prevent that from happening, you can place the `only` keyword at the start of an `@media` rule to make older browsers ignore the entire rule, like so:

```
@media only screen and (color)
```

Modern browsers know that the `only` keyword is just present to throw off older browsers, and they continue reading everything after it. The `only` keyword is used with every Cool Shoes & Socks `@media` rule.

Applying Styles to Specific Media Features

Media features, such as `color` shown previously, are used in media queries to describe the requirements of the environment you want to apply styles to.

Many features accept optional `min-` or `max-` prefixes to express "greater than or equal to" and "less than or equal to." For example, `min-width: 100px` is true for anything with a width of 100px or more.

In this section, below the name of each media feature, you see which media type a media feature applies to and whether it supports `min-` and `max-` prefixes.

Let's look at what each media feature describes and then implement some of them into the Cool Shoes & Socks stylesheet.

width

Applies to: all visual and tactile (a device that allows for being touched) media types | Accepts min/max prefixes: yes

`width` describes the width of the viewport on a targeted display, that is, the width of the browser's viewing area.

Assuming you want to apply styles to a viewport with a width equal to or less than 960px, such as the 700px wide browser window shown in Figure 16-5, you can use the `max-width` media feature:

```
@media only screen and (max-width: 960px) {
    /* styles to be applied to screen devices when the viewport has
 a width equal to or less than 960px */
}
```

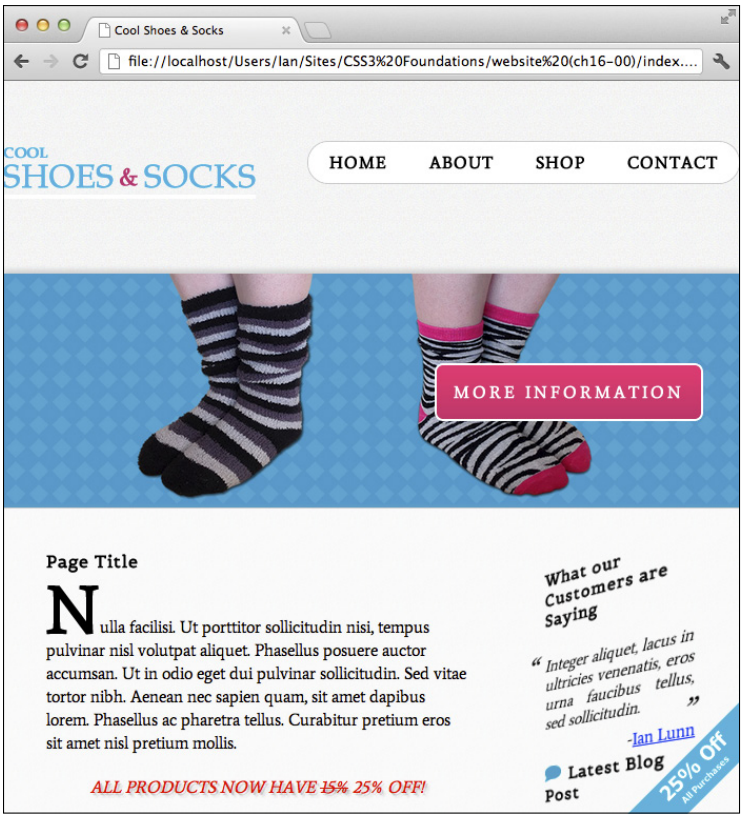


FIGURE 16-5 The Cool Shoes & Socks page viewed in a browser with a width of 700px.

Using the query `@media only screen and (width: 960px)` applies only to viewports with an exact width of 960px.

height

Applies to: all visual and tactile media types | Accepts min/max prefixes: yes

`height` describes the height of the viewport on a targeted display, that is, the height of the browser's viewing area.

Media queries tend to use the `width` media feature more than `height`, but this feature still has its uses.

For example, you could query the height of the browser to move important elements—ones that you want the users to see as soon as they visit the page—further up the page when their viewport has a small height. You may decide that the "25% Off" banner on the Cool Shoes & Socks page would get in the way of more important content when the browser height is smaller than usual and therefore use a media query to hide it below a certain height.

```
@media only screen and (max-height: 500px) {
    /* styles to be applied to screen devices when the viewport has
a height equal to or less than 500px */
}
```

device-width

Applies to: all visual and tactile media types | Accepts min/max prefixes: yes

Whereas the media feature `width` is the width of the viewport, `device-width` is the width of the screen.

`device-width` is often used to target devices such as mobiles and tablets. Consider the following example:

```
@media only screen and (min-device-width: 320px)

and (max-device-width: 480px)
```

With this query, all screen devices with a screen width ranging between 320px and 480px have the specified styles applied to them. This range of widths applies to many mobile devices both in portrait and landscape orientations, but not bigger devices such as tablets and desktop computers.

Unlike the viewport, the size of a screen obviously can't be changed.

device-height

Applies to: all visual and tactile media types | Accepts min/max prefixes: yes

`device-height` describes the height of a screen. As with the `height` media feature, it is not as commonly used as `width` or `device-width` but shares similar purposes as those described for `height`.

orientation

Applies to: bitmap (devices that output an image, such as screen or print) media types | Accepts min/max prefixes: no

The `orientation` media feature takes two values, `portrait` and `landscape`, and is written as follows:

```
@media only screen and (orientation: portrait)
@media only screen and (orientation: landscape)
```

`orientation` is `portrait` when the value of the `height` media feature is greater than or equal to the value of the `width` media feature; otherwise, `orientation` is `landscape`.

aspect-ratio

Applies to: bitmap media types | Accepts min/max prefixes: yes

`aspect-ratio` is the ratio between the `width` and `height` media features (the width and height of the viewport). As with many of the media features, `aspect-ratio` tends to be used with a `min-` or `max-` prefix.

This media query describes a viewport with a width that is equal to or greater than its height:

```
@media only screen and (min-aspect-ratio: 1/1)
```

device-aspect-ratio

Applies to: bitmap media types | Accepts min/max prefixes: yes

`device-aspect-ratio` is the ratio between the `device-width` and `device-height` (the width and height of a screen).

The following media query describes a device's screen with an aspect ratio of 16/9:

```
@media only screen and (device-aspect-ratio: 16/9)
```

color, color-index, monochrome, resolution, scan, and grid

The following media features are used for more specialized pages and applications:

- `color`—Describes the number of bits per color component of the output device
- `color-index`—Describes the number of entries in the color lookup table of the output device
- `monochrome`—Describes the number of bits per pixel in a monochrome frame buffer
- `resolution`—Describes the resolution of the output device
- `scan`—Describes the scanning process of `tv` output devices
- `grid`—Queries whether the output device is grid or bitmap

For more information on these media features, see the Media Queries module at `www.w3.org/TR/css3-mediaqueries/#device-aspect-ratio`.

Adding Media Queries to Cool Shoes & Socks

Now that you know what media features are available, you can modify the layout of Cool Shoes & Socks specifically for mobile devices and then move on to adjusting the tablet experience and desktop experience when the browser window is narrower than 960px.

Media Queries for Mobile Devices

As you saw in Figure 16-4, Cool Shoes & Socks when viewed on a smaller device gets messy, so let's tidy it up and make the experience better for smaller devices.

1. At the bottom of styles.css, add the following `@media` rule:

   ```
   /* Mobile (portrait and landscape)*/
   @media only screen and (max-device-width: 480px) {

   }
   ```

   This rule applies to devices that have a screen width up to 480px—many mobile devices. It's a good idea to include a comment above the `@media` rule to remind you and/or your colleagues what exactly a media query is targeting.

2. Inside the `@media` rule, add the following declarations:

```
#header .logo {
    display: block;
    float: none;
    margin: 0 auto;
}

#header {
    padding: 0 10px;
}

#header nav, .sidebar, #content{
    float: none;
    width: 100%;
}

.banner-ad {
    display: none;
}
```

The first two rule sets, `#header .logo` and `#header`, center the logo and add padding to either side of the header.

The third rule set, `#header nav, .sidebar, #content`, is one you will see a lot for mobile layouts (the selectors may be different but the declarations the same). Because the desktop version of Cool Shoes & Socks has multiple columns, when the width of the page is much smaller on devices such as mobile, there isn't enough room for two columns. When you set the header navigation, sidebar, and content area to `float: none;` and `width: 100%;`, those elements all stretch out into one column, giving them the space they need to become more readable.

The fourth rule set, `.banner-ad`, hides the "25% Off" banner, which, as mentioned previously, may get in the way of content. What's more, fixed position elements don't work well on mobile devices, so it's a good idea to avoid them where possible.

The mobile layout isn't quite right yet, so add a few more rule sets to the same `@media` rule.

3. Inside the `@media` rule and below the other rule sets you just added, add the following:

```
#header nav > ul {
    border: none;
}

#header nav li {
    border-radius: 20px;
    border: #ccc solid 1px;
```

```
    margin: 0 0 5px 0;
    width: 100%;
}

#header nav ul ul {
    display: none;
}

.sidebar {
    border-top: 3px dotted #ccc;
    clear: both;
    margin: 60px 0 0 0;
    padding: 20px 0 0 0;
}

.sidebar, .sidebar aside:hover,
.sidebar aside:nth-child(3):hover {
-webkit-transform: none;
    -moz-transform: none;
    -o-transform: none;
    -ms-transform: none;
    transform: none;
}

#footer li {
    display: block;
    float: none;
    line-height: 200%;
}
```

The first rule, `#header nav > ul`, removes the border around the navigation, and the second rule, `#header nav li`, adds borders around each navigation link instead. Because mobile devices often have touch capabilities now, it is good practice to make sure any link on a page has a wide target area so the users can easily press it without having to concern themselves about being too accurate.

The third rule set, `#header nav ul ul`, hides the drop-down menus. It's a good idea to hide complex drop-down menus on touch devices. Just remember to make sure the top-level pages have links to their child pages or provide an alternative navigation.

The fourth rule set, `.sidebar`, changes the layout of the sidebar somewhat. Now that the sidebar sits below the content (rather than to its side as in the desktop version), it's given a top border to visually separate it and some margin and padding on top to push it a little further away from the bottom of the content area.

The fifth rule set, `.sidebar, .sidebar aside:hover, .sidebar aside:nth-child(3):hover`, overrides the transforms when the sidebar elements are hovered over. Because the mobile layout is a smaller space, it's better to hide these animated transforms.

The last rule set, `#footer li`, tidies up the links in the footer.

Finally, modify the product showcase for mobile devices.

4. Still inside the `@media` rule and below the other rule sets you just added, add the following:

```
.showcase {
    height: 200px;
    overflow: hidden;
}

.showcase li {
    -webkit-animation: none;
    -moz-animation: none;
     -ms-animation: none;
    -o-animation: none;
    animation: none;
}

.showcase img {
    height: auto;
    margin-left: -50%;
    max-width: 200%;
    width: 200%;
}
```

5. Save styles.css.

Because the product showcase shrinks a little too small on mobile devices, the first rule set, `.showcase`, increases its height and also hides any overflowing content, which works well with the changes made in the third rule set.

The second rule set, `.showcase li`, removes the cycling animation from the product showcase. Note that in Opera Mobile Emulator, the browser doesn't support animations, so the users don't see them anyway. Other browsers that do support them, such as Safari on the iPhone, don't support the animation particularly well, so that has been removed from the mobile experience.

The last rule set, `.showcase img`, makes the product images twice as wide (and thus twice as tall). It also overwrites the inhertied `max-width: 100%;` declaration, because in this

case, having the image bigger than its container is desired. To keep the product image in the center, you give it a `margin-left` of `-50%`.

With all these styles applied that are specific to mobile devices, the mobile experience is much improved. Figure 16-6 shows how Cool Shoes & Socks looks in Opera Mobile Emulator; the logo is centered, and the navigation is much more suited to a touch device.

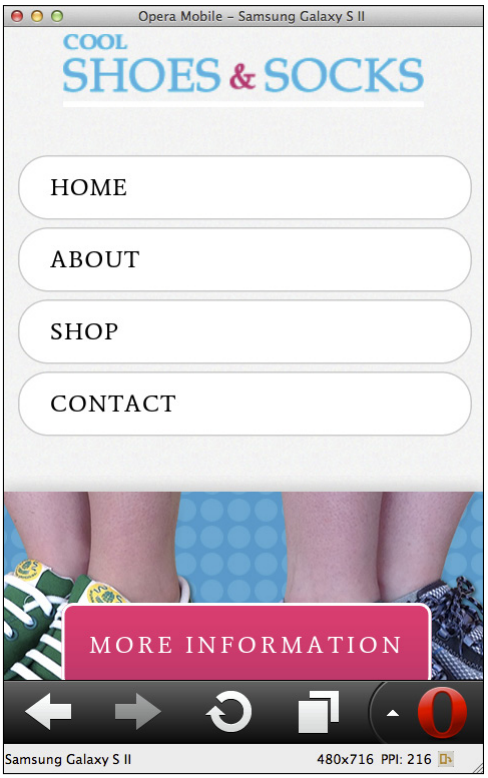


FIGURE 16-6 The Cool Shoes & Socks web page viewed in Opera Mobile Emulator with styles applied to it that are specific to mobile devices.

Figure 16-7 shows that the product showcase has been slightly modified for the mobile experience, and the content is all in one easier-to-read column.

If you rotate the device into landscape, you see the same rules still apply, but thanks to the fluid layout you built, the page stretches out to make use of the extra width, as shown in Figure 16-8.

FIGURE 16-7 The product showcase and content area.

FIGURE 16-8 The same Cool Shoes & Socks web page viewed in landscape view.

You may feel the product showcase when viewed in landscape needs tweaking, to bring the product image up a small amount, as shown in Figure 16-9.

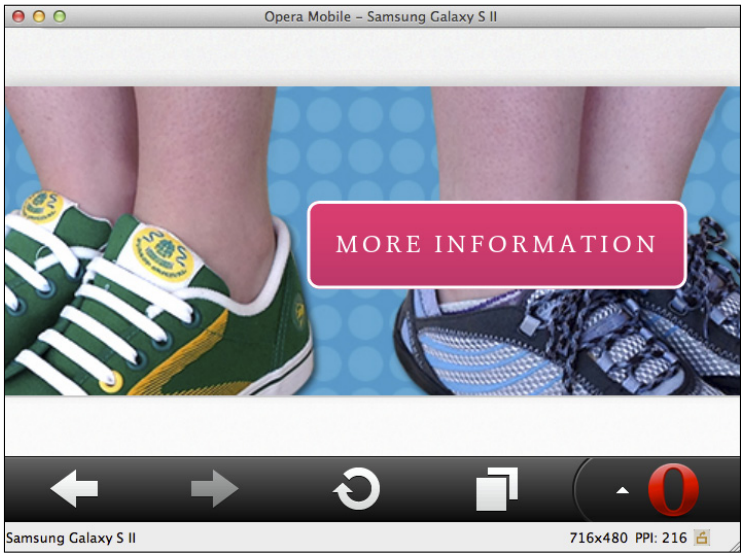


FIGURE 16-9 The product showcase when viewed in landscape.

I personally like the slightly abstract nature of the image in this way and am happy to have it stay like that. If you want to change it, you could add another media query that targets mobile devices only when in landscape orientation, like so:

```
/* Mobile (landscape)*/
@media only screen and (max-device-width: 480px) and (orientation:
  landscape) {
    .showcase img {
        margin-left: -25%;
       width: 150%;
    }
}
```

Project files update (ch16-01): If you haven't followed the previous instructions and are comfortable working from here onward or would like to reference the project files up to this point, you can download them from `www.wiley.com/go/treehouse/css3foundations`.

Media Queries for Tablets and Narrow-Size Desktop Browsers

Cool Shoes & Socks now provides a much better experience for mobile devices with a width up to 480px. The desktop experience looks great at 960px and higher, but there's still a gap between 480px and 960px that may need some additional styles. This gap covers tablet devices in addition to desktop experience when the browser is narrower than 960px.

Let's start with the desktop browser below 960px. In Figure 16-10, the browser window has been reduced to a width of 800px.

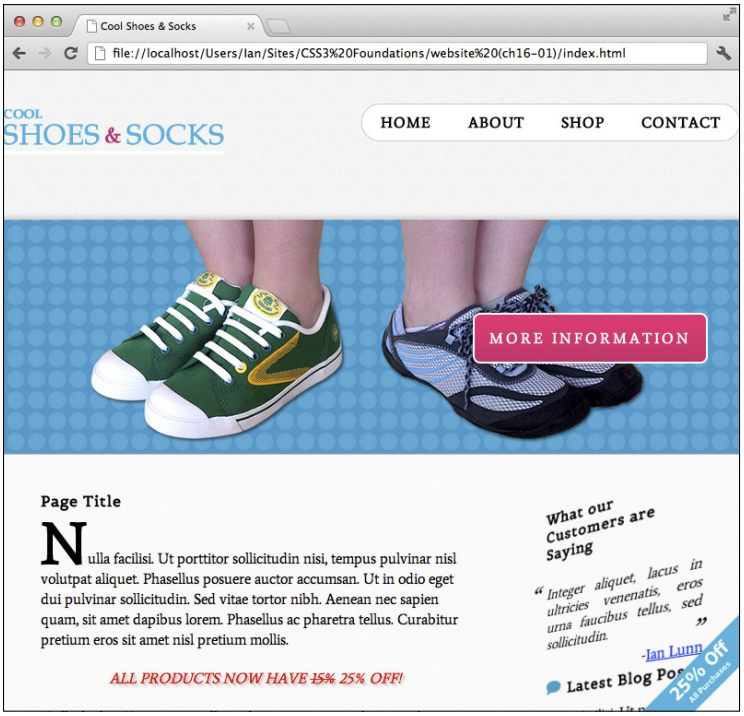


FIGURE 16-10 The Cool Shoes & Socks page viewed in a browser at a width of 800px.

For the most part, the normal desktop experience is sufficient, although notice that the logo and navigation in the header touch the sides of the browser window. You can change that:

1. At the bottom of styles.css, add the following `@media` rule:

```
/*viewport 960px and below*/
@media only screen and (max-width: 960px) {
    #header {
        padding: 0 40px;
    }
}
```

2. Save styles.css.

This media query applies to screen devices where the viewport width is 960px or less. Because the maximum width of the website is 960px, whenever the browser window is equal to or less than that, the header has some space on either side of it.

Now, what if the width of the browser window is reduced even more? Figure 16-11 shows that at a width of 700px, the navigation and logo no longer have enough space to sit aside each other. The content and sidebar sitting side by side are also starting to look a little cramped.

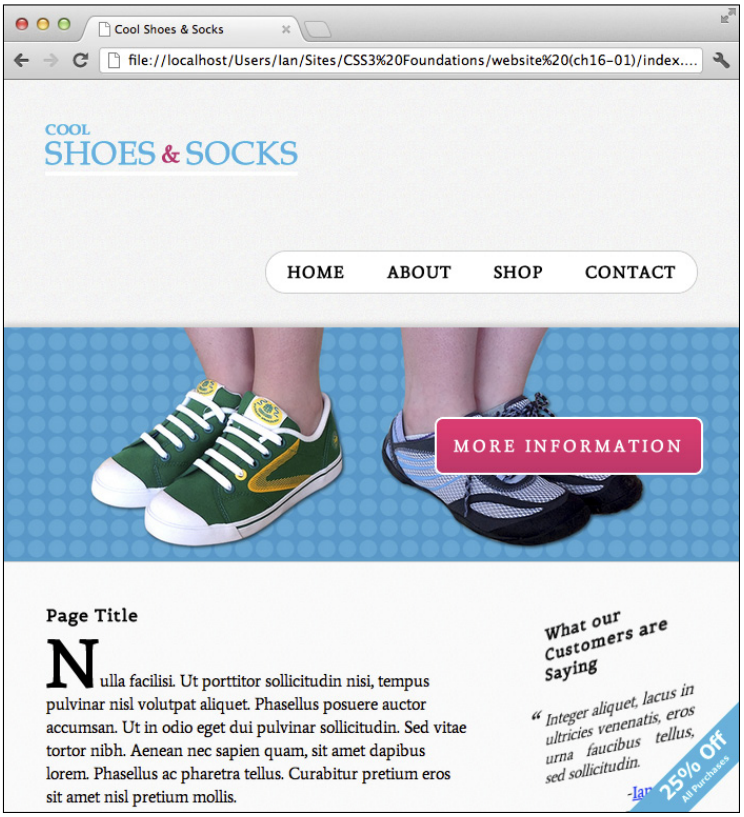


FIGURE 16-11 The Cool Shoes & Socks page viewed in a browser at a width of 700px.

The point where the navigation moves down below the logo is at about 710px; this point can be determined by resizing the window slowly or by using a tool such as responsivepx (`www.responsivepx.com`). This is known as a breakpoint. You could add a few styles to change the layout of the navigation at 710px or below, but because the content area starts to get too squashed as this point, a better approach would be to adopt a similar layout to the mobile layout.

Rather than write another `@media` rule and lots more styles, adapt the `@media` rule that was written for mobile devices.

3. In styles.css, find the following `@media` rule:

```
/* Mobile (portrait and landscape)*/
@media only screen and (max-device-width: 480px)
```

4. Change that `@media` rule and the comment that describes its purpose to the following:

```
/* Mobile (portrait and landscape) and viewports up to
a 735px width */
@media only screen and (max-width: 735px) {
```

When you change the last query to `max-width: 735px`, the mobile layout applies to mobile devices and viewports equal to or below a width of `735px`. Although it was determined that the breakpoint of the navigation was at `710px`, the navigation got a bit too close to the logo, so it's been increased to `735px`.

The mobile layout disabled the animation on the product showcase, which is undesired, so make another `@media` rule specific to mobile devices only and move the showcase rule sets into that.

5. From the `@media` rule `@media only screen and (max-width: 735px)`, remove the following:

```
.showcase {
    height: 200px;
    overflow: hidden;
}

.showcase li {
    -webkit-animation: none;
    -moz-animation: none;
    -ms-animation: none;
    -o-animation: none;
    animation: none;
}

.showcase img {
    height: auto;
    margin-left: -50%;
    max-width: 200%;
    width: 200%;
}
```

6. Above `@media only screen and (max-width: 735px)`, add the following:

```
/* Mobile (portrait and landscape) */
@media only screen and (max-device-width: 480px) {
```

```
    .showcase {
        height: 200px;
        overflow: hidden;
    }

    .showcase li {
        -webkit-animation: none;
        -moz-animation: none;
        -ms-animation: none;
        -o-animation: none;
        animation: none;
    }

    .showcase img {
        height: auto;
        margin-left: -50%;
        max-width: 200%;
        width: 200%;
    }
}
```

7. Save styles.css.

   When you move the showcase rule sets out of the `@media` rule that applies to mobile devices and viewports up to a width of 735px, the slideshow begins animating again in desktop browsers (where browsers support CSS3 animations). To keep it from working on mobile devices, you then add an `@media` rule specific to mobile again.

   With the desktop experience below a viewport width of 735px now taking on the mobile layout, the newsletter box looks a little stretched, as shown in Figure 16-12, so you can change that.

8. At the bottom of styles.css, add the following:

```
/* viewport between 480px and 735px */
@media only screen and (min-width: 480px) and (max-width:
  735px) {
    #newsletter {
        margin: 0 auto;
        width: 50%;
    }
}
```

This rule causes the newsletter box to be 50% wide when the viewport is between a width of 480px and 735px.

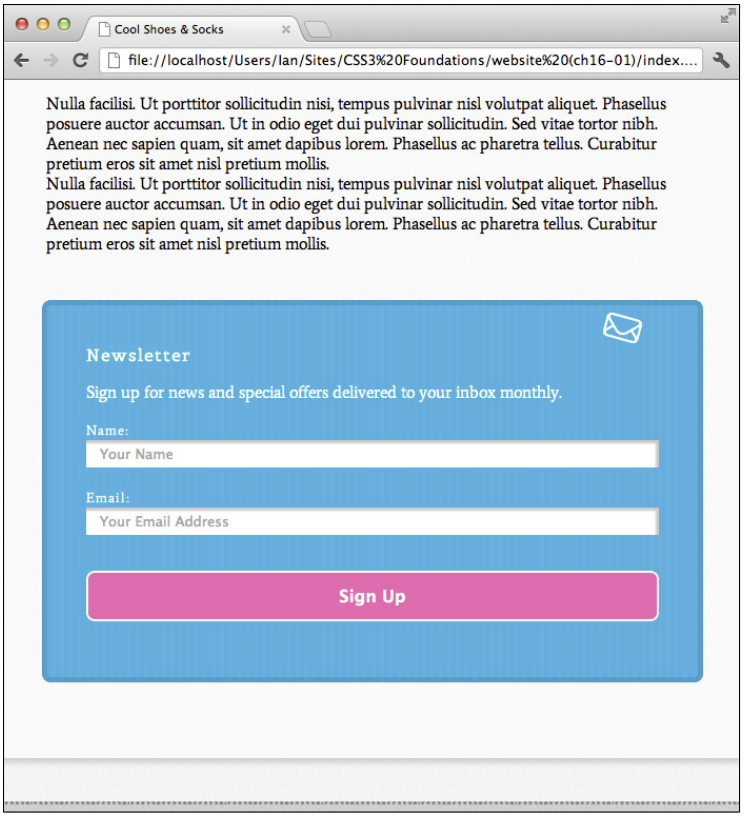


FIGURE 16-12 The newsletter box is a little too stretched when viewed in a browser window at 700px.

Summary

In this chapter, you made the final changes to Cool Shoes & Socks, and you now have a web page that is responsive and adapts to present itself in the best possible way for the particular device it is being viewed on!

At times, you may question why media queries are necessary when mobile browsers attempt to automatically scale web pages for better user experience, but as time goes on and many more differing devices are released, media queries will truly show their strength. Implementing them now doesn't just future proof a web page; it offers a superior experience for your users, too.

chapter seventeen

Final Steps and Conclusion

COOL SHOES & SOCKS is now a truly modern web page. It looks and functions great both on mobile and desktop devices and, with the approach you've taken, should stand the test of time, continuing to be a great page as technology and devices advance. Of course, because it is only one page, you will almost certainly want to create more. The page you created throughout *CSS3 Foundations* makes for a great template that you can easily duplicate and modify for existing pages.

Project files update (ch17-00): If you haven't followed the previous instructions and are comfortable working from here onward or would like to reference the project files up to this point, you can download them from `www.wiley.com/go/treehouse/css3foundations`.

Final Steps

When you're happy with the site, what's next? The page will need some final adjustments to get it ready to show the world!

Removing Production Code and Preparing to Go Live

In Chapter 2, I mentioned that all styles should be placed in one stylesheet where possible. This step is optional but one that can improve the performance of a web page. Cool Shoes & Socks consists of four stylesheets: reset.css, styles-ie.css, styles-ie7.css, and styles.css. Because styles-ie.css and styles-ie7.css are conditional stylesheets, they have to stay separate, but you can combine reset.css and styles.css.

1. In styles.css, find the following `@import` rule and delete it:

   ```
   @import url("reset.css");
   ```

2. Open reset.css, highlight all the styles, and copy them.
3. In styles.css, paste the copied reset.css styles where the `@import` rule was located.
4. Save styles.css.

   Now that the reset styles are included in styles.css, you can minify them. Minifying is also optional but good for performance. Minifying a stylesheet causes all spaces to be removed from that file, reducing the file size—so it can be downloaded faster—and making it easier for a browser to read.

   Many minifying tools are available, but my personal recommendation is `www.cssminify.net`, which is one of the easiest and quickest to use.

5. Create a new file, call it styles.min.css, and save it in the css folder.
6. In styles.css, highlight all the styles and copy them.
7. In your browser, navigate to `www.cssminify.net`.
8. In the text area on the `www.cssminify.net` site, paste the styles from styles.css.
9. From the Compression drop-down menu, select Highest and click Minify.

   When you click Minify, spaces and returns are stripped out of the styles in the text area.

10. Highlight and copy all the minified styles from the text area.
11. Return to styles.min.css and paste the minified styles.
12. Save styles.min.css.

    Now that you have a minified version of styles.css, index.html needs to reference it.

13. In index.html, find the reference to styles.css and change it to reference styles.min.css:

    ```
    <!--[if ! lte IE 6]><!--><link rel="stylesheet"
    href="css/styles.min.css" type="text/css" /><![endif]-->
    ```

14. Save index.html.

Cool Shoes & Socks now uses the minified styles from styles.min.css instead of the production styles found in styles.css.

Testing, Testing, Testing

In Chapter 15, you prepared Cool Shoes & Socks to begin modifying it with media queries. Having modified the site in this chapter and Chapter 16, you should test it again to be doubly sure it is cross browser compatible. Although testing can sometimes be tiresome, it is a very important step and should be your last step before making a page live for everybody to see.

Going Live! Uploading to a Web Server

When you're satisfied that a web page is cross browser compatible, to make it live, you need to upload it to a web server, most commonly done using FTP software.

Many text editors nowadays come with built-in FTP software, and if this is the case, consult the instructions provided with your text editor to set it up. Alternatively, dedicated FTP programs are available, some free and some paid.

My recommendations are Cyberduck (`www.cyberduck.ch`) and FileZilla (`www.filezilla-project.org`), both free and both available on Mac and Windows.

If you choose to use one of these FTP clients, see the documentation made available on its website for instructions on how to use it.

The Future Web

With Cool Shoes & Socks, you created a page suited not just to the modern web but also the future web. Its robust, responsive layout will make it appear and function great across many different types of devices now and in the future. Will CSS offer anything more for the future, though? Of course!

Because the CSS3 specification is still being modified, you will be able to use many new technologies in the near future. Here are a couple:

- Filter Effects (`http://dvcs.w3.org/hg/FXTF/raw-file/tip/filters/index.html`), which changes the style of an image similar to what you can do in photo-editing applications such as Adobe Photoshop.
- Flexible Box Layout (`www.w3.org/TR/css3-flexbox/`) and Regions (`http://dev.w3.org/csswg/css3-regions/`), which make laying out a page easier and more robust.

What's more, work has already started on new CSS modules, which propose new selectors and media features, with no doubt lots more exciting goodies on the way, too!

The web is a fascinating technology to be a part of. And now styling beautiful content is easier than ever. If you can dream it, you can create it.

Index

C

D

G

H

I

M

N

O

P

Q

R

S

T

X

Z