learn enough

HTML, CSS AND LAYOUT TO BE DANGEROUS

LEE DONAHOE
MICHAEL HARTL

FREE SAMPLE CHAPTER |

Praise for Learn Enough Tutorials

"I have nothing but fantastic things to say about @LearnEnough courses. I am just about finished with the #javascript course. I must say, the videos are mandatory because @mhartl will play the novice, and share in the joy of having something you wrote actually work!"

—Claudia Vizena

"I must say, this Learn Enough series is a masterpiece of education. Thank you for this incredible work!"

—Michael King

"I want to thank you for the amazing job you have done with the tutorials. They are likely the best tutorials I have ever read."

—Pedro Iatzky

This page intentionally left blank

Learn Enough HTML, CSS and Layout to Be Dangerous

Learn Enough Series from Michael Hartl

Visit **informit.com/learn-enough** for a complete list of available publications.

The **Learn Enough** series teaches you the developer tools, Web technologies, and programming skills needed to launch your own applications, get a job as a programmer, and maybe even start a company of your own. Along the way, you'll learn technical sophistication, which is the ability to solve technical problems yourself. And Learn Enough always focuses on the most important parts of each subject, so you don't have to learn everything to get started—you just have to learn enough to be dangerous. The Learn Enough series includes books and video courses so you get to choose the learning style that works best for you.

Pearson
Addison-Wesley

twitter.com/informIT

informIT.com
the trusted technology learning source

Learn Enough HTML, CSS and Layout to Be Dangerous

An Introduction to Modern Website Creation and Templating Systems

Lee Donahoe
Michael Hartl

Addison-Wesley
Boston • Columbus • New York • San Francisco • Amsterdam • Cape Town
Dubai • London • Madrid • Milan • Munich • Paris • Montreal • Toronto • Delhi • Mexico City
São Paulo • Sydney • Hong Kong • Seoul • Singapore • Taipei • Tokyo

Cover image: smalvik/Shutterstock
Figures 1.5-1.7, 5.7, 9.2(b), 16.11: GitHub, Inc.
Figures 1.8, 1.10, 3.2, 13.8, 13.9, 14.12, 15.29(b), 17.1, 17.3-17.5: Google LLC
Figures 1.9, 1.12, 1.13, 2.1-2.7, 2.9-2.11, 3.1, 3.3-3.5, 3.8-3.10, 4.1-4.4, 4.6, 4.7, 4.9-4.16, 5.6, 5.10, 6.1-6.5, 7.4, 7.5, 7.9-7.17, 7.20-7.23, 8.2-8.4, 8.6, 8.8-8.14, 8.18-8.25, 8.27-8.33, 8.36, 8.38, 9.3, 9.6, 9.7, 9.9-9.11, 9.14, 9.16-9.20, 9.29-9.39, 9.43, 9.44, 10.3-10.8, 10.10, 10.11, 10.13, 10.14, 10.17, 10.18, 11.08, 11.12, 11.15-11.22, 12.2, 12.5, 12.6, 12.8, 12.9-12.13, 13.3-13.7, 13.11-13.16, 13.20, 13.22, 13.23, 14.3, 14.4, 14.8-14.10, 14.13, 14.14, 14.17, 15.29(a), 17.6-17.8: Apple Inc.
Figure 4.15, page 92: Icon courtesy of Twitter, Inc.
Figures 5.4, 14.1: Microsoft
Figures 8.15, 9.4, 9.15, 9.42, 13.19: Amazon.com, Inc.
Figure 15.46: Caniuse.com
Figure 16.1: Hover
Figures 16.8-16.10, 17.2: Cloudflare, Inc.

Library of Congress Control Number: 2022935502

ISBN-13: 978-0-13-784310-7
ISBN-10: 0-13-784310-0

ScoutAutomatedPrintCode

Contents

Preface xvii
About the Authors xxiii

PART I HYPERTEXT MARKUP LANGUAGE 1

Chapter 1 Basic HTML 3

1.1 Introduction 6
1.2 HTML Tags 8
1.2.1 Exercises 11
1.3 Starting the Project 12
1.3.1 Exercises 17
1.4 The First Tag 17
1.4.1 Exercises 20
1.5 An HTML Skeleton 20
1.5.1 Exercises 27

Chapter 2 Filling in the Index Page 29

2.1 Headings 29
2.1.1 Exercise 31
2.2 Text Formatting 31

2.2.1 Emphasized Text 32
2.2.2 Strong Text 34
2.2.3 Exercises 35
2.3 Links 35
2.3.1 Exercises 38
2.4 Adding Images 41
2.4.1 Hotlinking 44
2.4.2 Exercises 48

Chapter 3 More Pages, More Tags 51

3.1 An HTML Page About HTML 51
3.1.1 Exercises 53
3.2 Tables 54
3.2.1 Block Elements 55
3.2.2 Inline Elements 59
3.2.3 Exercises 60
3.3 Divs and Spans 62
3.3.1 Exercises 66
3.4 Lists 66
3.4.1 Exercise 68
3.5 A Navigation Menu 68
3.5.1 Exercises 72

Chapter 4 Inline Styling 73

4.1 Text Styling 74
4.1.1 Exercises 79
4.2 Floats 79
4.2.1 Exercise 82
4.3 Applying a Margin 82
4.3.1 Exercises 85
4.4 More Margin Tricks 85
4.4.1 Exercise 88
4.5 Box Styling 88
4.5.1 Exercises 89
4.6 Navigation Styling 90
4.6.1 Exercises 92

4.7 A Taste of CSS 93
4.7.1 Internal Stylesheets 93
4.7.2 External Stylesheets 96
4.7.3 Exercises 97
4.8 Conclusion 98

PART II CASCADING STYLE SHEETS AND PAGE LAYOUT 101

Chapter 5 Introduction to CSS 103

5.1 You're a Front-End Developer 106
5.1.1 So, What Is Front-End Development? 108
5.2 CSS Overview and History 109
5.2.1 CSS Is Always Changing 110
5.2.2 How Did CSS Develop? 112
5.2.3 The Bog of Eternal Subjectivity 115
5.3 Sample Site Setup 116
5.3.1 Exercise 121
5.4 Start Stylin' 121
5.4.1 Exercises 128
5.5 CSS Selectors 128
5.5.1 Exercises 132

Chapter 6 The Style of Style 133

6.1 Naming Things 134
6.2 When and Why 137
6.3 Priority and Specificity 140
6.3.1 Exercises 145
6.4 How to Be a Good Styling Citizen 145
6.4.1 Exercises 156

Chapter 7 CSS Values: Color and Sizing 157

7.1 CSS Color 157
7.1.1 Hexadecimal Colors 158
7.1.2 Setting Color and Transparency via `rgb()` and `rgba()` 161
7.1.3 Exercises 163

7.2 Introduction to Sizing 163
7.3 Pixels (and Their Less-Used Cousin, the Point) 164
7.3.1 Exercise 168
7.4 Percentages 169
7.4.1 Percentage Fonts 174
7.4.2 Exercises 174
7.5 `em` 175
7.5.1 Exercises 181
7.6 `rem` Isn't Just for Dreaming 181
7.6.1 Exercises 184
7.7 `vh`, `vw`: The New(er) Kids on the Block 184
7.7.1 Exercises 189
7.8 Just Make It Look Nice 190
7.8.1 Exercises 191

Chapter 8 The Box Model 193

8.1 Inline vs. Block 193
8.1.1 `display: none` 194
8.1.2 `display: block` 195
8.1.3 `display: inline` 196
8.1.4 `display: inline-block` 197
8.1.5 `display: flex` 199
8.1.6 Exercises 199
8.2 Margins, Padding, and Borders 199
8.2.1 Margin Weirdness 202
8.2.2 Exercises 206
8.3 Floats 206
8.3.1 Clearing Floats 208
8.3.2 Exercises 214
8.4 A Little More About the `overflow` Style 214
8.5 Inline Block 219
8.5.1 Exercises 223
8.6 Margins for Boxes 223
8.6.1 An Exception: `margin: auto` 229
8.6.2 Yet Another Exception: Negative Margins 231
8.6.3 Exercises 234

8.7 Padding… Not Just for Chairs 234
8.7.1 Exercise 235
8.8 Fun with Borders 235
8.8.1 Border Radius 238
8.8.2 Making Circles 238
8.8.3 Line Height 244
8.8.4 Syncing Up 245
8.8.5 Exercises 249

Chapter 9 Laying It All Out 251

9.1 Layout Basics 251
9.2 Jekyll 253
9.2.1 Installing and Running Jekyll 254
9.2.2 Exercises 259
9.3 Layouts, Includes, and Pages (Oh My!) 259
9.3.1 Layouts/Layout Templates 259
9.3.2 Includes 261
9.3.3 Pages/Page Templates 261
9.3.4 Posts, and Post-Type Files 261
9.4 The Layout File 261
9.4.1 Exercises 263
9.5 CSS File and Reset 264
9.5.1 Exercises 274
9.6 Includes Intro: Head and Header 275
9.6.1 Page Header: Up Top! 277
9.6.2 Navigation and Children 280
9.6.3 Exercise 284
9.7 Advanced Selectors 284
9.7.1 Pseudo-Classes 284
9.7.2 Exercises 286
9.7.3 `first-child` 287
9.7.4 Exercise 288
9.7.5 Siblings 288
9.7.6 Exercise 291

9.8 Positioning 291
9.8.1 A Real Logo 304
9.8.2 Exercise 308
9.9 Fixed Header 309
9.9.1 Exercise 312
9.10 A Footer, and Includes in Includes 312
9.10.1 Exercise 325

Chapter 10 Page Templates and Frontmatter 327

10.1 Template Content 327
10.1.1 Exercises 330
10.2 There's No Place Like Home 330
10.2.1 Exercises 342
10.3 More Advanced Selectors 342
10.3.1 The `:before` and `:after` Pseudo-Elements 343
10.3.2 The `:before` and `:after` CSS Triangle 347
10.3.3 Exercises 356
10.4 Other Pages, Other Folders 356
10.4.1 Exercises 360

Chapter 11 Specialty Page Layouts with Flexbox 361

11.1 Having Content Fill a Container 363
11.1.1 Exercises 371
11.2 Vertical Flex Centering 371
11.2.1 Exercises 375
11.3 Flexbox Style Options and Shorthand 375
11.3.1 Flex Container Properties 375
11.3.2 Flex Item Properties 376
11.3.3 Exercises 381
11.4 Three-Column Page Layout 381
11.4.1 Exercises 386
11.5 A Gallery Stub 386
11.5.1 Exercises 395

Chapter 12 Adding a Blog 397

12.1 Adding Blog Posts 398

12.1.1 Blog Index Structure 402
12.1.2 Exercises 411
12.2 Blog Index Content Loop 412
12.2.1 Exercises 418
12.3 A Blog Post Page 419
12.3.1 Exercises 427

Chapter 13 Mobile Media Queries 429

13.1 Getting Started with Mobile Designs 429
13.1.1 Exercise 433
13.1.2 How to See in Mobile (Without Looking at Your Phone) 434
13.2 Mobile Adaptation 438
13.2.1 Exercise 449
13.3 Mobile Viewport 449
13.3.1 Exercise 453
13.4 Dropdown Menu 453
13.4.1 The Hitbox 454
13.4.2 Making the Dropdown 455
13.4.3 Exercise 463
13.5 Mobile Dropdown Menu 463
13.5.1 Exercises 473

Chapter 14 Adding More Little Touches 475

14.1 Custom Fonts 475
14.1.1 Installing Vector Image Fonts 477
14.1.2 Loading Text Fonts via a CDN 483
14.1.3 Exercises 488
14.2 Favicons 488
14.2.1 Exercise 490
14.3 Custom Title and Meta Description 490
14.3.1 Custom Title 492
14.3.2 Custom Descriptions 494
14.3.3 Exercise 497
14.4 Next Steps 497

Chapter 15 CSS Grid 499

15.1 CSS Grid at a High Level 501
15.2 A Simple Grid of Content 504
15.2.1 Grid Columns and the Grid `fr` Unit 507
15.2.2 Grid Rows and Gaps 510
15.2.3 Exercises 515
15.3 `minmax`, `auto-fit`, and `auto-fill` 515
15.3.1 Using Grid `auto-fit` 516
15.3.2 Relative Spanning Columns 522
15.3.3 Leveling Up CSS Grid Understanding 524
15.3.4 Exercises 527
15.4 Grid Lines, Areas, and Layouts 527
15.4.1 Getting Started with Grid Lines 529
15.4.2 The Simple Grid Layout 534
15.4.3 Named Lines and Areas 540
15.4.4 Overlapping Using Grid 545
15.4.5 Source-Independent Positioning 546
15.4.6 Finishing the Layout 550
15.4.7 Exercises 555
15.5 Grid on the Inside 556
15.5.1 Setting Up the Page 559
15.5.2 Adding a Global Grid and Header Positioning 563
15.5.3 Using Building Blocks and Justifying 570
15.5.4 More Column Positioning 574
15.5.5 Using Overlapping in a Feature Section 575
15.5.6 Starting at a Specific Column and Self-Aligning 582
15.5.7 Grid Inside a Grid Inside a Page 584
15.5.8 Exercises 589
15.6 Conclusion 589

PART III CUSTOM DOMAINS 591

Chapter 16 A Name of Our Own 593

16.1 Custom Domain Registration 594
16.1.1 What to Register? 594
16.1.2 You've Got a Domain, Now What? 598

16.2 Cloudflare Setup 599
16.2.1 Cloudflare Features 599
16.2.2 Cloudflare Signup 604
16.2.3 Connecting Registrar Nameservers 604
16.3 Custom Domains at GitHub Pages 606
16.3.1 Configuring Cloudflare for GitHub Pages 607
16.3.2 Configuring GitHub Pages 610
16.3.3 Cloudflare Page Rules 613
16.3.4 Profit!! 618

Chapter 17 Custom Email 619

17.1 Google Mail 619
17.1.1 Google Workspace Signup 621
17.2 MX Records 622
17.3 Site Analytics 626
17.3.1 Add Snippet 627
17.4 Conclusion 630

Index 635

This page intentionally left blank

Preface

Learn Enough HTML, CSS and Layout to Be Dangerous teaches you how to make modern websites using HyperText Markup Language (HTML) and Cascading Style Sheets (CSS). This tutorial includes several much-neglected yet essential techniques for page layout, including more advanced CSS techniques like flexbox and grid. It also covers the use of a static site generator to make websites that are easy to maintain and update. Finally, this tutorial shows you how to register and configure custom domains, including both custom URLs and custom email addresses. You can think of *Learn Enough HTML, CSS and Layout to Be Dangerous* as "a website in a box": everything you need (and nothing you don't) to design, build, and deploy modern, professional-grade websites.

The only prerequisites for *Learn Enough HTML, CSS and Layout to Be Dangerous* are knowledge of the Unix command line, a text editor, and version control with Git (as covered, for example, by *Learn Enough Developer Tools to Be Dangerous*). These prerequisites allow us to use good software development practices throughout the tutorial. This includes using a text editor to ensure readable code formatting and using version control to track changes in our projects. It also enables frequent deployment to production (for free!) using GitHub Pages.

The skills you'll develop in this tutorial are valuable whether your interest is in collaborating with developers or becoming a developer yourself. No matter what your

goals are—level up in your current job, start a new career, or even start your own company—*Learn Enough HTML, CSS and Layout* will help get you where you want to go. To get you there as quickly as possible, throughout the tutorial we'll focus on the most important aspects of the subject, grounded in the philosophy that you don't have to learn everything to get started—you just have to learn enough to be *dangerous*.

In addition to teaching you specific skills, this tutorial also helps you develop *technical sophistication*—the seemingly magical ability to solve practically any technical problem. Technical sophistication includes concrete skills like version control and HTML, as well as fuzzier skills like Googling the error message and knowing when to just reboot the darn thing. Throughout *Learn Enough HTML, CSS and Layout*, we'll have abundant opportunities to develop technical sophistication in the context of real-world examples.

Finally, although the individual parts of this tutorial are as self-contained as possible, they are also extensively cross-referenced to show you how all the different pieces fit together. You'll learn how to use CSS to style your HTML elements into a flexible multicolumn layout, use a static site generator to put the same elements on every page without repeating any code, and then deploy your site to the live Web using a custom domain of your choice. The result is an integrated introduction to the foundations of front-end web development that's practically impossible to find anywhere else.

HyperText Markup Language

Part I of *Learn Enough HTML, CSS and Layout to Be Dangerous*, also known as *Learn Enough HTML to Be Dangerous* (https://www.learnenough.com/html), is an introduction to HyperTextMarkup Language, the language of the World Wide Web. It doesn't assume any prior knowledge of web technologies (though readers of *Learn Enough Developer Tools to Be Dangerous* will quickly realize they got a big head start when developing a sample website using version control).

Like all Learn Enough tutorials, *Learn Enough HTML to Be Dangerous* is structured as a technical narrative, with each step carefully motivated by real-world uses. Chapter 1 starts with a "hello, world!" page that you'll immediately deploy to production (!). We'll then fill in the index page with formatted text, links, and images in Chapter 2, expanding it into a multiple-page site with more advanced features like tables and lists in Chapter 3. Finally, we'll add some inline styling in Chapter 4, which will allow us to see the effect of simple style rules on plain HTML elements.

The result of finishing *Learn Enough HTML to Be Dangerous* is a mastery of the core HTML needed for making static websites. It also gives

you a big head start on learning how to develop dynamic web applications with technologies like JavaScript (*Learn Enough JavaScript to Be Dangerous* (https://www.learnenough.com/javascript)) or Ruby and Ruby on Rails (*Learn Enough Ruby to Be Dangerous* (https://www.learnenough.com/ruby) and the *Ruby on Rails Tutorial* (https://www.railstutorial.org/)).

Cascading Style Sheets and Page Layout

Building on the simple styling techniques introduced in Chapter 4 of Part I, Part II—also known as *Learn Enough CSS and Layout to Be Dangerous* (https://www.learnenough.com/css-and-layout)—covers both web design with Cascading Style Sheets and front-end web development with a static site generator. We know of no comparable tutorial that brings all this material together in one place, and the result is the ability to make and deploy websites that are attractive, maintainable, and 100% professional-grade.

In Chapter 5, we'll learn the basics of CSS declarations and values by starting with a few super-simple elements on a sample page. We'll end with a first introduction to the essential technique of CSS selectors to target particular page elements for styling. In Chapter 6, we'll discuss aspects of selectors that are important to get right at the beginning of a project, with a focus on managing complexity and maintaining flexibility by choosing good names for things (including an introduction to CSS color conventions).

Chapter 7 introduces two of the most important kinds of CSS values: colors and sizes. These lay an essential foundation for Chapter 8 on the box model, which determines how different elements fit together on the page.

In Chapter 9 and Chapter 10, we'll take the page that we've been working on and factor it into a layout using a static site generator called Jekyll to build professional-grade websites that are easy to maintain and update. In Chapter 11, we'll learn how to make flexible page layouts using flexbox, adding layouts for a photo gallery page (covered in *Learn Enough JavaScript to Be Dangerous*) and a blog with posts.

In Chapter 12, we'll add the blog itself, showing how to use Jekyll to make a professional-grade blog without black-box solutions like WordPress or Tumblr. Because a large and growing amount of web traffic comes from mobile devices, in Chapter 13 we'll cover the basics of using CSS and media queries to make mobile-friendly sites without violating the DRY ("Don't Repeat Yourself") principle.

As a concluding step in developing the main sample application, in Chapter 14 we'll add the kinds of little details that make a site feel complete. The result will be an industrial-strength, nicely styled site deployed to the live Web.

Finally, as a special bonus, in Chapter 15 we'll introduce a more recent and advanced layout technique known as CSS grid. The result is a largely self-contained discussion of how to use grid to accomplish some of the same effects mentioned in previous chapters, as well as some effects you can only accomplish easily with grid.

Custom Domains

In Part III, also known as *Learn Enough Custom Domains to Be Dangerous* (https://www.learnenough.com/custom-domains), you'll learn how to associate your website with a custom domain. This means your site will live at a domain like example.com instead of example.someoneelsesdomain.com—in other words, at a domain you control and that no one can ever take away.

Chapter 16 shows you how to register a custom domain, including guidance on how to pick a good domain name and a discussion of the pros and cons of various top-level domains (TLDs). You'll also learn how to use Cloudflare to configure the DNS settings for your custom domain. As part of this, you'll learn how to use Secure Sockets Layer/Transport Layer Security (SSL/TLS) to make sure your site is secure and how to redirect URLs for a more pleasant user experience.

Chapter 17 shows you how to use custom email addresses with your domain using Google Workspace. The result is the ability to use yourname@example.com instead of yourname152@gmail.com. As a special bonus, you'll learn how to use another Google service, Google Analytics, to monitor traffic to your site and gain insight into how visitors are using it.

Additional Features

In addition to the main tutorial material, *Learn Enough HTML, CSS and Layout to Be Dangerous* includes a large number of exercises to help you test your understanding and to extend the material in the main text. The exercises include frequent hints and often include the expected answers, with community solutions available by separate subscription at www.learnenough.com.

Final Thoughts

Learn Enough HTML, CSS and Layout to Be Dangerous covers everything you need to know to make a website for a personal homepage, hobby, or business—it's basically a one-stop shop for all things "Web." After learning the techniques covered in this tutorial, and especially after developing your technical sophistication, you'll know

everything you need to design and deploy professional-grade websites. You'll also be ready for a huge variety of other resources, including books, blog posts, and online documentation. *Learn Enough JavaScript to Be Dangerous*, which builds on this tutorial to make a website with an interactive image gallery, is especially recommended. You can even go on to learn dynamic, database-backed web development with *Learn Enough Ruby to Be Dangerous* and the *Ruby on Rails Tutorial*.

Learn Enough Scholarships

Learn Enough is committed to making a technical education available to as wide a variety of people as possible. As part of this commitment, in 2016 we created the Learn Enough Scholarship program (https://www.learnenough.com/scholarship). Scholarship recipients get free or deeply discounted access to the Learn Enough All Access subscription, which includes all of the Learn Enough online book content, embedded videos, exercises, and community exercise answers.

As noted in a 2019 RailsConf Lightning Talk (https://youtu.be/AI5wmnzzBqc?t=1076), the Learn Enough Scholarship application process is incredibly simple: just fill out a confidential text area telling us a little about your situation. The scholarship criteria are generous and flexible—we understand that there are an enormous number of reasons for wanting a scholarship, from being a student, to being between jobs, to living in a country with an unfavorable exchange rate against the U.S. dollar. Chances are that, if you feel like you've got a good reason, we'll think so, too.

So far, Learn Enough has awarded more than 2,500 scholarships to aspiring developers around the country and around the world. To apply, visit the Learn Enough Scholarship page at www.learnenough.com/scholarship. Maybe the next scholarship recipient could be you!

This page intentionally left blank

About the Authors

Lee Donahoe is Learn Enough cofounder and an entrepreneur, designer, and frontend developer. When he was 16 his father handed him a tutorial on HTML, and for more than 25 years since then he has been creating things for the Web. In addition to doing the design and front-end development for Learn Enough (https://www.learnenough.com/), Softcover (https://www.softcover.io/), and the *Ruby on Rails™ Tutorial* (https://www.railstutorial.org/), he is also a cofounder and frontend developer for Coveralls (https://www.coveralls.io/), a leading test coverage analysis service, and also for Buck Mason (https://www.buckmason.com/), a Los Angeles based clothing company that crafts timeless men's and women's clothing. Lee is a graduate of USC, where he studied economics as well as multimedia and creative technologies.

Michael Hartl (https://www.michaelhartl.com/) is the creator of the *Ruby on Rails™ Tutorial*, one of the leading introductions to web development, and is cofounder and principal author at Learn Enough. Previously, he was a physics instructor at the California Institute of Technology (Caltech), where he received a Lifetime Achievement Award for Excellence in Teaching. He is a graduate of Harvard College, has a PhD in Physics from Caltech, and is an alumnus of the Y Combinator entrepreneur program.

This page intentionally left blank

CHAPTER 9
Laying It All Out

Now that we've got a good base of CSS knowledge, it's time to learn how to put everything together into a real website. This chapter and the next is where we really kick things into high gear, with material you're unlikely to see in any other CSS tutorial. To get started, our first step will be to transform our previous work into a more manageable set of *templates* and *page layouts* that can be easily reused and updated (in accordance with the DRY principle (Box 5.2)).

Along the way, we'll add more styling as a way to learn more complex aspects of CSS, while refining our design to be more suitable for use as a personal or business website. Combined with Chapter 10, the result will be a professional-grade example that shows a variety of aspects of modern site design.

9.1 Layout Basics

There are an infinite number of ways that you can design content layouts for the Web, but over the years certain conventions have become common to many sites, as shown in Figure 9.1. These may include elements like a header that contains site navigation and a logo (which typically links to the homepage); a hero section (Section 7.7); paragraph-style content with optional asides; and a page footer containing repetition of some elements from the header, as well as things like links to About or Contact pages, privacy policy, etc. These commonalities are the result of years of trial and error, and by incorporating such familiar elements into our site, we help new visitors orient themselves and find what they're looking for.

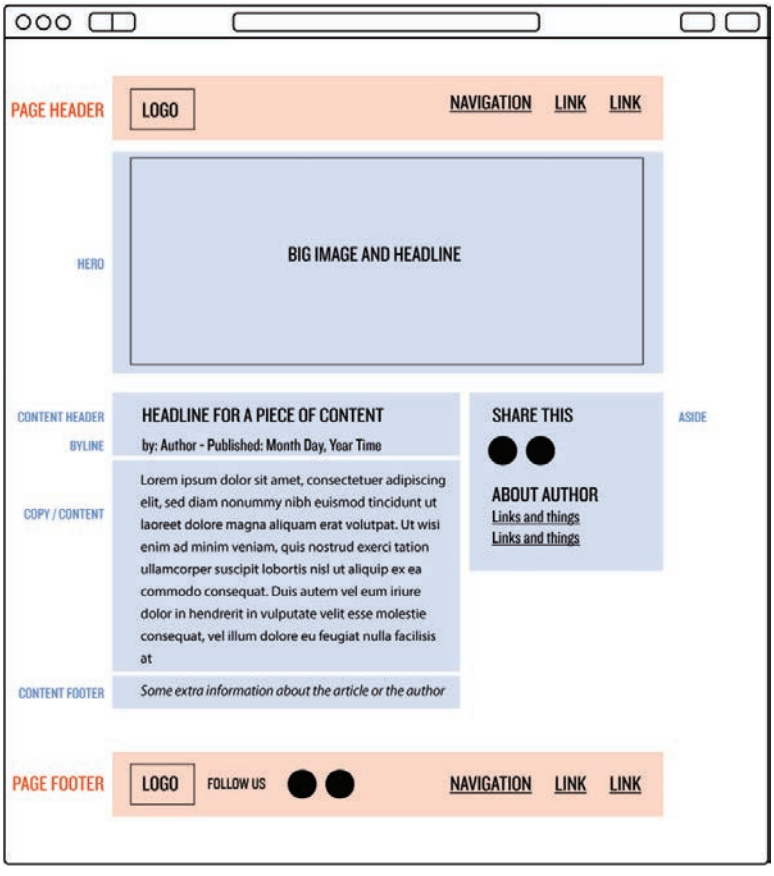

Figure 9.1: Elements of a typical web page.

One thing you may notice from Figure 9.1 is that many elements, such as the header and footer, are the same (or nearly the same) on every page of our site. If we made each page by hand, that would make our markup ridiculously repetitive—if we wanted to make a change, updating all those pages would be a nightmare.

This is an issue we faced repeatedly in Part I, where we simply copied and pasted common elements like navigation links onto every individual page. Such repetition is a violation of the DRY principle (Box 5.2), and in Box 3.2 we promised to teach you how to use a *templating system* to solve this problem. In this section, we'll fulfill this promise by installing and using the *Jekyll* static site generator to eliminate duplication in our layout.

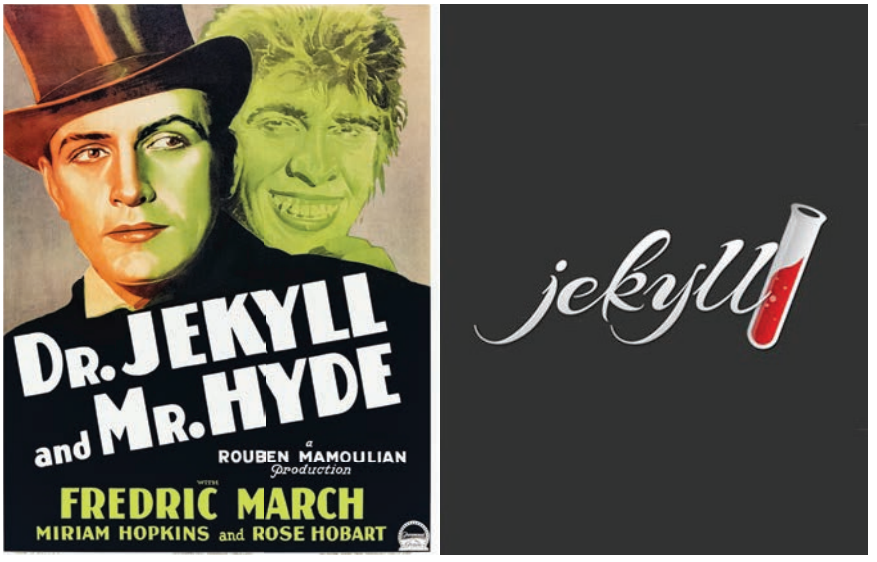

Figure 9.2: Not Jekyll and Hyde… rather, Jekyll the static site generator!

9.2 Jekyll

When building a professional-grade website, it's essential to use a system capable of supporting templates to eliminate duplication. To accomplish this, we'll be using *Jekyll* (https://jekyllrb.com/) (Figure 9.2[1]), a free and open-source program for generating static websites (that is, sites that don't change from visit to visit).[2]

By learning Jekyll, you'll cultivate the skills needed to develop and deploy a real website—skills that are transferable to other static site generators (such as Middleman and Hugo) and to full-blown web frameworks (like Ruby on Rails (https://www.railstutorial.org/)). Learning the template language used by Jekyll (called *Liquid*) is also a valuable skill in itself, as Liquid is widely used in systems like the Shopify ecommerce platform.[3]

1. Poster image courtesy of BFA/Alamy Stock Photo.

2. Making *dynamic* sites that allow user registration, login, input, etc. requires using a full web application framework. In future Learn Enough tutorials, we'll cover two such frameworks, Sinatra and Rails (in *Learn Enough Ruby to Be Dangerous* (https://www.learnenough.com/ruby) and the *Ruby on Rails Tutorial*, respectively).

3. Indeed, as noted in Section 9.3, Liquid was originally developed by Shopify cofounder Tobi Lütke for exactly this purpose.

In addition to supporting templates, Jekyll also includes a bunch of other useful features:

- Write content in Markdown (the lightweight markup format we first discussed in Chapter 6 of *Learn Enough Developer Tools to Be Dangerous*) in your text editor of choice.
- Write and preview your content on your site locally in your dev environment.
- Publish changes via Git (which also gives you an automatic off-site backup).
- Host your site for free on GitHub Pages.
- No database management.

Originally developed by GitHub cofounder Tom Preston-Werner, Jekyll is used by millions of people around the world and is an industrial-strength tool for creating static websites. For example, the fundraising platform for U.S. President Barack Obama's 2012 reelection campaign, which handled 81,548,259 pageviews and raised over $250 million, was built using Jekyll:

> *By using Jekyll, we managed to avoid the complexity that comes with most CMSes (databases, server configuration) and instead were able to focus on things like optimizing the UI and providing a better user experience. To work in this environment, the most a front-end engineer had to learn was the Liquid template language that Jekyll uses, and boy is that simple.*[4]

9.2.1 Installing and Running Jekyll

Jekyll is written in the Ruby programming language, and is distributed as a Ruby *gem*, or self-contained package of Ruby code. As a result, installing Jekyll is easy once you have a properly configured Ruby development environment.

If your system is not already configured as a dev environment, you should consult *Learn Enough Dev Environment to Be Dangerous* (https://www.learnenough.com/dev-environment) at this time. This step might prove challenging, especially if you decide to configure your native system, but in the long run the effort is well worth the reward.

4. Originally published at http://kylerush.net/blog/meet-the-Obama-campaigns-250-million-fundraising-platform/ (since removed). Quoted selection has been lightly annotated and copyedited.

Once you've got a working dev environment, you can install Jekyll using *Bundler*, a manager for Ruby gems. We can install Bundler using the **gem** command, which comes with Ruby:

```
$ gem install bundler -v 2.3.14
```

Next, we need to create a so-called **Gemfile** to specify the Jekyll gem:

```
$ touch Gemfile
```

Then use a text editor to fill the **Gemfile** with the contents shown in Listing 9.1.

Listing 9.1: Adding the Jekyll gem.
Gemfile

```
source 'https://rubygems.org'

gem 'jekyll', '4.2.2'
gem 'webrick', '1.7.0'
```

If you run into any trouble, check the **Gemfile** at https://github.com/mhartl/mhartl.github.io to see if it has been updated.

Finally, we can install the jekyll gem using **bundle install** (with a little extra code to ensure that we're using the right version of Bundler):

```
$ bundle _2.3.14_ install
```

Although Jekyll is designed to work with a system of templates (Section 9.3), in fact it can work with a single file, such as our current **index.html**. To see how it works, we can run the Jekyll server in our project directory (using **bundle exec** to ensure that the right version of Jekyll gets run):

```
$ bundle _2.3.14_ exec jekyll serve
```

If you're working on a native system or a virtual machine (as opposed to a cloud IDE), at this point the Jekyll app should be available at the URL http://localhost:4000, where localhost is the address of the local computer and 4000 is the *port number* (Box 9.1). The result should look something like Figure 9.3.

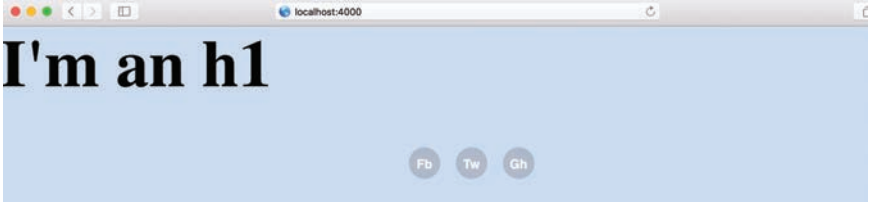

Figure 9.3: No more URL pointing to a file—you're running on a server now!

Box 9.1: Server Ports

If you look at the URL for the Jekyll site, you'll notice that it ends in ":4000". That is the *server port*. If you end a URL with a colon followed by a number, you are telling the browser to connect to that port on the server… so what does that mean?

You can think of server ports as being like individual phone numbers for different services that run on a computer. The default port number for the World Wide Web is port 80, so http://www.learnenough.com:80 is the same thing as http://www.learnenough.com, while the default port for a secure connection is 443, so https://learnenough.com:443 is the same thing as https://learnenough.com (with `https` in place of `http`). Other common port numbers include 21 (`ftp`), 22 (`ssh`), and 23 (`telnet`).

In the context of developing applications on a development machine, using port numbers allows us to solve the important problem of being able to run two or more apps simultaneously. Suppose, for example, that we wanted to run two different Jekyll websites on our development server. By default, both of them would be located at localhost:4000, but this would cause a conflict because the browser would have no way of knowing which site to serve when visiting that address. The solution is to add an extra piece of information, the port number, which allows the computer to distinguish between, say, app #1 running on localhost:4000 and app #2 running on localhost:4001.

As noted above, Jekyll's default server port is 4000, but we can set a different port number using the `--port` command-line option as follows:

```
$ bundle _2.3.14_ exec jekyll serve --port 4001
```

To connect to this second server, we would then type localhost:4001 into our browser's address bar.

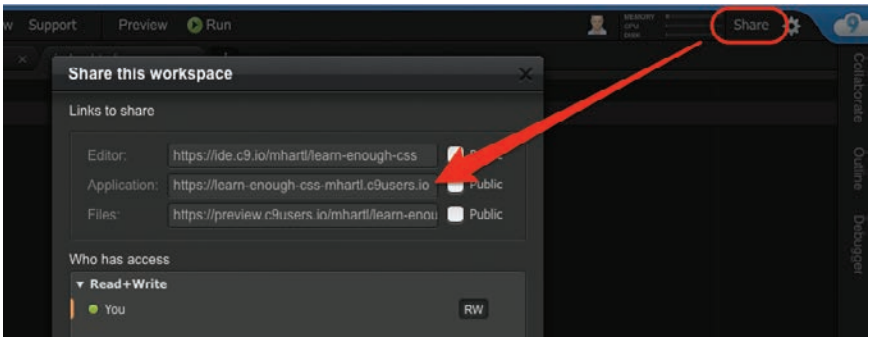

Figure 9.4: Sharing the URL on the cloud IDE.

If you're using the cloud IDE (https://www.learnenough.com/dev-environment-tutorial#sec-cloud_ide) suggested in *Learn Enough Dev Environment to Be Dangerous*, you'll have to pass options for the port number (Box 9.1) and host IP number when running the **`jekyll`** command:

```
$ bundle _2.3.14_ exec jekyll serve --port $PORT --host $IP
```

Here **`$PORT`** and **`$IP`** should be typed in literally; they are *environment variables* provided by the cloud IDE to make the development site accessible on an external URL. Once the server is running, you can visit it by selecting Share and then clicking on the server URL, as shown in Figure 9.4. The result, apart from the browser URL, should be the same as for the local system shown in Figure 9.3. (For simplicity, in what follows we sometimes refer to localhost:4000, but users of the cloud IDE should use their personal URL instead. *Mutatis mutandis*.)

After starting the Jekyll server, you should find a new folder in your project called **`_site`** (with a leading underscore):

```
$ ls
_site       index.html
```

This folder contains the output from the Jekyll server as it builds your site from the source files (currently just **`index.html`**).

The **`_site`** directory and all its contents are generated by Jekyll every time a file is saved, and if you were to make any changes in the **`_site`** folder, they will be automatically overwritten. As a result, you should never make changes in any of the

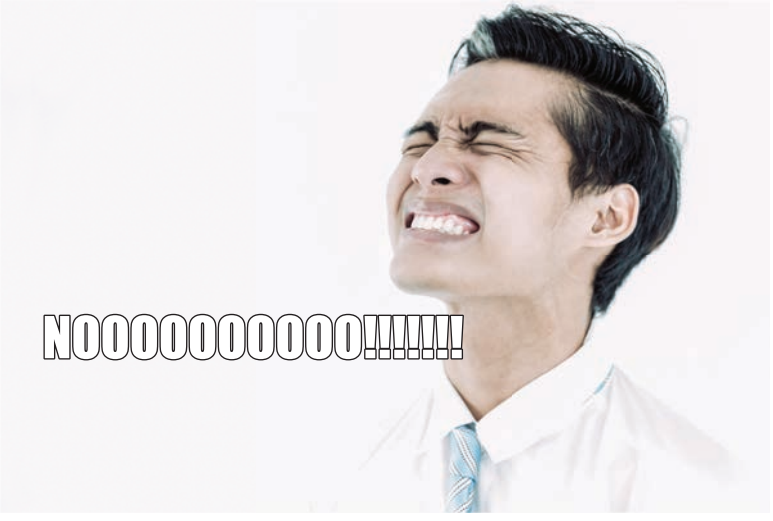

Figure 9.5: TFW changes accidentally made in generated files get overwritten.

`_site` files themselves—they would only be overwritten by Jekyll. There's nothing more frustrating than accidentally working on updates in an automatically generated folder, only to have your changes overwritten by an uncaring static site generator (Figure 9.5).[5]

Because all its content is generated by Jekyll, it's a good idea to ignore the `_site` directory by adding it to your `.gitignore` file, and there's a Bundler configuration directory called `.bundle` that should also be ignored:

```
$ echo _site/ >> .gitignore
$ echo .bundle >> .gitignore
$ git add .gitignore
$ git commit -m "Ignore the generated site and Bundler directories"
```

You should also add the `Gemfile` (and the associated auto-generated `Gemfile.lock` file) to the repository:

```
$ git add -A
$ git commit -m "Add a Gemfile"
```

5. Image courtesy of mangostar/123RF.

9.2.2 Exercises

1. Try starting Jekyll on a non-standard port like `1234`.

9.3 Layouts, Includes, and Pages (Oh My!)

One of the most powerful features of Jekyll is its ability to factor different parts of a website into reusable pieces. To accomplish this, Jekyll uses a system of folders and conventional names for files, along with a mini-language called *Liquid*. Originally developed by Tobi Lütke, cofounder of online store powerhouse Shopify,[6] Liquid is a system for adding content to a site using what are in effect simple computer programs.

Files inside a Jekyll project can be static, meaning that they do not get processed by the Jekyll engine, or they can be dynamic and get constructed with Jekyll magic. (The *site* is still static because it consists of static files on the server, even if those files are generated dynamically by Jekyll. In other words, the files don't change once they've been generated by Jekyll, so the results are the same for every visitor of the site.)

There are four main types of magic objects/files that the Jekyll engine can use in an automated way to build your site:

- Layouts/layout templates
- Includes
- Pages/page templates
- Posts

We'll discuss each of these in abstract terms for reference, but their exact uses won't become clear until we see some concrete examples starting in Section 9.4.

9.3.1 Layouts/Layout Templates

Anything in the special `_layouts` directory (which we'll create in Section 9.4) can have Jekyll magic, meaning those files get read by the engine looking for Liquid tags and other Jekyll formatting.

6. Tobi is also an alumnus of the Rails core team.

One of the key parts of many Jekyll pages is *frontmatter*, which is *metadata* at the top of an HTML file (in YAML format) that identifies the kind of layout to be used, a page-specific title, etc. A fairly complicated example might look like this, where the frontmatter is everything between the two triple-dashes `---`:

```
---
layout: post
title: This is the title of the post
postHero: images/shark.jpg
author: Me, Myself, and I
authorTwitter: https://twitter.com/mhartl
gravatar: https://gravatar.com/avatar/ffda7d145b83c4b118f982401f962ca6?s=150
postFooter: Additional information, and maybe a <a href="#">link or two</a>
---

<div>
  <p>Lorem ipsum dolor sit paragraph.</p>
<div>
```

In a simpler but still common example, the frontmatter identifies only the page layout template to be used when rendering the page:

```
---
layout: default
---

<div>
  <p>Lorem ipsum dolor sit paragraph.</p>
<div>
```

We'll see the effects of this sort of code starting in Section 9.4.

If there is no frontmatter in a layout file, then it is a true layout, and it needs to have a full HTML page structure. If there *is* frontmatter, then the file is a layout template that can be built into other layouts, and it doesn't need to have a full HTML page structure.

Layouts are often the most base-level objects, defining a standard page with a `DOCTYPE`, `html`/`head`/`body` tags, `meta` tags, stylesheet links, JavaScript, etc., and they usually pull in snippets like a site header or site footer. You often need only one default layout for a site, but you can also use layout templates for things like blogs (Section 12.3).

Layouts have the special ability to load content, like posts, using a generic Liquid tag that looks like this: `{{ content }}`. We'll see a short example of this in an exercise (Section 9.6.3), and we'll apply it to our full site in Chapter 10.

9.3.2 Includes

Files in the `_includes` folder can have Jekyll magic even though they don't need frontmatter, and these files are always intended to be built into something else. Includes tend to be little snippets of a site that get repeated on many pages, such as the header and footer (Figure 9.1) or a standard set of social media links. Includes will be covered in Section 9.6.

9.3.3 Pages/Page Templates

Any other HTML file in the project directory is a *page*. If there is no frontmatter in the file it is a *static page*, and Jekyll magic will not work (Liquid tags go unprocessed). If a page has frontmatter, though, it will need to specify a layout, and then all the Jekyll magic will be available. We'll cover pages more in Chapter 10.

9.3.4 Posts, and Post-Type Files

Posts are self-contained pieces of content, such as blog posts or product details, that are saved as files in the `_posts` directory. Some forms of content (like blog posts) are typically organized by date, while others (like product descriptions) are organized based on other attributes into *collections*. We'll discuss posts further in Chapter 12; collections are beyond the scope of this tutorial, but you can read about them in the Jekyll documentation on collections (https://jekyllrb.com/docs/collections/).

9.4 The Layout File

Let's start playing around with a Jekyll layout by adapting our site into the framework. The end result of this section will be a page that looks exactly like the current `index.html`, but which is created in a way that will give us greater power and flexibility down the road. This includes getting a first taste of templates and frontmatter (which we'll cover in greater depth in Chapter 10).

This isn't how you would normally go about creating a site if you were starting from scratch. Layout files are usually pretty bare-bones (as we'll see in Section 10.1), and a more common development process is to create a spartan layout using the command `jekyll new` and then start doing the real work in the pages and includes. In our case, though, we've already done a lot of work in our single `index.html` file; using it as our initial layout means that, as we learn about different aspects of Jekyll, we can pull the parts we need out of the layout, thereby showing how a whole site can be sliced up and reassembled.

As we explained in Section 9.3, the Jekyll convention for layouts is to place these files in a directory called `_layouts` (with a leading underscore), which you should create in the root directory of your application (`repos/<username>.github.io`):

```
$ mkdir _layouts
```

Any HTML file in the `_layouts` directory can serve as a layout, so to get started we'll copy the existing `index.html` into the layouts directory to create a default layout:

```
$ cp index.html _layouts/default.html
```

At this point, your project files should look something like Figure 9.6.

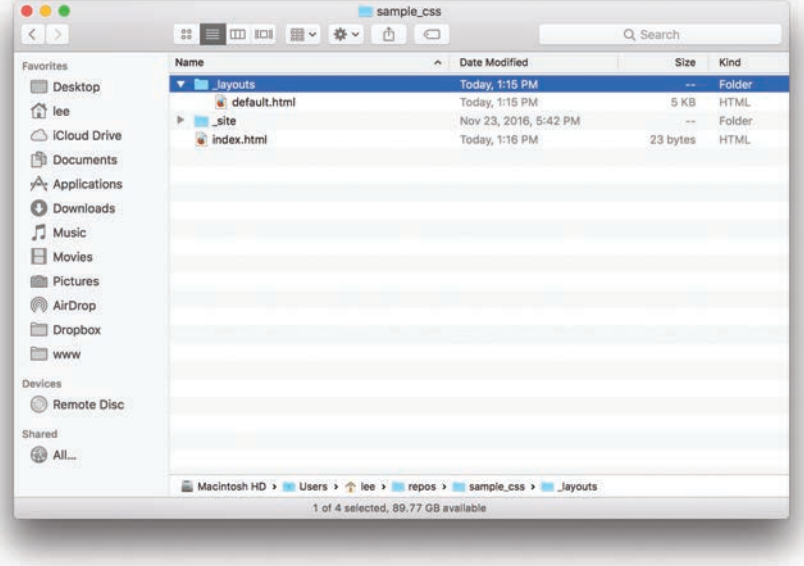

Figure 9.6: Your files and directories should look like this.

To get our site back up and visible, replace the entire contents of **`index.html`** with the code shown in Listing 9.2.

Listing 9.2: The site index with Jekyll frontmatter.
`index.html`

```
---
layout: default
---
```

As mentioned in Section 9.3, the content in Listing 9.2 is known as the Jekyll *frontmatter*, and by adding it to the **`index.html`** file we've turned a static page into a Jekyll *page template*.

The frontmatter is the secret sauce that lets Jekyll know that it needs to read through an HTML page to see if it should process any of the content. By specifying **`layout: default`**, we've arranged for Jekyll to use **`default.html`** as the page layout. Because **`default.html`** is currently a fully self-contained page, the result of visiting http://localhost:4000 is to render our entire test page (Figure 9.3). In other words, Jekyll just takes the contents of **`default.html`** and inserts it into **`index.html`**.

As mentioned in Section 5.4, this sort of transformation, where we change the underlying code without changing the result, is known as *refactoring*. It may seem like we've done nothing, but we'll see in Section 9.6 how this new structure lets us slice and dice our website into reusable pieces.

9.4.1 Exercises

1. To see the way frontmatter affects how pages are built, delete the frontmatter in **`index.html`**, and write "Hello world." Save the file and refresh the page.
2. Revert your changes from Exercise 1, and change the layout to one called **`test`**. Then create a new file in the **`_layouts`** directory called **`test.html`**, and add in some text like "Hello again, world."
3. In the root directory of your project, create a new file called **`tested.html`** and add in some text in it like "For the third time, hello world!" Now, in your browser go to http://localhost:4000/tested.html to see what happens.

9.5 CSS File and Reset

Now that we've refactored our test page into a layout (`default.html`) and a page template (`index.html`), we're going to start the process of breaking our monolithic HTML/CSS file into its component parts. The first step is to create a standalone CSS file with a *reset* that eliminates troublesome browser defaults for margins, padding, etc. (Listing 7.18). Then we'll pull all the CSS out of the test site's `style` block and put it into the same external file.

To get started, create a new folder in the project directory called `css`, and then create a new file in that directory called `main.css`, either using the terminal like in Listing 9.3, or by just adding the folders and files in your text editor.

Listing 9.3: Creating a new CSS folder and blank document in the terminal.

```
$ mkdir css
$ touch css/main.css
```

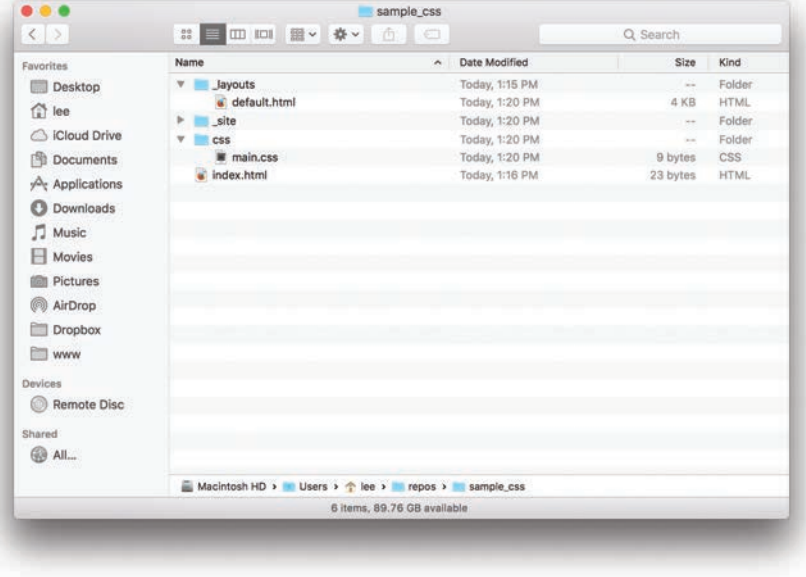

Figure 9.7: The new `css` folder and `main.css` file.

You have to name your directory exactly `css`, because Jekyll automatically looks for CSS files in that location, but you can use whatever filename makes you happy for the actual CSS file.

After you've created the folder and file as in Listing 9.3, your project directory should look something like Figure 9.7.

Recall from the discussions in Section 7.5 and Section 7.7 that browsers have built-in default styling for many common elements. Those browser defaults can differ from browser to browser, and if we were to allow them to remain it would mean that many elements on the page would start with styles we didn't pick. No self-respecting and properly perfectionist developer wants to leave the appearance of important elements up to the browser makers, so we'll apply a full *CSS reset* to create a blank slate for our designs.

Recall that we created a mini-version of a CSS reset in Listing 7.18, where we reset the margin and padding for `html` and `body` tags. Now it's time to upgrade our site to use an industrial-strength reset. The resulting CSS may look intimidating, but don't worry—we're putting it in Listing 9.4 precisely so that you can copy and paste it without having to understand the details.[7]

Listing 9.4: A standard CSS reset.
css/main.css

```
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: inherit;
  vertical-align: baseline;
}
/* HTML5 display-role reset for older browsers */
```

7. Recall that the code listings are available at https://github.com/learnenough/learn_enough_html_css_and_layout_code_listings.

```
article, aside, details, figcaption, figure,
footer, header, hgroup, menu, nav, section {
  display: block;
}
body {
  line-height: 1;
}
blockquote, q {
  quotes: none;
}
blockquote:before, blockquote:after,
q:before, q:after {
  content: '';
  content: none;
}
table {
  border-collapse: collapse;
  border-spacing: 0;
}
strong, b {
  font-weight: bold;
}
em, i {
  font-style: italic;
}
a img {
  border: none;
}
/* END RESET*/
```

Note that the CSS in Listing 9.4 doesn't need to be wrapped with the **`style`** tags the way the styles in the HTML file did; as we'll see in Listing 9.7, the browser understands from the link that everything inside the file is CSS.

We see in Listing 9.4 that most of the standard HTML elements get some sort of styling applied to them. The big block of selectors at the top is pretty much every HTML element in the spec forced to have margin and padding set to zero, a border of zero, and told to inherit font styles. This might seem a little extreme to target every element, but when we are making a custom website there is no reason to leave browser defaults for things like margin, padding, and border in place—otherwise, we could end up having to *undo* styling all over our stylesheet. It's better to undo a lot of stuff right off the bat, and then only add positive styling later on.

Also, don't think that the above reset styling is something set in stone (Figure 9.8[8]). If later in your development career you find yourself adding the same styling to every

8. Etching image courtesy of World Archive/Alamy Stock Photo; tablet graphic courtesy of Oleksiy Mark/Shutterstock.

Figure 9.8: Reset rules aren't set in stone… or any other kind of tablet.

(say) `table` tag on every site you design, it's probably best just to add that to your reset. As usual, the DRY principle applies (Box 5.2).

With the reset added, we're now in a position to move the custom CSS style developed so far in the tutorial into `main.css`. This involves first opening `default.html` and cutting all the CSS inside the `style` tag, leaving the tag empty (Listing 9.5).

Listing 9.5: The default layout with CSS cut out.
_layouts/default.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>Test Page: Don't Panic</title>
    <meta charset="utf-8">
    <style>
```

```
    </style>
  </head>
  <body>
  .
  .
  .
  </body>
</html>
```

Next, paste the CSS into **`main.css`** (possibly using something like Shift-Command-V, which pastes at the proper indentation level), and then delete the mini-reset targeting only **`html, body`** that we added before since it is now redundant. The full resulting code is shown in Listing 9.6.

Listing 9.6: The entire CSS file up to this point.
`css/main.css`

```
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: 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;
}
blockquote, q {
  quotes: none;
}
blockquote:before, blockquote:after,
```

```
q:before, q:after {
  content: '';
  content: none;
}
table {
  border-collapse: collapse;
  border-spacing: 0;
}
strong, b {
  font-weight: bold;
}
em, i {
  font-style: italic;
}
a img {
  border: none;
}
/* END RESET*/

/* GLOBAL STYLES */
h1 {
  font-size: 7vw;
  margin-top: 0;
}
a {
  color: #f00;
}

/* HERO STYLES */
.full-hero {
  background-color: #c7dbfc;
  height: 50vh;
}

/* SOCIAL STYLES */
.social-link {
  background: rgba(150, 150, 150, 0.5);
  border-radius: 99px;
  box-sizing: border-box;
  color: #fff;
  display: inline-block;
  font-family: helvetica, arial, sans;
  font-size: 1rem;
  font-weight: bold;
  height: 2.5em;
  line-height: 1;
  padding-top: 0.85em;
  text-align: center;
  text-decoration: none;
  vertical-align: middle;
```

```
  width: 2.5em;
}
.social-list {
  list-style: none;
  padding: 0;
  text-align: center;
}
.social-list > li {
  display: inline-block;
  margin: 0 0.5em;
}

/* BIO STYLES */
.bio-wrapper {
  font-size: 24px;
  margin: auto;
  max-width: 960px;
  overflow: hidden;
}
.bio-box {
  border: 1px solid black;
  box-sizing: border-box;
  float: left;
  font-size: 1rem;
  margin: 40px 1% 0;
  padding: 2%;
  width: 23%;
}
.bio-box h3 {
  color: #fff;
  font-size: 1.5em;
  margin: -40px 0 1em;
  text-align: center;
}
.bio-box img {
  width: 100%;
}
.bio-copy {
  font-size: 1em;
  line-height: 1.5;
}
.bio-copy a {
  color: green;
}
```

As you can verify by refreshing the browser, the page is now completely unstyled (Figure 9.9).

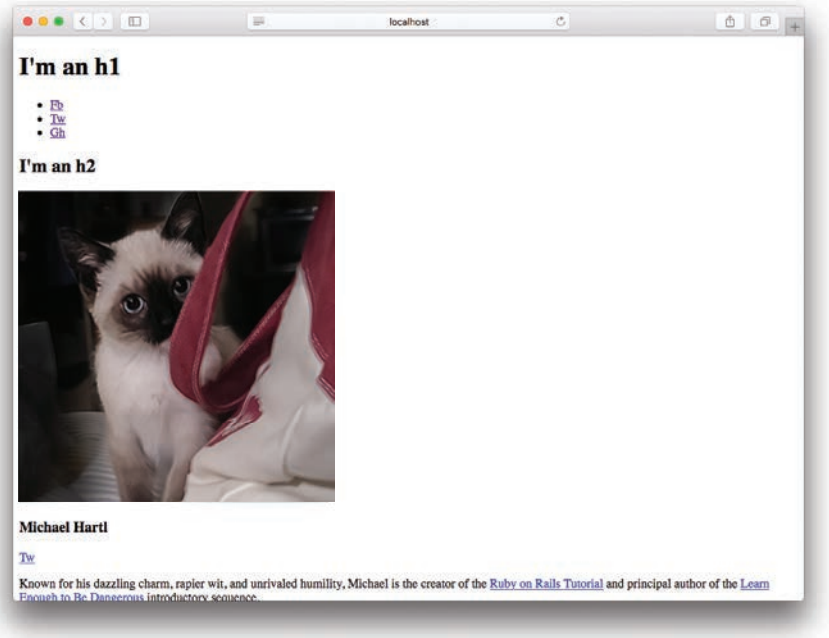

Figure 9.9: It's been a long time since our site was this naked and unstyled.

To restore the styling, all we need to do is tell the layout page about `main.css`. The way to do this is to replace the `style` tags in the `head` section with a link to our stylesheet, as shown in Listing 9.7.

Listing 9.7: Using a `link` tag to load `main.css`.
_layouts/default.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>Test Page: Don't panic</title>
    <meta charset="utf-8">
    <link rel="stylesheet" href="/css/main.css">
  </head>
  .
  .
  .
```

The `link` tag in Listing 9.7 tells the browser that it will be loading a stylesheet (`rel` is short for "relationship"), and then specifies a URL (in this case an absolute one that looks at the site's root directory by starting the URL with a forward slash)[9] that leads to the file.

It's important to understand that using the `link` tag to load an external stylesheet has nothing to do with Jekyll; this general technique works even on hand-built websites that don't use any site builder. The stylesheet doesn't actually need to be local, either—theoretically, it can be anywhere on the Internet—but for our purposes, we want to use a local file so that it's easy to make changes.

Now when you refresh the browser the styles should be properly applied, and the page will pretty much look how it did before our refactoring, although there will be some places where things don't look right because of the CSS reset (Figure 9.10).

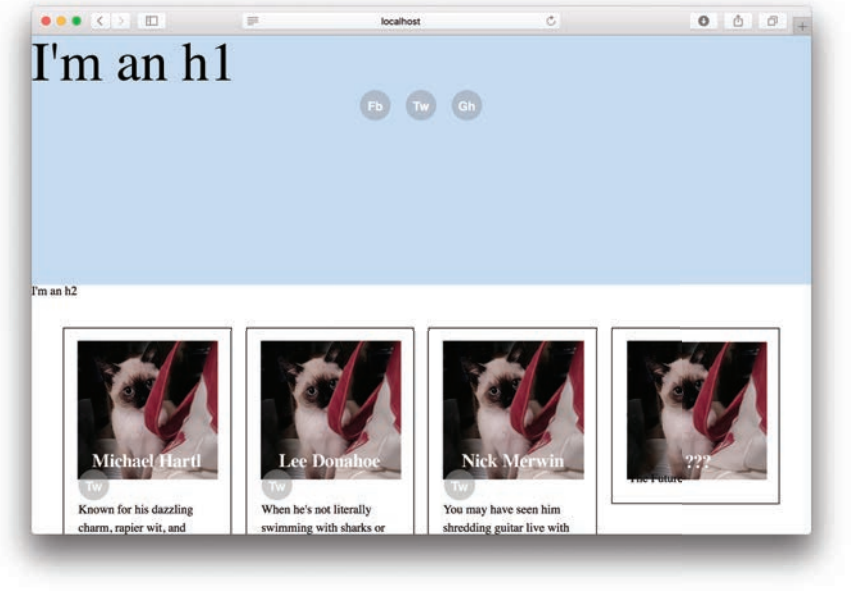

Figure 9.10: Same old page, with some minor oddities.

9. Recall from Section 2.4 that paths can be either relative (local to the computer serving the file) or absolute (accessed by a full URL). For example, the path `css/main.css` is relative, while `/css/main.css` is absolute.

Before moving on, let's make a few minor changes to prove that we know how to update styles via the CSS file. Ever since we started with this page, the fonts have looked a little… old-school. Let's add in a general style to the page **body** that will cascade down to every element on the page and change all body text to a nice, clean, sans-serif font (Listing 9.8).

Listing 9.8: A good spot for this would be in the "Global Styles" section of the CSS file. *css/main.css*

```
/* GLOBAL STYLES */
body {
  font-family: helvetica, arial, sans;
}
```

When you save your work and refresh the browser, everything should still look the way it did before, but with all-new fonts across the page (Figure 9.11).

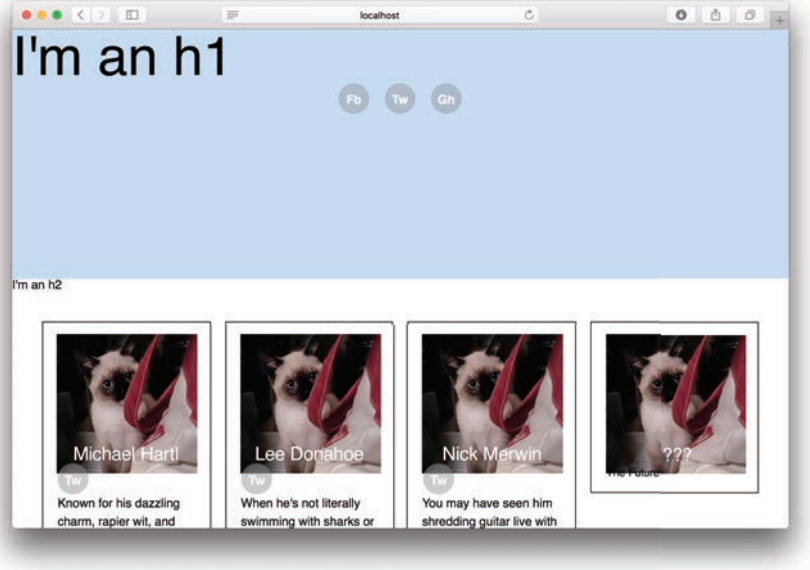

Figure 9.11: Same old page, all-new fonts.

Finally, in order to avoid the overlap between the bio box and social links, we'll change the CSS for the latter to be `display: block` with a margin, as shown in Listing 9.9.

Listing 9.9: Fixing up the social link spacing.
index.html

```
.bio-box img {
  width: 100%;
}
.bio-box .social-link {
  display: block;
  margin: 2em 0 1em;
}
.bio-copy {
  font-size: 1em;
}
```

The result appears in Figure 9.12.

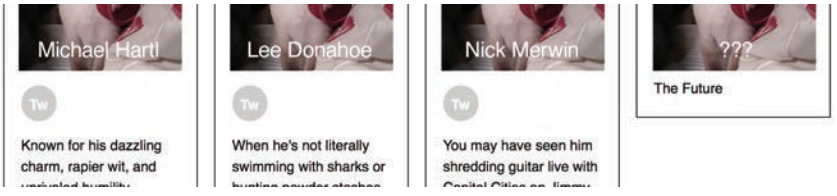

Figure 9.12: Better spacing for the social links.

9.5.1 Exercises

1. Create a second CSS file in the `css` folder, and add a second link to this new CSS file in the head of the document (making sure that this second link comes after the original CSS link). In your new CSS file, add a style that changes the `.full-hero` background color to a color of your choice. This shows that the order in which stylesheets load affects which styles take priority.
2. Rename the new CSS to `reset.css`, and move the stylesheet link above the link to `main.css`. Now cut and paste the entire reset section from `main.css` into the new CSS file (overriding the style added in Exercise 1). Save everything and make

sure that your test page looks the same in your browser. You've made your reset portable!

9.6 Includes Intro: Head and Header

Now that we've factored the CSS into a separate file (and added a CSS reset), it's time to start slicing up the default page into reusable pieces. As discussed in Section 9.3, Jekyll provides *includes* to help us with this important task. (*Note*: In this context, the word "include" is used as a *noun*, which is not standard English but is standard in the world of static site generators. This usage also changes the pronunciation; the verb form is "in-CLUDE", but the noun form is "IN-clude".)[10]

Includes are supposed to be the smallest/most reusable snippets of site code. They are usually loaded into either layouts or templates, but in fact can be used anywhere on a site—you can even have includes call other includes (Figure 9.13).[11] Since these snippets of code are intended to get dropped into the site almost anywhere, you should

Figure 9.13: You can put includes in includes, so your includes have includes.

10. This distinction exists in many other English words, such as AT-tri-bute (noun)/at-TRI-bute (verb) and CON-flict (noun)/con-FLICT (verb).

11. Image courtesy of vividpixels/123RF.

always try to make sure that any includes you create have code that is portable and self-contained.

Jekyll includes are located in a dedicated folder called `_includes` (as with `_layouts`, the underscore is important). Go ahead and create that folder now, together with a new file called `head.html` (Listing 9.10).

Listing 9.10: Creating the includes folder and adding in a new file.

```
$ mkdir _includes
$ touch _includes/head.html
```

At this point, your project folder should look something like Figure 9.14.

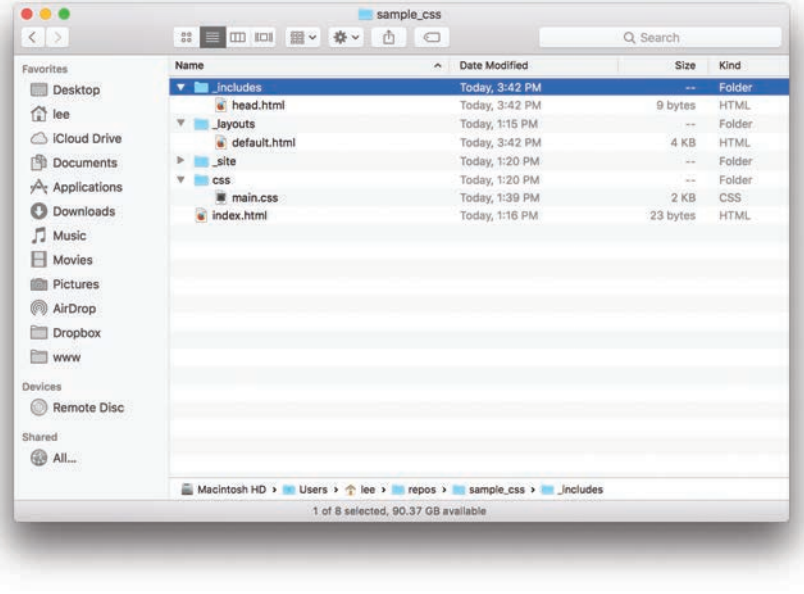

Figure 9.14: The project directory with added includes.

As you might have guessed, we're going to use `head.html` to hold the `head` tag and its contents. The way to do this is first to cut that content out of `default.html`, and then paste it into `head.html` (possibly using Shift-Command-V to paste with the proper indentation), as shown in Listing 9.11.

Listing 9.11: Moving `head` to its own file.
`_includes/head.html`

```
<head>
  <title>Test Page: Don't Panic</title>
  <meta charset="utf-8">
  <link rel="stylesheet" href="/css/main.css">
</head>
```

To include the contents of **`head.html`** back into the **`default.html`** layout, we'll use our first example of the Liquid language mentioned in Section 9.3, which looks like this:

```
{% include head.html %}
```

Here **`include`** is a Liquid command to include the file in question (in this case, **`head.html`**). The special syntax **`{% … %}`** tells Jekyll to replace the contents of that line with the result of evaluating the code inside. Because Jekyll automatically knows to look in the **`_includes`** directory, the result will be to insert the contents of **`head.html`**.

Replacing the original **`head`** section with the corresponding Liquid snippet gives the code shown in Listing 9.12.

Listing 9.12: Including the site head using Liquid.
`_layouts/default.html`

```
<!DOCTYPE html>
<html>
  {% include head.html %}
  <body>
```

After making these changes, you should refresh your browser to confirm that the page still works.

9.6.1 Page Header: Up Top!

At the top of a typical web page, you will usually find some sort of site-level *navigation* that takes users from page to page on the site, and also includes site branding.

This section is often referred to as the *site header* (Figure 9.15) (not to be confused with the **head** tag, which is the HTML header). Implementing such a header site-wide is a perfect application of Jekyll includes.

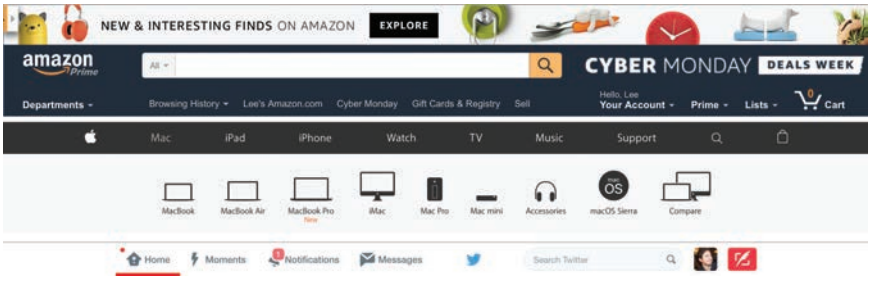

Figure 9.15: Some site headers from popular websites.

To get started, let's add a new Liquid tag to **header.html** (which we'll create in a moment) at the top of the **default.html** file, as shown in Listing 9.13.

Listing 9.13: Including the header HTML.
_layouts/default.html

```
<!DOCTYPE html>
<html>
  {% include head.html %}
  <body>
    {% include header.html %}
    <div class="full-hero hero-home">
      <h1>I'm an h1</h1>
      <ul class="social-list">
```

Next, create a new blank document in the **_includes** folder called **header.html**:[12]

```
$ touch _includes/header.html
```

The header itself will use two *semantic elements* (i.e., elements that have meaning): **header** to contain the header and **nav** for the navigation links, which (as with the

12. You can of course use your text editor to create the file rather than using **touch**.

social links in Section 8.5) are organized as an unordered list `ul`. We'll also use the classes `"header"` and `"header-nav"` to make it easier to apply styles across a range of browsers (Box 9.2). The resulting code appears in Listing 9.14.

Listing 9.14: The basic structure of our site header.
_includes/header.html

```
<header class="header">
  <nav>
    <ul class="header-nav">
      <li><a href="/">Home</a></li>
      <li><a href="#">Nav 1</a></li>
      <li><a href="#">Nav 2</a></li>
      <li><a href="#">Nav 3</a></li>
    </ul>
  </nav>
  <a href="/" class="header-logo">Logo</a>
</header>
```

Save and refresh your browser and now you'll see your new site header (Figure 9.16). (We'll explain the placement of the logo in Section 9.6.2.)

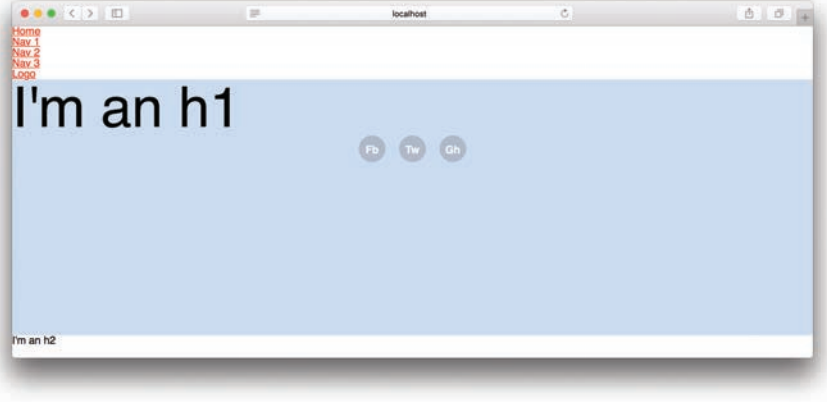

Figure 9.16: Our not-very-attractive header.

Box 9.2: Style Note: Style HTML5 Elements with Classes

To ensure maximum backward compatibility, it's not a good idea to target the newer HTML5 semantic elements like `header` and `nav` directly. There are inevitably going to be some users who visit your site on an old browser that doesn't support them—though luckily there are fewer such cases with each passing year.

When an old browser encounters new HTML tags, it sees them as regular `div`s, and any styles targeting those tags are ignored. To avoid this situation, it's better to give such elements classes, and then target your styles at the classes.

For example, we want to avoid styling `header` directly:

```
header {
  background: #000;
}
```

Instead, we'll give the `header` tag a class `"header"` (like in Listing 9.14), and then target that class (note the leading dot):

```
.header {
  background: #000;
}
```

This way, our styles will work even in older browsers.

9.6.2 Navigation and Children

Now, let's style that ugly header!

The end goal for our design is to create a traditional sort of header, with a logo on the left-hand side that will send users back to the homepage, and site navigation at the top right. As a final step, we'll change the position of the header so that it will sit on top of content below it.

The first thing that we are going to do is move the navigation to the right and put the **`li`**s into a horizontal row by changing their display property to **`inline-block`**. The result, which we suggest inserting immediately after the global styles, appears in Listing 9.15.

Listing 9.15: Adding header styles.
css/main.css

```
/* HEADER STYLES */
.header-nav {
  float: right;
}
.header-nav > li {
  display: inline-block;
}
```

Note in Listing 9.15 that we've used the more advanced child selector **>** to target the **li**s (as discussed before in Box 8.1). That is to make sure that if we wanted to put a second level of links into the menu, only the direct children would be **inline-block** (which we will in fact do in Section 13.4).

After saving and refreshing, you'll see that the menu has moved (Figure 9.17).

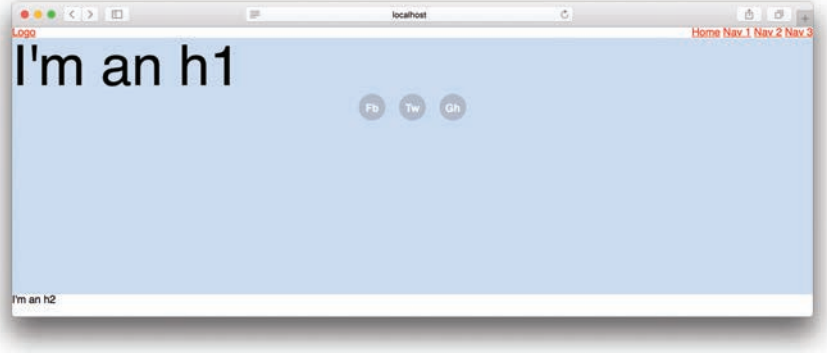

Figure 9.17: Navigation moved to the right and all in a line.

You might have wondered why the logo is *below* the navigational list in Listing 9.14 even though it comes first when viewing the header from left to right. The reason is that we knew all along that we were going to float the navigation to the right side of the screen, and if the logo appeared before the navigation in the HTML order then the menu would start at the *bottom* of the logo. This is because even a floating element respects the line height and position of normal block or inline elements that come before it, which in this case would lead to unwanted space around the logo. You can

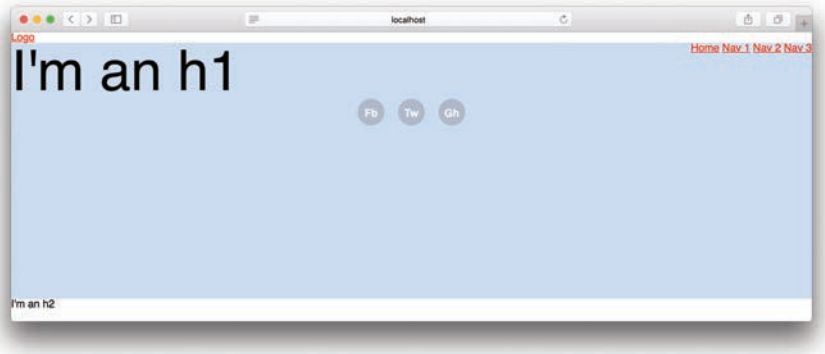

Figure 9.18: Switching the logo to come first adds unwanted space.

check this yourself by switching the positions of the logo and nav links; you'll see that the menu starts lower as a result (Figure 9.18).

Now let's add in some padding on the list items and make those links a little more stylish. We are going to add some padding to move the navigation away from the edges of the page:

```
padding: 5.5vh 60px 0 0;
```

We are also going to give each `li` in the navigation a bit of left margin so that it isn't bumping right up against its neighbor:

```
margin-left: 1em;
```

For the links themselves, we'll change the color and the size, make the font bold so that it is easier to read, get rid of the default link underlines (as is done in about 99% of site headers), and also automatically transform the text to be uppercase:

```
color: #000;
font-size: 0.8rem;
font-weight: bold;
text-decoration: none;
text-transform: uppercase;
```

Here we've used `#000` instead of `black`; as noted in Section 7.1.1, it's important to learn how to use these two interchangeably.

After adding the appropriate selectors, the styling changes look like Listing 9.16.

Listing 9.16: Styling the navigational links.
css/main.css

```
.header-nav {
  float: right;
  padding: 5.5vh 60px 0 0;
}
.header-nav > li {
  display: inline-block;
  margin-left: 1em;
}
.header-nav a {
  color: #000;
  font-size: 0.8rem;
  font-weight: bold;
  text-decoration: none;
  text-transform: uppercase;
}
```

Your page navigation should now look like Figure 9.19.

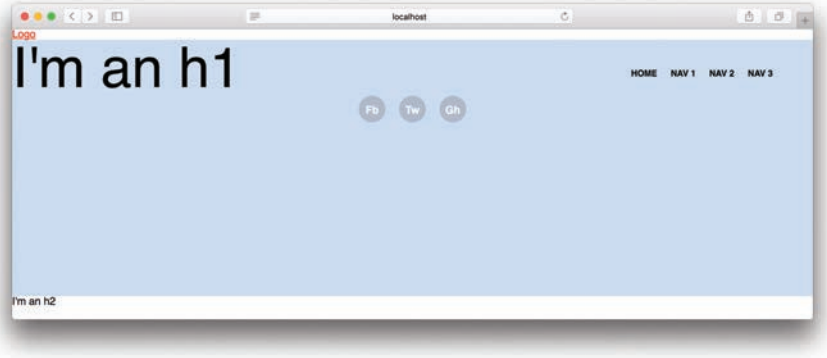

Figure 9.19: Navigational links are now a bit more stylish.

So how did we come up with those exact styles? The values came from just adding a couple of styling rules, and then tweaking the numbers until things looked good. Design isn't always a systematic process—often you just need to make changes and then play around with the numbers until you get something you like. When designing

websites, there tends to be an extended period of experimentation, so don't worry if it takes you time to get things right when you work on your own!

9.6.3 Exercise

1. You can load dynamic text into includes. To try this, add the extra code `{{ include.content }}` somewhere in your `header.html` include, and then in the layout change the include tag to `{% include header.html content="This is my sample note." %}`.

9.7 Advanced Selectors

In order to add an extra bit of polish to the site header, we are going to introduce a few more advanced CSS selectors, and then we'll continue to add in more styling for the rest of our page. These advanced selectors include *pseudo-classes*, *first-child/last-child*, and *siblings*.

9.7.1 Pseudo-Classes

It's always nice to have links do something when a user rolls over them, especially since we removed the underlines from the links in Listing 9.16. Those underlines on links are called *design affordances*, and they are there to give users the suggestion that something will happen if they move the cursor to the link and click.

Some people may argue that all links on a site should have some affordance that clearly marks them as something clickable, either with underlines or by making them look like buttons (HOLY WAR!!!). At this point in time, though, the design convention of putting plain-text links that don't have underlines (or some other special style) in a header is something that most Internet users are now accustomed to. You just know that the things at the top of the page are clickable.

Without underlines or other immediately visible affordances, though, it is important to show users a response to rolling over the link with their cursor (including on mobile (Box 9.3)). You really want people to know that they are interacting with an element that does something after they perform an action.

Box 9.3: Style Note: Mobile Hover Consideration

Mobile users don't see rollover states, so you always need to be sure that the things you are designing will make sense to both mobile and desktop users. One way to do this is to make sure that you also style things so that there is a change when the link is actively clicked.

You might think that this would be something that happens automatically no matter what, but if you make any styling changes that alter links from their browser default, you will actually need to use the `:active` pseudo-class to define how you want a link to appear when someone interacts with it.

If you do end up removing all hints that something is clickable for your site on desktop, you might want to consider using a mobile media query to add in some hints specifically for mobile users. We'll be discussing this further in the context of *media queries* in Chapter 13.

All HTML links have a set of what are called *pseudo-classes* that allow developers to style different interactions with the link:

- **`:hover`**: Styles what happens when a user rolls over the link (applies to any element, not just links)
- **`:active`**: Styles what happens when a user clicks the link
- **`:visited`**: Styles what the link should look like if a user has already visited the linked page

The way to add a pseudo-class to a style declaration is by combining the element or class name with the pseudo-class, like this:

```
.header-nav a:hover {
  color: #ed6e2f;
}
```

This use of the **`:hover`** pseudo-class arranges to change the color of the link when the user's mouse hovers over it. (For now we've just picked a random orange color that will stand out nicely against the blue background.)

We'll add a second change as well, which is to make the logo partially transparent on hover using the **`opacity`** property. The combined result appears in Listing 9.17.

Listing 9.17: Adding hover states to the navigational links.
css/main.css

```
.header-nav a:hover,
.header-nav a:active {
  color: #ed6e2f;
}
.header-logo:hover,
.header-logo:active {
  opacity: 0.5;
}
```

Note that we've added the same styling to the `:active` pseudo-class in order to give mobile users feedback as well (as discussed in Box 9.3).

Save your styles and refresh, and now the nav links will turn orange on rollover, and the logo will turn 50% transparent (the opacity style works like a decimal percentage), as shown in Figure 9.20.

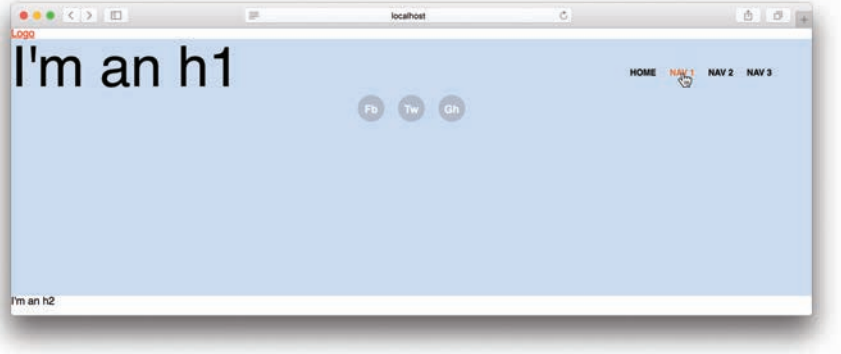

Figure 9.20: Muuuuch better.

There are a bunch of other very useful pseudo-classes that are regularly used in designing layouts. We'll talk about some of these throughout the rest of this section, and we'll see further examples in Section 13.5.

9.7.2 Exercises

1. Now that you've seen how to style rollovers, try styling the `.social-link`s to have rollover states where the background color changes.

2. As stated in this section, psuedo-classes like `:hover` don't just apply to links. Try adding a hover state that changes the background color of the `.full-hero` element.

9.7.3 `first-child`

In order to indicate that the Home link in the navigation menu is particularly important, let's arrange for it always to be a different color from the others. We could do this with a separate class, but since Home is always going to be the first link in the menu we can target it using what is called the `first-child` pseudo-class. This pseudo-class applies the corresponding styles only to the first child of the parent element. (There's also a `last-child` pseudo-class, which we'll use in Section 13.4, and many others that are beyond the scope of this tutorial.)

Let's make the Home link work the *opposite* of the styling for the other links, so that it's orange by default and black on rollover. To use the `first-child` pseudo-class, we need to make sure that whatever we're targeting is contained in a wrapper, and that there is nothing else in the wrapper. That just means that when you are using the child pseudo-classes, you need the elements to be inside some other HTML element.

If there is anything like text, or an HTML element of a different type, between the top of the parent and the element you are trying to target, the first and last child pseudo-classes won't work, but in this case we *are* going to target the first `li` in `.header-nav` (Listing 9.18). The `ul` with the class `.header-nav` is our wrapper, and the `li`s are all children that can be targeted.

Listing 9.18: Changing the appearance of just the first link.
css/main.css

```
.header-logo:hover,
.header-logo:active {
  opacity: 0.5;
}
.header-nav > li:first-child a {
  color: #ed6e2f;
}
.header-nav > li:first-child a:hover {
  color: #000;
}
```

Note how specific we are in Listing 9.18: We're using the child selector to target only `li`s that are direct children of the `.header-nav` class. You don't technically need this level of precision, but later on we will add in a dropdown menu in the header (Section 13.4), and if we target styles too generally then we'll make styling the dropdown difficult.

Now when you save and refresh the first link should look different (Figure 9.21).

Figure 9.21: Making the first nav link orange.

9.7.4 Exercise

1. We mentioned that there are other child selector types. Try out `:last-child` by changing the color of the link that is in the last `li` in the page header.

9.7.5 Siblings

Let's look at two additional advanced selectors, and then after seeing how they work, we'll use one to add another little style detail to our site navigation. CSS supports two sibling selectors, both of which are written like the child selector `>` when making a declaration:

- The *adjacent sibling* `+`: Selects a single element only if it is right next to the primary element in the declaration. For example, `h2 + p` selects a `p` tag only if it is immediately preceded by an `h2` tag.

- The *general sibling* `~`: Selects all elements of the type in the declaration if they follow the primary element. For example, `h2 ~ p` applies to *all* `p` tags preceded by an `h2` tag.

Let's hop out of working on the header for a second to create an example to use with the sibling selectors. In your `default.html` file, replace the `h2` tag with the HTML from Listing 9.19.

Listing 9.19: Replacing the `h2` and adding some text.
_layouts/default.html

```
<h2>THE FOUNDERS</h2>
<p>
  Learn Enough to Be Dangerous was founded in 2015 by Michael Hartl, Lee Donahoe,
  and Nick Merwin. We believe that the kind of technical sophistication taught by
  the Learn Enough tutorials can benefit at least a billion people, and probably
  more.
</p>
<p>Test paragraph</p>
```

We can target the paragraph that directly follows the `h2` with the style shown in Listing 9.20.

Listing 9.20: Adding an adjacent sibling selector.
css/main.css

```
h2 + p {
  font-size: 0.8em;
  font-style: italic;
  margin: 1em auto 0;
  max-width: 70%;
  text-align: center;
}
```

Notice that only the first paragraph is styled (Figure 9.22).

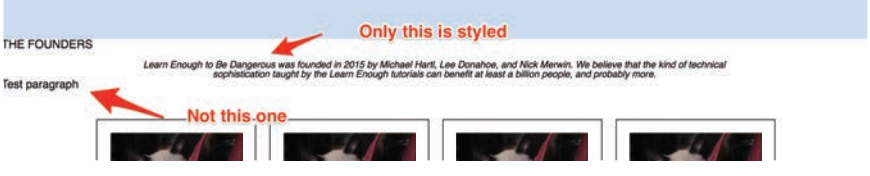

Figure 9.22: Only the `p` immediately after the `h2` is styled.

Now if we change to the general sibling selector `~` as in Listing 9.21, both paragraphs will get styled (Figure 9.23).

Listing 9.21: The general selector targets all elements that come after a specified element.

css/main.css

```
h2 ~ p {
  font-size: 0.8em;
  font-style: italic;
  margin: 1em auto 0;
  max-width: 70%;
  text-align: center;
}
```

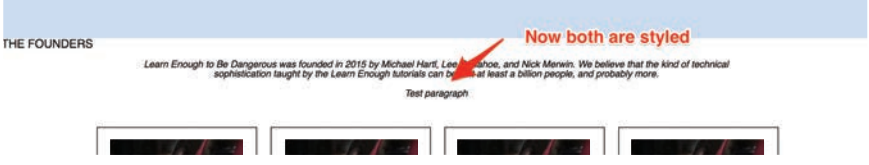

Figure 9.23: All `p` tags after the `h2` are now styled the same.

You may also have noticed from Figure 9.23 that the `p`s in the `.bio-box`es below aren't styled. That is because the sibling selectors don't pass styles to elements that are wrapped inside any other elements. They only work on elements inside the same parent.

Looking back to the header, we can use a sibling selector in the site header navigation to target all the `li`s after the first `li`, and add in a little extra styling to help visually separate the links using the styles in Listing 9.22. You might have seen something like this online: a little vertical line between navigational links to help separate them from other links in a list. Let's use a *sibling selector* to add some divider lines.

Listing 9.22: Using the general sibling selector to add styling to the header navigation.

css/main.css

```
.header-nav > li {
  display: inline-block;
  margin-left: 1em;
}
```

```
.header-nav > li ~ li {
  border-left: 1px solid rgba(0, 0, 0, 0.3);
  padding-left: 1em;
}
```

The rule `.header-nav > li ~ li` in Listing 9.22 says to apply the subsequent rules to all `li` elements next to an initial `li` inside an element with class `".header-nav"`—in other words, every `li` in the menu after the first one. This way, the divider lines appear before every menu item except the first (Figure 9.24).

Figure 9.24: Menu divider lines.

Now that the navigation is fairly spiffy, let's turn our attention to the logo, which will give us a chance to learn a little bit about CSS positioning.

9.7.6 Exercise

1. What if you didn't use the `~` in Listing 9.22, but rather the adjacent sibling selector? Would the divider line appear before every menu item?

9.8 Positioning

In this section, we are going to take a look at how positioning works in CSS, focusing on the site logo, and then we'll finish off the header design. CSS positioning can be a little tricky, and honestly there are people who work with CSS all the time who regularly get confused trying to get positioning to work right. So if this section seems

long and loaded with examples, just bear with us and work through it all—you'll find that understanding CSS positioning is an essential skill.

When you style an element's position, there are two basic possibilities:

1. Have the browser draw the element in its natural position in the normal flow of content on the page.
2. Remove the target from the flow of content and display it in a different place using directional styles—left, right, top, and bottom—and an additional dimension, the so-called `z-index`.

When an element is moved around out of its natural position with directional styles, it doesn't affect other elements in the document—it either covers them up or is hidden behind them. It becomes like a ship cast adrift, torn free from its mooring on the page.

While it might be self-explanatory to move something left or right, or to change its top or bottom position, you might not be familiar with the idea of a `z-index`. The `z-index` property (usually a nonnegative number, 0 by default—negatives put elements behind everything) determines whether an element is displayed above or below other elements, as in farther "into" the screen or farther "out" toward the viewer. It's an element's 3D position.

You can think of this like looking down at a big stack of papers—the higher the `z-index` number is, the higher up the stack toward you the element is. A `z-index` of `0` would be the bottommost piece of paper. We'll see a concrete example of the `z-index` in Section 9.9.

In order to change those directional styles, we first need to alter an element's `position` property. The `position` style in CSS can be given five different values (though one of them isn't really used). We'll start with one of the most common one, *static*.

- `position: static` (Figure 9.25)
  - This is the default positioning of elements in the flow of content.
  - An element that has no position style set, or has `position: static`, will ignore directional styles like left, right, top, and bottom.

Figure 9.25: How `position: static` affects elements.

- `position: absolute` (Figure 9.26)
  - Positions the element at a specific place by taking it out of the document flow, either within a parent wrapper that has a `position:` value other than `static`, or (if there is no parent) a specific place in the browser window. It is still a part of the page content, which means when you scroll the page, it moves with the content.
  - Also lets you define a `z-index` property.
  - Because the element is removed from the document flow, the width or height is determined either by shrinking to the content inside or by setting dimensions in CSS. It behaves kind of like an element set to `inline-block`.
  - Causes any float that is set on the object to be ignored, so if you have both styles on an element you might as well delete the float.

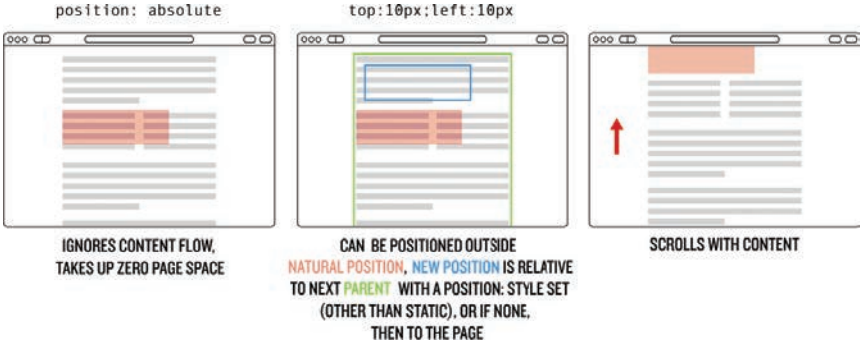

Figure 9.26: How `position: absolute` affects elements.

- `position: relative` (Figure 9.27)
  - This is like static in that it respects the element's starting position in the flow of content, but it also allows directional styles to be applied that nudge the element away from the boundary with other elements.
  - It allows absolutely positioned items to be contained within, as though the relatively positioned element were a separate canvas. In other words, if an absolutely positioned element is inside a relatively positioned element, a style of `top: 0` would cause the absolutely positioned element to be drawn at the top of the relatively positioned element rather than at the top of the page.
  - Also allows you to change the `z-index` of the element.

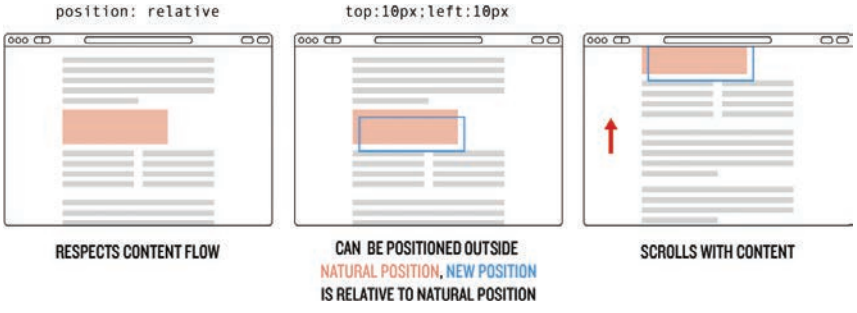

Figure 9.27: How `position: relative` affects elements.

- `position: fixed` (Figure 9.28)
  - Positions the element at a specific place within the browser window totally separate from the page content. When you scroll the page, it won't move.
  - Lets you set `z-index`.
  - Has the same need to have dimensions set as `position: absolute`; otherwise, it will be the size of the content inside.
  - Also causes floats to be ignored.

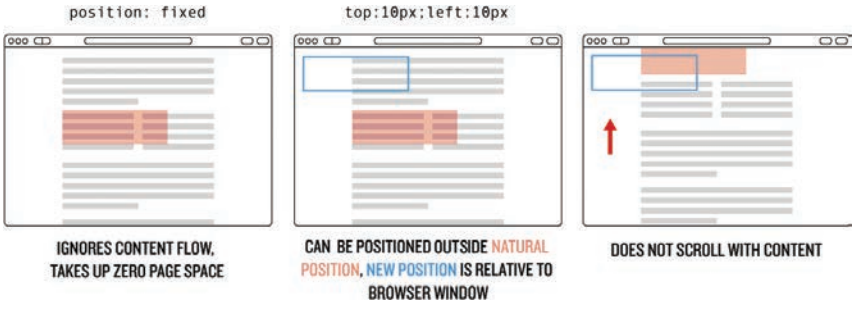

Figure 9.28: How `position: fixed` affects elements.

- `position: inherit`
  - This is not very common, so we aren't going to discuss it other than to say it makes the element inherit the position from its parent.

Let's play around with some examples. First, let's add in some styles for the header to better see the boundaries and to give it dimensions (Listing 9.23).

Listing 9.23: Added styles for the `.header` class.
css/main.css

```
/* HEADER STYLES */
.header {
  background-color: #aaa;
  height: 300px;
  width: 100%;
}
```

Let's now absolutely position the `.header-logo` and set it to `50px` from the bottom (Listing 9.24).

Listing 9.24: Adding an initial `position: absolute` to the logo.
css/main.css

```
.header-nav > li:first-child a:hover {
  color: #fff;
}
.header-logo {
  bottom: 50px;
  position: absolute;
}
```

Now save and refresh… where did the logo go (Figure 9.29)?

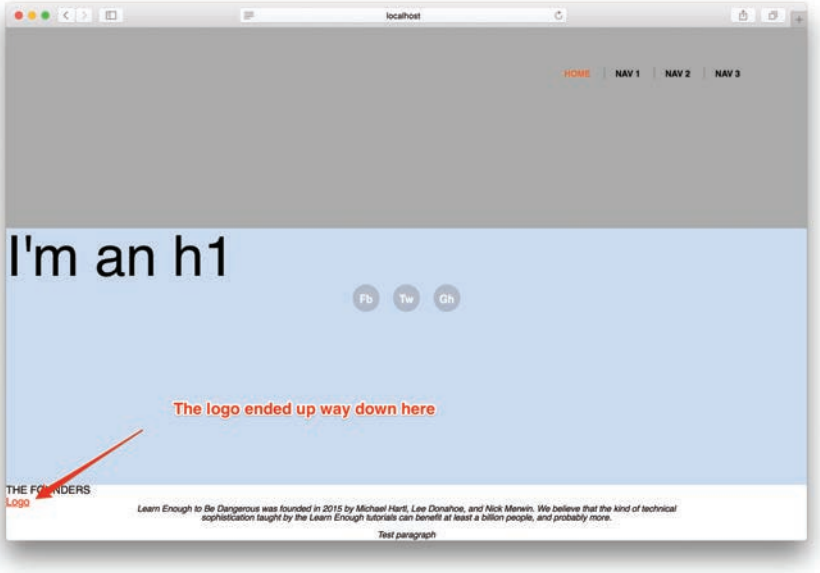

Figure 9.29: The parent container has no position style set.

The logo link ended up way at the bottom because the parent element that wraps the `.header-logo` doesn't have any `position` style applied. Also, if you scroll the page up and down you'll notice that the `.header-logo` still moves with the page. Let's constrain the logo to stay within the header by adding a position property, as shown in Listing 9.25.

Listing 9.25: Setting a position other than static on the wrapper.
css/main.css

```
.header {
  background-color: #aaa;
  height: 300px;
  position: relative;
  width: 100%;
}
```

With the `position` rule in Listing 9.25, the `.header-logo` will now be `50px` from the bottom of the gray header box, and any positions that we give to `.header-logo` will be determined based on the boundaries of the `.header` container (Figure 9.30). The way that the position is based off of the boundaries of the parent is what we meant when we said that setting a parent wrapper to `position: relative` made it like a separate canvas—everything inside that is absolutely positioned takes its place based on the dimensions of the parent.

Figure 9.30: The absolutely positioned `.header-logo`.

Note here that when an element is absolutely positioned, the directional styles don't add or subtract distance—setting `bottom: 50px` doesn't move it *toward* the bottom, but rather sets the position `50px` *from* the bottom. So `right: 50px` puts the element `50px` from the right edge.

Negative positions work as well, and as long as the overflow of the parent wrapper isn't set to `hidden`, the absolutely positioned element will get placed outside the boundaries of the parent (Listing 9.26).

Listing 9.26: Trying out negative positioning on our object.
css/main.css

```
.header-logo {
  bottom: -50px;
  position: absolute;
  right: 50px;
}
```

After adding that style and refreshing your browser, the logo should be in a position similar to what is shown in Figure 9.31.

Figure 9.31: Positioning the logo on the right-hand side.

You might be asking, "Well, what happens if I set both a top *and* bottom, or a left *and* right?" The answer is that, for whatever reasons, the top and left properties will take priority and the bottom and right will be ignored.

Another thing to consider is when you set a position property, you are manipulating elements and messing around with the natural page flow, which means that it is possible to cause misalignments. So if you add `left: 200px` to the `.header`, the

width of the element (which is 100%) isn't recalculated. Instead, the entire `.header` box is pushed over by 200px, and your browser window will have horizontal scrollbars and look broken (Figure 9.32).

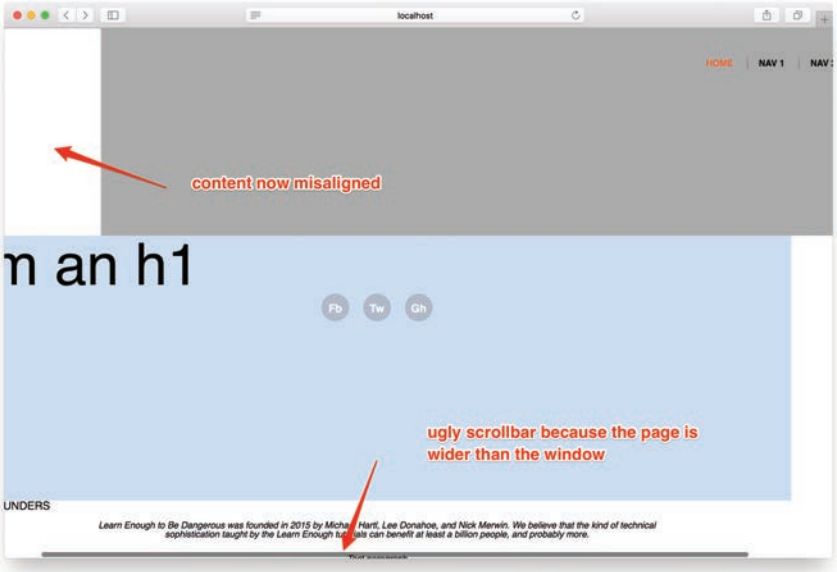

Figure 9.32: This sort of thing looks sloppy.

You have to be careful!

While we are still just playing around in the positioning sandbox, we should take a look at ways to deal with a situation that comes up anytime positioning in CSS is discussed: How do you center an absolutely positioned object horizontally and vertically in a way that allows the object to be any size… and allows the wrapper to be any size?

Let's first look at an old method where the object that we are centering has a set height and width—centering this is easy. Give the logo a width and height, remove the old positioning, and change the background to better see the object (Listing 9.27).

Listing 9.27: Adding height and width dimensions to the logo.
css/main.css

```
.header-logo {
  background-color: #000;
  height: 110px;
  position: absolute;
  width: 110px;
}
```

Now let's center it.

You might think that centering the element would be as simple as giving the `.header-logo` class a style of `left: 50%` and `top: 50%`—that should put it in the middle, both horizontally and vertically, right (Listing 9.28)?

Listing 9.28: Positioning the `.header-logo` in the center?
css/main.css

```
.header-logo {
  background-color: #000;
  height: 110px;
  left: 50%;
  position: absolute;
  top: 50%;
  width: 110px;
}
```

Well, no, the reason this didn't work is that when the browser positions an object, it calculates the distance using the same-named edge—so when you apply `top: 50%`, it moves the top edge (not the center point) of `.header-logo` 50% away from the top of `.header`; similarly, applying `left: 50%` tells the browser to move the left edge 50% away from the left of `.header`. The result is that the object we are trying to position is off-center by half of its width and height (Figure 9.33).

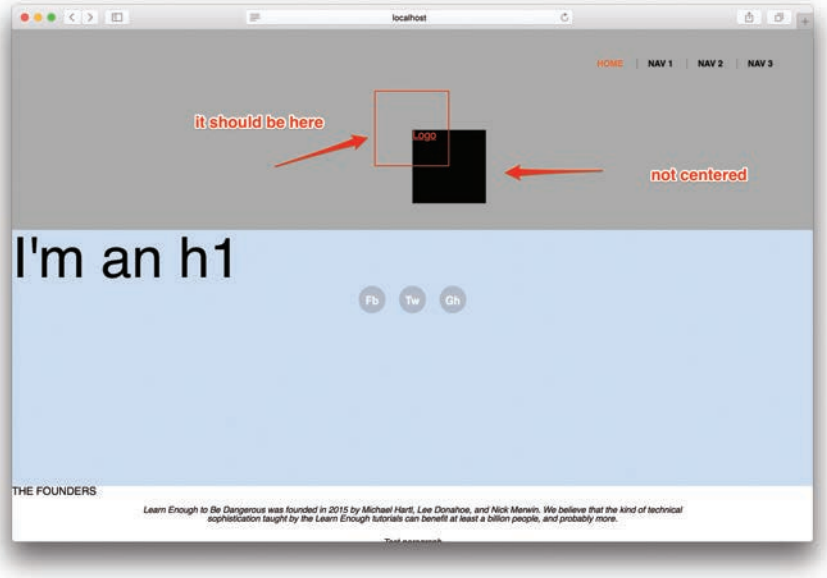

Figure 9.33: The red box in the expected position if centered vertically and horizontally.

How do we solve this and get our object in the actual center? The older method mentioned above was to use a negative margin (Section 8.6.2) to move the object up and left. This only works if you know the size of the object, though, since trying to use something like a percentage would move the object based on the size of the parent (recall from Section 7.4 that percentage values are based on the size of the parent object). Since the height and width of the box are `110px`, half of that is `55px` (Listing 9.29).

Listing 9.29: Adding in the negative margins to position the black box in the right spot.
css/main.css

```
.header-logo {
  background-color: #000;
  height: 110px;
  left: 50%;
  margin: -55px 0 0 -55px;
  position: absolute;
  top: 50%;
  width: 110px;
}
```

That works just fine, but you'd always be limiting yourself to centering only objects with fixed dimensions (Figure 9.34).

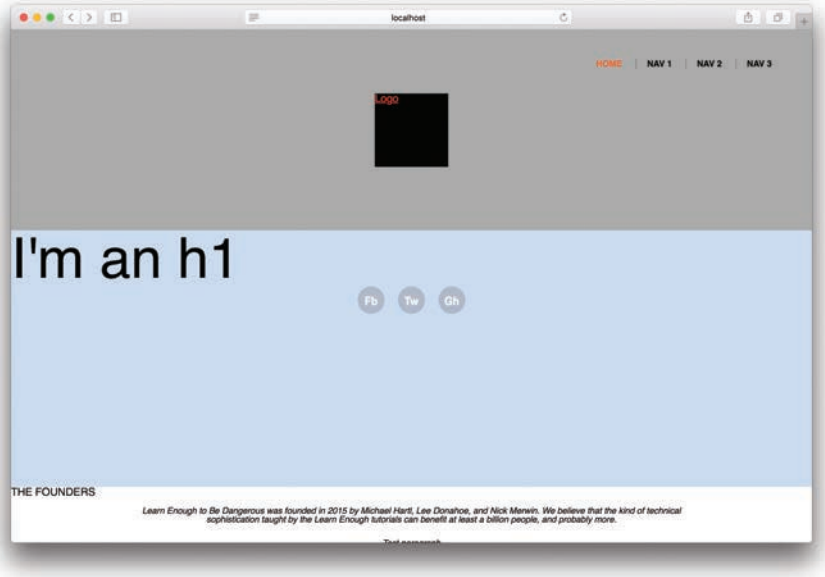

Figure 9.34: Negative margins worked!

If you wanted to make a slightly bigger (or smaller) centered object, you'd have to recalculate sizes and margins, and then make changes to your CSS. That's too much work, and it wouldn't work at all with dynamically sized elements. Thankfully there is a better, relatively new CSS style called `transform` that can help. The `transform` property allows developers to do all sorts of amazing things like move objects around, rotate them, and simulate three-dimensional movement.

The upside for centering objects is that this new style calculates all these movements based on the object itself. So if we move it 50% to the left using `transform`, the browser looks at the object's width, and then moves it to the left 50% of its own width, not the width of the parent.

The actual style declaration looks like this: `transform: translate(x, y)`—where `x` is replaced by the distance along the x-axis (left is negative, right is positive), and the same for the y-axis (up is negative, down is positive). So, to move our object left and up half its width and height, we'd add the `transform` style like you see in Listing 9.30 (make sure to remove the margin styling that we added in Listing 9.29).

Listing 9.30: Moving an object using `transform`.
css/main.css

```
.header-logo {
  background-color: #000;
  height: 110px;
  left: 50%;
  position: absolute;
  top: 50%;
  transform: translate(-50%, -50%);
  width: 110px;
}
```

Now when you save your work and refresh the browser you'll have a black box in the center of the gray header. It doesn't matter what dimensions you give for either the `.header-logo` or `.header`—you'll always have a vertically and horizontally centered object. To try it out, delete the height and width that we gave the `.header-logo`.

When you save and refresh your browser, the now-smaller box will still be centered vertically and horizontally (Figure 9.35).

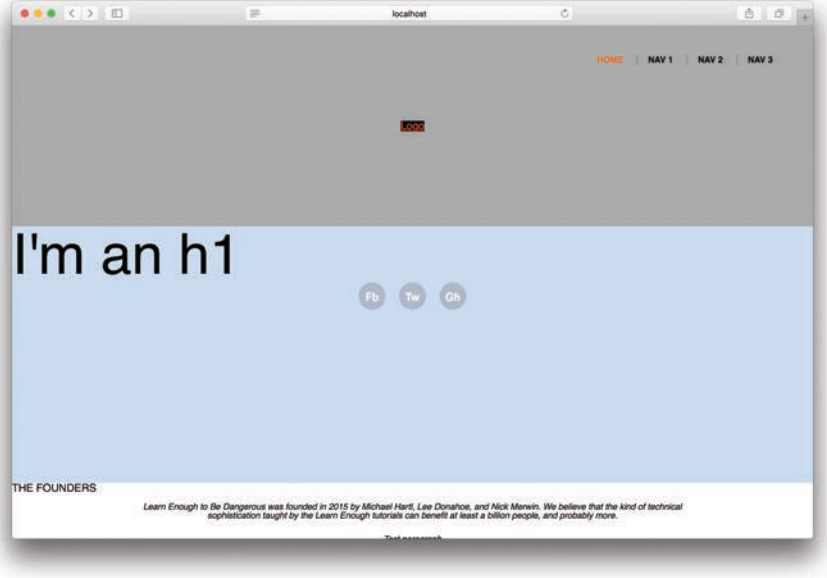

Figure 9.35: No matter what size the object is, it stays right in the center.

9.8.1 A Real Logo

All right, enough positioning playtime. Let's get back to making this site look good by putting an actual logo in that `.header-logo`. In your project directory, add a new folder called `images` (Figure 9.36):

```
$ mkdir images
```

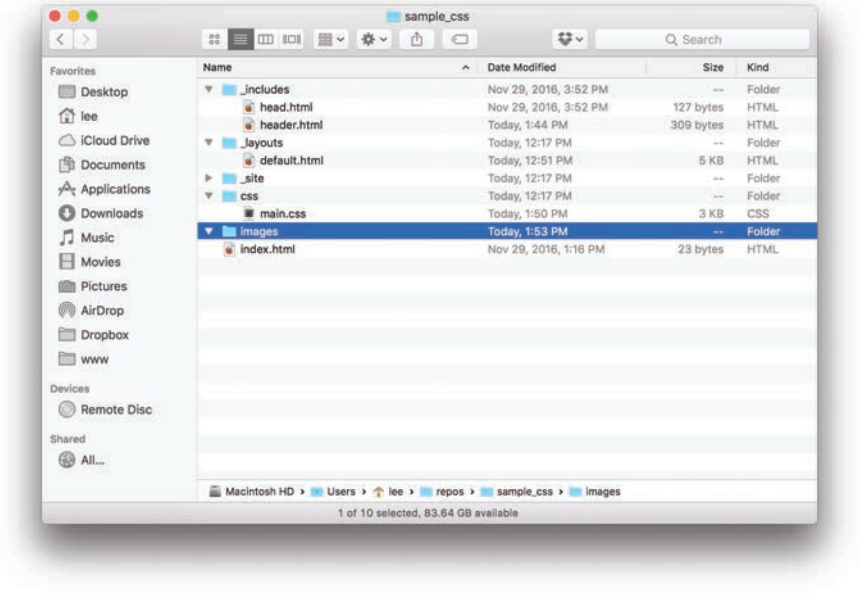

Figure 9.36: New images folder in your project directory.

Then use this `curl` command to grab the logo image off the Learn Enough servers:

```
$ curl -o images/logo.png -L https://cdn.learnenough.com/le-css/logo.png
```

Now let's put the image into the `header.html` (Listing 9.31). The result appears in Figure 9.37.

Listing 9.31: Replacing the word *logo* with a logo image.
_includes/header.html

```
<header class="header">
  <nav>
    <ul class="header-nav">
      <li><a href="/">Home</a></li>
      <li><a href="#">Nav 1</a></li>
      <li><a href="#">Nav 2</a></li>
      <li><a href="#">Nav 3</a></li>
    </ul>
  </nav>
  <a href="/" class="header-logo">
    <img src="/images/logo.png" alt="Learn Enough">
  </a>
</header>
```

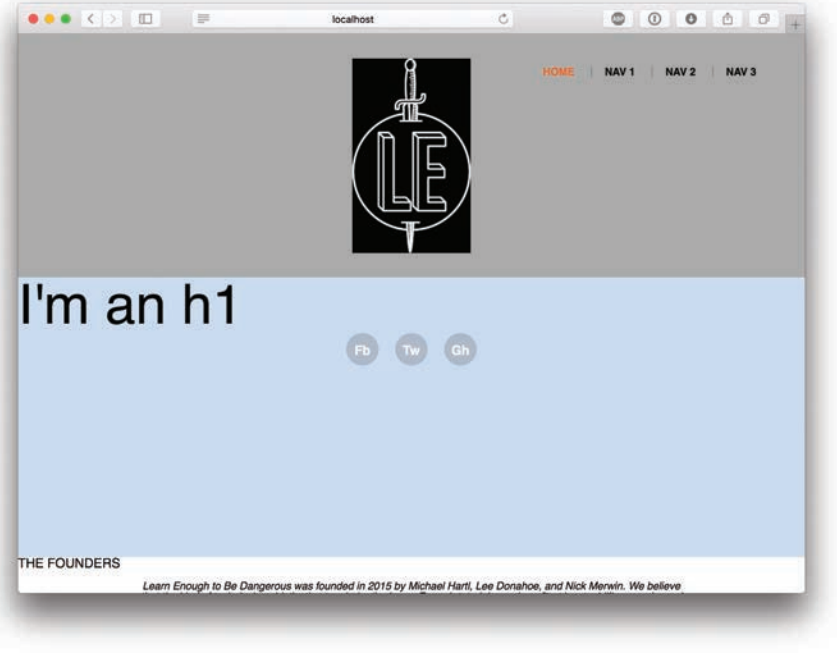

Figure 9.37: The initial (sub-optimal) logo placed on the page.

Now we are going to make a whole lot of changes to whip this part of the site into shape. As in Section 9.6.2, we aren't going to go through and give a reason why each value is the exact number we chose. Styling a section of a site is a non-linear process at times, and you'll likely need to experiment a lot if you are doing this on your own starting from a blank slate.

First, we are going to make the header background color black and any text in the header white as follows:

```
.header {
  background-color: #000;
  color: #fff;
}
```

That's also going to require that we change the color of the links, as well as the rollover color for the first-child link in the navigation:

```
.header-nav > li:first-child a:hover {
  color: #fff;
}
```

We'll also need to change the background color of our little divider lines so that it is partially transparent white instead of partially transparent black:

```
border-left: 1px solid rgba(255, 255, 255, 0.3);
```

Then we are going to move the `.header-logo` into the top left, and shrink the image a bit:

```
.header-logo {
  background-color: #000;
  box-sizing: border-box;
  display: block;
  height: 10vh;
  padding-top: 10px;
  position: relative;
  text-align: center;
  width: 10vh;
}
.header-logo img {
  width: 4.3vh;
}
```

We chose `10vh` for the size of the link, and for the image we set the width to be 4.3% of the height of the container (`4.3vh`). We got those values after playing around with different numbers and settling on this size for a balance of readability while not taking up too much space.

You'll notice that most of the sizing styles are on the link that wraps the image and not on the image itself. The reason we did that was so that if there is a problem downloading the image, or a delay, there is still a nice, big clickable link in the header.

Putting everything together gives us Listing 9.32, which includes all the styling for the site header so far.

Listing 9.32: Changing up the styling for the header and logo.
css/main.css

```
/* HEADER STYLES */
.header {
  background-color: #000;
  color: #fff;
}
.header-logo {
  background-color: #000;
  box-sizing: border-box;
  display: block;
  height: 10vh;
  padding-top: 10px;
  position: relative;
  text-align: center;
  width: 10vh;
}
.header-logo:hover,
.header-logo:active {
  background-color: #ed6e2f;
}
.header-logo img {
  width: 4.3vh;
}
.header-nav {
  float: right;
  padding: 5.5vh 60px 0 0;
}
.header-nav > li {
  display: inline-block;
  margin-left: 1em;
}
.header-nav > li ~ li {
  border-left: 1px solid rgba(255, 255, 255, 0.3);
  padding-left: 1em;
}
```

```
.header-nav a {
  color: #fff;
  font-size: 0.8rem;
  font-weight: bold;
  text-decoration: none;
  text-transform: uppercase;
}
.header-nav a:hover,
.header-nav a:active {
  color: #ed6e2f;
}
.header-nav > li:first-child a {
  color: #ed6e2f;
}
.header-nav > li:first-child a:hover {
  color: #fff;
}
```

Save and refresh, and your header should look like Figure 9.38. That logo's lookin' sharp!

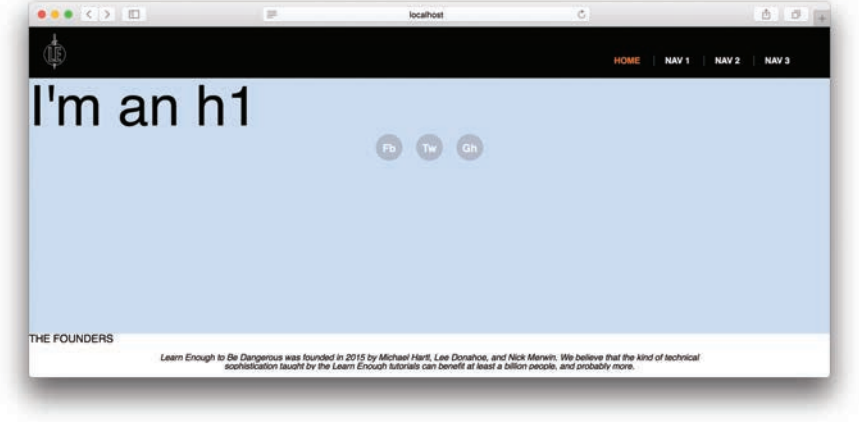

Figure 9.38: The header, now styled.

9.8.2 Exercise

1. Try moving the `ul` that contains the social links to the bottom-left corner of the `.full-hero` using the positioning rules you've learned. What changes are you going to need to make to `.full-hero` to allow the social links to remain inside?

2. To see why we gave dimensional styling and an `alt` tag to our image, try removing the image source link to simulate the browser not finding the file.

9.9 Fixed Header

You may have noticed the recent design trend where the header sticks to the top of the screen as you scroll down the page. This is called a *fixed header*—the header is styled to use `position: fixed` to take the header entirely out of the page content and stick it to the top of the user's browser. If your site has a bunch of different sections that your users need to navigate to, a fixed header can be a good solution to keep them from getting annoyed that they always have to scroll to the top to do something new.

The way to implement a fixed header is to change the positioning of the header to `fixed` while specifying a `z-index` for the header. Recall from the beginning of Section 9.8 that the `z-index` determines whether an element is drawn in front of or behind other elements. We'll want to give our header a large value for `z-index`, which will force the browser to draw the element above other elements (i.e., closer to the user using our stack-of-paper analogy).

The styles to change the positioning value and set a `z-index` are shown in Listing 9.33.

Listing 9.33: Fixing the header's position means that content will now scroll under it.
css/main.css

```
.header {
  background-color: #000;
  color: #fff;
  position: fixed;
  width: 100%;
  z-index: 20;
}
```

When you check the work in your browser, you'll find that the header is now pinned to the top of the screen, and when you scroll, all the content will scroll underneath.

The resulting black bar at the top looks cool, but what if we were to put a border around the entire page? It could look interesting to have a dark area around the whole site to frame the content. We can arrange for this with the styling shown in Listing 9.34.

Listing 9.34: Just for fun, let's put a border around the entire site.
css/main.css

```
/* GLOBAL STYLES */
html {
  box-shadow: 0 0 0 30px #000 inset;
  padding: 0 30px;
}
```

Listing 9.34 introduces the **box-shadow** style, which is a relatively new CSS style that lets you add drop shadows to HTML elements, and the declaration that we added is a shorthand for **box-shadow: x-axis y-axis blur size color inset**. We aren't going to go any deeper into it, but if you want to play around with box shadows there are a number of sites that let you fiddle with the settings, such as CSSmatic box shadow (https://www.cssmatic.com/box-shadow).

After applying the code in Listing 9.34, your page should look like Figure 9.39.

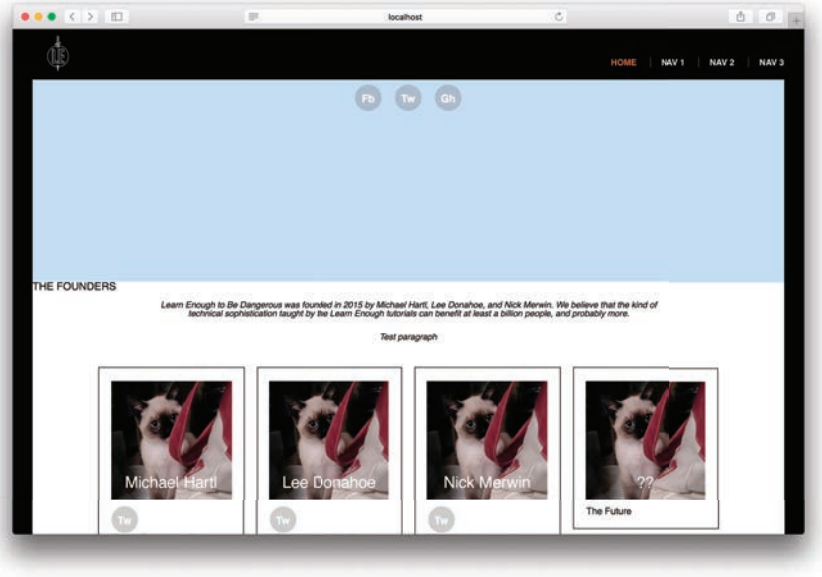

Figure 9.39: Box shadow inset around the entire page. Nifty.

After saving and refreshing, you might have noticed that the logo in the header now looks a little off since it isn't right up in the corner anymore. This is because we increased the padding on the entire site by `30px` for the black border. Let's use a negative value (`-30px`) on the positioning to get it back in place, as shown in Listing 9.35.

Listing 9.35: Using a negative value to move the logo back into place.
css/main.css

```
.header-logo {
  background-color: #000;
  box-sizing: border-box;
  display: block;
  height: 10vh;
  left: -30px;
  padding-top: 10px;
  position: relative;
  text-align: center;
  width: 10vh;
}
```

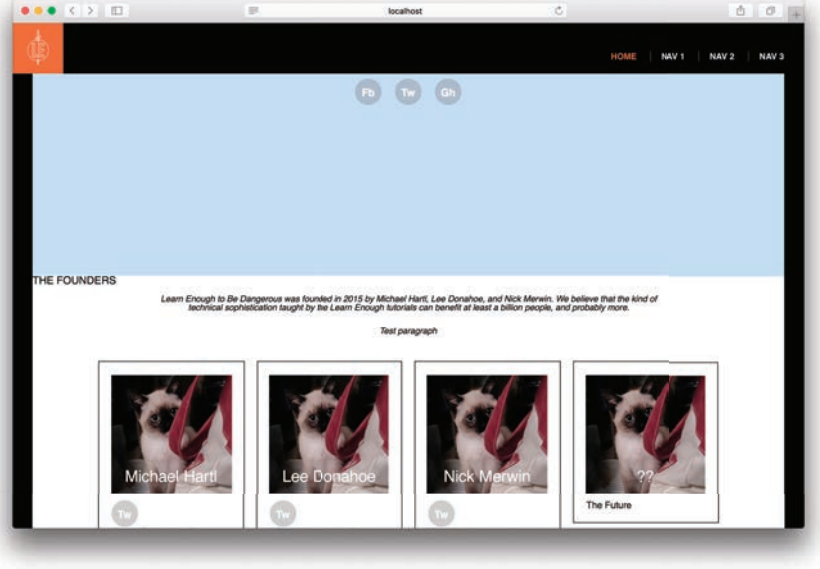

Figure 9.40: A completed page header.

The fixed final header should now look like Figure 9.40 (shown as it should appear with the mouse cursor on the logo, making it orange).

One thing you might have noticed is that after adding fixed positioning to the header, the big `h1` text in the hero is covered. We'll tackle this issue in Section 10.2.

Now that we've got the header squared away, let's turn our attention to the other end of the site.

9.9.1 Exercise

1. To see why it is important to define the `z-index` of the header, try setting the value to `1`, and then add styles to the `.social-list` class to set `position: relative` and `z-index: 40`. Then scroll the page.

9.10 A Footer, and Includes in Includes

After creating and styling a site header, a natural next step is to style the page footer. This is the navigational/informational section that can be found at the bottom of a site (Figure 9.41).

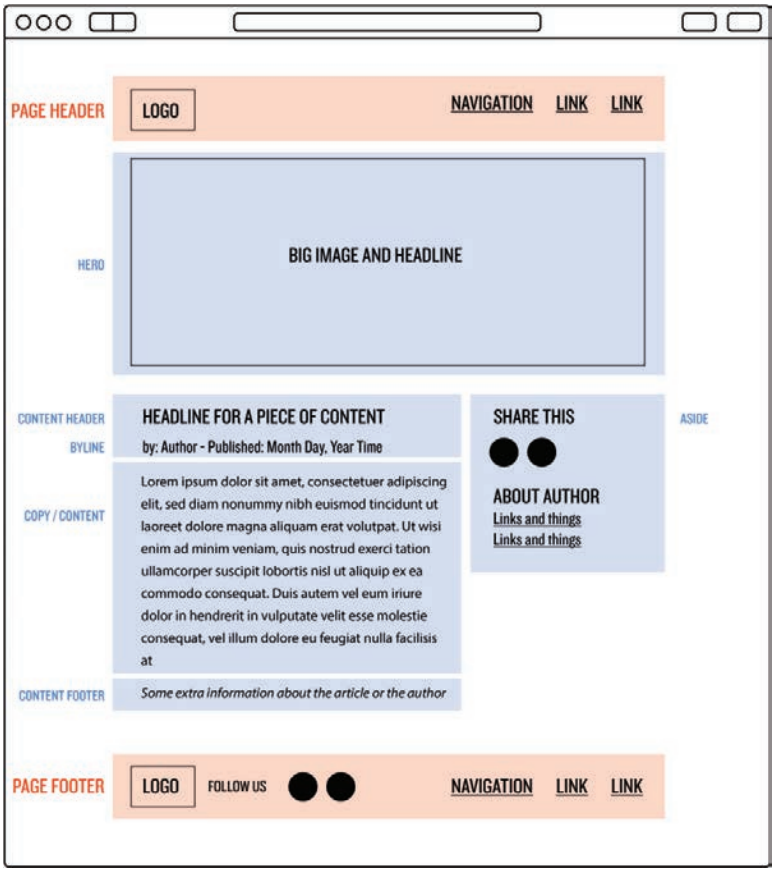

Figure 9.41: A refresher on the elements of a typical web page, including a page footer.

Often, the footer is a partial replication of the navigational elements from the header (just styled in a slightly different way), but many sites add to that a bunch of other content—everything from store locations and hours to additional content links.

Since the footer is found at the end of the page and contains ancillary information, you don't really need to worry about space (there's plenty of room at the bottom!). What we mean by that is that you can think of the footer as extra space, where users aren't *required* to see everything there. Many sites, such as Amazon, have a lot of content in a giant footer at the bottom of the page (Figure 9.42).

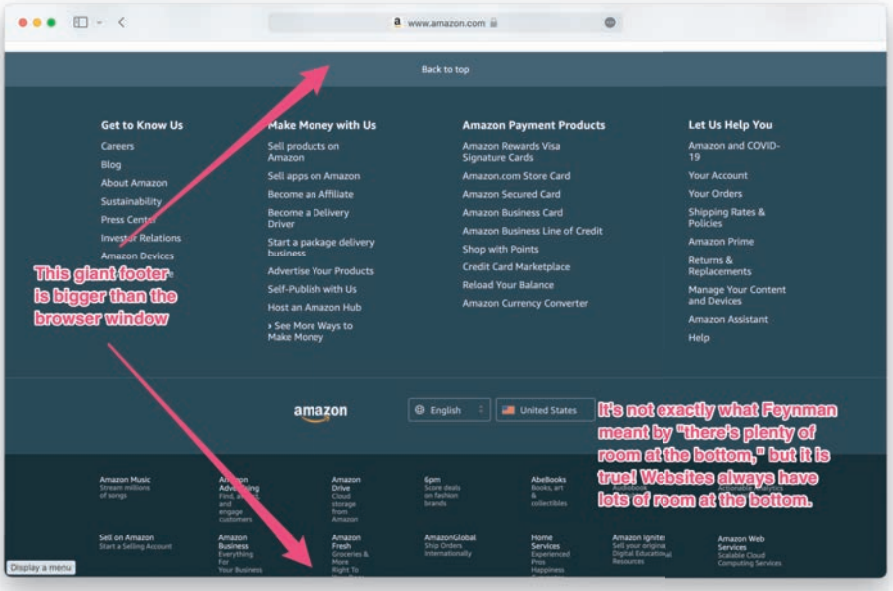

Figure 9.42: A giant footer.

We'll start by creating a new `footer.html` file inside the `_includes` folder:

```
$ touch _includes/footer.html
```

Next, we'll add some HTML. We're going to wrap the footer in another HTML5 semantic tag, the `footer` tag. As with the `header` tag, this is a semantic element that works just like a standard `div`, but gives automated site readers (such as web spiders and screen readers for the visually impaired) a better idea of what the purpose is of the content inside. We are also going to add in a logo link similar to the one in the header. The result appears in Listing 9.36.

Listing 9.36: Adding in the basic footer structure.
_includes/footer.html

```
<footer class="footer">
  <a class="footer-logo" href="/">
    <img src="/images/logo.png" alt="Learn Enough"/>
  </a>
```

```
  <h3>Learn Enough <span>to Be Dangerous</span></h3>
</footer>
```

To include the footer in the default layout, we'll follow the model from Listing 9.12 and use Liquid to insert the contents of `footer.html` just before the closing `body` tag in `default.html` (Listing 9.37).

Listing 9.37: Add in the Liquid tag to the default layout.
_layouts/default.html

```
    .
    .
    .
    </p>
    {% include footer.html %}
  </body>
</html>
```

Now let's add some styling as well. We'll give the footer a black background, like the header, and we'll give it some padding. We'll make sure that the content inside is easy to read by using `vh` units, which causes our padding to take up a large portion of the screen:

```
background-color: #000;
padding: 10vh 0 15vh;
```

We'll also constrain the size of the logo so that it isn't a giant image, and style the `h3` and the span that is inside it (just to add a little design detail to give some of the text a different color). All together the footer styling looks like Listing 9.38.

Listing 9.38: The initial styles for the footer.
css/main.css

```
/* FOOTER STYLES */
.footer {
  background-color: #000;
  padding: 10vh 0 15vh;
  text-align: center;
}
.footer-logo img {
```

```
  width: 50px;
}
.footer h3 {
  color: #fff;
  padding-top: 1.5em;
  text-transform: uppercase;
}
.footer h3 span {
  color: #aaa;
}

/* HERO STYLES */
```

Save and refresh, and the result should appear as in Figure 9.43.

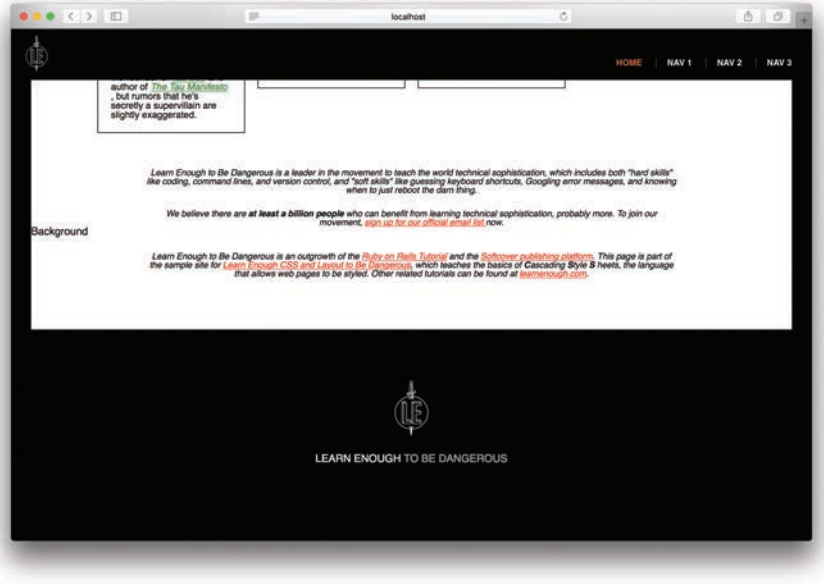

Figure 9.43: The first stab at the footer is looking pretty good.

And it looks... not too bad!

But let's make it a little more useful and also add in the navigational links from the header. You could just copy and paste the HTML from the header, but if you added a new page you'd have to edit your navigation in two spots… we hope the mere

suggestion of that is making your programmer's itch flare up again. Since those nav links are always going to be the same in both the header and the footer, we can create a new include to include in includes (thereby fulfilling the promise from Figure 9.13—it wasn't (just) a joke!).

We don't want to take the outer `ul` from Listing 9.14 since it has a `header-nav` class applied to it (well, you *could* add that in the include, then unstyle all the header styles, and then restyle to fit the footer—but that would be a lot of unnecessary work). So the content of our new include will just be the `li`s and the links—in other words, the content that definitely needs to be repeated.

To eliminate repetition in the links, let's create a new file in the `_includes` directory and name it `nav-links.html`:

```
$ touch _includes/nav-links.html
```

Then cut the `li`s and links out of the `.header-nav` and paste them into the new include, as shown in Listing 9.39.

Listing 9.39: We've cut and pasted in the `li`s and links.
_includes/nav-links.html

```
<li><a href="/">Home</a></li>
<li><a href="">Nav 1</a></li>
<li><a href="">Nav 2</a></li>
<li><a href="">Nav 3</a></li>
```

With the code in Listing 9.39, we can replace the links in the header file with a Liquid tag, as shown in Listing 9.40.

Listing 9.40: Updating the header with an include and a second class.
_includes/header.html

```
<ul class="header-nav nav-links">
  {% include nav-links.html %}
</ul>
```

Note that we've also added a `.nav-links` class in Listing 9.40 so we can add styling to the links that will be shared between the header and footer. Before, we were targeting and styling the links using the class `.header-nav` (introduced in

Listing 9.14), but now that the links are going to be in multiple places, that isn't a good name to use to target the styling common to both the header and the footer.

Now that we've factored the nav links into a separate include, let's add them to the navigation section in the footer. In order to allow footer-specific styling, we'll also add a **`footer-nav`** class (in analogy with the header's **`header-nav`** class), as well as the general **`nav-links`** class added in Listing 9.40. The result appears in Listing 9.41.

Listing 9.41: The new Liquid tag to load the links in the footer.
`_includes/footer.html`

```
<footer class="footer">
  <a class="footer-logo" href="/">
    <img src="/images/logo.png" alt="Learn Enough"/>
  </a>
  <nav>
    <ul class="footer-nav nav-links">
      {% include nav-links.html %}
    </ul>
  </nav>
  <h3>Learn Enough <span>to Be Dangerous</span></h3>
</footer>
```

Now let's add some styling. First, we should move some of the styles that before were defined on **`.header-nav a`** over to **`.nav-links a`**, and change the class that is targeting the **`:hover`** and **`:active`** states from **`.header-nav`** to **`.nav-link`**, as in Listing 9.42.

Listing 9.42: Moving link styling into a new **`.nav-links`** class.
`css/main.css`

```
.header-nav a {
  color: #fff;
}
.nav-links a {
  font-size: 0.8rem;
  font-weight: bold;
  text-decoration: none;
  text-transform: uppercase;
}
.nav-links a:hover,
.nav-links a:active {
  color: #ed6e2f;
}
```

Again, the idea is that we want navigational links to look similar between the header and footer, and then for any changes that are specific to one location or the other by targeting the links using either the `.header-nav` or the `.footer-nav` class.

Finally, we'll add footer-specific styles, as shown in Listing 9.43.

Listing 9.43: New styling for footer navigation and links.
css/main.css

```
.footer-nav li {
  display: inline-block;
  margin: 2em 1em 0;
}
}
.footer-nav a {
  color: #ccc;
}
```

When you save and refresh, you'll have a nice header and footer, both pulling their navigational links from the same place (Figure 9.44).

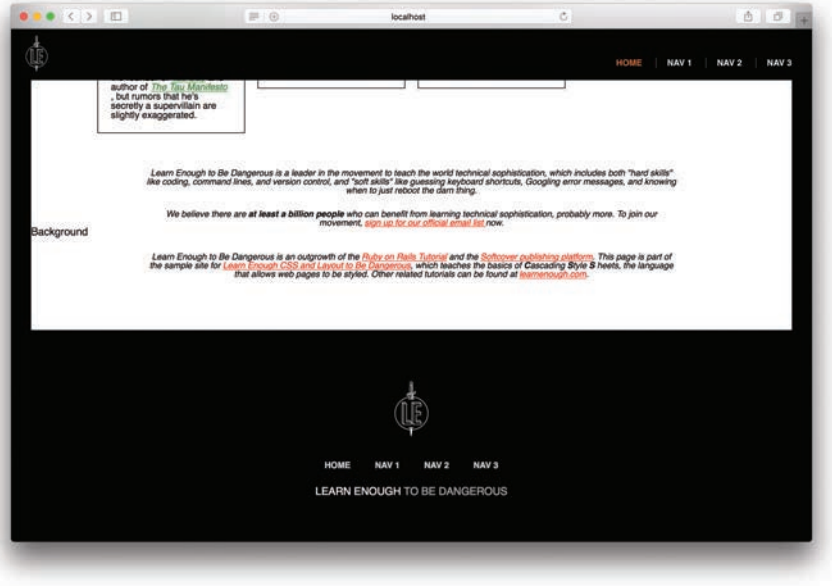

Figure 9.44: Styled header and footer with nav links from an include.

If you want to double-check and sync up all your styles, Listing 9.44 has the current state of the CSS declarations for the site.

Listing 9.44: The full header and footer styles.
css/main.css

```
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: 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;
}
blockquote, q {
  quotes: none;
}
blockquote:before, blockquote:after,
q:before, q:after {
  content: '';
  content: none;
}
table {
  border-collapse: collapse;
  border-spacing: 0;
}
strong, b {
  font-weight: bold;
}
em, i {
  font-style: italic;
```

```
}
a img {
  border: none;
}
/* END RESET*/

/* GLOBAL STYLES */
html {
  box-shadow: 0 0 0 30px #000 inset;
  padding: 0 30px;
}
body {
  font-family: helvetica, arial, sans;
}
h1 {
  font-size: 7vw;
  margin-top: 0;
}
a {
  color: #f00;
}
h2 ~ p {
  font-size: 0.8em;
  font-style: italic;
  margin: 1em auto 0;
  max-width: 70%;
  text-align: center;
}

/* HEADER STYLES */
.header {
  background-color: #000;
  color: #fff;
  position: fixed;
  width: 100%;
  z-index: 20;
}
.header-logo {
  background-color: #000;
  box-sizing: border-box;
  display: block;
  height: 10vh;
  left: -30px;
  padding-top: 10px;
  position: relative;
  text-align: center;
  width: 10vh;
}
.header-logo:hover,
.header-logo:active {
```

```
  background-color: #ed6e2f;
}
.header-logo img {
  width: 4.3vh;
}
.header-nav {
  float: right;
  padding: 5.5vh 60px 0 0;
}
.header-nav > li {
  display: inline-block;
  margin-left: 1em;
}
.header-nav > li ~ li {
  border-left: 1px solid rgba(255, 255, 255, 0.3);
  padding-left: 1em;
}
.header-nav a {
  color: #fff;
}
.nav-links a {
  font-size: 0.8rem;
  font-weight: bold;
  text-decoration: none;
  text-transform: uppercase;
}
.nav-links a:hover,
.nav-links a:active  {
  color: #ed6e2f;
}
.header-nav > li:first-child a {
  color: #ed6e2f;
}
.header-nav > li:first-child a:hover {
  color: #fff;
}

/* FOOTER STYLES */
.footer {
  background-color: #000;
  padding: 10vh 0 15vh;
  text-align: center;
}
.footer-logo img {
  width: 50px;
}
.footer h3 {
  color: #fff;
  padding-top: 1.5em;
  text-transform: uppercase;
```

```
}
.footer h3 span {
  color: #aaa;
}
.footer-nav li {
  display: inline-block;
  margin: 2em 1em 0;
}
.footer-nav a {
  color: #ccc;
}

/* HERO STYLES */
.full-hero {
  background-color: #c7dbfc;
  height: 50vh;
}

/* SOCIAL STYLES */
.social-list {
  list-style: none;
  padding: 0;
  text-align: center;
}
.social-link {
  background: rgba(150, 150, 150, 0.5);
  border-radius: 99px;
  box-sizing: border-box;
  color: #fff;
  display: inline-block;
  font-family: helvetica, arial, sans;
  font-size: 1rem;
  font-weight: bold;
  height: 2.5em;
  line-height: 1;
  padding-top: 0.85em;
  text-align: center;
  text-decoration: none;
  vertical-align: middle;
  width: 2.5em;
}
.social-list > li {
  display: inline-block;
  margin: 0 0.5em;
}

/* BIO STYLES */
.bio-wrapper {
  font-size: 24px;
  margin: auto;
```

```
  max-width: 960px;
  overflow: hidden;
}
.bio-box {
  border: 1px solid black;
  box-sizing: border-box;
  float: left;
  font-size: 1rem;
  margin: 40px 1% 0;
  padding: 2%;
  width: 23%;
}
.bio-box h3 {
  color: #fff;
  font-size: 1.5em;
  margin: -40px 0 1em;
  text-align: center;
}
.bio-box img {
  width: 100%;
}
.bio-box .social-link {
  display: block;
  margin: 2em 0 1em;
}
.bio-copy {
  font-size: 1em;
}
.bio-copy a {
  color: green;
}
```

Finally, in case you haven't been doing your own Git commits and deploys, now would be a good time to do one:

```
$ git add -A
$ git commit -m "Finish initial layout"
```

You'll discover that GitHub Pages is fully Jekyll-aware, and automatically generates and displays the site based on the contents of the repository—free static site hosting!

9.10.1 Exercise

1. **(challenging)** In the same manner that we just made the header links modular, first create a new include that makes the social links in the hero into an include that can be inserted into other places on the site. Then use the correct include tag to put it back where it originally was, and also a second include that builds the social links into a new `ul` in the footer.

This page intentionally left blank

Index

Symbols

– (en dash), 56
/ (forward slash), 9

A

About pages, 21
absolute sizing, 167
access, personal access tokens, 15
active links, 447. *See also* links
adaptation, mobile, 438–449
adding
  analytics snippets, 627
  background images, 334
  backgrounds, 334
  banners, 547–549
  blogs, 398–399
  borders, 207–208, 347
  classes, 134, 201, 522–523
  classes to links, 149–150
  column templates, 506–507
  comments, 155
  content in CSS grid, 551–555, 573
  content in HTML, 517
  content loops, 412–418
  CSS, 125
  CSS classes, 129
  CSS styles, 123
  dimensions, 195
  drop shadows, 310
  dummy elements, 384
  elements, 201
  favicons, 488–490
  files, 276, 397–399
  folders, 117
  footer structures, 314 (*see also* footers)
  Gravatar hotlinks, 46–47
  height, 173, 185, 216, 299–300
  "Hello, world!", 25
  hover states, 286, 287
  HTML, 456
  HTML to headers, 567–568
  images, 41–48
  `index.html`, 117, 118
  index pages, 358
  inputs, 463, 464
  Jekyll gems, 255
  labels, 463, 464
  links, 36, 37, 39–40, 381, 425, 489
  Liquid tags, 315
  lists, 67
  margin declarations, 223, 228
  margins, 86–87
  metadata, 494–497
  navigation links, 69–71
  negative margins, 301, 302
  padding, 467
  pages, 52, 357–360
  paragraphs, 31–32, 37–38

positioning, 467
post pages, 416, 417
records, 609, 610
rows, 55, 56
rules, 95, 130
siblings, 288–289
styles, 73–74, 150, 201, 336–337, 344, 352–356, 361, 391, 503, 518, 581 (*see also* inline styling)
styles to headers, 280–284
`meta` tag, 24
tags, 61
text, 39
text to boxes, 209
titles, 490–494
Twitter links, 39–40
wrappers, 169, 184–185, 393
addresses
custom, 622
IP (Internet Protocol), 594, 600
adjacent siblings, 288–289
`:after` method, 213
`:after` pseudo-element, 343–356
`.alert` class, 137, 138
aligning. *See also* moving
child elements, 519
content, 572–573
self-aligning, 582–583
vertical flex centering, 371–375 (*see also* centering)
`align-items` property, 371
alpha levels, 161
`alt` attribute, 41
analytics, site, 626–629
anchoring background images, 336, 338
animation, 390
annotations, 9. *See also* text
applications, Gravatar, 45
applying
borders, 235–236
flexbox, 367
formatting (*see* formatting)
italics, 32–33
`margin: auto`, 229–230
`minmax`, 520
styles, 73–74, 346 (*see also* inline styling)
areas
banners, 555–556
naming, 540–544
A records, 608–610
`a` tag, adding, 61
attacks
Cloudflare, 603 (*see also* Cloudflare)
DDoS, 599, 603
attributes, 24
`alt`, 41
configuring, 466
`auto-fill`, 515–516, 524–527
`auto-fit`, 515–522, 524–527, 584
auto-sizing, 509, 518
avatars, 45, 46

B

`background` declaration, 140
`background.position` style, 338–339
backgrounds
adding, 334
colors, 161 (*see also* backgrounds)
formatting, 161
images, 336–337
resizing images, 338
banners
adding, 547–549
areas, 555–556
moving, 548–549
base-level objects, 260. *See also* layouts
`:before` pseudo-element, 343–356
beginning tags, 9
behavior, margins, 202–205
BEM (Block Element Modifier), 137
Berners-Lee, Tim, 6, 7
Block Element Modifier. *See* BEM
block elements, 54, 55–58, 193–199, 511
`blockquote` tag, 65, 68, 74–79
blogs
adding, 398–399
columns, 405, 406
content loops, 412–418
directories, 398–399
formatting, 414

frontmatter in, 398, 399–400
index pages, 398, 402–411, 448, 449
posting, 398–411
post pages, 419–427
previewing, 425–426
sizing pages, 407 (*see also* pages)
width, 426
`body` tag, 20, 25, 29, 110
bold, 7, 34, 35
borders, 193, 235–250, 310
adding, 207–208, 347
applying, 235–236
box models, 199–206
inline/block elements, 193–199
line height, 244–246
making circles, 238–244
radius, 238
styles, 236–237
syncing index pages, 244–249
zero-height/zero-width elements, 347, 348
Bos, Bret, 113
boxes. *See also* box models
content, 579–580
inline styling, 88–90
restyling, 354–355
scrolling, 215
spacing, 224
box models, 191, 511
adding text to, 209
borders, 199–206, 235–250
floats, 206–214
inline/block elements, 193–199
inline blocks, 219–223
`margin: auto`, 229–230
margins, 199–206, 223–233
`overflow` method, 211–212, 214–218
padding, 199–206, 234–235
properties, 201
rounding, 238–244
sizing, 208
breakpoints, 431, 432
browsers
CSS support, 497
functionality, 7
grid overlay, 527–529 (*see also* CSS grid)
modifying windows, 425
refreshing, 19
resizing windows, 434–438
building blocks, grid (CSS), 570–574

C

caches, edge caching, 602
`calc()` function, 586
callouts, 371, 442
containers, 371
titles, 372
Cascading HTML Style Sheets. *See* CHSS
Cascading Style Sheets. *See* CSS (Cascading Style Sheets)
CDNs (content delivery networks), 483–488
cells
formatting, 513
table data, 55, 56, 58
centering, 519. *See also* moving
headings, 79
images, 87
lists, 583
vertical flex, 371–375
characters
character entity references, 56
sequences of, 9
checkboxes, adding, 464, 465
child elements, 171, 519
child selectors, 220
Chrome
mobile views, 438 (*see also* mobile media queries)
resizing windows, 434–438
web inspector, 437
CHSS (Cascading HTML Style Sheets), 113
circles, formatting, 238–244
classes
adding, 134, 201, 522–523
adding links to, 149–150
adding names to unordered lists, 219
`.alert`, 137, 138
combining, 151
CSS (Cascading Style Sheets), 94, 129
`feature`, 576
`grid`, 564

grid-banner, 547
grid-content, 534–540
grid-expand, 545
HTML5, 280
naming, 134, 135, 136, 184
pseudo-classes, 284–286
targeting, 140
when to use, 137–140
clearing, 208–214. *See also* deleting
Cloudflare, 593
benefits of, 599
configuring, 599–606
connecting registrar nameservers, 604–606
features of, 599–604
GitHub Pages configuration, 610–613
page rules, 613–618
signup, 604
CNAME records, 610, 616
code
blogs, 400 (*see also* blogs)
detecting screen size, 430, 431
frontmatter, 260
Ruby programming language, 254
snippets, 626, 627
code tag, 52, 53
collapsing margins, 186, 189, 204–205
collections, 261
colors
backgrounds, 161
border styles, 236–237
configuring transparency, 161–163
CSS, 157–163
formatting links, 162
hexadecimal, 77, 158–160
naming, 129, 157
pickers, 160
styles, 148
text, 7
color tag, 110
columns
defining, 502, 503, 504
flexbox, 406
formatting, 361, 362
gaps, 514
grid (CSS), 507–510
layouts, 564
on mobile screens, 520
positioning, 574–575
relative spanning, 522–524
self-aligning, 582–583
sizing, 509–510
starting, 537–538, 582–583
structure, 564
templates, 506–507, 509, 586
three-column layouts, 381–386, 445
two-column layouts, 409
combining
classes, 151
styles, 138, 139, 140
command lines
creating tags, 51
first tags, 17
commands
curl, 304
gem, 255
open, 19
rgb(), 161
rgba(), 161
unzip, 387
comments, 407
adding, 155
CSS, 106, 139
HTML, 65
styles, 373–374
computers. *See* desktops
conditionals, 492–493
configuring
attributes, 466
Cloudflare, 599–606
page layouts, 559–563
transparency, 161–163
wrappers, 296–297
connecting
registrar nameservers, 604–606
servers, 256
containers
callouts, 371
content filling, 363–368
CSS grid, 502, 503
examples of, 215

parent, 508–509
styles, 369
content
adding in CSS grid, 551–555, 573
aligning, 572–573
boxes, 579–580
CSS grid, 501–504 (*see also* CSS grid)
elements, 543
filling containers, 363–368
formatting columns, 362–363
gallery stubs, 386–395
loops, 412–418
moving, 566
overlapping, 545–546
positioning, 340–341
replacing, 328
templates and, 327–330 (*see also* page templates)
variables, 329–330
wrapping, 339–340, 339–340, 365
content delivery networks. *See* CDNs
conventions, naming, 136. *See also* naming
copying images, 42. *See also* moving
corners, rounding, 238–240
creating. *See* formatting
Creative Commons licenses, 48
CSS (Cascading Style Sheets), 6, 8
adding, 125
adding rules, 130–131
adding styles, 344
animation, 390
box models (*see* box models)
classes, 94, 129
colors, 157–163 (*see also* colors)
comments, 106, 139
defining, 126, 127
development of, 112–115
dropdown menus, 455–463
elements, 131
files, 264–275
flexbox (*see* flexbox)
footers, 312–325
formatting post pages, 419–424
front-end development, 106–109
grids (*see* CSS grid)
hexadecimal colors, 158–160
history of, 109–116
implementations of, 114
inline styling, 93–98
layouts, 251 (*see also* layouts)
media queries, 431 (*see also* media queries)
naming, 123, 134–137
overview of, 103–106
positioning, 291–309
priority and specificity, 140–146
resets, 265–267
rules, 104, 140–146, 145–156
sample site setups, 116–120
selecting text styles, 190–191
selectors, 108, 128–131, 149
sizing, 163–164 (*see also* sizing)
specialty page layouts, 361–363 (*see also* specialty page layouts)
styles, 121–127, 133 (*see also* styles)
subjectivity of, 115–116
table data cells, 58
technical sophistication, 106
values, 157
when to use classes/ids, 137–140
CSS grid, 497
adding in content, 573
`auto-fill`, 515–516, 524–527
`auto-fit`, 515–522, 524–527
building blocks, 570–574
columns, 503, 504, 507–510 (*see also* columns)
creating HTML files for, 501–504
elements, 520–503
finishing layouts, 550–556
footers, 540, 542
formatting layouts, 524–527
`fr`(functional) units, 507–510
gaps, 510–515
global grids, 563–569
grid layouts, 529–540
grid lines, 529–533
grid overlays, 527–529
grids inside a grid, 584–589
headers, 540, 542
inside elements, 556–589

justifying, 570–574
minmax, 515–516
modifying, 504, 506
named lines/areas, 540–544
overlapping, 545–546, 576–580
overview of, 499–504
padding, 550–556
page setups, 559–563
positioning columns, 574–575
positioning headers, 563–569
relative spanning columns, 522–524
rows, 502, 503, 504, 510–515 (*see also* rows)
source-independent positioning, 547–550
starting columns, 582–583
subgrids, 557–558
curl command, 304
curves, borders, 238–240
custom addresses, 622
custom domains, 117, 593–594. *See also* domains
Cloudflare page rules, 613–618
configuring Cloudflare, 599–606
DNS, 597–599
GitHub pages, 606–618
purchasing, 598
registering, 594–598
TLDs, 594–597
custom email, 619. *See also* email
Google Workplace signup, 621–622
MX records, 622–626
site analytics, 626–629
customizing
blogs, 400 (*see also* blogs)
elements, 104, 105
favicons, 488–490
fonts, 475–488
metadata, 494–497
titles, 490–494

D

DDoS (Distributed Denial of Service) attacks, 599, 603
declarations, 123
adding style, 150
background, 140
overriding, 142
default.html, 327. *See also* page templates
default port numbers, 256
default templates, 365, 366. *See also* templates
defining
CSS (Cascading Style Sheets), 126, 127
tables with headers, 55
deleting
elements, 195
spacing, 514
densities of pixels, 164, 166. *See also* pixels
deploying GitHub pages, 120
design. *See also* formatting
affordances, 284
bogs, 398–401 (*see also* blogs)
elements, 477
inline styling, 73–74 (*see also* inline styling)
layouts (*see* layouts)
mobile media queries, 429–432 (*see also* mobile media queries)
modular systems and, 139
pixels, 165 (*see also* pixels)
responsive, 429
Safari, 427
sizing, 180 (*see also* sizing)
UX (user experience), 108
viewing screens, 434–438
web, 105
desktops, development, 426
detecting screen size, 430, 431
development
of CSS, 112–115
desktops, 426
environments, 107
front-end, 106–109
inline styling, 73–74 (*see also* inline styling)
mobile-first, 426–427
mobile-ready prototypes, 450
dimensions
adding, 195, 299–300
CSS grid, 504
flexbox, 502
post pages, 423
directories
adding index pages, 358–359

blogs, 398–399
formatting, 13
`images`, 42, 43
Jekyll, 262 (*see also* Jekyll)
`display: block` style, 195–196
`display: flex` style, 199
`display: inline-block` style, 197–198
`display: inline` style, 196–197
`display: none` style, 194–195
`display` property, 425
displays
detecting screen size, 430, 431
fonts, 484
modifying properties, 196
pixels, 158 (*see also* pixels)
viewing screens, 434–438
Distributed Denial of Service. *See* DDoS (Distributed Denial of Service) attacks
divisions, 62–66
`div` tag, 62, 65, 90
DNS (Domain Name System), 593, 597–599
MX records, 622
records, 598, 607–609
`DOCTYPE`, 23
Document Object Models. *See* DOMs
documents. *See also* HTML (Hypertext Markup Language)
CSS (Cascading Style Sheets), 106 (*see also* CSS (Cascading Style Sheets))
margins, 82–87
navigating, 8
non-linked, 9
outlines, 29
types, 23
Domain Name System. *See* DNS (Domain Name System)
domains, 593–594
Cloudflare page rules, 613–618
configuring Cloudflare, 599–606
custom, 117
DNS, 597–599
GitHub pages, 606–618
names, 594
purchasing, 598
registering, 594–598
TLDs, 594–597
DOMs (Document Object Models), 121
Don't Repeat Yourself. *See* DRY
dots and pluses trick (Gmail), 620
downloading images, 43, 48
downward-pointing triangles, 348
dropdown menus, 212, 453–463
adding HTML, 456
CSS, 457–460
hitboxes, 454–463
mobile, 463–473
drop shadows, adding, 310
DRY (Don't Repeat Yourself), 107, 122, 252, 265, 565
dummy elements, adding, 384
dynamic sites, 253

E

edge caching, 602
editing DNS records, 609
elements. *See also* styles
adding, 201
adding dummy, 384
`:after` pseudo-element, 343–356
applying borders to, 235–236 (*see also* borders)
`:before` pseudo-element, 343–356
block, 54, 55–58, 193–199, 511
child, 171, 519
content, 543
covering multiple columns with CSS, 523–524
CSS, 131
CSS grid, 504, 506
customizing, 104, 105
deleting, 195
design, 477
`display: none` style, 194–195
hiding, 467, 468
HTML, 25
HTML5, 280
inline, 54, 59–60, 193–199, 210, 340
inline blocks, 219–223
lists, 66–68
overlapping, 545–546

parent, 170, 171
positioning, 291–309, 540
sections, 577
semantics, 278, 280
sizing, 372
stacking, 438–439, 443
stretching with flexbox, 364
styles, 167
viewing, 577
of web pages, 252
wrapping, 169
zero-height/zero-width, 347, 348
email
Gmail, 619–622
Google Workplace signup, 621–622
MX records, 622–626
site analytics, 626–629
emphasized text, 32–33
empty boxes, 82
empty declarations, styles, 171
empty style tags, 93
`em` tag, 33
em units, sizing, 175–181
encryption, 600, 601
en dash (–), 56
ending tags, 9
environments
development, 107
variables, 257
errors, 24
explicit positioning, 578
Extensible Markup Language. *See* XML (Extensible Markup Language)
external stylesheets, 93, 96–97, 110

F

fault-tolerance, 20
`feature` class, 576
files
adding, 276, 397
CSS, 106, 264–275 (*see also* CSS (Cascading Style Sheets))
layouts, 264
`.png` files, 488
posts/post-type, 261
filtering spam, 619
finishing layouts, 550–556
`first-child` pseudo-class, 287–288
first tags, 17–20
fixed headers, 309–312
flags, `!important`, 143, 145
flexbox
applying, 109, 367
columns, 405, 406
comparing to CSS grid, 502, 503 (*see also* CSS grid)
dimensions, 502
flex containers, 363–364 (*see also* containers)
flex direction rule, 367
gallery stubs, 386–395
properties, 375, 445
specialty page layouts, 361–363 (*see also* specialty page layouts)
stretching elements with, 364
styles, 375–381
three-column layouts, 381–386
two-column layouts, 409
vertical flex centering, 371–375
`flex-grow` property, 367
flex items, 363–364, 366, 369
properties, 376–381
rules, 372
floating, 79–82, 443
box models, 206–214
clearing, 208–214
images, 194
folders, 357–360. *See also* documents; files
adding, 117
includes, 276
Font Awesome, 477, 478, 479, 480, 481, 482
fonts, 10. *See also* inline styling; text
customizing, 475–488
favicons, 488–490
formatting, 243, 244
installing vector image, 477–483
loading text via CDNs, 483–488
Open Sans, 484
percentages, 174
selecting text styles, 190–191
sizing, 167, 175–181, 182

stacking, 486
types of, 476
footers, 252, 312–325
CSS grid, 540
styles, 315, 318–325
`footer` tag, 314
foreach loops, 413
formatting. *See also* layouts; styles
auto-sizing, 509, 518
backgrounds, 161
blogs, 398–401, 414 (*see also* blogs)
borders, 235–250
cells, 513
circles, 238–244
columns, 361–362
CSS, 116–120
directories, 13
divisions, 62–66
dropdown menus, 453–463
emphasized text, 32–33
favicons, 488–490
fonts, 243, 244, 475–488
grids inside a grid, 584–589
homepages, 330–342
HTML, 21–23
inline styling, 73–74 (*see also* inline styling)
Jekyll, 259 (*see also* Jekyll)
layouts, 251, 524–527 (*see* layouts)
lists, 66–68
loops, 411
margins, 82–87
metadata, 494–497
named areas, 540
navigation menus, 68–72
`overflow` method, 211–212, 214–218
page rules, 613–614
pixels, 163 (*see also* sizing)
positioning, 291–309
post pages, 419–424
repositories, 14, 15, 120
sizing, 180 (*see also* sizing)
source-independent positioning, 547–550
spans, 62–66, 456–457
strong text, 34, 35
styles, 124
tables, 54–61, 362–363
tabs, 51
tags, 10–11
text, 6, 31–35, 240
titles, 490–494
viewing screens, 434–438
width, 426
forwarding
permanent, 615
URLs, 615, 616
forward slash (/), 9
`fr`(functional) units, 507–510
front-end development, 106–109
frontmatter, 260, 263, 398, 399–400
functionality of browsers, 7
functional units (`fr`), 507–510

G

galleries
adding links, 381
mobile styles, 445, 446, 447, 448
stubs, 386–395
gaps. *See also* spacing
columns, 514
CSS grid, 510–515
Gecko engine, 111, 114
`gem` command, 255
general siblings, 289
generic restricted TLDs, 595. *See also* TLDs (top-level domains)
generic TLDs, 595. *See also* TLDs (top-level domains)
GitHub pages, 12, 13
blogs (*see* blogs)
configuring CloudFlare, 607–610
custom domains, 606–618
deploying, 120
repositories, 14, 15, 19
results to, 26
settings, 610–613
templates, 16
global grids, 563–569, 586
Gmail (Google Mail), 619–622
dots and pluses trick, 620
spam filtering, 619
Google Analytics, 626–629

Google Fonts CDN service, 484
Google Workplace signup, 621–622
graphics, `.png` files, 488. *See also* images
Gravatar, 45, 46–47
grayscale, 160. *See also* colors
green, 157
grid (CSS), 497
  adding in content, 573
  `auto-fill`, 515–516, 524–527
  `auto-fit`, 515–522, 524–527
  building blocks, 570–574
  columns, 507–510
  creating HTML files for, 501–504
  elements, 502–503
  finishing layouts, 550–556
  footers, 540, 542
  formatting layouts, 524–527
  `fr`(functional) units, 507–510
  gaps, 510–515
  global grids, 563–569
  grid layouts, 529–540
  grid lines, 529–533
  grid overlays, 527–529
  grids inside a grid, 584–589
  headers, 540, 542
  inside elements, 556–589
  justifying, 570–574
  `minmax`, 515–516
  modifying, 504, 506
  named lines/areas, 540–544
  overlapping, 545–546, 575–580
  overview of, 499–504
  padding, 550–556
  page setups, 559–563
  positioning columns, 574–575
  positioning headers, 563–569
  relative spanning columns, 522–524
  rows, 510–515
  source-independent positioning, 547–550
  starting columns, 582–583
  subgrids, 557–558
`grid-auto-rows`, 521
`grid-banner` class, 547
`grid` class, 564
`grid-content` class, 535–540
`grid-expand` class, 545
grids
  CSS, 109
  global, 563–569
  on mobile screens, 515
  padding, 514
groups, styles, 155–156

H

hacking, 602
Håkon Lie, 113
headers, 23, 252, 275–284
  adding HTML to, 567–568
  adding margins, 86–87
  CSS grid, 540
  fixed, 309–312
  mobile media queries, 440, 441, 442
  pages, 278–279
  positioning, 563–569
  styles, 280–284, 565
  tables, 55, 56
  tags, 30
  updating, 317
`header` tag, 62, 64, 314
`head.html`, 276–278
headings
  centering, 79
  index pages, 29–31
`head` tag, 21
height. *See also* sizing
  adding, 173, 185, 216, 299–300
  borders, 244–246
  columns, 361
  `overflow` method, 216, 217–218
  `vh` (viewport height), 184–189
  zero-height/zero-width elements, 347, 348
"Hello, world!," 6, 18, 25
hero styles, 334, 335
hexadecimal colors, 158–160
  counting n, 158
  HTML, 77
  RGB, 158, 160 (*see also* colors)
hiding
  elements, 467, 468
  `overflow` method, 216, 217–218

highlighting syntax, 18
histories, CSS (Cascading Style Sheets), 109–116
hitboxes, 454–463
homepages, 330–342
  blogs, 425 (*see also* blogs)
  content, 350–353 (*see also* content)
  previewing, 425–426
  uploading HTML, 331–333
  wrapping, 393
horizontal layouts, 445, 446
hotlinking, 42, 44–47, 46–47
hover rollovers, 455
HTML (Hypertext Markup Language), 6, 7
  adding, 456
  adding to headers, 567–568
  applying realistic, 146–147
  box models, 194 (*see also* box models)
  comments, 65
  creating files for CSS grid, 501–501
  documents (*see* documents)
  elements, 25
  formatting, 21–23
  headers, 278–279 (*see also* headers)
  hexadecimal colors, 77
  inline styling, 73–74 (*see also* inline styling)
  pages (*see* pages)
  skeletons, 20–27
  starting projects, 12–17
  tags, 8–12 (*see also* tags)
  uploading, 331–333
  writing, 10–11
HTML5, 7, 42
  classes, 280
  elements, 280
`html` tag, 21
hyperlinks, 35. *See also* links
hypertext, 35. *See also* links
Hypertext Markup Language. *See also* HTML (Hypertext Markup Language)

I

ICANN, 595
icon fonts, 476, 477, 479, 481, 482. *See also* fonts
id style, 137–140, 146
images, 6
  adding, 41–49, 334
  background, 336–337
  centering, 87
  copying, 42
  downloading, 48
  floating, 79–82, 194
  Gravatar, 45, 46–47
  hotlinking, 42, 44–47
  links, 49
  margins, 82–87
  moving, 79–82
  `.png` files, 488
  resizing, 80, 81, 338, 339, 336
  sizing, 165, 166
  vector image fonts, 477–483
`images` directory, 42, 43
`img` tag, 41, 82
  adding, 61
implementation of CSS (Cascading Style Sheets), 114
implicit row sizes, 511–513. *See also* rows; sizing
`!important` flag, 143, 145
includes, 275
  folders, 276
  Jekyll, 261
`index.html`, adding, 117, 118
index pages, 6, 19
  adding, 358
  blogs,398–411, 448, 449 (*see also* blogs)
  content loops, 412–418
  filling in, 29
  headings, 29–31
  homepages (*see* homepages)
  images, 41–48
  links, 35–40
  post pages, 419–427
  syncing, 245–249
  text formatting, 31–35
  updating, 330–334
inline blocks, 219–223
inline elements, 54, 59–60, 193–199, 210, 344
inline styling, 48, 122
  boxes, 88–90

CSS, 93–98
floats, 79–82
margins, 82–87
navigation menus, 90–93
resizing, 80
text styling, 74–79
inputs, 463, 464–465
installing
iOS simulators, 449
Jekyll, 253–259
vector image fonts, 477–483
interfaces, 599
page rules, 613–614
UI (user interface), 108
internal stylesheets, 93, 110
Internet Protocol. *See* IP (Internet Protocol)
iOS simulators, 449
IP (Internet Protocol), 594, 600
italics, 32–33, 74. *See also* inline styling
items, flex, 363–364, 366, 369
properties, 376–381
rules, 372

J

Jekyll, 252, 253–259. *see also* layouts
adding pages, 357–360
blogs (*see* blogs)
content loops, 412–418
CSS files, 264–275
directories, 262
`first-child` pseudo-class, 287–288
fixed headers, 309–312
flexbox (*see* flexbox)
footers, 312–325
includes, 261
installing, 254–259
layouts, 259–261, 264
page templates, 261 (*see also* page templates)
post pages, 419–427
posts/post-type files, 261
selectors, 284–286
sibling pseudo-class, 288–291
variables, 329–330
justifying
center, 519 (*see also* centering; moving)
CSS grid, 570–574

K

Karlton, Phil, 134

L

labels, 463
adding, 463, 464
configuring attributes, 466
landscape orientation, 431
layouts
applying flexbox, 109
blogs (*see* blogs)
CCS files, 264–275
columns, 564
CSS grid, 497 (*see also* CSS grid)
files, 264
finishing, 550–556
`first-child` pseudo-class, 287–288
fixed headers, 309–312
flexbox (*see* flexbox)
fonts, 475–488
footers, 312–325
formatting, 524–527
frontmatter, 260 (*see also* frontmatter)
grids, 529–540 (*see also* CSS grid)
headers/`head.html`, 275–284
horizontal, 445, 446
includes, 261
Jekyll, 252, 253–259, 259–261
nesting, 400–401
overview of, 251–253
page templates (*see* page templates)
positioning, 291–309
selectors, 284–286
sibling pseudo-class, 288–291
specialty page, 361–363 (*see also* specialty page layouts)
three-column, 361–362, 445
three-column page, 381–386
two-column, 409
vertical, 445
licenses, Creative Commons, 48
lightgray, 157
lines
grid, 529–533 (*see also* CSS grid)
height, borders, 244–246
named, 540–544

links, 6, 8
  active, 447
  adding, 36, 37, 39–40, 381, 425, 489
  adding classes, 149–150
  adding navigation, 69–71
  adding style declarations, 150
  Font Awesome, 477, 478, 479, 480
  formatting colors, 162
  hotlinking, 42, 44–47
  images, 48
  index pages, 35–40
  inline/block elements, 197
  moving styling, 318
  non-clickable, 462
  pseudo-classes, 284–286
  rounding corners on, 238
  spacing, 274
  styles, 318–325
  styling navigation, 280–284
`link` tag, 97, 271–272
Liquid, 253, 259, 277–278. *See also* Jekyll
  blogs, 413 (*see also* blogs)
  replacing content, 328
  tags, 259, 315, 317
lists, 6, 66–68
  adding, 67
  centering, 583
  ordered, 68
  unordered, 68, 219
`li` tag, 66–68
loading
  text fonts via CDNs, 483–488
  websites, 19
local servers, viewing, 451
logos, 304–309, 477
loops
  content, 412–418
  foreach, 413
  formatting, 411
  terminating, 413
Lütke, Tobi, 253, 259

M

macOS, 449, 450
`margin: auto`, 229–230
margins
  adding, 86–87, 301, 302
  behavior, 202–205
  for boxes, 223–233
  box models, 199–206
  collapsing, 186, 187, 204–205
  formatting, 82–87
  modifying, 224
  negative, 231–233
  resetting, 186
  rules, 94
  sizing, 224, 225, 227, 229
Markdown, content, 399, 413
matching URLs (uniform resource locators), 616
measurements, 163. *See also* sizing
  CSS, 164 (*see also* CSS (Cascading Style Sheets))
  pixels, 166 (*see also* pixels)
media queries, 109, 187, 431. *See also* mobile media queries
menus
  divider lines, 290–291
  dropdown, 212, 453–463
  mobile dropdown, 463–473
  navigation, 68–72, 90–93
  optimizing for small screens, 467–468
merging styles, 139, 140
metadata, 23, 260, 494–497
`meta` property, 452
`meta` tag, 24, 41
methods. *See also* styles
  `:after`, 213
  `overflow`, 211–212, 214–218
`minmax`, 515–516, 520
mobile-first development, 426–427
mobile hover navigation, 285
mobile media queries, 429–432
  dropdown menus, 453–463
  headers, 440, 441, 442
  mobile adaptation, 438–449
  mobile dropdown menus, 463–473
  mobile viewports, 449–453
  stacking elements, 438–439
  styles, 429–432

viewing screens, 434–438
mobile-ready prototypes, 450
mobile screens
columns on, 521
grids on, 515
models, box. *See* box models
modifying
browser windows, 425
CSS grid, 504, 506
display properties, 196
font sizes, 175, 176 (*see also* sizing)
images, 80
margins, 224
styles, 307–309
thumbnails, 445, 446
modular systems, 139
moving
banners, 548–549
content, 339, 566
headings, 79
images, 42, 79–82
`inline-block` elements, 222
link styling, 318
objects, 304
relationships, 121
self-aligning, 582–583
text, 74–79
Mozilla Developer Network CSS Reference, 106
MX records, 622–626

N

named lines/areas, 540–544
naming
classes, 134, 135, 136, 184
colors, 130, 157
conventions, 136
CSS, 124
domains, 593–594 (*see also* domains)
styles, 134–137
wrapping font names, 487
navigating
adding links, 381, 425
documents, 8
dropdown menus, 453–463
mobile dropdown menus, 468–471
mobile hover navigation, 285
styling links, 280–284
navigation menus, 68–72, 90–93
negative margins, 231–233, 301, 302
negative positioning, 297–298
nesting layouts, 400–401
new generic TLDs, 596. *See also* TLDs (top-level domains)
non-clickable links, 462
non-linked documents, 9
numbers
hex, 158, 159
ports, 255

O

Object Oriented CSS. *See* OOCSS
objects
base-level, 260 (*see also* layouts)
moving, 304
negative positioning, 297–298
OOCSS (Object Oriented CSS), 137
open command, 19
Open Sans fonts, 484
optimizing
fonts, 475–488
menus for small screens, 467–468
SEO, 615
options section, 587–588
ordered lists, 68
outlines, documents, 29. *See also* documents
overflow, examples of, 215
`overflow` method, 211–212, 214–218
overlapping
elements, 545–546
sections, 575–580
overlays, grid, 527–529. *See also* CSS grid
overriding
declarations, 142
styles, 139

P

padding, 82, 88, 193, 445
adding, 467
box models, 199–206, 234–235
grid (CSS), 550–556

grids, 514
resetting, 186
values, 234
page layouts, 251, 559–563. *See also* layouts
specialty (*see* specialty page layouts)
three-column, 381–386
pages, 261, 262
adding, 52, 357–360
block elements, 54, 55–58
Cloudflare rules, 613–618
content loops, 412–418
CSS grid, 501–502 (*see also* CSS grid)
divisions, 62–66
fonts, 475–488
headers, 278–279
index (*see* index pages)
inline elements, 59–60
lists, 66–68
navigation menus, 68–72
post, 419–427
refreshing, 416
resizing, 434–438
sizing, 407
spans, 62–66
tables, 54–61
titles, 26, 490–494
variables, 401–402
page templates, 261, 263
adding pages, 357–360
homepages, 330–342
selectors, 343–356 (*see also* selecting)
template content, 327–330
variables, 329–330
paragraphs. *See also* text
adding, 31–32, 37–38
adding images, 43
links, 35–40 (*see also* links)
styles, 148
parameters, queries, 45, 46
parent containers, 508–509. *See also* containers
parent elements, 170, 171, 175
Pei-Yuan Wei, 112
percentages
fonts, 174
modifying margins from pixels to, 224
sizing, 169–175
Perl, 69
permanent forwarding, 615
personal access tokens, 15
pixels, 158
measurements, 166
modifying margins to percentages from pixels, 224
sizing, 163, 164–168 (*see also* sizing)
plain text, 9, 175
`.png` files (Portable Network Graphics), 488
points, sizing, 164–168
Portable Network Graphics. *See* `.png` files
portrait orientation, 431
ports
numbers, 255
servers, 256
positioning, 291–309
adding, 467
columns, 574–575
content, 339–340
elements, 540
explicit, 578
headers, 563–569
vertical, 340
posting
blogs, 398–411
building blogs, 416
post pages, 419–427
posts/post-type files, 261
Preston-Werner, Tom, 254
previewing blogs, 425–426
priority, CSS (Cascading Style Sheets), 140–146
projects
first tags, 17–20
starting, 12–17
propagation, DNS, 600. *See also* DNS (Domain Name System)
properties
`align-items`, 371
borders, 199
box models, 201
`display`, 425
flexbox, 445
flex containers, 375–376

flex-grow, 367
flex items, 376–381
floats, 206–214
margins, 199
meta, 452
modifying display, 196
text-align, 78
z-index, 292
prototypes, mobile-ready, 450
pseudo-classes, 284–286
first-child, 287–288
inputs, 464–465
sibling, 288–291
pseudo-elements
:after, 343–356
:before, 343–356

Q

queries
media, 109, 189
mobile media, 430–431 (*see also* mobile media queries)
parameters, 45, 46
quotations, blockquote tag, 74. *See also* blockquote tag

R

radius, borders, 238
Raisch, Robert, 112
records
adding, 609, 610
CNAME, 610, 616
DNS, 598, 607–609
MX, 622–626
A records, 608–610
TXT, 622, 623, 624
red, 157
red, green, blue. *See also* RGB
redirects, 615
refreshing
browsers, 19
pages, 416
registering custom domains, 594–598
registrar nameservers, connecting, 604–606
relationships, moving, 121
relative sizing, 164
relative spanning columns, 522–524
remote origin, 14
removing. *See also* deleting
elements, 195
rows, 517
spacing, 513
rem units, sizing, 181–184
replacing
content, 328
inline styles with classes, 95
repositories
creating, 14, 15, 120
GitHub pages, 19
searching, 16
resets
CSS, 265–267
rules, 267
styles, 186
resizing
images, 338, 339, 336
inline styling, 80
windows, 434–438
responsive design, 429, 435–436
restoring element displays, 195
restyling, 354–355. *See also* styles
RGB (red, green, blue), 77, 158, 160. *See also* colors
rgba() command, 161
rgb() commands, 161
right margins, adding, 84. *See also* margins
Right Way, 52
rollovers, hover, 455
root em units. *See* rem units
rounding borders, 238–240
rows. *See also* columns
adding, 56
CSS grid, 510–515
defining, 502, 503, 504, 510
removing, 517
starting, 536, 539
tables, 55
Ruby programming language, 254
rules
adding, 95, 130–131
Cloudflare page, 613–618

CSS, 104, 140–146, 145–156 (*see also* CSS (Cascading Style Sheets))
flexbox, 406 (*see also* flexbox)
flex direction, 367
flex items, 372
margins, 94
modifying font sizes, 175, 176
resets, 267
styles, 88
targeting classes, 140
running Jekyll, 254–259

S

Safari
developer tools, 427
resizing windows, 434–438
web inspector, 436
Scalable and Modular Architecture for CSS. *See* SMACSS
screens. *See also* displays; viewing
columns on mobile, 521
detecting size of, 430, 431
grids on mobile, 515
optimizing menus for small, 467–468
viewing, 434–438
scrolling. *See also* moving
boxes, 215
`overflow` method, 217–218
search engine optimization. *See* SEO (search engine optimization)
searching repositories, 16
sections
adding styles, 352–356, 391
elements, 578
options, 587–588
overlapping, 575–580
sizing, 352–353
Secure Sockets Layer. *See* SSL (Secure Sockets Layer)
security
attacks (*see* attacks)
encryption, 600, 601
hacking, 602
selecting
fonts, 243, 244, 475–488, 476
text styles, 190–191
TLDs, 595, 596, 597
selectors, 221–222, 284–286
`:after` pseudo-element, 343–356
`:before` pseudo-element, 343–356
CSS, 108, 128–131, 149
page templates, 343–356
pseudo-classes, 284–286
siblings, 288–291
self-aligning, 582–583
semantics, 10–11, 278, 280
SEO (search engine optimization), 615
sequences of characters, 9
servers
connecting, 256
connecting registrar nameservers, 604–606
Jekyll (*see* Jekyll)
ports, 256
settings, GitHub pages, 610–613. *See also* configuring
setups. *See also* formatting
setups, CSS, 116–120
sharing URLs (uniform resource locators), 257
shorthand notation, flex items, 376–381
sibling pseudo-class, 288–291
signup
Cloudflare, 604
Google Workplace, 621–622
site analytics, 626–629
site generators, 253
site headers, 278–279. *See also* headers
sizing
absolute, 167
auto-sizing, 509, 518
box models, 208
columns, 509
CSS, 163–164
detecting screen size, 430, 431
elements, 372
em units, 175–180
fonts, 167, 175–180, 182
images, 165, 166
inline styling, 80
landscape orientation, 431
margins, 224, 225, 227, 229

pages, 407
percentages, 169–175
pixels, 164–168
portrait orientation, 431
relative, 164
`rem` units, 181–184
Safari, 434
sections, 352–353
selecting text styles, 190–191
styles, 307
text, 175, 176
`vh` (viewport height), 184–189
`vw` (viewport width), 184–189
windows, 426
skeletons, HTML (Hypertext Markup Language), 20–27
SMACSS (Scalable and Modular Architecture for CSS), 137
smartphones, 166, 430, 431, 432. *See also* mobile media queries
snippets, adding analytics, 627
source-independent positioning, 547–550
spacing
boxes, 224
inline/block elements, 193–199
links, 274
margins, 225, 227, 229
removing, 514
spam, filtering, 619
spans, 62–66
formatting, 456–457
relative spanning columns, 522–524
`span` tag, 62, 64, 76, 78
special tags, 9. *See also* tags
specialty page layouts, 361–363
content filling containers, 363–368
flexbox styles, 375–381
gallery stubs, 386–395
three-column page layouts, 381–386
vertical flex centering, 371–375
specificity, CSS (Cascading Style Sheets), 140–146
sponsored generic TLDs, 595. *See also* TLDs (top-level domains)
SSL (Secure Sockets Layer), 599, 600, 601
stacking
elements, 438–439, 443
fonts, 486
starting
columns, 537–538, 582–583
Jekyll, 254–258
projects, 12–17
rows, 536, 539
static site generators, 109
strings, 9, 10
`strong` tag, 9, 10, 18, 34, 35, 61
strong text, 34, 35
structure. *See also* design
adding blog post, 420–421
columns, 564
of index pages, 402–411
styles, 6
adding, 150, 201, 336–337, 344, 352–353, 361, 391, 503, 520, 578
adding comments, 155–156
animation, 390
applying, 346
`auto-fit`, 510
`background.position`, 338–339
borders, 236–237
circles, 238–244
colors, 148
columns, 405, 406
combining, 138, 139, 140
comments, 373–374
containers, 369
CSS, 121–127, 133
`display: block`, 195–196
`display: flex`, 199
`display: inline`, 196–197
`display: inline-block`, 197–198
`display: none`, 194–195
dropdown menus, 455–463
elements, 167
empty declarations, 171
favicons, 488–490
flexbox, 375–381
fonts, 475–488
footers, 315, 318–325
formatting, 124

gallery stubs, 386–395
groups, 155–156
headers, 280–284, 565
hero, 334, 335
id (*see* id style)
inline blocks, 219–223
inline styling, 48, 73–74 (*see also* inline styling)
links, 318–325
mobile dropdown menus, 463–471
mobile media queries, 430–438
modifying, 307–309
moving link styling, 318
naming, 134–137
overriding, 139
paragraphs, 148
positioning, 291–309
priority and specificity, 140–146
resetting, 186
rules, 88, 145–156
selecting text, 190–191
sizing, 307
`transform`, 302–303
when to use classes/id style, 137–140
width, 426
wrappers, 350
stylesheets, 93, 96–97
`style` tag, 267–268
subdomains, 608. *See also* domains
subgrids, 557–558
switching color names to color values, 158
symbols, 24
syncing index pages, 245–249
syntax, highlighting, 18

T

table data cells, 55, 56
CSS (Cascading Style Sheets), 58
tables, 6, 54–61
block elements, 54, 55–58
defining with headers, 55
formatting, 362–363
headers, 55, 56
inline elements, 59–60
rows, 55
using as layouts, 362–363
`table` tag, 54, 267
tabs, creating, 51
tags
`a`, 61
adding, 61
beginning, 9
`blockquote`, 65, 68, 74–79
`body`, 21, 25, 29, 110
`code`, 52, 53
`color`, 110
`div`, 62, 65, 90
`em`, 33
empty style, 93
ending, 9
first, 17–20
`footer`, 314
formatting, 10–11
`head`, 21
`header`, 62, 64
headers, 30
`html`, 21
HTML, 8–12
`img`, 41, 61, 82
inline styling, 73–74 (*see also* inline styling)
`li`, 66–68
`link`, 97, 271–272
Liquid, 259, 315, 317
`meta`, 24, 41, 452
`span`, 62, 64, 76, 78
`strong`, 9, 10, 18, 34–35, 61
`style`, 267–268
`table`, 54, 267
`title`, 23, 29
`tags.html`, 51
targeting classes, 140
technical sophistication, 4, 32, 33, 59, 106
templates, 251. *See also* layouts
columns, 506–507, 509, 586
content, 327–330
default, 365
GitHub pages, 16
homepages, 330–342
Jekyll, 259–261
page templates (*see* page templates)

support, 253 (*see also* Jekyll)
variables, 329–330
templating systems, 52, 104, 108, 252
terminating loops, 413
text
adding, 39
annotations, 9
bold, 7, 34, 35
colors, 7
emphasized, 32–33
formatting, 6, 8, 31–35, 240
index pages, 31–35
inline styling, 74–79
links, 35–40 (*see also* links)
plain, 9
selecting styles, 190–191
sizing, 175, 176
strong, 34, 35
`text-align` property, 78
text editors, "Hello, world!," 18
three-column layouts, 361–362, 381–386, 445
thumbnails, modifying, 445, 446
titles
callouts, 372
customizing, 490–494
page, 27
`title` tag, 23, 29
TLDs (top-level domains), 594–597
TLS (Transport Layer Security), 599, 600, 601
tokens, personal access, 15
top-level domains. *See* TLDs (top-level domains)
`transform` style, 302–303
transparency, configuring, 161–163
Transport Layer Security. *See* TLS (Transport Layer Security)
Twitter, adding links, 39–40
two-column layouts, 405, 409. *See also* layouts
TXT records, 622, 623, 624
types of documents, 23

U

UI (user interface), 108
Unicode, 24
units, `fr`(functional), 507–510
unordered lists, 68, 219
`unzip` command, 387
updating
class names, 136
headers, 317
index pages, 330–334
uploading HTML (Hypertext Markup Language), 331–333
URLs (uniform resource locators), 16, 36, 256. *See also* links
adding pages, 357–358
forwarding, 615, 616
matching, 615
redirects, 615
sharing, 257
UX (user experience), 108

V

values, 83, 84, 123
CSS, 157
padding, 234
variables
definition of, 329–330
environments, 257
page, 401–402
vector image fonts, 477–483
vendor prefixing, 111
verification, TXT records, 623, 624
vertical flex centering, 371–375
vertical layouts, 445
vertical positioning, 340
`vh` (viewport height), 184–189
viewing
`display: block` style, 195–196
`display: flex` style, 199
`display: inline-block` style, 197–198
`display: inline` style, 196–197
`display: none` style, 194–195
elements, 578
local servers, 451
pixels, 158
screens, 434–438
viewports, mobile, 449–453
`vw` (viewport width), 184–189

W

W3C (World Wide Consortium), 7, 113
warnings, 24
web design, 105, 251. *See also* layouts
web developers, 6, 7
WebKit browsers, 114
web pages, 62, 252. *See also* pages
websites, 16. *See also* page templates; projects
  custom elements of, 104, 105
  domains, 593–594 (*see also* domains)
  fonts, 475–488
  headers, 278–279
  homepages, 330–342
  loading, 19
web sizing, 164. *See also* sizing
width. *See also* sizing
  adding, 299–300
  blogs, 426
  margins, 225, 227, 229
  modifying browser windows, 425
  `vw` (viewport width), 184–189
  windows, 509
  zero-height/zero-width elements, 347, 348
wildcards, 614, 615, 616
windows
  auto-sizing, 509, 518
  modifying browsers, 425
  resizing, 434–438
  sizing, 434
  viewing, 434–438
  width, 509
World Wide Consortium. *See* W3C (World Wide Consortium)
wrappers
  adding, 169, 185, 393
  configuring, 296–297
  styles, 350
wrapping
  content, 339–340, 339–340, 365
  elements, 169
  font names, 487
  homepages, 393
writing. *See also* projects
  blogs (*see* blogs)
  HTML, 10–11
  styles, 133 (*see also* styles)
WYSIWYG (What You See Is What You Get), 9, 397

X

x-axis, 339
XML (Extensible Markup Language), 42

Y

y-axis, 339

Z

zero-height/zero-width elements, 347, 348
`z-index` property, 292
zooming in/out, 164. *See also* pixels