



# Learn to Code **HTML & CSS**

Develop & Style Websites

Shay Howe

# Learn to Code **HTML & CSS**

Develop & Style Websites

Shay Howe



VOICES THAT MATTER™

## **LEARN TO CODE HTML & CSS: DEVELOP & STYLE WEBSITES**

**Shay Howe**

### **NEW RIDERS**

[www.newriders.com](http://www.newriders.com)

To report errors, please send a note to [errata@peachpit.com](mailto:errata@peachpit.com)

New Riders is an imprint of Peachpit, a division of Pearson Education.

Copyright © 2014 by W. Shay Howe

**Project Editors:** Michael J. Nolan and Nancy Peterson

**Development Editor:** Jennifer Lynn

**Production Editor:** David Van Ness

**Copyeditor:** Jennifer Needham

**Technical Editor:** Chris Mills

**Indexer:** Karin Arrigoni

**Proofreader:** Darren Meiss

**Cover Designer:** Shay Howe

**Interior Designer:** Mimi Heft

**Compositor:** WolfsonDesign

### **NOTICE OF RIGHTS**

All rights reserved. No part of this book may be reproduced or transmitted in any form by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of the publisher. For information on getting permission for reprints and excerpts, contact [permissions@peachpit.com](mailto:permissions@peachpit.com).

### **NOTICE OF LIABILITY**

The information in this book is distributed on an “As Is” basis without warranty. While every precaution has been taken in the preparation of the book, neither the author nor Peachpit shall have any liability to any person or entity with respect to any loss or damage caused or alleged to be caused directly or indirectly by the instructions contained in this book or by the computer software and hardware products described in it.

### **TRADEMARKS**

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and Peachpit was aware of a trademark claim, the designations appear as requested by the owner of the trademark. All other product names and services identified throughout this book are used in editorial fashion only and for the benefit of such companies with no intention of infringement of the trademark. No such use, or the use of any trade name, is intended to convey endorsement or other affiliation with this book.

ISBN 13: 978-0-321-94052-0

ISBN 10: 0-321-94052-0

9 8 7 6 5 4 3 2 1

Printed and bound in the United States of America

*For you.*

*One way or another this book ended up in your hands. I'm excited to see what you do with it, and I hope the knowledge within this book makes as large an impact on your life as it has on my own.*

## About the Author



Born and raised in the small town of Lima, Ohio, Shay Howe grew up disassembling remote controls and other electronics in hopes of learning how they worked. When the Internet was introduced, he was fascinated and immediately began learning all he could about it. Upon graduating from high school, he moved to Tempe, Arizona, where he attended the University of Advancing Technology and received a Bachelor of Arts degree in digital multimedia with a focus in web design.

Currently living in Chicago, Illinois, Shay is a designer and front-end developer with a passion for solving problems while building creative and intuitive websites. He specializes in web and product design and front-end development, specialties that he regularly writes and speaks about.

Shay is co-founder of Chicago Camps, which hosts low-cost, high-value technology events in the Chicago area. He is also co-organizer of Refresh Chicago and UX Happy Hour, which help to refresh the creative, technical, and professional culture of New Media endeavors.

Perhaps most importantly, though, Shay is the undisputed office table tennis champion.

## Acknowledgments

To everyone who helped make this book a reality, from the bottom of my heart, I cannot thank you enough!

There are so many people who have helped me in my career and with this book that it's going to be impossible to thank them all. I will undoubtedly forget someone important, and I apologize to whoever that may be. That said, I have to begin by thanking my family and friends. There's no way this book would ever exist without their help and support.

My wife, Becky, was encouraging from day one and has always been supportive of my endeavors, no matter if they seemed like good ideas or not. Our pup, Gatsby, who makes me smile every day, kept my feet warm all winter while I was writing, using them as his bed under my desk. All of the thanks in the world would not be enough for my parents, Wes and Deb, who have provided me with more support and guidance than I could have ever dreamed. I love them all.

Before this was a book it was a website, and that website received feedback from some of the best in the business. I'm incredibly thankful to Jeff Cohen, Mike Gibson, Scott Robbin, Christopher Webb, Russell Schoenbeck, Dan Kim, Chris Mills, Bruce Lawson, Christian Heilmann, and many others for their initial feedback on these lessons. Of course the website itself wouldn't have existed without the help of Darby Frey, who has had my back for years and is easily one of the best guys I know.

I tapped on quite a few friends for content for this book, and I was overwhelmed by all of their contributions. I owe two-handed high fives to Aaron Irizarry, Adam Connor, AJ Self, Arman Ghosh, Bermon Painter, Brad Smith, Candi Lemoine, Carolyn Chandler, Chris Mills, Dan Denney, Darby Frey, Erica Decker, Estelle Weyl, Jen Myers, Jenn Downs, Jennifer Jones, Leslie Jensen-Inman, Maya Bruck, Russ Unger, Tessa Harmon, Victoria Pater, Vitaly Friedman, and Zoe Mickley Gillenwater. Next time I see each of them the high fives are payable in full, and dinner is on me.

I owe a ton of thanks to the New Riders family who helped bring this all together. Michael Nolan eased my fear of writing a book and gave me a gracious introduction to New Riders. Jennifer Lynn deserves an award for keeping this book on track and helping make sense of the content within it. Chris Mills did a fantastic job of making sure all of the right topics were covered in an understandable manner. Jennifer Needham put my words to work, making me sound better than I ever imagined. Mimi Heft always went the extra mile and was incredibly patient with me. Nancy Peterson handled every request I threw at her with ease and kept the entire team on the same page. They're all superheroes in my eyes, and they made writing this book an amazing experience.

Much of the content within this book has been heavily influenced by those who have written books and publications before me, and who continue to be thought leaders within the industry. Without the contributions of Jeffrey Zeldman, Eric Meyer, Dan Cederholm, Dave Shea, Andy Budd, Jeremy Keith, Cameron Moll, Ethan Marcotte, Chris Coyier, and others, it's hard to say what I'd know.

Today the Mozilla Developer Network and Dev.Opera communities are publishing some of the best content on HTML and CSS; they have become staples within a long list of great resources. They must be thanked, too, for their amazing contributions.

When not in the office I do my best to stay involved in the community, and to that end I help organize a handful of different events in the Chicago area. While writing this book I fell behind on my duties, and I must thank Russ Unger and Brad Simpson from Chicago Camps and Jon Buda and Anthony Zinni from Refresh Chicago for picking up my slack. They all helped to carry my portion of the work without hesitation, and I'm thankful and honored to have them as partners in crime.

Many people have generously offered words of wisdom and lent an ear from time to time. For that, I must thank Bill DeRouchey, Bill Scott, Brad Wilkening, Braden Kowitz, Brandon Satrom, Carl Smith, Chris Courtney, Chris Eppstein, Crystal Shuller, Dale Sande, Dave Giunta, Dave Hoover, Debra Levin Gelman, Derek Featherstone, Dustin Anderson, Fabian Alcantara, Greg Baugues, Hampton Catlin, Jack Toomey, Jason Kunesh, Jason Ulaszek, JC Grubbs, Jim and Jen Remsik, Jonathan Snook, Keith Norman, Luis D. Rodriguez, Michael Boeke, Michael "Doc" Norton, Michael Parenteau, Milton Jackson, Nishant Kothary, Peter Merholz, Sam Rosen, Samantha Soma, Tim Frick, Todd Larsen, and Todd Zaki Warfel.

Last, but certainly not least, I must thank the late Matt Puchlerz. He taught me more than he'll ever know, and I wouldn't be where I am today without him. I am forever grateful for Matt's friendship, and I miss him dearly.

# Contents

- Introduction . . . . . x
- LESSON 1 Building Your First Web Page 1**
  - What Are HTML & CSS? . . . . . 2
  - Understanding Common HTML Terms . . . . . 2
  - Setting Up the HTML Document Structure . . . . . 4
  - Understanding Common CSS Terms . . . . . 7
  - Working with Selectors . . . . . 9
  - Referencing CSS. . . . . 11
  - Using CSS Resets . . . . . 12
  - Summary . . . . . 16
- LESSON 2 Getting to Know HTML 17**
  - Semantics Overview . . . . . 18
  - Identifying Divisions & Spans. . . . . 18
  - Using Text-Based Elements . . . . . 20
  - Building Structure. . . . . 23
  - Creating Hyperlinks . . . . . 29
  - Summary . . . . . 35
- LESSON 3 Getting to Know CSS 36**
  - The Cascade . . . . . 37
  - Calculating Specificity . . . . . 38
  - Combining Selectors . . . . . 40
  - Layering Styles with Multiple Classes. . . . . 42
  - Common CSS Property Values . . . . . 44
  - Summary . . . . . 52

<b>LESSON 4 Opening the Box Model</b>	<b>53</b>
How Are Elements Displayed? . . . . .	54
What Is the Box Model? . . . . .	56
Working with the Box Model . . . . .	56
Developer Tools . . . . .	67
Summary . . . . .	73
<b>LESSON 5 Positioning Content</b>	<b>74</b>
Positioning with Floats . . . . .	75
Positioning with Inline-Block . . . . .	87
Creating Reusable Layouts . . . . .	90
Uniquely Positioning Elements . . . . .	95
Summary . . . . .	98
<b>LESSON 6 Working with Typography</b>	<b>99</b>
Adding Color to Text . . . . .	100
Changing Font Properties . . . . .	101
Applying Text Properties . . . . .	113
Using Web-Safe Fonts . . . . .	123
Embedding Web Fonts . . . . .	124
Including Citations & Quotes . . . . .	128
Summary . . . . .	131
<b>LESSON 7 Setting Backgrounds &amp; Gradients</b>	<b>132</b>
Adding a Background Color . . . . .	133
Adding a Background Image . . . . .	134
Designing Gradient Backgrounds . . . . .	142
Using Multiple Background Images . . . . .	152
Exploring New Background Properties . . . . .	153
Summary . . . . .	156



<b>LESSON 8 Creating Lists</b>	<b>157</b>
Unordered Lists . . . . .	158
Ordered Lists . . . . .	158
Description Lists . . . . .	160
Nesting Lists . . . . .	162
List Item Styling . . . . .	163
Horizontally Displaying List . . . . .	166
Summary . . . . .	176
<b>LESSON 9 Adding Media</b>	<b>178</b>
Adding Images . . . . .	179
Adding Audio . . . . .	189
Adding Video . . . . .	191
Adding Inline Frames . . . . .	193
Semantically Identifying Figures & Captions . . . . .	201
Summary . . . . .	203
<b>LESSON 10 Building Forms</b>	<b>204</b>
Initializing a Form . . . . .	205
Text Fields & Textareas . . . . .	205
Multiple Choice Inputs & Menus . . . . .	208
Form Buttons . . . . .	211
Other Inputs . . . . .	212
Organizing Form Elements . . . . .	212
Form & Input Attributes . . . . .	215
Login Form Example . . . . .	217
Summary . . . . .	228

LESSON 11 Organizing Data with Tables229

Creating a Table. . . . . 230

Table Structure . . . . . 234

Table Borders . . . . . 238

Table Striping . . . . . 242

Aligning Text. . . . . 244

Completely Styled Table. . . . . 248

Summary . . . . . 265

LESSON 12 Writing Your Best Code266

HTML Coding Practices . . . . . 267

CSS Coding Practices. . . . . 273

Summary . . . . . 281

Index . . . . . 282

# Introduction

I come from a family of educators. My parents are both teachers, as is my brother. I was the only one in my family not to become a teacher. That said, I love helping others, spreading the knowledge I have about web design, and teaching when possible. To that end, I often speak at different conferences and schools, as well as host the occasional workshop. When doing so, I continually receive questions about HTML and CSS. So, I wrote this book to be that ideal, all-encompassing resource for learning HTML and CSS.

Traditionally, you'll see books that teach HTML first and then CSS, keeping the two languages completely separate. But when they're taught independently, things don't really come together until the very end, which is frustrating for someone new to HTML and CSS. I wanted to take a different approach, teaching both languages at the same time so that you can see the fruits of your labor sooner rather than later. This book aims to bring instant gratification to the web design process.

It was also important to me that the book be project based, providing a completed website for readers who work through the book from start to finish. Not everyone learns by reading alone, so I wanted to provide a tangible website to allow people to learn experientially.

And let's face it, HTML and CSS can be a little daunting at first. After all, the languages are ever changing, and the evolution requires a steady stream of up-to-date material. This book is written in a workshop-style format, with 12 easy-to-digest lessons. Starting with the basics, each lesson builds upon the previous one and breaks down the barriers to entry, showing you how you can start writing HTML and CSS today in practical examples. In fact, you build a simple web page in the first lesson. Then, in subsequent lessons, you learn not only how to make this web page more robust, but also how to create additional design-savvy and interactive web pages that, when combined, form an entire functional website.

*Learn to Code HTML & CSS* covers the latest technologies as well as the foundations of HTML and CSS that were set years ago. It also covers a range of topics and skills, from beginning to advanced. So if you're looking to become a web designer or developer and you want to learn HTML and CSS, then this book is for you.

## Lesson 4

# Opening the Box Model

We've familiarized ourselves with HTML and CSS; we know what they look like and how to accomplish some of the basics. Now we're going to go a bit deeper and look at exactly how elements are displayed on a page and how they are sized.

In the process we'll discuss what is known as the box model and how it works with HTML and CSS. We're also going to look at a few new CSS properties and use some of the length values we covered in Lesson 3. Let's begin.



# How Are Elements Displayed?

Before jumping into the box model, it helps to understand how elements are displayed. In Lesson 2 we covered the difference between block-level and inline-level elements. To quickly recap, block-level elements occupy any available width, regardless of their content, and begin on a new line. Inline-level elements occupy only the width their content requires and line up on the same line, one after the other. Block-level elements are generally used for larger pieces of content, such as headings and structural elements. Inline-level elements are generally used for smaller pieces of content, such as a few words selected to be bold or italicized.

## Display

Exactly how elements are displayed—as block-level elements, inline elements, or something else—is determined by the `display` property. Every element has a default display property value; however, as with all other property values, that value may be overwritten. There are quite a few values for the `display` property, but the most common are `block`, `inline`, `inline-block`, and `none`.

We can change an element's `display` property value by selecting that element within CSS and declaring a new `display` property value. A value of `block` will make that element a block-level element.

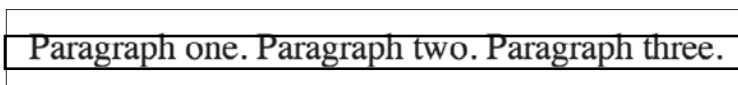
```
1. p {  
2.   display: block;  
3. }
```

A value of `inline` will make that element an inline-level element.

```
1. p {  
2.   display: inline;  
3. }
```

Things get interesting with the `inline-block` value. Using this value will allow an element to behave as a block-level element, accepting all box model properties (which we'll cover soon). However, the element will be displayed in line with other elements, and it will not begin on a new line by default.

```
1. p {  
2.   display: inline-block;  
3. }
```



**Figure 4.1** Three paragraphs displayed as `inline-block` elements, sitting one right next to the other in a horizontal line

## The Space Between Inline-Block Elements

One important distinction with inline-block elements is that they are not always touching, or displayed directly against one another. Usually a small space will exist between two inline-block elements. This space, though perhaps annoying, is normal. We'll discuss why this space exists and how to remove it in the next lesson.

Lastly, using a value of `none` will completely hide an element and render the page as if that element doesn't exist. Any elements nested within this element will also be hidden.

```
1. div {  
2.   display: none;  
3. }
```

Knowing how elements are displayed and how to change their `display` is fairly important, as the `display` of an element has implications on how the box model is rendered. As we discuss the box model, we'll be sure to look at these different implications and how they can affect the presentation of an element.

# What Is the Box Model?

According to the box model concept, every element on a page is a rectangular box and may have width, height, padding, borders, and margins (see **Figure 4.2**).

That's worth repeating: Every element on a page is a rectangular box.



**Figure 4.2** When we look at each element individually, we can see how they are all rectangular, regardless of their presented shapes

Every element on every page conforms to the box model, so it's incredibly important. Let's take a look at it, along with a few new CSS properties, to better understand what we are working with.

## Working with the Box Model

Every element is a rectangular box, and there are several properties that determine the size of that box. The core of the box is defined by the width and height of an element, which may be determined by the `display` property, by the contents of the element, or by specified `width` and `height` properties. `padding` and then `border` expand the dimensions of the box outward from the element's width and height. Lastly, any `margin` we have specified will follow the border.

Each part of the box model corresponds to a CSS property: `width`, `height`, `padding`, `border`, and `margin`.

Let's look these properties inside some code:

```
1. div {  
2.     border: 6px solid #949599;  
3.     height: 100px;
```

```

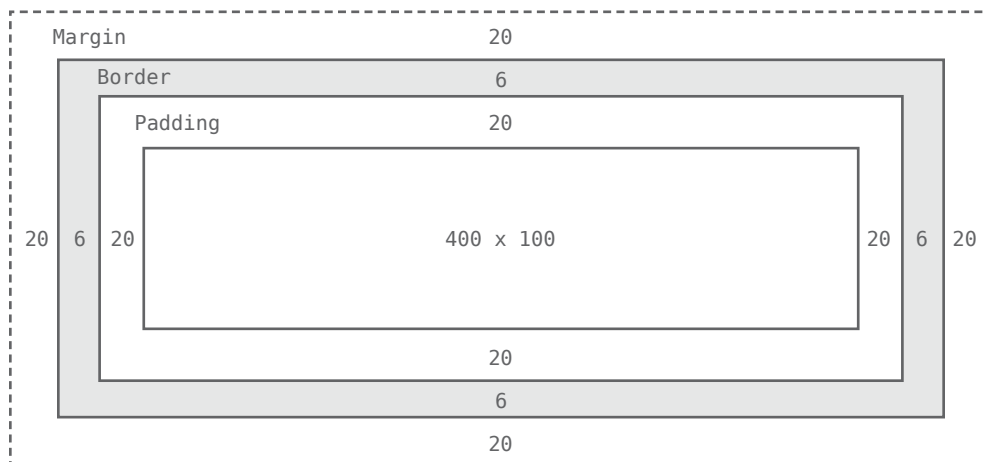
4.   margin: 20px;
5.   padding: 20px;
6.   width: 400px;
7. }

```

According to the box model, the total width of an element can be calculated using the following formula:

$$\text{margin-right} + \text{border-right} + \text{padding-right} + \text{width} + \text{padding-left} + \text{border-left} + \text{margin-left}$$

In comparison, according to the box model, the total height of an element can be calculated using the following formula:

$$\text{margin-top} + \text{border-top} + \text{padding-top} + \text{height} + \text{padding-bottom} + \text{border-bottom} + \text{margin-bottom}$$


**Figure 4.3** The box model broken down, including a base height and width plus paddings, borders, and margins

Using the formulas with the box shown in **Figure 4.3**, we can find the total height and width of our example.

- **Width:**  $492\text{px} = 20\text{px} + 6\text{px} + 20\text{px} + 400\text{px} + 20\text{px} + 6\text{px} + 20\text{px}$
- **Height:**  $192\text{px} = 20\text{px} + 6\text{px} + 20\text{px} + 100\text{px} + 20\text{px} + 6\text{px} + 20\text{px}$



The box model is without question one of the more confusing parts of HTML and CSS. We set a `width` property value of 400 pixels, but the actual width of our element is 492 pixels. By default the box model is additive; thus to determine the actual size of a box we need to take into account padding, borders, and margins for all four sides of the box. Our width not only includes the `width` property value, but also the size of the left and right padding, left and right borders, and left and right margins.

So far a lot of these properties might not make a whole lot of sense, and that's all right. To clarify things, let's take a close look at all of the properties—`width`, `height`, `padding`, `border`, and `margin`—that go into forming the box model.

## Width & Height

Every element has default width and height. That width and height may be 0 pixels, but browsers, by default, will render every element with size. Depending on how an element is displayed, the default height and width may be adequate. If an element is key to the layout of a page, it may require specified `width` and `height` property values. In this case, the property values for non-inline elements may be specified.

## Width

The default width of an element depends on its `display` value. Block-level elements have a default width of 100%, consuming the entire horizontal space available. Inline and inline-block elements expand and contract horizontally to accommodate their content. Inline-level elements cannot have a fixed size, thus the `width` and `height` properties are only relevant to non-inline elements. To set a specific width for a non-inline element, use the `width` property:

```
1. div {  
2.     width: 400px;  
3. }
```

# Height

The default height of an element is determined by its content. An element will expand and contract vertically as necessary to accommodate its content. To set a specific height for a non-inline element, use the `height` property:

```
1. div {  
2.     height: 100px;  
3. }
```

## Sizing Inline-Level Elements

Please keep in mind that inline-level elements will not accept the `width` and `height` properties or any values tied to them. Block and inline-block elements will, however, accept the `width` and `height` properties and their corresponding values.

# Margin & Padding

Depending on the element, browsers may apply default margins and padding to an element to help with legibility and clarity. We will generally see this with text-based elements. The default margins and padding for these elements may differ from browser to browser and element to element. In Lesson 1 we discussed using a CSS reset to tone all of these default values down to zero. Doing so allows us to work from the ground up and to specify our own values.

## Margin

The `margin` property allows us to set the amount of space that surrounds an element. Margins for an element fall outside of any border and are completely transparent in color. Margins can be used to help position elements in a particular place on a page or to provide breathing room, keeping all other elements a safe distance away. Here's the `margin` property in action:

```
1. div {  
2.     margin: 20px;  
3. }
```

One oddity with the `margin` property is that vertical margins, `top` and `bottom`, are not accepted by inline-level elements. These vertical margins are, however, accepted by block-level and inline-block elements.

## Padding

The `padding` property is very similar to the `margin` property; however, it falls inside of an element's border, should an element have a border. The `padding` property is used to provide spacing directly within an element. Here's the code:

```
1. div {  
2.     padding: 20px;  
3. }
```

The `padding` property, unlike the `margin` property, works vertically on inline-level elements. This vertical padding may blend into the line above or below the given element, but it will be displayed.

### Margin & Padding on Inline-Level Elements

Inline-level elements are affected a bit differently than block and inline-block elements when it comes to margins and padding. Margins only work horizontally—`left` and `right`—on inline-level elements. Padding works on all four sides of inline-level elements; however, the vertical padding—the `top` and `bottom`—may bleed into the lines above and below an element.

Margins and padding work like normal for block and inline-block elements.

## Margin & Padding Declarations

In CSS, there is more than one way to declare values for certain properties. We can use longhand, listing multiple properties and values one after the other, in which each value has its own property. Or we can use shorthand, listing multiple values with one property. Not all properties have a shorthand alternative, so we must make sure we are using the correct property and value structure.

The `margin` and `padding` properties come in both longhand and shorthand form. When using the shorthand `margin` property to set the same value for all four sides of an element, we specify one value:

```
1. div {  
2.     margin: 20px;  
3. }
```

To set one value for the top and bottom and another value for the left and right sides of an element, specify two values: top and bottom first, then left and right. Here we are placing margins of 10 pixels on the top and bottom of a `<div>` and margins of 20 pixels on the left and right:

```
1. div {  
2.     margin: 10px 20px;  
3. }
```

To set unique values for all four sides of an element, specify those values in the order of top, right, bottom, and left, moving clockwise. Here we are placing margins of 10 pixels on the top of a `<div>`, 20 pixels on the right, 0 pixels on the bottom, and 15 pixels on the left.

```
1. div {  
2.     margin: 10px 20px 0 15px;  
3. }
```

Using the `margin` or `padding` property alone, with any number of values, is considered shorthand. With longhand, we can set the value for one side at a time using unique properties. Each property name (in this case `margin` or `padding`) is followed by a dash and the side of the box to which the value is to be applied: `top`, `right`, `bottom`, or `left`. For example, the `padding-left` property accepts only one value and will set the left padding for that element; the `margin-top` property accepts only one value and will set the top margin for that element.

```
1. div {  
2.     margin-top: 10px;  
3.     padding-left: 6px;  
4. }
```

When we wish to identify only one `margin` or `padding` value, it is best to use the long-hand properties. Doing so keeps our code explicit and helps us to avoid any confusion down the road. For example, did we really want to set the `top`, `right`, and `left` sides of the element to have margins of 0 pixels, or did we really only want to set the `bottom` margin to 10 pixels? Using longhand properties and values here helps to make our intentions clear. When dealing with three or more values, though, shorthand is incredibly helpful.

### Margin & Padding Colors

The `margin` and `padding` properties are completely transparent and do not accept any color values. Being transparent, though, they show the background colors of relative elements. For margins, we see the background color of the parent element, and for padding, we see the background color of the element the `padding` is applied to.

## Borders

Borders fall between the padding and margin, providing an outline around an element. The `border` property requires three values: `width`, `style`, and `color`. Shorthand values for the `border` property are stated in that order—`width`, `style`, `color`. In longhand, these three values can be broken up into the `border-width`, `border-style`, and `border-color` properties. These longhand properties are useful for changing, or overwriting, a single border value.

The `width` and `color` of borders can be defined using common CSS units of length and color, as discussed in Lesson 3.

Borders can have different appearances. The most common style values are `solid`, `double`, `dashed`, `dotted`, and `none`, but there are several others to choose from.

Here is the code for a 6-pixel-wide, solid, gray border that wraps around all four sides of a `<div>`:

```
1. div {  
2.     border: 6px solid #949599;  
3. }
```



**Figure 4.4** Different border sizes and styles

## Individual Border Sides

As with the `margin` and `padding` properties, borders can be placed on one side of an element at a time if we'd like. Doing so requires new properties: `border-top`, `border-right`, `border-bottom`, and `border-left`. The values for these properties are the same as those of the `border` property alone: width, style, and color. If we want, we can make a border appear only on the bottom of an element:

```
1. div {  
2.     border-bottom: 6px solid #949599;  
3. }
```

Additionally, styles for individual border sides may be controlled at an even finer level. For example, if we wish to change only the width of the bottom border we can use the following code:

```
1. div {  
2.     border-bottom-width: 12px;  
3. }
```

These highly specific longhand border properties include a series of hyphen-separated words starting with the border base, followed by the selected side—`top`, `right`, `bottom`, or `left`—and then width, style, or color, depending on the desired property.

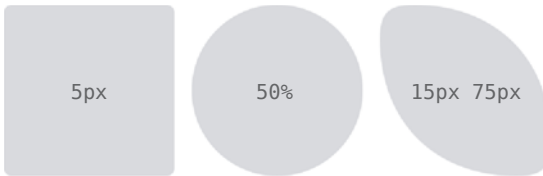
## Border Radius

While we're looking at borders and their different properties, we need to examine the `border-radius` property, which enables us to round the corners of an element.

The `border-radius` property accepts length units, including percentages and pixels, that identify the radius by which the corners of an element are to be rounded. A single value will round all four corners of an element equally; two values will round the `top-left`/`bottom-right` and `top-right`/`bottom-left` corners in that order; four values will round the `top-left`, `top-right`, `bottom-right`, and `bottom-left` corners in that order.

When considering the order in which multiple values are applied to the `border-radius` property (as well as the margin and padding properties), remember that they move in a clockwise fashion starting at the top left of an element.

```
1. div {  
2.   border-radius: 5px;  
3. }
```



**Figure 4.5**  
Different border-radius sizes

The `border-radius` property may also be broken out into longhand properties that allow us to change the radii of individual corners of an element. These longhand properties begin with `border`, continue with the corner's vertical location (`top` or `bottom`) and the corner's horizontal location (`left` or `right`), and then end with `radius`. For example, to change the top-right corner radius of a `<div>`, the `border-top-right-radius` property can be used.

```
1. div {  
2.   border-top-right-radius: 5px;  
3. }
```

## Box Sizing

Until now the box model has been an additive design. If you set the width of an element to 400 pixels and then add 20 pixels of padding and a border of 10 pixels on every side, the actual full width of the element becomes 460 pixels. Remember, we need to add the width, padding, and border property values together to get the actual, full width of an element.

The box model may, however, be changed to support different calculations. CSS3 introduced the `box-sizing` property, which allows us to change exactly how the box model works and how an element's size is calculated. The property accepts three primary values—`content-box`, `padding-box`, and `border-box`—each of which has a slightly different impact on how the box size is calculated.

## Content Box

The `content-box` value is the default value, leaving the box model as an additive design. If we don't use the `box-sizing` property, this will be the default value for all elements. The size of an element begins with the `width` and `height` properties, and then any `padding`, `border`, or `margin` property values are added on from there.

```
1. div {  
2.     -webkit-box-sizing: content-box;  
3.     -moz-box-sizing: content-box;  
4.     box-sizing: content-box;  
5. }
```

### Browser-Specific Properties & Values

What are all those hyphens and letters on the `box-sizing` property?

As CSS3 was introduced, browsers gradually began to support different properties and values, including the `box-sizing` property, by way of vendor prefixes. As parts of the CSS3 specification are finalized and new browser versions are released, these vendor prefixes become less and less relevant. As time goes on, vendor prefixes are unlikely to be a problem; however, they still provide support for some of the older browsers that leveraged them. We may run across them from time to time, and we may even want to use them should we wish to support older browsers.

Vendor prefixes may be seen on both properties and values, all depending on the CSS specification. Here they are shown on the `box-sizing` property. Browser vendors were free to choose when to use a prefix and when not to. Thus, some properties and values require vendor prefixes for certain browser vendors but not for others.

Moving forward, when a property or value needs a vendor prefix, the prefix will only be used in the introduction of that property or value (in the interest of keeping our code digestible and concise). Do not forget to add the necessary vendor prefixes when you're actually writing the code.

For reference, the most common vendor prefixes are outlined here:

- Mozilla Firefox: `-moz-`
- Microsoft Internet Explorer: `-ms-`
- Webkit (Google Chrome and Apple Safari): `-webkit-`



## Padding Box

The padding-box value alters the box model by including any padding property values within the width and height of an element. When using the padding-box value, if an element has a width of 400 pixels and a padding of 20 pixels around every side, the actual width will remain 400 pixels. As any padding values increase, the content size within an element shrinks proportionately.

If we add a border or margin, those values will be added to the width or height properties to calculate the full box size. For example, if we add a border of 10 pixels and a padding of 20 pixels around every side of the element with a width of 400 pixels, the actual full width will become 420 pixels.

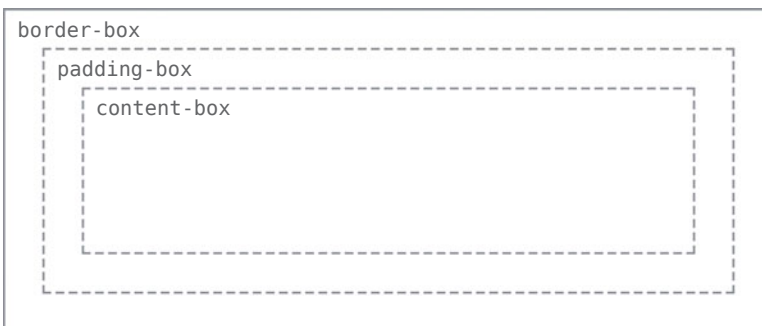
```
1. div {  
2.   box-sizing: padding-box;  
3. }
```

## Border Box

Lastly, the border-box value alters the box model so that any border or padding property values are included within the width and height of an element. When using the border-box value, if an element has a width of 400 pixels, a padding of 20 pixels around every side, and a border of 10 pixels around every side, the actual width will remain 400 pixels.

If we add a margin, those values will need to be added to calculate the full box size. No matter which box-sizing property value is used, any margin values will need to be added to calculate the full size of the element.

```
1. div {  
2.   box-sizing: border-box;  
3. }
```



**Figure 4.6**  
Different box-sizing values allow the width of an element—and its box—to be calculated from different areas

## Picking a Box Size

Generally speaking, the best `box-sizing` value to use is `border-box`. The `border-box` value makes our math much, much easier. If we want an element to be 400 pixels wide, it is, and it will remain 400 pixels wide no matter what padding or border values we add to it.

Additionally, we can easily mix length values. Say we want our box to be 40% wide. Adding a padding of 20 pixels and a border of 10 pixels around every side of an element isn't difficult, and we can still guarantee that the actual width of our box will remain 40% despite using pixel values elsewhere.

The only drawback to using the `box-sizing` property is that as part of the CSS3 specification, it isn't supported in every browser; it especially lacks support in older browsers. Fortunately this is becoming less and less relevant as new browsers are released. Chances are we're safe to use the `box-sizing` property, but should we notice any issues, it's worth looking into which browser those issues are occurring with.

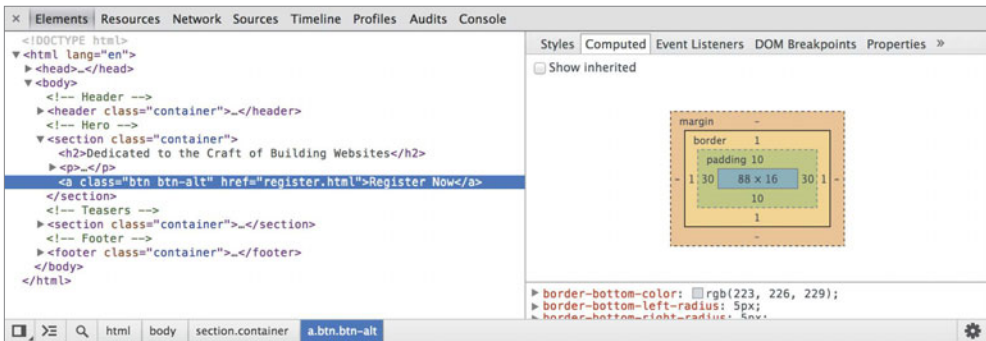
## Developer Tools

Most browsers have what are known as *Developer Tools*. These tools allow us to inspect an element on a page, see where that element lives within the HTML document, and see what CSS properties and values are being applied to it. Most of these tools also include a box model diagram to show the computed size of an element.

To see the Developer Tools in Google Chrome, click "View" within the menu bar and navigate to "Developer" and then "Developer Tools." This loads a drawer at the bottom of the browser window that provides a handful of tools for inspecting our code.

Clicking the magnifying glass at the bottom of this drawer enables us to hover over and then click on different elements on the page to review more information about them. After selecting an element, we'll see a handful of tabs on the right-hand side of the Elements panel within our Developer Tools. Selecting the "Computed" tab will show us a breakdown of the box model for our selected element.

Play around with the Developer Tools, be it in Google Chrome, Mozilla Firefox, Apple Safari, or other browsers; there is much to learn from looking at our code. I generally leave the Developer Tools open at all times when writing HTML and CSS. And I frequently inspect the code of other websites to see how they are built, too.



**Figure 4.7** The Google Chrome Developer Tools, which help us to inspect the HTML and CSS on any page

The box model is one of the most confusing parts of learning how to write HTML and CSS. It is also one of the most powerful parts of HTML and CSS, and once we have it mastered, most everything else—like positioning content—will come to us fairly easily.

## In Practice

Let's jump back into our Styles Conference website to center it on the page and add some more content.

1. Let's start by adjusting our box size to use the border-box version of the box model, which will make sizing all of our elements much easier. Within our `main.css` file, just below our reset, let's add a comment to identify the code for what will become our grid and help determine the layout of our website. We're putting this below our reset so that it falls in the proper position within the cascade.

From there, we can use the universal selector, `*`, along with universal pseudo-elements, `*:before` and `*:after`, to select every imaginable element and change the box-sizing to border-box. Remember, we're going to want to include the necessary vendor prefixes for the box-sizing property, as it is a relatively new property.

```
1.  /*
2.  =====
3.  Grid
4.  =====
5.  */
6.
7.  *,
8.  *:before,
```

```
9.  *:after {
10.    -webkit-box-sizing: border-box;
11.    -moz-box-sizing: border-box;
12.    box-sizing: border-box;
13. }
```

2. Next we'll want to create a class that will serve as a container for our elements. We can use this container class on different elements to set a common width, center the elements on the page, and apply some common horizontal padding.

Just below our universal selector rule set, let's create a selector with a class of `container`. Within this selector let's set our width to 960 pixels, our left and right padding to 30 pixels, our top and bottom margins to 0, and our left and right margins to `auto`.

Setting a width tells the browser definitively how wide any element with the class of `container` should be. Using a left and right margin of `auto` in conjunction with this width lets the browser automatically figure out equal left and right margins for the element, thus centering it on the page. Lastly, the left and right padding ensures that our content isn't sitting directly on the edge of the element and provides a little breathing room for the content.

```
1.  .container {
2.    margin: 0 auto;
3.    padding-left: 30px;
4.    padding-right: 30px;
5.    width: 960px;
6.  }
```

3. Now that we have a container class available to use, let's go ahead and apply the class of `container` throughout our HTML to the `<header>` and `<footer>` elements on each page, including the `index.html`, `speakers.html`, `schedule.html`, `venue.html`, and `register.html` files.

```
1.  <header class="container">...</header>
2.
3.  <footer class="container">...</footer>
```

4. While we're at it, let's go ahead and center the rest of the content on our pages. On the home page, our `index.html` file, let's add the class of `container` to each `<section>` element on the page, one for our hero section (the section that introduces our conference) and one for our teasers section.

```
1.  <section class="container">...</section>
```

Additionally, let's wrap all of the `<h1>` elements on each page with a `<section>` element with the class of `container`.

```
1. <section class="container">
2.
3.   <h1>...</h1>
4.
5. </section>
```

We'll come back and adjust these elements and classes later, but for now we're headed in the right direction.

5. Now that all of our content is centered, let's create some vertical spacing between elements. For starters let's place a 22-pixel bottom margin on a few of our heading and paragraph elements. We'll place and comment on these typography styles below our grid styles.

```
1. /*
2.  =====
3.   Typography
4.  =====
5. */
6.
7.   h1, h3, h4, h5, p {
8.     margin-bottom: 22px;
9.   }
```

We intentionally skipped `<h2>` and `<h6>` elements, as the design does not call for margins on `<h2>` elements and as we won't be using any `<h6>` elements at this time.

6. Let's also try our hand at creating a border and some rounded corners. We'll start by placing a button within the top `<section>` element on our home page, just below the header.

Previously we added an `<a>` element within this `<section>` element. Let's add the classes of `btn` and `btn-alt` to this anchor.

```
1. <a class="btn btn-alt">...</a>
```

Now let's create some styles for those classes within our CSS. Below our typography rule set, let's create a new section of the CSS file for buttons.

To begin let's add the `btn` class and apply some common styles that can be shared across all buttons. We'll want all of our buttons to have a 5-pixel `border-radius`. They should be displayed as `inline-block` elements so we can add padding around all four sides without issue; we'll remove any `margin`.

```
1.  /*
2.  =====
3.  Buttons
4.  =====
5.  */
6.
7.  .btn {
8.    border-radius: 5px;
9.    display: inline-block;
10.   margin: 0;
11. }
```

We'll also want to include styles specific to this button, which we'll do by using the `btn-alt` class. Here we'll add a 1-pixel, solid, gray border with 10 pixels of padding on the top and bottom of the button and 30 pixels of padding on the left and right of the button.

```
1.  .btn-alt {
2.    border: 1px solid #dfe2e5;
3.    padding: 10px 30px;
4.  }
```

Using both the `btn` and `btn-alt` classes on the same `<a>` element allows these styles to be layered on, rendering all of the styles on a single element.

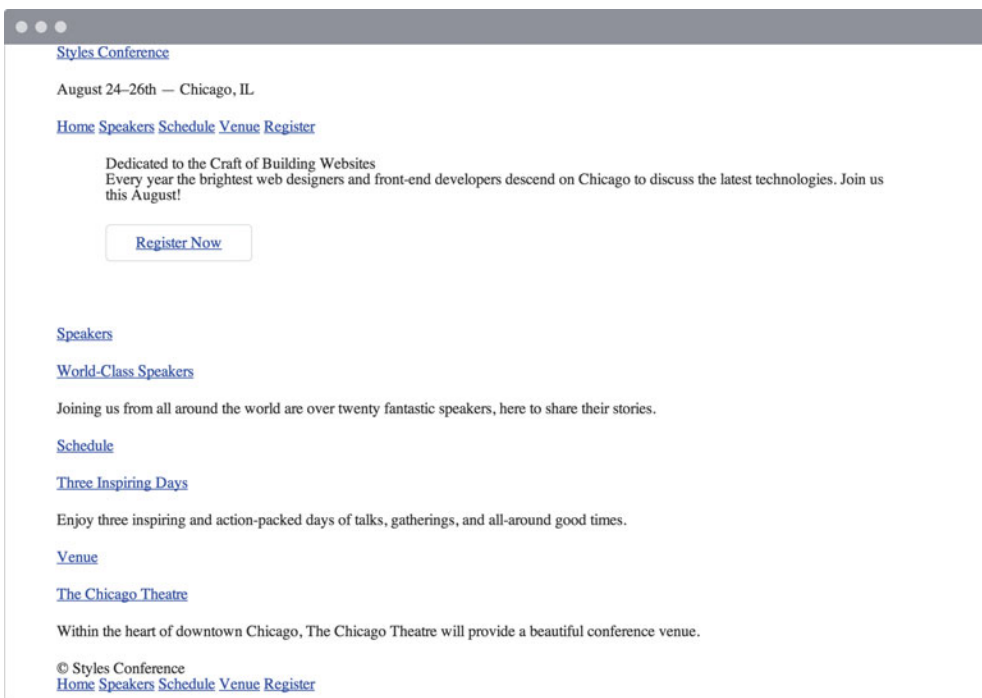
7. Because we're working on the home page, let's also add a bit of padding to the `<section>` element that contains our `<a>` element with the classes of `btn` and `btn-alt`. We'll do so by adding a class attribute value of `hero` to the `<section>` element, alongside the `container` class attribute value, as this will be the leading section of our website.

```
1.  <section class="hero container">
2.    ...
3.  </section>
```

Next we'll want to create a new section within our CSS file for home page styles, and, once we're ready, we'll use the class of `hero` to apply padding around all four sides of the `<section>` element.

```
1.  /*
2.  =====
3.  Home
4.  =====
5.  */
6.
7.  .hero {
8.      padding: 22px 80px 66px 80px;
9.  }
```

Our website is starting to come together, especially the home page, as shown in **Figure 4.8**.



**Figure 4.8** Our Styles Conference home page, taking shape after a few updates

The source code for the exercises within this lesson can be found at <http://learn.shayhowe.com/html-css/opening-the-box-model/>.

## The Universal Selector

In the first step of this exercise we were introduced to the *universal selector*. In CSS the asterisk, `*`, is the universal selector, which selects every element. Rather than listing every single element imaginable, we can use the asterisk as a catch-all to select all elements for us.

The `:before` and `:after` pseudo-elements also mentioned in this step are elements that can be dynamically generated with CSS. We're not going to be using these elements within our project; however, when using the universal selector it's a good practice to also include these pseudo-elements in case they should ever appear.

## Summary

Take a second and pat yourself on the back. I'll wait.

Learning all the different parts of the box model is no small feat. These concepts, although briefly introduced, take quite a bit of time to fully master, and we're on the right path toward doing so.

In brief, within this lesson we talked about the following:

- How different elements are displayed
- What the box model is and why it's important
- How to change the size, including the height and width, of elements
- How to add margin, padding, and borders to elements
- How to change the box sizing of elements and the effects this has on the box model

Now that we have a better understanding of how elements are displayed and sized, it's time to move into positioning these elements.



# Index

- (hyphen), 38
- ; (semicolon), 8, 274
- : (colon), 8
- . (period), 10
- { } (curly brackets), 8, 9, 274
- & (ampersand), 31
- # (hash sign), 10, 46
- < > (angle brackets), 2
- % unit notation, 51

## A

- <a> element, 29
- absolute lengths, 50
- absolute paths, 30
- absolute positioning, 96-98
- absolute value, 96-98
- action attribute, 205
- Adobe Kuler, 47
- alert message example, 136-137
- alignment
  - float values and, 114
  - images, 185-186
  - list items, 170
  - text, 114, 121, 122
  - vendor prefixes, 278-279
  - vertical, 91
- alpha channels, 48
- alt attribute, 179, 271
- alt (alternative) text, 179
- alternative (alt) text, 179
- ampersand (&), 31
- anchor elements, 29
- anchor links, 3, 29
- anchor tags, 3
- angle brackets < >, 2
- <article> element, 25
- <aside> element, 26, 76, 78
- aspect ratio, 180
- attributes
  - for, 213
  - action, 205
  - alt, 179, 271
  - audio, 189-190
  - autoplay, 189, 191
  - charset, 5
  - checkbox, 209
  - cite, 129, 130-131
  - class, 3, 10
  - cols, 208
  - colspan, 237-238
  - container class, 244
  - controls, 189, 191
  - datetime, 255
  - described, 3-4
  - disabled, 215
  - headers, 234
  - height, 180-181
  - hidden, 215
  - hidden, 215
  - href, 3, 11, 12, 30
  - id, 3, 10, 213, 234, 256
  - ID, 172
  - intro ID value, 267
  - loop, 189, 191
  - method, 205
  - multiple, 210
  - name, 205, 208, 210
  - placeholder, 216
  - poster, 192
  - preload, 189-190, 191
  - rel, 11
  - required, 216
  - reversed, 158-160
  - rows, 208
  - rowspan, 237-238
  - scope, 232, 234, 255
  - seamless, 194-195
  - selected, 210
  - src, 3, 179, 189, 190
  - start, 159
  - target, 31
  - type, 190, 205-208
  - value, 160, 208
  - width, 180-181
- audio, 189-191

- audio controls, 192
- audio fallbacks, 190-191
- audio file formats, 193
- audio files, 189-191
- <audio> element, 189-191
- autoplay attribute, 189, 191

## B

- <b> element, 21-22
- background color, 37-38, 242-243
- background images, 134-137
  - background-position, 155-156
  - centering, 135
  - code example, 136-137, 152-153
  - considerations, 134
  - hyperlink paths, 134
  - vs. image elements, 183
  - multiple, 152-153
  - positioning, 135
  - repeating, 134
  - shorthand values, 134, 136, 152
  - specifying size for, 153-155
  - specifying surface area, 155-156
- background pattern, 180
- background property, 133, 134, 142, 164, 183
- background-clip property, 155-156
- background-color property, 133
- background-image property, 134, 142, 183
- background-origin property, 155-156
- background-position property, 135
- background-repeat property, 134
- backgrounds, 132-156
  - color, 133, 137
  - considerations, 132
  - fallback options, 133
  - new CSS properties, 153-156
  - practice example, 137-141
  - transparent, 133
- background-size property, 153-155
- block elements, 18, 59
- block positioning images, 182
- block values, 54, 182
- block-level elements, 29, 54, 55, 182
- <blockquote> element, 128, 130-131
- <body> element, 4, 5
- bold text, 21-22
- border property, 62-64

- border-box value, 67, 155-156
- border-collapse property, 238-239, 241, 243
- borders
  - adding to rows, 241-242
  - box model, 62-64
  - images, 182
  - padding and, 60
  - radius, 63-64
  - sides, 63
  - size, 62-63
  - styles, 62-63
  - tables, 238-242
- border-spacing property, 240-241
- box model, 53-73
- box model
  - border box, 66
  - borders, 62-64
  - box sizing, 64-67
  - content box, 65
  - described, 56
  - element display, 54-55
  - element height, 57-58, 59
  - element padding, 60-62
  - element width, 57-58
  - margins, 59-62
  - padding box, 66
  - practice exercise, 68-73
  - working with, 56-67
- boxes
  - content, 65
  - padding, 66
  - sizing, 64-67
- box-shadow property, 116
- box-sizing property, 64-67
- braces { }, 8, 9
- browsers
  - audio file formats, 190, 193
  - Chrome, 65, 67-68
  - cross-browser compatibility, 12-13
  - cross-browser testing, 13
  - developer tools, 67-68
  - Firefox, 65
  - Google Chrome, 65, 67-68
  - Internet Explorer, 65
  - Safari, 65
  - vendor prefixes, 65, 279
  - video file formats, 193
- <button> element, 211

- buttons
  - background color, 43
  - font size, 43
  - forms, 208-209, 211
  - radio, 208-209
  - styles, 138-139

## C

- capitalize value, 116
- <caption> element, 234
- captions
  - figures, 202
  - table, 234
- cascade, 37-38
- cascading properties, 37-38
- Cascading Style Sheets. *See* CSS
- cells, combining, 237-238
- cf class, 83
- characters
  - encodings, 2
  - hexadecimal colors, 46, 277
  - special, 28
- charset attribute, 5
- check boxes, 209
- checkbox attribute, 209
- Chrome browser, 65, 67-68
- citations, 128, 129, 130-131
- cite attribute, 129, 130-131
- <cite> element, 128, 130-131
- class attribute, 3, 10
- class selectors, 10, 38
- class values, 270
- classes
  - multiple, 42-43
  - names, 275
  - pseudo-classes, 106
  - tips for, 275, 276
  - values, 275
- clear property, 80
- clearfix, 83
- clearfix class, 83
- clearing floats, 80
- closing tags, 3
- code validation, 6
- coding best practices, 266-281
  - CSS, 273-280
  - general guidelines, 281
  - HTML, 267-272
  - reusable layouts, 90-94
- col value, 232
- colon (:), 8
- color
  - background, 37-38, 133, 137, 242-243
  - borders, 62
  - gradients, 146-147
  - hexadecimal values, 46-47, 277
  - HSL/HSLa, 49-50
  - keyword, 44-45
  - links, 137-138
  - margins and, 62
  - opacity, 48
  - padding and, 62
  - RGB/RGBa, 48
  - sRGB, 44
  - in tables, 242-243
  - text, 100, 138
  - transparent, 48
- color channels, 46-50
- color property, 100
- color stops, 146-147
- color values, 42-50
- color wheel, 47
- cols attribute, 208
- colspan attribute, 237-238
- comments
  - in CSS, 19, 273, 274
  - in HTML code, 19
- contain keyword value, 154
- container class, 69
- container class attribute, 252
- content, 74-98. *See also* media
  - absolute positioning, 96-98
  - centering, 69
  - embeddable, 3
  - grouping, 25
  - positioning with floats, 75-86
  - positioning with inline-block, 87-89
  - related, 26
  - relative positioning, 95-96
  - reusable layouts, 90
  - self-contained, 25
  - semantic decisions and, 25, 267-268
  - separating from style, 271
  - source for, 3
  - in tables, 234-235
  - wrapping, 79

- content boxes, 65
- content-box value, 65, 155-156
- controls attribute, 189, 191
- cover keyword value, 154
- creative works, citing, 128
- cross-browser compatibility, 12-13
- cross-browser testing, 13
- CSS (Cascading Style Sheets)
  - best practices, 273-280
  - calculating specificity, 38-39
  - cascading properties, 37-38
  - class names/values, 275
  - code validators, 6
  - color values, 42-50
  - comments in, 19, 273, 274
  - considerations, 2
  - described, 2, 36
  - dropping units from zero values, 278
  - good vs. bad code examples, 273-280
  - length values, 50-52
  - modularized styles, 280
  - multiple lines and, 274
  - property values, 44-52
  - referencing, 11-12
  - reusable layouts, 90
  - shorthand alternatives. *See* shorthand values
  - spacing and, 274
  - terminology, 7-9
  - units of measurement, 50-52
  - vendor prefixes, 278-279
- .css extension, 11
- CSS pseudo-classes, 106
- CSS resets, 12-15, 28
- CSS selectors
  - IDs and, 275
  - tips for, 275
- CSS3 gradient generators, 146
- CSS3 gradients, 146
- curly brackets { }, 8, 9, 274

## D

- data, table, 231-232, 254
- datetime attribute, 255
- <dd> element, 160-161
- description lists, 160-161
- developer tools, 67-68

- dialogue citation, 129
- dialogue quotation, 129
- disabled attribute, 215
- display property, 54-55, 167, 182
- display value, 77, 169
- <div> element, 18-19, 25, 272
- divisions, 18-19, 25
- <dl> element, 160-161
- <!DOCTYPE html> declaration, 4, 5
- Dreamweaver, 4
- drop-down lists, 209-210
- <dt> element, 160-161

## E

- elements
  - absolute positioning, 96-98
  - block-level, 18, 29, 54, 55
  - borders, 62-64
  - classifying, 3
  - described, 2
  - displaying, 54-55
  - floating, 76
  - height of, 57-58, 59
  - hiding, 55
  - identifying, 3
  - indenting, 5
  - inline, 18, 54, 55
  - margins, 59-62
  - nested, 5
  - padding, 60-62
  - relative positioning, 95-96
  - self-closing, 5
  - text-based, 20-23
  - width of, 57-58
- em unit notation, 51
- em units, 51
- <em> element, 22-23, 276
- email addresses
  - linking to, 30-31
  - validation, 216
- Eric Meyer's reset, 12, 13
- error message styles, 216
- external citation, 130-131
- external quotation, 130
- external style sheets, 11, 12

## F

- fallback options
  - audio, 190–191
  - backgrounds, 133
  - fonts, 101
  - video, 191, 193
- fields, text, 205–207
- fieldsets, 214
- `<figcaption>` element, 202
- `<figure>` element, 201–202
- figures, 201–202
- file input, 212
- files
  - adding to forms, 212
  - audio, 189–191
  - comments, 19
  - CSS, 273, 275, 278
  - external, 4, 24
  - gradient image, 142
  - links to, 24
  - organizing, 19
- Firefox browser, 65
- `:first-of-type` pseudo-class selector, 261
- `float` property, 75, 77, 79, 114, 167, 182
- floating
  - clearing floats, 80
  - considerations, 95
  - containing floats, 80–83
  - content, 75–86
  - images, 182–183
  - lists, 167–168
- font families, 101
- font property, 104
- font variants, 102
- `@font-face` at-rule, 124
- `font-family` property, 101, 124
- fonts
  - bold, 102–103
  - considerations, 99, 125
  - described, 100
  - embedded, 99, 124–127
  - example code, 105–106
  - fallback options, 101
  - Google Fonts, 125
  - italics, 102
  - licensing issues, 125
  - practice exercise, 106–113
  - properties, 101–113
  - shorthand values, 104
  - size, 51, 101
  - styles, 102
  - vs. typefaces, 100
  - web-safe, 123–124
  - weights, 102–103
- `font-size` property, 101
- `font-style` property, 102
- `font-variant` property, 102
- `font-weight` property, 102–103, 126, 127
- `<footer>` element, 26, 28
- footers, 26, 235
- for attribute, 213
- `<form>` element, 205
- forms, 204–228
  - adding files to, 212
  - adding to pages, 205
  - buttons, 208–209, 211
  - check boxes, 209
  - disabling elements/controls, 215
  - drop-down lists, 209–210
  - example code, 217–219
  - fieldsets, 214
  - hidden inputs, 212
  - initializing, 205
  - input attributes/values, 215–217
  - labels, 213
  - legends, 214–215
  - login, 217–219
  - multiple selections, 210
  - organizing elements in, 212–215
  - overview, 204
  - placeholder controls, 216
  - practice example, 219–226
  - required values, 216
  - text fields, 205–207
  - textareas, 208
  - validation, 216

## G

- gif format, 180
- Google Chrome browser, 65, 67–68
- Google Fonts, 125
- gradient backgrounds, 142–151
  - changing direction of, 143–144
  - color stops, 146–147

- considerations, 142
- CSS3, 146
- example code, 147-148
- linear, 142-144
- practice example, 148-151
- radial, 145-146
- vendor prefixes, 142

gradients

- background. See gradient backgrounds

grid class attribute, 91, 92, 171, 195, 220, 252.

group class, 81

## H

<h> element, 5, 20, 24

hash sign (#), 10, 46

<head> element, 4, 5, 11, 24

<header> element, 24, 27

headers

- table, 232-234, 235
- text, 24, 27

headers attribute, 234

headings, 5, 20

height attribute, 180-181

height property, 56, 58, 59, 180

hexadecimal colors, 46-47, 277

hexadecimal values, 100, 133, 147

hidden attribute, 215

hidden inputs, 212

hiding elements, 55

:hover pseudo-class, 106

href attribute, 3, 11, 12, 30

hsl() function, 49

HSLa value, 133

HSL/HSLa colors, 49-50

HTML (HyperText Markup Language), 2-4

HTML code

- best practices, 267-272
- class values, 270
- comments in, 19
- considerations, 2
- described, 2
- divisions, 18-19
- document structure, 268-269
- example of basic code, 4-5
- good vs. bad code examples, 267-272

headings, 20

hyperlinks. See hyperlinks

ID values, 270

inline styles and, 271

paragraphs, 21

refactoring code, 272

referencing CSS in, 11-12

removing code, 272

reusable layouts, 90

semantics in, 18, 267-268

spans, 18-19

standards-compliant markup, 267

structural elements, 23-29

syntax organization, 269-270

terminology, 2-4

text-based elements, 20-23

validators, 6

version, 4

HTML document structure, 4-7

.html extension, 4

<html> element, 4, 5

hyperlink reference. See href

hyperlinks

- adding, 32-35
- anchor, 3, 29
- background images, 134
- to citations, 128, 129
- colors, 137-138
- creating, 29-35
- described, 29
- to email addresses, 30-31
- navigation, 24
- opening links in new window, 31
- to other pages of website, 30
- to parts of same page, 32
- to quotations, 129
- specifying, 3

HyperText Markup Language. See HTML

hyphen (-), 38

## I

<i> element, 22-23

icons, 180

id attribute, 3, 10, 213, 234, 256

ID attributes, 172

ID selectors, 10, 38, 39

- ID values, 270
- `<iframe>` element, 193-195
- image elements, 183
- image formats, 180
- images, 179-188
  - adding to pages, 179
  - alignment, 185-186
  - `alt` attribute, 271
  - aspect ratio, 180
  - background. See background images
  - borders, 182
  - distorted, 180
  - embedded, 179
  - floating, 182-183
  - flush left/right, 182-183
  - margins, 182-183
  - padding, 182
  - positioning, 181-183
  - practice exercise, 183-188
  - sizing, 180-181
  - spacing, 182-183
- `<img>` element, 179, 181, 183
- indenting text, 115
- `index.html` file, 15
- inline elements, 18
- inline frames, 193-195
- inline styles, 11, 271
- `inline` value, 54, 166-167
- inline-block elements
  - positioning content with, 87-89
  - removing spaces between, 88-89
  - sizing, 59
  - space between, 55
- `inline-block` value, 55, 166-167
- inline-level elements, 59
- `<input>` element, 205
- `inside` property value, 165, 166
- internal style sheets, 11
- Internet Explorer, 65
- `intro` ID attribute value, 267
- italicized text, 22-23, 102

## J

- `jpg` format, 180

## K

- key selector, 40
- keyword color values, 44-45, 47

## L

- `<label>` element, 213
- labels, 213
- `:last-child` pseudo-class selector, 170, 241, 253
- `:last-of-type` pseudo-class selector, 261
- leading, 103-104
- legends, 214-215
- length values, 50-52
- letter spacing, 117
- letter-spacing property, 117
- `<li>` element, 158
- linear gradients, 142-143
- `linear-gradient()` function, 143, 149
- `line-height` property, 103-104
- `<link>` element, 11-12, 125-126
- links
  - adding, 32-35
  - anchor, 3, 29
  - background images, 134
  - to citations, 128, 129
  - colors, 137-138
  - creating, 29-35
  - described, 29
  - to email addresses, 30-31
  - navigation, 24
  - opening links in new window, 31
  - to other pages of website, 30
  - to parts of same page, 32
  - to quotations, 129
  - specifying, 3
- list item markers
  - floating and, 167
  - setting content of, 163-165
  - using images as, 164-165
- list items
  - alignment, 170
  - styling, 163-166
- lists, 157-177
  - changing values in, 160
  - considerations, 157
  - description, 160-161

- drop-down, 209-210
- floating, 167-168
- horizontally displaying, 166-169
- navigational, 168-169
- nesting, 162-163
- numbered, 158-160
- ordered, 158-160
- overview, 157
- practice example, 169-176
- reverse order, 158-160
- sample code, 168-169
- unordered, 158

- list-style property value, 166
- list-style-position property, 165-166
- list-style-type property, 163-165
- login forms, 217-219
- loop attribute, 189, 191
- lowercase value, 116

## M

- "magic corners," 144
- mailto:, 31
- main.css file, 12
- margin property, 59-62, 182-183
- margins
  - images, 182-183
  - overview, 59-62
- measurement, units of, 50-52
- media, 178-203. *See also* content
  - audio, 189-191
  - considerations, 178
  - embedded, 193
  - images. *See* images
  - inline frames, 193-195
  - video, 191-193
- media player, 192
- <meta> element, 5
- method attribute, 205
- mp3 format, 190
- multiple attribute, 210

## N

- name attribute, 205, 208, 210
- <nav> element, 24
- navigation menus, 33-34, 168-169

- navigational links, 24
- navigational lists, 168-169
- nested elements, 5
- nesting lists, 162-163
- none value, 55
- Normalize.css, 12-13
- Notepad++, 4
- :nth-child pseudo-class selector, 242-243
- number sign (#), 10, 46
- numbered lists, 158-160

## O

- offset class, 96
- ogg format, 190
- <ol> element, 158-160
- :only-of-type pseudo-class selector, 261
- opacity, 48
- opening tags, 3
- <option> elements, 208
- ordered lists, 158-160
- outside property value, 165

## P

- <p> element, 5, 21
- padding, 60-62, 66, 182
- padding property
  - box model, 60-62, 66
  - tables, 260-262
- padding-box value, 66, 155-156
- pages. *See* web pages
- paragraphs, 21
- paths
  - absolute, 30
  - hyperlink, 134
  - relative, 30
- pattern, background, 180
- percentages, 51
- performance, 276
- period (.), 10
- photographs, 180
- pixels, 50
- placeholder attribute, 216
- placeholder controls, 216
- png format, 180
- position property, 95-98



- poster attribute, 192
- pound sign (#), 10, 46
- preload attribute, 189-190, 191
- properties
  - background, 133, 134, 142, 164, 183
  - background-clip, 155-156
  - background-color, 133
  - background-image, 134, 142, 183
  - background-origin, 155-156
  - background-position, 135
  - background-repeat, 134
  - background-size, 153-155
  - border, 62-64
  - border-collapse, 238-239, 241, 243
  - border-spacing, 240-241
  - box-shadow, 116
  - box-sizing, 64-67
  - cascading, 37-38
  - cascading properties, 37-38
  - clear, 80
  - color, 100
  - described, 8
  - display, 54-55, 167, 182
  - float, 75, 77, 79, 114, 167, 182
  - font, 104
  - font-based, 101-113
  - font-family, 101, 124
  - fonts, 101-113
  - font-size, 101
  - font-style, 102
  - font-variant, 102
  - font-weight, 102-103, 126, 127
  - height, 56, 58, 59, 180
  - letter-spacing, 117
  - line-height, 103-104
  - list-style, 166
  - list-style-position, 165-166
  - list-style-type, 163-165
  - margin, 59-62, 182-183
  - padding, 60-62, 66, 260-262
  - position, 95-98
  - text, 113-123
  - text-align, 114, 244-247
  - text-based, 101-123
  - text-decoration, 114
  - text-indent, 115
  - text-shadow, 115-116

- text-transform, 116
  - vertical-align, 244
  - width, 57-58, 180
  - word-spacing, 117
- prose citation, 129
- prose quotation, 129
- pseudo-class selectors, 261
- px unit notation, 50

## Q

- <q> element, 128, 129
- quotations, 128, 129, 130

## R

- radial gradients, 145-146
- radial-gradient() function, 145
- radio buttons, 208-209
- rel attribute, 11
- relative lengths, 51
- relative paths, 30
- relative positioning, 95-96
- relative value, 95-96
- required attribute, 216
- reusable layouts, 90-94
- reversed attribute, 158-160
- rgb() function, 48
- rgba() function, 48
- RGB/RGBA colors, 48
- root directory, 12
- row value, 232
- rows
  - adding borders to, 241-242
  - gradient background, 148
  - styles, 139-140
  - table, 230
- rows attribute, 208
- rowspan attribute, 237-238

## S

- Safari browser, 65
- scope attribute, 232, 234, 255
- seamless attribute, 194-195
- <section> elements, 25, 27, 76, 140, 149
- <select> element, 210

- selected attribute, 210
- selectors
  - additional, 11
  - calculating specificity, 38–39
  - class, 10, 38
  - combining, 40–42
  - described, 8
  - ID, 10, 38, 39
  - spaces within, 41
  - specificity within, 42
  - type, 9, 38
  - working with, 9–11
- semantic elements, 267–268
- semantics, 18, 267–268
- semicolon (;), 8, 274
- shorthand values
  - background images, 134, 136, 152
  - borders, 62
  - example of, 276–277
  - fonts, 104
  - hexadecimal color values, 46, 277
  - list-style property, 166
  - margins, 60–61
  - padding, 60–61
  - tips for, 276
- `<small>` element, 28
- `<source>` elements, 190–191
- spaces
  - between inline-block elements, 88–89
  - within selectors, 41
- spacing
  - borders, 240–241
  - CSS and, 274
  - images, 182–183
  - inline-block elements and, 55
- `<span>` element, 18–19
- spans, 18–19
- special characters, 28
- specificity points, 38, 42
- specificity weight, 38–39, 42
- `src` attribute, 3, 179, 189, 190
- sRGB color, 44
- `start` attribute, 159
- striping, 242–243
- `<strong>` element, 21–22
- style sheets, 11. *See also* CSS
- styles. *See also* CSS
  - borders, 62–63
  - buttons, 138–139
  - error messages, 216
  - fonts, 102
  - layering, 42–43
  - list items, 163–166
  - multiple classes, 42–43
  - rows, 139–140
  - separating content from, 271
  - tables, 248–252, 260–262
- Styles Conference website. *See* websites
- Sublime Text, 4
- submit button, 211

## T

- table data, 231–232, 254
- `<table>` element, 230
- tables, 229–265
  - aligning text in, 244–247
  - borders, 238–242
  - captions, 234
  - color in, 242–243
  - combining cells, 237–238
  - contents of, 235–236
  - creating, 230–234
  - footers, 235
  - headers, 232–234, 235
  - overview, 229
  - padding, 260–262
  - practice example, 252–264
  - rows, 230
  - striping, 242–243
  - structure, 234–238
  - styling, 248–252, 260–262
  - table body, 235
- tags
  - anchor, 3
  - closing, 3, 190, 211
  - described, 3
  - end, 208
  - opening, 3, 190, 211
  - start, 208
- target attribute, 31
- `<tbody>` element, 235
- `<td>` element, 231–232, 234

- terminology
  - CSS, 7-9
  - HTML, 2-4
- text
  - aligning, 114, 121-122
  - aligning in tables, 244-247
  - bold, 21-22, 102-103
  - citations, 128, 129, 130-131
  - color, 100, 138
  - example code, 118-119
  - indenting, 115
  - inline changes, 116
  - italicized, 22-23, 102
  - leading, 103-104
  - letter spacing, 117
  - line height, 103-104
  - practice exercise, 119-123
  - properties, 113-123
  - quotations, 128, 129, 130
  - shadows, 115-116
  - small caps, 102
  - underlined, 114, 119-120
  - word spacing, 117
- text decoration, 114
- text editors, 4
- text fields, 205-207
- text-align property, 114, 244-247
- <textarea> element, 208
- textareas, 208
- text-based elements, 20-23
- text-decoration property, 114
- text-indent property, 115
- text-shadow property, 115-116
- text-transform property, 116
- TextWrangler, 4
- <tfoot> element, 235
- <th> element, 232, 234
- <thead> element, 235
- <time> element, 255
- <title> element, 5
- <tr> element, 230
- tracking, 117
- transparency, 48
- transparent backgrounds, 133
- .txt extension, 4
- type attribute, 190, 205-208

- type selectors, 9, 38
- typeface weights, 103
- typefaces
  - considerations, 99, 125
  - described, 100
  - vs. fonts, 100
  - licensing issues, 125
- typography, 99-131

## U

- <u> element, 158
- underlined text, 114, 119-120
- units of measurement, 50-52
- unordered lists
  - described, 158
  - practice example, 169-176
  - sample code, 168-169
- uppercase value, 116
- url ( ) function, 134
- URLs, 193

## V

- validation
  - code, 6
  - email, 216
  - forms, 216
  - standards-compliant markup, 267
- value attribute, 160, 208
- values
  - described, 8-9
- vendor prefixes, 65, 142, 278-279
- vertical alignment, 91
- vertical margins, 60
- vertical padding, 60
- vertical-align property, 244
- video, 191-193
- video controls, 192
- video fallbacks, 193
- video file formats, 193
- video hosting websites, 193
- video player, 193
- <video> element, 191-193
- Vimeo embedded video, 193

## W

wav format, 190

web browsers

- audio file formats, 190, 193

- Chrome, 65, 67-68

- cross-browser compatibility, 12-13

- cross-browser testing, 13

- developer tools, 67-68

- Firefox, 65

- Google Chrome, 65, 67-68

- Internet Explorer, 65

- Safari, 65

- vendor prefixes, 65, 279

- vendor prefixes and, 65, 279

- video file formats, 193

web pages

- adding forms to, 205

- adding images to, 179

- building structure, 23-29

- links on, 32

Webkit, 65

web-safe fonts, 123-124

websites

- adding audio, 189-191

- adding container class to, 69

- adding content. *See* content

- adding CSS to, 13-15

- adding figures/captions, 201-202

- adding forms. *See* forms

- adding images. *See* images

- adding inline frames, 193-200

- adding links, 32-35

- adding multiple pages, 32-35

- adding navigation menu, 33-34

- adding new pages, 34

- adding structure to, 26-29

- adding video, 191-193

- adjusting box size, 68-69

- creating, 6-7

- links to pages on, 30

- positioning images, 181-183

- reusable layouts, 90-94

- video hosting, 193

width attribute, 180-181

width property, 57-58, 180

word spacing, 117

word-spacing property, 117

## Y

YouTube videos, 193

## Z

zero values, 278