LEARNING WEB DESIGN *Fifth Edition*
INSTRUCTOR'S GUIDE

by Jennifer Niederst Robbins

Learning Web Design, 5e provides a thorough foundation in HTML, CSS, and image production, as well as an introduction to scripting with JavaScript. It requires no previous coding experience, yet by the end of the book, students will be able to create simple web pages according to industry best practices.

It uses a "classroom-in-a-book" approach that starts with the absolute basics, then builds skills gradually. Important concepts such as accessibility and progressive enhancement are introduced early so readers always know "why" things are done in addition to "how."

Simple assignments and assessments are built into the text:

Quizzes
: "Test Yourself" quizzes at the end of every chapter help students test their mastery of important concepts.

Hands-on Exercises
: Exercises throughout the book give students the opportunity to try out key techniques. The source code for all exercises is provided at this location:

  learningwebdesign.com/5e/materials/

NOTE

All the exercises in the book use software that comes with the operating system or is freely available. No purchases are required to follow along with the book.

INSTRUCTOR'S GUIDE CONTENTS

ORGANIZATION

The book is divided into five parts:

Part I: Getting Started
: Before diving into code, the chapters in this section get students oriented with web design, including common roles, equipment, and how the web and web pages work. I recommend introducing the concepts outlined in **Chapter 3**

in class because they provide important context to the detailed lessons in subsequent chapters.

NOTES

Part II: HTML for Structure

The HTML lessons begin in **Chapter 4** with an overview of how a simple page is constructed. This serves as a springboard to learning basic HTML syntax. From there, chapters are topic based. **Chapters 5** through **7** cover markup for text content, links, and images, respectively, and form the minimum training in HTML before moving on to CSS. **Chapters 8** through **10** (Tables, Forms, and Embedded Media, respectively) cover specialized content that could be introduced at a later time or as needed.

Part III: CSS for Presentation

Chapter 11 provides an overview of how Cascading Style Sheets work, including rule syntax, so it is an essential starting point. **Chapters 12** through **16** cover CSS properties organized by topic (text, colors, the box model, floats/positioning, Flexbox and Grid, respectively). **Chapter 17, Responsive Web Design** ties many of these lessons together into a technique for designing web pages that adapt to screen size. **Chapter 18** introduces interactive properties for transitions and animation. **Chapter 19** is a grab bag of common CSS-related techniques that don't quite fit neatly into the prior chapter topics, but are worth knowing. Finally, although not all CSS-related, **Chapter 20** introduces tools that are common to the modern web developer's toolbox.

Part IV: JavaScript for Behavior

The two chapters in **Part IV** give readers a taste of JavaScript and how it is used to manipulate web pages. The goal is to get students familiar with JavaScript syntax so they can recognize what's going on in a script. It is not a complete course in JavaScript or programming, which is beyond the scope of this overview book; however, resources for further learning are provided.

Part V: Web Images

Part V gets away from code to focus on the ins and outs of producing images in a way that is appropriate for the web. It includes descriptions of the various image file types, an image production strategy for responsive web design, tips for image optimization, as well as a whole chapter dedicated to the SVG (Scaleable Vector Graphic) format.

Part VI: Appendices

The Appendices include answers to the chapter quizzes, a table of HTML5 global attributes, a table of all CSS3 and 4 selectors, and a history of HTML leading to HTML5.

ONE POSSIBLE COURSE STRATEGY

Learning Web Design is used as a textbook in a wide variety of courses on web design in departments as diverse as Computer Science, Art and Design, and even

Journalism. Instructors may have different priorities, contexts, and approaches to teaching from the book. What follows is just one possible path through the content for students with no prior experience with code.

NOTES

1. **Get students oriented with the web as a medium.**

   The chapters in **Part I** provide the background information for getting started. In particular, students should understand browser-server interactions, the parts of a URL, and the components of a web page (**Chapter 2**). In addition, a general introduction to the concepts of standards, progressive enhancement, responsive design, accessibility, and performance (**Chapter 3**) will give important context to following lessons.

2. **Focus on the fundamentals of semantic markup with HTML.**

   Before making pages look nice, students should get a good feel for marking up documents in a way that accurately describes the content. **Chapter 4** is an essential introduction to HTML syntax and minimal document structure. From there, cover text markup (**Chapter 5**), links (**Chapter 6**), and the `img` element (the first part of **Chapter 7**).

   In addition to the exercises in the book, you might have students write and mark up their own pages, such as a résumé or a page for an organization they care about (a club, church, sports team, or local business, for example).

NOTE

*The chapters on special elements, table markup (**Chapter 8**), web forms (**Chapter 9**), and embedded media (**Chapter 10**), may be skipped initially to get to CSS sooner. You can swing back to them later.*

3. **Spend some time on image production.**

   When covering image markup, you could make a detour to discuss where to get images (**Chapter 23**), image formats (**Chapter 23**), and web image production techniques (**Chapter 24**). You may also introduce the SVG format (**Chapter 25**) and SVG markup (the second part of **Chapter 7**) at this time.

4. **Introduce basic styling with CSS.**

   With solid markup skills established, you can begin making content look better with CSS. **Chapter 11** provides the essentials needed to get started with CSS syntax. From there, spend time on formatting text (**Chapter 12**), colors and backgrounds (**Chapter 13**), and padding, margins, and borders (**Chapter 14**). These three chapters introduce enough properties to format a presentable 1-column page layout. Have them make their browser narrow to see how it might look on a smartphone.

   In addition, I would introduce the CSS reset technique (**Chapter 20**) at this point. Browser defaults can create unexpected results, and removing them will help beginners see accurate results of the style sheets they write.

5. **Move on to CSS page layout properties.**

   Now students can start moving styled images around on the page, starting simply with floating and positioning (**Chapter 15**). **Chapter 16** introduces the CSS layout tools Flexbox and Grid. Each of these techniques should be covered carefully as they are the primary, modern methods for arranging elements in rows and columns. The code gets a little complex because the

specifications provide so many options, and all of us are new to these techniques. I've done my best to present the specs in as clear a manner as possible.

This is the opportunity for students to transform their 1-column web pages into multi-column layouts.

In addition, because Flexbox and Grid are not universally supported, this would be a good time to discuss vendor prefixes (**Chapter 13**) and providing fallbacks with feature detection (**Chapter 19**).

NOTES

6. **Get some experience with Responsive Web Design.**

   At this point, students should have all the basic skills necessary to move on to creating responsive layouts. Cover all the concepts in **Chapter 17** and see if students can adapt their own pages to make them responsive as I have done for the Black Goose Bakery page in **Exercise 17-1**.

   This is also an opportunity to go back and cover responsive image markup in the third section of **Chapter 7** (the media queries will make more sense now). as well as responsive SVGs at the end of **Chapter 25**.

7. **Get a basic familiarity with JavaScript and the DOM.**

   Chapters 21 introduces the building blocks of JavaScript, so students should be able to generally understand what a script is doing when they read through it. Similarly, **Chapter 22** introduces some of the ways JavaScript uses the DOM to access and change parts of a page.

   Teaching how to write JavaScript is beyond the scope of these two chapters, so if scripting is a major part of your course, you will likely need additional resources. The site *JavaScript.info* has many detailed tutorials.

8. **Try out these techniques and topics as time allows.**

   The following topics are all good to know but are not essential building blocks of web design (in my opinion, anyway), so they may be covered as embelleshments after the basics are mastered.

   — Tables, forms, and embedded media (**Chapters 8**, **9**, and **10**), plus table and form styling tips (**Chapter 19**).

   — Transitions and transforms (**Chapter 18**). The exercises in this chapter are especially fun to play around with.

   — How to write in Sass syntax (**Chapter 20** gets you started and there are tutorials online).

   — Getting your own hosting and domain name (Supplemental article available at *learningwebdesign.com/articles*).

   — Git and GitHub (**Chapter 20**) are useful tools for web professionals. You can cover the command-line interface to Git or use tools that provide a graphical user interface if that is more intuitive for your class.

WHAT'S NEW IN THE FIFTH EDITION

NOTES

If you are familiar with past editions of *Learning Web Design*, you'll find that most of the book will look familiar—there's just a lot more of it!

The following are new or expanded topics in the fifth edition:

- Improving accessibility with ARIA (**Chapter 5, Marking Up Text**).
- An expanded introduction to structured data (**Chapter 5, Marking Up Text**).
- **Chapter 7, Adding Images** has been expanded to include markup options for adding SVG images to the page as well as new markup patterns for responsive images (the `srcset` attribute and the `picture` and `source` elements). Note also that the desription of the `alt` attribute has been updated according to best practices.
- Because HTML5 is now an established standard, the fourth-edition **Chapter 10, What's Up, HTML5?** has been dismantled and redistributed, and XHTML has been de-emphasized in the new edition. The history of HTML has been moved to **Appendix D**.
- HTML5 video, audio, and canvas features are in the new **Chapter 10, Embeded Media**, which also includes `iframe` and `object` elements.
- **Chapter 11, Introducing Cascading Style Sheets** includes new sections on CSS units of measurement and browser developer tools. The section describing the cascade has been rewritten and reorganized for accuracy and clarity.
- Flexbox and Grid are now covered in the all-new **Chapter 16, CSS Layout with Flexbox and Grid**. There is also a supplemental article (PDF) on a third layout technique, multicolumn layout (*learningwebdesign.com/articles*).
- There is now an entire chapter (**Chapter 17**) dedicated to Responsive Web Design.
- CSS feature detection with `@supports` rule and Modernizr is now included in **Chapter 19, More CSS Techniques**.
- **Chapter 20, Modern Web Development Tools** is a new chapter that covers essential tools used by developers, such as command-line tools, CSS pre- and post-processors, build tools, and Git and GitHub. The intent is to give students insight to the way developers work in the real world.
- **Part V, Web Images** has been completely rewritten to reflect current image tools and best practices, including strategies for creating images for responsive layouts. There is a new emphasis on PNG-8, which now is more versatile and has smaller file sizes than the traditional GIF.
- This is the first edition to contain a tutorial on creating favicons, the icons that appear next to the page title in tabs and bookmark lists (**Chapter 23, Web Image Basics**).

NOTES

- Because SVG has become well supported and is well suited to responsive layouts, there is a whole new chapter about this versatile image format (**Chapter 25, SVG**).
- Finally, there is a table of all the global attributes (attributes that can be used with every HTML element) in the new **Appendix B**.

The following topics have been removed from the book due to space concerns and or because the techniques have become outdated.

- The section on border images in **Chapter 14, Thinking Inside the Box** has been moved to a PDF available online at *learningwebdesign.com/articles/.*
- **Chapter 15, Floating and Positioning**, no longer covers using floats and positioning for columned layouts as that technique is no longer recommended.
- **Chapter 16, Page Layout with CSS** from the fourth edition has been removed from the book and adapted to a supplemental PDF article (*learningwebdesign.com/articles/*). It is still relevant only if you need to support Internet Explorer 8 and other old browsers.

USING LWD AS A REFERENCE BOOK

Although the book is written in a manner that is appropriate for step-by-step learning, it is thorough enough to use as a reference book for HTML and CSS.

- Every HTML element, attribute, and CSS property is listed in the index.
- The HTML elements and their respective attributes are listed at the end of the HTML chapter in which they are introduced. Topic-based chapters make the end-of-chapter tables intuitive to find.
- All the HTML5 global attributes are listed in **Appendix B**.
- CSS properties are listed at the end of the CSS chapter in which they appear. Again, topic-based chapters make them easy to get to.
- CSS selectors (which are introduced gradually over several chapters) are provided all in one place in **Appendix C**.

CLASS EQUIPMENT AND SOFTWARE REQUIREMENTS

The following are guidelines for setting up a work environment for your students to follow along with the exercises in the book:

- Students should have a working knowledge of the operating system used in the classroom, including how to open, save, and close files in various applications and in browser windows.

NOTES

- macOS and Windows are supported with examples in the book, so either is fine. Linux should be fine as well, although you may need to find an alternative text editor.
- Code can be written in TextEdit or Notepad. No special code editors are necessary, although if they are available that will make work easier.
- Use an up-to-date browser such as Chrome, Firefox, Safari, Opera, or Microsoft Edge. Some of the CSS properties in the book are not supported by Internet Explorer 9 and earlier, so avoid old versions of IE if possible. Expect browser display to vary somewhat from the figures in the book.
- The image production chapters use Photoshop (free trial version) or GIMP (always freely available).
- The source materials provided for the exercises require 22 MB of space and are available at *learningwebdesign.com/5e/materials*. Students may produce several more MB of files, but in general, the file size for web projects is small.
- I recommend that students have their own copies of the exercise files, either on their own account on a central server or on a portable USB storage drive (thumb drive).

CHAPTER REVIEW

PART I: GETTING STARTED

Chapter 1: Getting Started in Web Design

Summary

This chapter starts with a description of the various disciplines included under the wider umbrella of "web design." The second half of the chapter covers the equipment and software that web designers and developers commonly use.

Takeaways

In **Chapter 1**, students will learn the following:

- The many roles and responsibilities that go into making a site, including content creation and management, various design disciplines, frontend and backend development, as well as other roles.
- That work may be handled by an enormous team or just one person, depending on the scale of the site.
- Not to worry about learning everything at once. They can learn gradually, focusing on the roles that they enjoy and work as part of a team.

- Common web development equipment, including: an up-to-date computer, a second or large monitor, and access to other computer and mobile devices for testing.
- Web design and development software categories: code editors, user interface and layout design programs, image creation/edition tools, and file management and transfer tools.

NOTES

Key terms in Chapter 1

Information Architect (Information Designer)
: Organizes content logically and for ease of findability, including but not limited to search functionality, site diagrams, and how the data is organized on the server.

Content Strategist
: Oversees all the text on the site and ensures that it supports the brand identity and marketing goals of the company. May also be responsible for planning content maintenance and extending the brand voice to social media.

User Experience (UX) Design
: A holistic approach to designing a product so the start-to-finish experience of using it is enjoyable and supports the brand and business goals of the company.

Interaction Design (IxD)
: Makes using the site as easy and delightful to use as possible at every point at which the user interacts with it.

User Interface (UI) design
: Focuses on the functional design of the elements on the page as well as specific tools (buttons, links, menus, and so on) that users need to accomplish tasks.

User-Centered Design
: An approach to design that focuses on users' needs, based on user research and frequent testing.

wireframes
: A simple diagram that shows the structure of a web page.

site diagram
: Indicates the structure of the site as a whole and how individual pages relate to one another.

storyboards
: A diagram that traces a path through a site or app from the point of view of a typical user.

persona
: A fictional user profile, based on user research, that is used as a reference during the design process.

NOTES

visual (graphic) design
: The visual style, or "look and feel," of the page.

frontend development
: Pertaining to aspects of the code that are handled by the browser: HTML, CSS, and JavaScript and DOM scripting.

authoring
: The process of preparing content by marking it up with HTML.

markup language
: A system for identifying and describing various components of a document with tags that describe their function and purpose.

DOM (Document Object Model)
: The standardized list of elements, document objects, and parts of the browser that can be accessed and manipulated using JavaScript or another scripting language.

backend development
: Pertaining to code processing that happens on the server and generally includes server software, web application languages (such as PHP or Ruby), and database software.

full-stack development
: Development that spans both backend and frontend responsibilities.

SEO (Search Engine Optimization)
: A discipline focused on producing site code that increases the chances it will be highly ranked in search results.

FTP (File Transfer Protocol)
: An internet protocol for moving files from one computer to another, such as from a local computer to a remote server.

terminal application
: Software for using a UNIX command-line interface.

Exercises in Chapter 1:

1-1: *Taking stock*
Students are asked to review their own goals, interests, and current skills as well as any equipment or software they might want to have access to.

Chapter 2: How the Web Works

Summary

This chapter is a basic lesson on how the web and web pages work, including some basic terminology used in web design and production. Knowing how things work "under the hood" provides context for understanding why things are done the way they are.

It starts by identifying the web (HTTP) as just one protocol used over the network of computers known as the internet and describes the functions of servers and browsers. There is a detailed discussion of the parts of a URL. We then look at the components of a simple web page, including first introductions to HTML, CSS, JavaScript, and images. The chapter wraps up with a diagram of the interactions between the browser and the server when you type in a URL or tap a link in the browser.

NOTES

NOTE

There is a supplemental article to this chapter, ***Getting Your Pages on the Web****, that describes how to register a domain name and how to find a server to host your site. It is available at* ***learningwebdesign.com/articles/****.*

Takeaways

In **Chapter 2**, students will learn the following:

- That the web uses a protocol called **HyperText Transfer Protocol (HTTP)**. It is just one of many protocols used to transfer information over the internet.
- **Web servers** (running special web software to handle HTTP transactions) deliver documents or data on request.
- Every computer connected to the internet has a unique **IP address**. The **Domain Name System (DNS)** matches domain names to IP numbers.
- A browser is the **client** software that requests documents and data from the server.
- A **URL (Uniform Resource Locator)** describes the location of a file on the network.
- Complete URLs include the protocol (`http://`), the domain name (with optional host name), and the pathname to the resource.
- Some parts of the URL may be filled in on the fly, so typing just a domain name into a browser implies the HTTP protocol and accesses a default document (usually called *index.html*) at the top level of the site directory.
- **HTTPS** is a secure version of HTTP that encrypts data submitted by the user through a web form.
- Web pages are made up of HTML files with (optional) CSS style sheets to affect how they are displayed and (optional) scripts written in JavaScript for interactivity.
- How separate image files are added to the layout by the browser on the fly.
- The series of interactions between the browser and server that happen between clicking/tapping a link and seeing the web page display in your browser.

Key terms in Chapter 2

NOTES

internet
: An international network of connected computers.

web
: One method for transferring data over the internet, using the HTTP protocol.

protocol
: A standardized method for transferring data or documents over a network, for example: FTP for file transfer, and SMTP for email.

server
: A computer running software that allows it to return documents or data on request. A web server is a computer running HTTP server software such as Apache or Microsoft IIS.

open source
: Software that is developed as a collaborative effort with the intent to make its source code and products available for free.

IP address
: A standardized number assigned to every computer and device connected to the internet.

domain name
: A name that is assigned to an IP address to make it easier to reference by humans.

DNS (Domain Name System) server
: A computer that uses a database to match IP addresses to domain names.

client
: A piece of software that requests a document or data from the server. On the web, a browser is the client software.

user agent
: Another word for the browser in a server/client transaction.

server-side
: Pertaining to applications and functions that run on the server computer.

client-side
: Pertaining to applications that run on the user's computer.

intranet
: A web-based network that runs on a private network within an organization.

firewall
: A security device (software) that prevents access to a private network.

rendering engine
: The part of a browser's code that is responsible for converting HTML/CSS/JavaScript into what you see rendered on the screen.

URL (Uniform Resource Locator)
: The address (location) of a file or resource on the web.

index file
: A default file on the server that is returned if no specific filename is provided in the URL.

HTTPS
: A more secure web protocol that encrypts form data submitted by the user when it is sent between the client and the server.

HTML (HyperText Markup Language)
: The markup language used to describe elements on documents that can be connected together with hypertext links.

source document
: The text document containing the content and markup for a web page.

tags
: Text strings between brackets (<>) that identify elements within an HTML document (for example, and to identify a paragraph). Most elements consist of an opening tag and a closing tag on either end of some content.

empty elements
: An HTML element that does not have content, but rather places something in the page flow.

Cascading Style Sheets (CSS)
: The styling language used on the web to affect the presentation of elements.

JavaScript
: The scripting language used on the web to add interactivity and behaviors to page elements.

W3C (World Wide Web Consortium)
: The standards body that develops and maintains all the technologies that are used on the web.

CERN
: The particle physics laboratory in Geneva, Switzerland, where Tim Berners-Lee first proposed a hypertext-based network in 1989.

NCSA
: The creator of the first graphical browser (NCSA Mosaic) that was responsible for boosting the web's popularity.

NOTES

Exercises in Chapter 2

***2-1**: View source*

Students are asked to view the source of various web documents in their browser. It is useful to know that you can view the source of any web page to see how it is constructed. At this point, they can just look around to see if there is anything familiar and marvel at the complexity of larger sites.

Chapter 3: Some Big Concepts You Need to Know

NOTES

Summary

Chapter 3 introduces ideas that form the foundation of modern web design, including dealing with the vast variety of mobile devices, the benefits of web standards, progressive enhancement, Responsive Web Design, accessibility, and performance. They are introduced here to provide context for discussions throughout the book. Students will gain a better understanding of important goals and *why* modern web developers build web pages the way they do.

NOTE

On the topic of web standards, make it clear that by "standards", we mean HTML, CSS, and all the technologies maintained by the W3C. They should be used as intended—for example, using HTML elements to accurately describe content, not to achieve a certain look.

Takeaways

In **Chapter 3**, students will learn the following:

- As web designers, we never know how the pages we create will be viewed (desktop or small screen, slow or fast connection speed, displayed on a screen or read aloud, with or without the fonts and images we specify, with or without JavaScript, and so on).
- A significant percentage of Americans (and more so elsewhere in the world) use their mobile devices as their only access to the internet.
- Using **web standards** (written by the W3C) as they are intended is the best guarantee for consistency and forward compatibility.
- **Progressive enhancement** is a strategy for dealing with unknown browser capabilities. The key to progressive enhancement is starting with a baseline experience (one that meets the content or activity goals of the site) and layering on more advanced features for browsers that support them.
- **Responsive Web Design** (introduced by Ethan Marcotte) is a strategy for dealing with unknown screen size. Responsive web pages use the same HTML source document but swap out styles based on the screen size, enabling the layout and its components to adapt to the screen.
- **M-dot sites** are completely separate sites (usually with reduced content or menu options) that get delivered when the server detects the browser is on a mobile device.
- Users with disabilities (vision, mobility, auditory, and/or cognitive impairments) may access web content with assistive devices, including screen readers and voice interfaces, keyboards, joysticks, foot pedals, magnifiers, Braille output, and more.

- A primary goal of good web production is to make pages load as quickly as possible (referred to as **performance**). Two general approaches are to limit file sizes and reduce the number of requests to the server.

NOTES

Key terms in Chapter 3

desktop computer
: Any computer or laptop with a separate screen and input device such as a mouse or trackpad.

mobile devices
: Smaller devices such as smartphones and tablets.

progressive enhancement
: A strategy for web production that deals with unknown browser capabilities by making sure that content and key interactions are available on all browsing and assistive devices, then adding features for browsers that support them.

graceful degradation
: An approach to web development that designs the fully enhanced experience first, then adds a series of fallbacks for non-supporting browsers.

Responsive Web Design
: A strategy for designing sites that delivers one HTML file and uses CSS to provide appropriate layout to devices based on the screen width.

server-side detection
: Server software and corresponding database that detect the type of device making the HTTP request, then return an appropriate document based on the device and its capabilities.

M-dot site
: A separate site served to mobile devices (using server-side detection) that may strip out certain content and functions based on how the site is used in a mobile context.

accessibility
: Features of a site and its underlying code that make it usable by visitors with assistive browsing and input devices.

WAI (Web Accessibility Initiative)
: The group at the W3C that oversees efforts across web technologies that make the web more accessible.

Section 508
: US Government Accessibility Guidelines that must be adhered to if you are developing a *.gov* website.

performance
: In web design, the speed at which a web page and its resources downloads, displays, and responds to user input.

waterfall chart
: A tool built into major web browsers that reveals all the requests made to the server and how long each one takes, as well as the total download and render time for the page.

NOTES

Exercises in Chapter 3

None.

PART II: HTML FOR STRUCTURE

Chapter 4: Creating a Simple Page (HTML Overview)

Summary

This chapter introduces HTML and its syntax via a series of five steps that create a simple web page. Students can work along with exercises that show the effects HTML tags and a style sheet have on the way the document is rendered in the browser. Revealing effects in small increments shows which parts of the code are responsible for the final product.

Takeaways

In **Chapter 4**, students will learn the following:

- How to configure TextEdit (macOS) and Notepad (Windows) for HTML documents
- A feel for how markup works and how browsers interpret it
- The recommended minimal structure of an HTML5 document
- How to name files correctly
- The importance of a descriptive **title**
- The syntax for **elements**, **empty elements**, and **attributes**
- Why **semantic** markup is important
- **Block** and **inline** elements
- HTML comments
- How images are added to the page
- The effect of a style sheet on the presentation of the page
- Troubleshooting tips for HTML

Key terms in Chapter 4

syntax
: The rules for how code must be structured to function properly.

markup
: The tags added around content that describe the semantic meaning and function of the content.

tag
: Consists of the element name (usually an abbreviation of a longer descriptive name) within angle brackets (< >).

element
: Consists of both the content and its markup (the start and end tags).

empty element
: An element that does not contain content, but provides a directive or embeds an external resource on the page.

attribute
: Instructions that clarify or modify an element.

document type declaration (DOCTYPE)
: Code at the beginning of an HTML document that tells the browser what version of HTML the document is written in.

root element
: The element that contains all the elements and content in the document. In HTML documents, the **`html`** element is the root element.

document head
: The section of the HTML document, defined with the **`head`** element, that contains elements pertaining to the document that are not part of the rendered content.

document body
: The section of the HTML document, defined by the **`body`** element, that contains all the content that displays in the browser.

metadata
: Information about the document, provided with the **`meta`** element in the **`head`** of the document.

character set
: A standardized collection of letters, numbers, and symbols.

coded character set
: A standardized collection of characters with their reference numbers (code points).

character encoding
: The manner in which characters are converted to bytes for use by computers.

Unicode (Universal Character Set)
: A super-character set that contains characters from all active modern languages.

NOTES

HTTP header
: Information about the document that the server sends to the user agent before sending the actual document.

semantic markup
: Marking up a document in a way that provides the most meaningful description of the content and its function.

DOM (Document Object Model)
: An API for accessing and manipulating the elements and attributes in the document based on the document structure.

Boolean attribute
: An attribute that describes a feature that is either on or off. In HTML syntax, Boolean attributes may be written as a single word (such as `checked` for a form `input` element).

valid HTML
: HTML that is written according to all the syntax rules for its declared version of HTML, without errors.

validator tools
: Software that checks an HTML source against the specification to be sure it is valid and error-free.

NOTES

Exercises in Chapter 4

4-1: *Entering content*
Students type in several paragraphs of text and open the document in the browser without any HTML markup. In the resulting page, everything runs together because line breaks are ignored.

4-2: *Adding minimal structure*
Minimal structure markup is added to the sample (`DOCTYPE`, `html`, `head`, `title`, `meta`, `body`). When viewed in the browser, the only difference is the title is added in the browser tab. Content is unaffected.

4-3: *Defining text elements*
Markup identifying basic text elements (`h1`, `h2`, `p`, and `em`) is added to the sample document. When displayed in the browser, the headings are now distinct from the paragraphs due to the browser's built-in default style sheet.

4-4: *Adding an image*
A logo image is added to the page with the `img` element.

4-5: *Adding a style sheet*
Students add a few style rules in a `style` element in the `head` of the document. Viewing the page in the browser shows how its presentation has changed. This is the first introduction to style rules in the book. These particular styles were chosen because they are intuitive even if you don't know CSS. It is easy to compare the rules to the results.

Chapter 5: Marking Up Text

NOTES

Summary

With the fundamentals of markup established, this chapter introduces all the HTML elements available for formatting various types of text content (both block and inline), with an emphasis on using markup in a meaningful and semantic way. The chapter closes with a section on how to "escape" special characters.

Takeaways

In **Chapter 5**, students will learn the following:

- Elements for basic text components, such as paragraphs (`p`) and headings (`h1–h6`)
- Elements for inserting breaks (`hr`, `br`, `wbr`)
- Elements for marking up unordered (`ul`, `li`), ordered (`ol`, `li`), and description (`dl`, `dt`, `dd`) lists
- Other content elements (`blockquote`, `pre`, `figure`, `figcaption`)
- Elements used to organize a page (`main`, `section`, `article`, `nav`, `aside`, `header`, `footer`, `address`)
- Inline (phrase) elements (`abbr`, `b`, `cite`, `code`, `data`, `del`, `dfn`, `em`, `i`, `ins`, `kbd`, `mark`, `q`, `s`, `samp`, `small`, `strong`, `sub`, `sup`, `time`, `u`, `var`)
- How to properly nest elements
- Generic elements (`div` for block elements; `span` for inline elements)
- The `id` attribute for identifying unique elements
- The `class` attribute for classifying elements as belonging to a group
- How **ARIA roles** improve accessibility
- How and why to escape special characters

Key terms in Chapter 5

document outline
: The outline created by the heading order within a document.

thematic break
: The point at which one topic has completed and another one is beginning. It can be indicated with the `hr` element (originally defined as a "horizontal rule").

unordered list
: A collection of items that appear in no particular order.

NOTES

ordered list
: A list in which the sequence of the items is important.

definition list
: A lisit that contains name and value pairs, including but not limited to terms and definitions.

nesting
: The containment of one element completely inside another element.

monospace font
: A font in which all the characters have the same width.

text-level semantic elements
: What the HTML5 spec calls elements that appear within the flow of text without introducing line breaks (previously called *inline elements*; also referred to as *phrasing content*).

structured data
: Standards that allow content to be machine-readable, enabling it to be used by computer programs (like a calendar app) and search engines. Microformats, Microdata, RDFa, and JSON-LD are examples of structured data standards.

global attributes
: Atttributes that can be used with any HTML element (for example, **`id`** and **`class`**).

ARIA roles
: The **`role`** attribute describes an element's function or purpose in the context of the document to improve accessibility—for example, **`role="alert"`** or **`role="menubar"`**.

escaped character
: A character represented by its Unicode number or a predefined name. For example, the < character must be escaped (represented as **`<`** or **`&#060;`** in the source so it is not mistaken for the beginning of a tag.

character entity reference
: A name or a number assigned to a character that is referenced when that character is escaped.

named character entity
: A predefined abbreviated name for the character— for example, **`&mdash;`** for an em dash (—).

numeric character entity
: An assigned numeric value that corresponds to the character's position in a coded character set such as Unicode.

Exercises in Chapter 5

NOTES

5-1: *Marking up a recipe*
Students add markup for paragraphs, headings, lists, and a blockquote to a sample recipe. Tags can be written into the provided source document or written right on the page.

5-2: *Identifying inline elements*
In this exercise, inline elements are added to a selection of HTML source code. The challenge is to find examples of `b`, `br`, `cite`, `dfn`, `em`, `i`, `q`, `small`, and `time` in the text content.

5-3: *Putting it all together*
This exercise is an opportunity to try out elements from the entire chapter. The content is provided, and readers are walked through the markup step by step.

Chapter 6: Adding Links

Summary

Chapter 6 introduces the anchor (`a`) element for adding links to text. It first looks at external links that use complete URLs, followed by relative links that use only a pathname. UNIX pathname syntax is introduced over several steps, making up the bulk of the chapter. Linking within a page (to fragments), targeting new browser windows, and mail and telephone links are also covered.

Takeaways

In **Chapter 6**, students will learn the following:

- Making a link using the **anchor** (`a`) element
- Linking to external web pages with a complete URL
- Linking to files on the same server using relative pathnames in UNIX syntax (within a single directory, down into subdirectories, and back up to higher directory levels)
- Linking to **fragments** within a web page
- Targeting new browser windows
- mailto and telephone links

Key terms in Chapter 6

anchor
: The HTML element (`a`) that creates a hyperlink

absolute URL
: A complete URL that includes the protocol and domain name in addition to the path to the resource

relative URL
: The location of a resource on the same server identified relative to the location of the current document

external link
: A link to a document or resource on a server other than the current server

pathname
: The notation used to point to a particular file or directory in which directory levels are separated by slashes (/)

root directory
: The directory that contains all of the files for the site

site root relative link
: A relative URL beginning with a slash (/) that is always relative to the root directory for the site

fragment
: A part of a web page

fragment identifier
: An `id` value given to an element that assigns a name to the fragment so it can be referenced by a link

mailto link
: A link that opens a preaddressed new mail message in the browser's designated email program

NOTES

Exercises in Chapter 6

6-1: *Make an external link*
We start out creating a basic link to an external web page.

6-2: *Link in the same directory*
This is the first of several exercises that get readers familiar with relative links. We start simply by linking to a file that is in the same directory as the current file.

6-3: *Link to a file in a subdirectory*
The next step is linking to a file that is within a subdirectory by including the directory name in the path.

6-4: *Link two directories down*
Building on the previous exercise, now we link to a file in a subdirectory within the subdirectory.

6-5: *Link to a higher directory*
This is a chance to try out the `../` convention for backing up a directory level.

6-6: *Link up two directory levels*
Here we link to a file that is two levels up in the hierarchy using `../../`

6-7: *Try a few more*

Students are given five more opportunities to try out relative pathnames based on the sample directory structure.

6-8: *Linking to a fragment*
In this exercise, students identify fragments in a long glossary with the `id` attribute and make letters at the top of the page link to them.

NOTES

Chapter 7: Adding Images

Summary

Chapter 7 focuses on embedding images in the content of the page (see **Note**). It is made up of three parts: basic image embedding with the `img` element, ways to embed SVG images, and responsive image techniques. The `img` element discussion is a necessary component of basic web design training, and SVG options should be addressed as well, given their rising popularity.

I recommend, however, that responsive images (methods for specifying a number of image options for use in responsive layouts) be saved for later in the course because the code is advanced and not strictly required. A logical point to swing back to responsive image markup is when discussing Responsive Web Design (**Chapter 17**).

NOTE

It is also possible to add images to a web page as a background image with CSS, which is covered in **Chapter 13, Colors and Backgrounds**.

Takeaways

In **Chapter 7**, students will learn the following:

Basic images

- The difference between raster and vector images.
- Images must be saved in JPEG, PNG, GIF, or SVG formats to be appropriate for web pages. Cutting-edge formats WebP and JPEG-XL are gaining in browser support.
- The `img` element and its attributes.
- The purpose and importance of alternative text (using the `alt` attribute).

SVG images

- Basic familiarity with the SVG image format as an XML text document.
- Three methods for adding an SVG to a page: `img` element, `object` element, and inline with the `svg` element.

Responsive Image Techniques

- **Responsive images** are a method for providing images tailored to the user's viewing environment without downloading more image data than necessary.

TIP

While talking about adding images to a web page, it might be a good time to address basic JPEG and PNG production and optimization, covered in **Chapters 23** *and* **24**. *SVG format is covered in* **Chapter 25**, *including a responsive SVG section that can be part of the Responsive Web Design discussion.*

NOTES

- The four categories of responsive image techniques: image options for high-resolution displays, options for multiple screen sizes, differently cropped versions, and alternative image formats.
- Use the `srcset` attribute in the `img` element with an **x-descriptor** to provide image versions targeted to various screen resolutions.
- Use the `source` element with `srcset` attribute, **w-descriptors**, and the `sizes` attribute to provide image versions to be used at a variety of screen sizes.
- Use the `picture` element to provide alternately cropped images based on screen size (the art direction scenario).
- Use the `picture` element to provide alternate image formats, delivering smaller cutting-edge image formats to browsers that support them.

Key terms in Chapter 7

bitmapped image (also called raster image)
: Images that are made up of a grid of tiny colored squares (pixels).

vector image
: An image made up of mathematically defined paths.

MIME type
: A resource's standardized media type, consisting of a general type (such as "image" or "audio"), a specific media type (such as JPEG or MP3), and a file suffix (*.jpeg* or *.mp3*).

replaced element
: An element that is replaced by an external file when the page is displayed.

disk cache
: Space used by browsers for temporarily storing files on the hard disk.

alternative text
: A text alternative to an image for those who are not able to see it.

standalone SVG
: An SVG document that is saved in its own file with the *.svg* suffix.

inline SVG
: An SVG that is embedded into an HTML document with the `svg` root element.

pixel
: A square of colored light used by displays to render images.

device pixels (also hardware pixels, physical pixels)
: The pixels that make up the screen display on a device.

resolution (ppi)
: A measurement of the number of pixels per inch (ppi).

NOTES

reference pixel (CSS pixels)
: A unit of measurement used by devices for layout purposes. They may or may not match up with the device/hardware pixels.

device-pixel-ratio
: The ratio of the number of device/hardware pixels to reference pixels. For example, on a 2x display, there would be two device pixels making up the width of one reference pixel.

x-descriptor
: Specifies the target device-pixel-ratio for a `srcset` attribute value in the `img` element. They are useful for telling the browser to choose the best image option based on screen resolution.

w-descriptor
: Provides the actual pixel width of the image with the `srcset` attribute. It is part of the system for providing viewport-based image selection.

viewport
: The canvas the web page is rendered on, usually corresponding to the size of the screen or browser window.

browser preloader
: A function in the browser that fetches images and other external resources from the browser so they are ready to go when the page displays.

media condition
: Part of a media query that describes a parameter to test for, such as the width of the viewport.

Exercises in Chapter 7

7-1: *Adding and linking images*

This exercise provides some practice for adding images to pages using the `img` element. Thumbnail images are added to a gallery page, then used as links to pages with larger versions of the images. The book lists steps for adding one image and linking it, and there are materials provided for readers to try it five more times.

7-2: *Adding an SVG to a page*

The exercise starts by making the logo in PNG format very large to see what happens to the image quality. Then it is replaced with an SVG version of the logo, and resized again to see that the quality stays the same. The second part of the exercise adds a row of social icons in SVG format and styles them with CSS rules.

7-3: *Adding responsive images*

Students are walked through the steps for adding responsive images for various screen widths using the `source` element.

Chapter 8: Table Markup

NOTES

Summary

This straightforward chapter covers the markup for structuring "tabular material" (tables). Because tables are a specialized type of content, it is okay to skip them if you are eager to get started with CSS. You can always come back and add tabular material later.

Takeaways

In **Chapter 8**, students will learn the following:

- That table markup should be used for tabular material (content arranged into rows and columns).
- Elements for basic table structure (`table`, `tr`, `th`, `td`).
- The role of **table headers**.
- How to span cells with `colspan` and `rowspan` attributes.
- Table accessibility features such as the `caption` element and the `scope` attribute for `th` elements.
- Adding a semantic layer to rows with the **row group elements** (`thead`, `tbody`, and `tfoot`).
- Adding a semantic layer to columns with the **column group elements** (`colgroup` and `col`).

Key terms in Chapter 8

tabular material
: Data arranged into rows and columns (tables)

table headers
: Cells that provide important information or context about the cells in the row or column they precede (`th`)

table data cells
: Cells that contain the main content of the table (`td`)

spanning
: Stretching a cell to cover several rows or columns, allowing more complex table structures

Exercises in Chapter 8

8-1: *Making a simple table*
This exercise asks students to write the HTML source for a simple table with two columns and six rows.

8-2: *Column spans*
This provides a chance to try the `colspan` attribute to span columns.

NOTES

8-3: *Row spans*
This time `rowspan` is used to span rows.

8-4: *The table challenge*
Students can put all of the previous table lessons together to recreate a complex table structure shown in the figure.

Chapter 9: Forms

Summary

This chapter starts with an introduction to how forms work: data is collected via controls on the page and an application or script on the server processes it. The bulk of the chapter consists of a rundown of the markup for various form control widgets. Throughout the chapter, there are opportunities to try them all out by marking up a pizza ordering form. The chapter closes with form accessibility features.

Takeaways

In **Chapter 9**, students will learn the following:

- Forms have two components: the form on the web page for collecting input and a program on the server that processes the collected information.
- The steps involved in a form submission.
- The `form` element and its attributes.
- What a **variable** is and how to identify it with the `name` attribute.
- What the form control content is and how it can be provided with the `value` attribute or included in the form content itself.
- Markup for file controls:
  - Single-line text-entry controls using the `input` element (simple text field and specialized fields such as password, email, telephone numbers, search, and URLs)
  - Multiline text-entry field (`textarea`)
  - Buttons, including submit and reset, and custom buttons: (`<input type=submit>`, `<input type=reset>`, `<input type=image>`, `<input type=button>`, and `<button>`)
  - Radio and checkbox buttons (and how they differ) (`<input type=radio>`, `<input type=checkbox>`)
  - Pop-up and scrolling menus (`select`, `option`, `optgroup`)
  - A file upload control (`<input type=file>`)
  - A hidden file control (`<input type=hidden>`)

NOTES

— Date and time controls (`input` element with `date`, `time`, `datetime-local`, `month`, and `week` types)

— Numerical inputs (`input` element with `number` and `range` types)

— Color selector (`<input type=color>`)

- How to make forms more accessible using `label`, `fieldset`, and `legend`.
- A few tips for improving the usability of web forms as recommended by Luke Wroblewski in his book *Web Form Design* (Rosenfeld Media).

Key terms in Chapter 9

form controls
: The buttons, input fields, and menus on the web page that collect information from the user.

variable
: A bit of information collected by a form, such as an email address or a date.

value
: The data associated with a given variable. It may be entered by the user via a form control or added by the author in the source with the `value` attribute.

radio buttons
: A form control made up of a collection of on/off buttons that is appropriate when only one option from the group is permitted (only one button may be selected at a time).

checkbox buttons
: A form control made up of a collection of on/off buttons that is appropriate when more than one selection is OK (multiple checkboxes in a group may be chosen).

form accessibility
: Markup added to forms that make them easier to understand and navigate with assistive devices.

implicitly associated labels
: A form control and its brief description nested within a single `label` element.

explicitly associated labels
: Matches the `label` element with its form control using the control's ID reference.

Exercises in Chapter 9

9-1: *Starting the pizza order form*
Using a "sketch" of the finished form, students start building a form for ordering a pizza. In this first step, they add the `form` element with text-entry controls and a submit button. The form points to a working PHP file so a thank-you message is returned when the form is submitted.

NOTES

9-2: *Adding radio buttons and checkboxes*
As it says in the title, radio buttons and checkboxes are added to the pizza ordering form.

9-3: *Adding a menu*
A pop-up menu is added to the form.

9-4: *Labels and fieldsets*
The pizza-ordering form is made accessible according to best practices with `label`, `fieldset`, and `legend` markup.

Chapter 10: Embedded Media

Summary

`Chapter 10` reviews the various types of media that can be embedded in a web page, including iframes, a multipurpose `object` element, video players, audio players, and canvas (a 2D raster drawing space that can be used for games and other interactive features).

Takeaways

In `Chapter 10`, students will learn the following:

- How to create a nested browser window for displaying an external web page using the `iframe` element.
- How to use the `object` and `param` elements to embed a variety of media types.
- An introduction to the various codecs and containers used by competing video and audio formats, including the best supported options.
- How to add a video to the page using the `video` element, along with fallback options.
- How to add an audio player to a page using the `audio` element, with fallbacks.
- Using the `track` element to attach synchronized text to audio and video (for example, subtitles, captions, descriptions, chapter titles, and other metadata).
- A brief introduction to the `canvas` element for creating drawings and interactive games and features on a web page using JavaScript.

Key terms in Chapter 10

embedded content
: Content that imports another resource into the document, or content from another vocabulary that is inserted into the document.

nested browsing context
: A viewport (browser window) that is embedded in another viewport.

encoding
: The algorithm used to convert a media source to 1s and 0s and how that data is compressed.

container format
: The file that packages the compressed media and its metadata.

HTML5 canvas
: An API for embedding a 2D drawing space on a web page that uses JavaScript functions for creating lines, shapes, fills animations, interactivity and so on. It is often used for interactive features and games.

NOTES

Exercises in Chapter 10

10-1: *Embedding a video with iframe*
In this exercise, students create an iframe on the page for embedding a YouTube video.

10-2: *Embedding a video player*
Two video files (provided) are added to a web page with the `video` element.

PART III: CSS FOR PRESENTATION

Chapter 11: Introducing Cascading Style Sheets

Summary

This chapter kicks off the Cascading Style Sheets (CSS) section of the book by introducing the basics of CSS. It is chock-full of essential information, including rule syntax, how styles are attached to the HTML document, and foundational concepts such as inheritance, the cascade (priority, specificity, and rule order), the box model, as well as the CSS units of measurement. Element selectors and grouped selectors are also introduced in this chapter.

Takeaways

In **Chapter 11**, students will learn the following:

- The benefits of using CSS for controlling **presentation**.
- The basics of how style sheets work in tandem with HTML markup.
- The parts of a style rule (**selector**, **declaration**, **property**, and **value**).
- The three methods for attaching style rules to HTML:
  — **external** style sheets (`link` and `@import`)
  — **embedded** style sheets (the `style` element in the `head` of the document)
  — **inline** styles (the `style` attribute applied directly to the element).

- How to write comments in style sheets (`/* … */`).
- How some elements **inherit** certain properties from the element in which they are contained (their parents).
- How the **cascade** determines which style rule applies when conflicting styles are applied to an element.
- The **priority** of style rule sources, from lowest to highest (browser default style sheet, user style sheet, author style sheet, styles marked `!important` by the author, styles marked `!important` by the user).
- How the specificity of the selector is used to settle conflicts in rules applying to a given element. More specific rules have more "weight" and override less specific rules.
- How the cascade uses **rule order** to determine which style rule wins when they have equal weight. The style listed last will be applied.
- The **box model** is part of the visual formatting system that assigns an implied rectangular box around all elements.
- A basic familiarity with the units of measurement in CSS, including the difference between **absolute** and **relative units**.
- Knowledge of extremely useful developer tools that are built into major browsers.

NOTES

Key terms in Chapter 11

presentation (also presentation layer)
: How the document is delivered to the user, whether rendered on a computer or mobile device screen, printed on paper, or read aloud by a screen reader.

style rule
: Instructions for how certain elements should be displayed.

selector
: The part of a style rule that identifies the element or elements to be affected.

declaration
: The part of a style rule with the rendering instructions. It is made up of one or more properties and values.

declaration block
: The curly brackets in a style rule and all the declarations they contain.

property
: A characteristic of the element, such as size, color, thickness, and so on.

value
: The specific setting for a property.

element type selector
: Selects all the elements in a document of a given element type.

grouped selector
: Provides a list of elements to be selected, separated by commas.

inline style
: A style rule added with the `style` attribute right in the opening tag of the element it is affecting.

inheritance (in CSS)
: Certain style properties are passed down from elements to the elements they contain (their descendents).

descendents
: All elements contained within a given element.

child
: The element(s) contained directly within a given element (with no intervening hierarchical levels).

parent
: The element that directly contains a given element.

ancestors
: All the elements higher than a particular element in the hierarchy.

siblings
: Two elements that share the same parent.

cascade
: A system for determining which style rule applies when there are conflicting rules applied to the same element.

user agent style sheet
: The default style sheet that comes built into the browser (**user agent**).

user style sheet
: Style rules created in the browser by the user.

author style sheet
: Styles created by the developer of the website.

specificity
: The concept that some selectors have more "weight" and therefore override rules with less specific selectors.

rule order
: In the cascade, if there are conflicting style rules of equal weight, whichever rule comes last will apply.

box model
: In CSS, all elements are contained in an implied rectangular box to which you can apply backgrounds, borders, padding, and margins.

grouped selectors
: Selectors combined in a comma-separated list so you can apply the same properties to them all.

NOTES

absolute units
: Units of measurement that have predefined meanings in the real world, such as inches or picas.

relative units
: Units that are based on the size of something else, such as ems and viewport units.

NOTES

Exercises in Chapter 11

***11-1**: A first look*
Readers will have the opportunity to create a simple style sheet in exercises throughout this chapter. In this first exercise, they simply take a look at the unstyled document in the browser to see the starting point.

***11-2:** Your first style sheet*
This basic introduction to writing style rules includes rules for changing the appearance of `h1` and `h2` headings, paragraphs, and image position.

***11-3:** Applying an inline style*
This exercise shows how adding an inline style overrides the styles applied with the `style` element.

Chapter 12: Formatting Text

Summary

This dense chapter covers the properties related to manipulating the appearance of text, including fonts, text color, text line alignment, decorations (like underlines) and list styles. In addition, we'll add Descendent, ID, and Class Name selectors to our selector toolkit, and take a more detailed look at specificity.

Takeaways

In `Chapter 12`, students will learn the following:

- The properties related to specifying the font for a text element, primarily `font-family`, `font-size`, `font-weight`, `font-style`, `font-variant`, and the shorthand `font` property.
- That fonts must be installed on the user's machine or downloaded as a web font in order to be used. You can specify a list of preferred fonts, ending with a generic font family, to be used as backups.
- The most widely used units for text size are **rem** and **em**. Pixels are also used, but are less flexible.
- An introduction to some of the advanced font features introduced in the CSS Font Module Level 3.
- Changing text color with the `color` property.

NOTES

- New selector types: descendant selectors (targeting elements only when they are contained within certain elements), ID, class, and universal selectors.
- How to calculate specificity by totaling up the number of IDs, classes, and elements in the selector. More specific selectors will override less specific selectors when there are conflicting styles applied to an element.
- Properties that affect whole lines of text, including line height (`line-height`), indents (`text-indent`), horizontal alignment (`text-align`).
- Other text properties such as underlining (`text-decoration`), changing capitalization (`text-transform`), letter spacing (`letter-spacing`), word spacing (`word-spacing`), and adding shadows (`text-shadow`).
- Properties for changing the presentation of bulleted and numbered lists (`list-style-type`, `list-style-position`, `list-style-image`).

Key terms in Chapter 12

typeface
: A particular design of glyphs (characters) that share common design features, for example Baskerville. Also known as a **font family**.

font
: One particular instance of a typeface that has a specific weight, style, condensation, slant, and so on. For example, Baskerville Bold Italic is one font of the typeface Baskerville. On computers, fonts are usually stored in different files.

rem
: Root em unit; it is equivalent to the font size of the `html` element.

em
: Em unit; it is equivalent to the font size of the current element.

posture
: Whether the font is vertical or slanted (italic or obilique).

contextual selector
: Selects elements based on their relationship to other elements (includes descendant, child, next-sibling, and subsequent-sibling selectors).

id selector
: Selects an element by the value of its `id` attribute (indicated by the `#` symbol).

class selector
: Selects an element or elements based on the value of their class attributes (indicated by the period [.] symbol).

universal selector
: Selects any element, like a wildcard in programming languages.

Exercises in Chapter 12

NOTES

12-1: *Formatting a menu*
A menu for the Black Goose Bistro will be embellished with text styles throughout the chapter as new properties are introduced. In this first exercise, all the text is set to the web-safe Verdana font, and the main heading is specified with a web font called Marko One.

12-2: *Setting font size*
In this step, the font size of the body text and headings is refined.

12-3: *Making text bold and italic*
This quick exercise formats list terms in bold and strong text in italic styles.

12-4: *Using the shorthand font property*
Here we merely convert a stack of font declarations into one declaration using the shorthand font property.

12-5: *Using selectors*
This exercise gives students a chance to try out descendent, ID, and class selectors to target specific elements in the bistro menu.

12-6: *Finishing touches*
Several more styles are added to the menu to practice using color, line-height, alignment, capitalization, shadow, as well as some new selector types.

Chapter 13: Colors and Backgrounds
(plus more selectors and external style sheets)

Summary

The primary purpose of this chapter is to introduce color and background properties; however, it also increases the number of selectors in the readers' toolkits and introduces external style sheets.

Takeaways

In **Chapter 13**, students will learn the following:

- How to use the **color** and **background-color** properties with color names, RGB values, and HSL values
- A thorough understanding of how the RGB color system works, and how that translates into the CSS syntax for specifying color: **rgb(#,#,#)**, **rgb(%, %, %)**
- How to specify RGB values with hexadecimal values (**#RRGGBB**)
- Specifying HSL color values: **hsl(#, % %)**
- Adding transparency to RGB and HSL values by including a fourth alpha value: **rgba(#,#,#,#)** and **hsla(#,%,%,#)**
- Using the **opacity** property to apply a level of transparency to an element

NOTES

- New selectors:
  - — Pseudo-classes for changing the appearance of elements based on user actions (`:link`, `:visited`, `:focus`, `:hover`, `:active`)
  - — Pseudo-elements that target elements not explicitly contained or marked up in the HTML source (`::first-line`, `::first-letter`, `::before`, `::after`)
  - — Attribute selectors that target elements based on their attribute names or values
- How to add tiling images to the background of an element with `background-image`, `background-repeat`, `background-position`, `background-origin`, `background-attachment`, and `background-size`
- Specifying background colors and all tiling image properties with the shorthand `background` property
- Adding multiple background images to a single element
- Using linear and radial gradients in element backgrounds
- Why vendor prefixes are required for certain CSS properties
- Two methods for adding external style sheets (text-only documents saved with the *.css* suffix): the `link` element in the `head` of the document and the `@import` rule in a style sheet

Key terms in Chapter 13

RGB color model (also called Truecolor)
: A color model that mixes colors from Red, Green, and Blue light. With 256 shades of light in each channel, this color model is capable of creating millions of colors.

HSL color model (Hue Saturation Luminosity)
: A color model that specifies colors by their Hue (in degrees around a circle of colors), Saturation (a percentage value), and Luminosity (a percentage value).

hexadecimal
: A numbering system that uses 16 digits: 0–9 and A–F (for representing the quantities 10–15). It is used in computing because it reduces the space it takes to store certain information.

alpha channel
: A fourth channel in image formats that contains transparency information.

RGBa
: A color model that specifies an RGB color with a transparency level. The alpha setting is specified on a scale from 0 (fully transparent) to 1 (fully opaque).

HSLa
: A color model that specifies an HSL color with a transparency level. The alpha setting is specified on a scale from 0 (fully transparent) to 1 (fully opaque).

foreground (of an element)
: Consists of the element's text and border.

background canvas
: The area in an element box that includes the content area and padding, extending behind the border to the margin edge.

background painting area
: The area in which fill colors are applied.

pseudo-class selector
: An implied class applied to elements in the same state (such as hover).

pseudo-element selector
: Selectors that insert implied or fictional elements into the document for styling.

generated content
: Content that browsers insert on the fly, such as list markers or content added with `::before` and `::after` pseudo-element selectors.

attribute selector
: Targets elements based on attribute names or values.

origin image
: The first image placed in the background from which tiling images extend.

attached background image
: A background image that stays in a fixed position relative to the browser window.

linear gradient
: A transition from one color to another along a straight line.

radial gradient
: A transition from one color to another that starts at a point and spreads out in a circular or elliptical shape.

color stop
: A point along a gradient line where pure color is positioned.

vendor prefix
: A browser-specific prefix added to a property to indicate that it is in an experimental mode (example: `-webkit-gradient`).

media queries
: A method for applying styles based on the medium used to display the document.

NOTES

Exercises in Chapter 13

NOTES

***13-1**: Adding color to a document*
Colors are added to various elements in a black and white menu to give it a more pleasing color palette.

***13-2**: Adding a tiling background image*
A background image is added to the background of the whole page when it is applied to the `body` element.

***13-3**: Controlling tile direction*
The `background-repeat` property is used to control the repeat pattern of images in the background of the page and in a header element.

***13-4:** Positioning background images*
Here students get a chance to play around with the position of several background images.

***13-5:** Fixed position*
A quick edit makes a single background image stay fixed in the browser window.

***13-6:** Convert to shorthand property*
All the background properties we've been applying are changed to a single background shorthand property.

***13-7:** Multiple background images*
This is an opportunity to put more than one image in the background of the header element.

***13-8:** Making an external style sheet*
All the styles in the `style` element get moved to an external style sheet document that is linked first with an `@import` rule and then with the `link` element.

Chapter 14: Thinking Inside the Box

Summary

`Chapter 14` covers all the box-related properties from the inside out: content dimensions, padding, borders, and margins. In addition, it looks at outlines, display roles, and drop shadows. Understanding the parts of the element box is an important first step to using CSS for page layout. Element widths and margins play a particularly large role in layout, so be sure they are mastered before moving on.

NOTE

There is a supplemental article to this chapter, **"Border Images,"** *that describes a technique for applying images around the edges of an element. It is available at* ***learningwebdesign.com/articles/****.*

Takeaways

NOTES

In **Chapter 14**, students will learn the following:

- The components of an element box: content area, inner edge, padding area, border, margin area, and outer edge.
- How to specify box dimensions (`width` and `height` properties).
- The difference between the content-box and border-box models for specifying element dimensions.
- Using the `overflow` values to specify what happens when content doesn't fit inside its box.
- Using the padding properties to specify an amount of space between the content and the border or inner edge.
- Using the border properties to add a border on one or more sides of an element.
- The convention for providing values for a shorthand property in a clockwise direction starting at the top: top, right, bottom, left (the mnemonic "TRouBLe" can help you remember).
- Making corners curved with the `border-radius` property.
- Adding margins on the outside of an element with the collection of margin properties.
- How neighboring top and bottom borders collapse instead of accumulating. Only the largest margin value is applied.
- How to assign a display type to elements to affect how they participate in the page layout.
- Adding soft drop shadows under element boxes.

Key terms in Chapter 14

box model
: A system in which every element in a document generates a rectangular box to which properties such as width, height, padding, borders, and margins can be applied.

content area
: The space that contains the content of the element.

inner edge
: The edges of the content area.

padding
: The area between the content area and an optional border.

border
: A line (or stylized line) that surrounds the element.

margin
: An optional amount of space added on the outside of the border.

outer edge
: The outside edges of the margin area form the outside edge of the element box.

visible element box
: The content, padding, and border (if there is one).

replaced elements
: HTML elements that get replaced by other external resources (such as an image).

non-replaced elements
: Elements that appear in the HTML source (like text).

content-box model
: A method for sizing an element by applying **`width`** and **`height`** properties to the content box only. Padding and border dimensions are additional.

border-box model
: A method for sizing an element that applies **`width`** and **`height`** properties to the visible box (including the padding and border).

outline
: An outline is like a border, but it is not calculated in the width of the element box, but rather lays on top of the rendered element.

collapsing margins
: Top and bottom margins overlap, and only the larger margin height is used (i.e., they are not additive).

display types
: Defines the type of element box the element generates and how it participates in the page layout, for example: inline, block, table, hidden.

Exercises in Chapter 14

14-1: *Adding a little padding*
This is the first of a series of exercises that make improvements to the Black Goose Bakery page using box model properties. In this exercise, dimensions and padding are added to various elements on the page. It also provides an opportunity to work with an external style sheet.

14-2: *Border tricks*
This exercise lets readers try out borders and border-radius properties on the bakery page.

14-3: *Adding margin space around elements*
Margins are adjusted around individual elements on the bakery page as well as around the edges of the page itself.

NOTES

NOTE

The bakery page will undergo additional improvements in **Chapters 15**, **16**, *and* **17**.

Chapter 15: Floating and Positioning

Summary

This is the first chapter that addresses moving element boxes around to break out of the normal document flow. Floating moves an element to the left or right and allows following text to flow around it. Positioning lifts up the element and places it in another position.

Takeaways

In **Chapter 15**, students will learn the following:

- The characteristics of the **normal flow**: **block** elements stack up and fill the available width of the window or other containing element, and **inline** elements line up next to one another to fill the width of block elements.
- Using the `float` property to shift an element to the left or right and allow text to flow around it.
- The behavior of floated inline and block elements.
- Ensuring an element begins below a floated element (i.e., stops wrapping) with the `clear` property.
- What happens when multiple elements are floated on the same page or within one element.
- How to use the `shape-outside` property to give text wraps more interesting shapes (circular, elliptical, along a path, or using an image).
- The four primary types of positioning:
  - **static** (the default)
  - **relative** (to its original position)
  - **absolute** (positioned with coordinates)
  - **fixed** (stays in one position in the viewport)
- How elements are positioned relative to their **containing blocks**. If they are not contained within another positioned element, then the `html` element forms the initial containing block.
- Specifying position with `top`, `right`, `bottom`, and `left` properties.
- Controlling how positioned elements stack up and overlap using the `z-index` property.

Key terms in Chapter 15

floating
: Moves an element as far as possible to the left or right and allows the following content to flow around it.

NOTES

NOTE

Floating and positioning, until recently, were the primary tools for achieving columned page layouts; however, that approach is obsolete now that better layout tools (Flexbox and Grid) are available. Float- and position-based layouts may still be used as a fallback for browsers that do not support the newer methods (mainly Internet Explorer 8). There is a PDF article online, **"Page Layout with Floating and Positioning,"** *that describes these techniques, but they should only be used if older, non-supporting browsers need to be supported. It is available at learningwebdesign.com/articles/.*

clearing
: Preventing an element from appearing next to a floated element and forcing it to start against the next available "clear" margin edge.

float containment
: A technique for expanding or holding open a containing element when all of its children have been floated and removed from the normal flow.

CSS shapes
: Non-rectangular wrap shapes around a floated element specified with the `shape-outside` property.

static positioning
: Elements are positioned as they appear in the normal document flow (the default).

relative positioning
: Moves the box relative to its initial position in the flow.

absolute positioning
: Removes the element from the document flow and positions it with respect to the viewport or other containing element box.

fixed positioning
: Stays in one position in the viewport as the document scrolls.

sticky positioning
: The positioned element behaves as though it is relatively positioned until it is scrolled to a position relative to the viewport, at which point it remains fixed.

containing block
: The box relative to which the position of an element is calculated. The containing box could be a positioned ancestor element or the `html` element (viewport).

z-index
: Defines the stacking order for overlapping positioned elements.

Exercises in Chapter 15

15-1: *Floating images*
Back to the Black Goose Bakery page, we use the `float` property to allow text to flow around the photos on the page.

15-2: *Adding shapes around floats*
Students are given the opportunity to add shaped wraps around the bread and muffin images.

15-3: *Absolute positioning*
This exercise uses absolute positioning to add an award graphic to the top of the home page.

15-4: *Fixed positioning*
The positioning type for the award graphic is changed from absolute to fixed, and the different behavior can be observed.

NOTES

Chapter 16: CSS Layout with Flexbox and Grid

NOTES

Summary

Chapter 16 covers the newer CSS layout tools and it should be treated as two big chapters. Browsers have only recently offered widespread support for the current Flexbox and Grid standards, but they are now acknowledged to be the "right" way to handle the layout of multi-part components and columned page layout.

NOTE

There is a third CSS layout tool that wouldn't fit in this chapter, so I put it in a separate PDF. The **"Multicolumn Layouts"** *article is available at* ***learningwebdesign.com/articles****.*

First, it is important to understand the purpose of teach tool. Flexbox arranges items along one axis (a row or a column), like beads on a string. Each flex item ("bead") can be set to stretch or shrink to fit the available space and the "string" may be set to wrap onto multiple lines. Flexbox is useful for items that line up, like menubars, complex header components, and gallery-like spaces where multliple images or product "cards" flow into a content area. Flexbox can be used to lay out a whole page, but that is not its strong suit because you can't force items on separate wrapped lines to line up with one another; it works on one axis only.

Grid Layout, on the other hand, provides a way to line up items into both rows and columns (two axes). The row and column tracks and the grid container itself can be set to be rigid, flexible, or based on the content within the cells. It is an amazingly versatile system.

Make it clear that Grid and Flexbox (as well as floating and positioning) can be used together; for example, laying out a page structure with a grid, then using Flexbox to handle components within that grid. Students should become familiar with both technques.

Both Flexbox and Grid offer ways to allow content to expand or shrink into available spaces, making them excellent solutions for web pages that will be viewed on a huge range of screen sizes. With that versatility, however, comes complexity. Both of these new specs can seem daunting at first because of the number of options, many of which are brand new concepts. Although each technique has a lot of moving parts and will require some attention to master, they are worth the time and effort.

Takeaways

In **Chapter 16**, students will learn the following:

Flexbox

- How Flexbox is useful for responsive layouts because items can adapt to the width of the viewport. Flexbox also makes it easy for items to be the same height and to center items horizontally and vertically (which were previously difficult to do with CSS properties).
- How to make an item a **flex container** by setting its `display` property to `flex`. Its children automatically become **flex items** within the container.

NOTE

With Flexbox, the trickiest part is getting used to the direction-agnostic terminology and mastering the alignment and flex properties.

- Flex items can be arranged on a horizontal or vertical axis, as specified with the `flex-direction` property. By default, it is the same direction as the writing direction for the document's language.
- The parts of a flex container: **main axis** (inclucing **main start**, **main size**, and **main end**) and **cross axis** (including **cross start**, **cross size**, and **cross end**). The main axis corresponds to the direction specified with `flex-direction`.
- Controlling whether flex items wrap onto new lines (`flex-wrap`).
- How to align flex items: `justify-content` sets alignment on the main axis; `align-items` sets alignment on the cross axis. If there are multiple wrapped lines, the `align-content` controls how they are aligned on the cross axis.
- Specifying how individual items shrink or expand using the `flex` property, including handy shortcut flex values for typical Flexbox scenarios.
- How to rearrange the order of flex items in the container (using the `order` property) independently of how they appear in the document source.
- Browser support issues with Flexbox.

Grid

- How grids work: Make an element a **grid container** (`display: grid`), set up the columns and rows in the grid (the **template**), assign each grid item to an area on the grid.
- Familiarity with the parts of the grid (**grid container**, **grid item**, **lines**, **tracks**, **cells**, **areas**, **block axis**, and **inline axis**).
- How to define grid tracks using `grid-template-rows` and `grid-template-columns` (also the `grid-template` and `grid` shorthand properties).
- An understanding of grid line numbering, explicit line naming, and implicit (automatic) line naming.
- All the options for defining the width and height of a grid track: lengths, % values, fractional units (`fr`), `min-content`, `max-content`, `auto`, `minmax()`, `fit-content()`, `repeat()`.
- Assigning names to areas on the grid so they can be used to place items on the grid (`grid-template-areas`).
- How to place items on the grid using lines (`grid-row-start`, `grid-row-end`, `grid-column-start`, `grid-column-end`, and the shorthand `grid-row` and `grid-column` properties).
- How to position items on the grid using named grid areas (`grid-area`).
- Using `grid-auto-rows`, `grid-auto-columns`, and `grid-auto-flow` properties to set parameters for implicit grid behavior.
- Alignment properties for aligning grid item elements within grid cells when there is extra room (`justify-items`, `align-items`, `justify-self`, `align-self`).
- Alignment properties for aligning grid tracks in a larger container (`justify-content`, `align-content`).

NOTES

NOTE

The thing that makes Grid feel complicated is that there are a lot of different ways to accomplish each task (such as identifying grid lines and specifying track sizes). Keep in mind that the system of setting up a grid and putting items in it is fairly straightforward.

The trick to mastering Grid is to become familiar with the various methods for specifying track size as well as taking advantage of Grid's features for filling grid cells, rows, and columns automatically.

Key terms in Chapter 16

NOTES

flex container
: An element that has its `display` set to `flex` (its children become flex items).

flex items
: The children of a flex container that line up along a specified axis.

nested flexbox
: Flex items can be made into flex containers by setting their `display` to `flex`.

flow
: The direction in which flex items are laid out as well as whether they are permitted to wrap onto additional lines.

main axis
: The flow direction specified for the flex container. For horizontal languages, when flow is set to `row`, the main axis is horizontal.

cross axis
: The cross axis is whatever direction is perpendicular to the main axis (vertical when flow is set to row).

main size
: The width of the container along the main axis if it is set to row (or the height of the container if it is set to column).

cross size
: The height along the cross axis if it is a row (or the width if it is a column).

flex
: The quality of flex items that allows them to resize to fit available space. It is concerned with how extra space is distributed *within* items.

relative flex
: When `flex-basis` is a value other than zero, extra space is divided up based on their flex ratios. An item with a flex ratio of 2 would get twice as much extra space allotted to it than an item with a flex ratio of 1 (although it may not end up twice as wide as 1 depending on its content).

absolute flex
: When `flex-basis` is 0, items are themselves sized proportionally according to their flex ratios. An item with a flex ratio of 2 would be twice as wide as one with a flex ratio of 1.

grid container
: An element that has its `display` set to `grid` (its children become grid items).

grid item
: Direct children of a grid container that end up positioned on the grid.

grid line
: The horizontal and vertical dividing lines of the grid. Grid lines may be assigned names and are numbered automatically.

NOTES

grid cell
: The smallest unit of a grid which is bordered by four adjacent grid cells.

grid area
: A rectangular area made up of one or more adjacent grid cells.

grid track
: Space between two adjacent grid lines (a generic name for column or row).

block axis
: The vertical axis of a grid (for languages written horizontally).

inline axis
: The horizontal axis of a grid (for languages written horizontally).

fractional unit (fr)
: A grid-specific unit of measurement that allows an item to expand and contract depending on the available space.

implicit grid behavior
: Automatic grid behaviors such as pouring grid items into cells sequentially if they haven't been explicitly positioned on the grid, or creating additional rows and columns on the fly to accommodate extra items.

Exercises in Chapter 16

16-1: *Making a navigation bar with Flexbox*
This exercise uses the basic Flexbox properties to turn the navigation list on the Black Goose Bakery page into a horizontal menu.

16-2: *A flexible online menu*
The sample page with multiple menu items (similar to the structure of a gallery or product listing page) provides the basis for exploring flex properties throughout this section. This first exercise creates a flex container and plays around with row direction, wrapping, and alignment to get a feel for how the properties work.

16-3: *Adjusting flex and order*
Here items are made to fill the available space using the `flex` property, and items within them are rearranged via nested flex containers and the `order` property.

16-4: *Setting up a grid*
The first step to creating a grid-based layout for a "Breads of the World" page is to set up a grid container and define its rows and columns. Because items are not placed on the grid, they flow in sequentially (that gets fixed in the next exercise).

16-5: *Placing items on a grid*
Students are asked to explicitly place grid items on the established grid using a variety of approaches with line numbers and names.

16-6: *A grid layout for the bakery page*
We return to the Black Goose Bakery page to give it a two-column layout using CSS Grid. Items are placed on the grid using named grid areas, which is a very straightforward approach for this simple page.

Chapter 17: Responsive Web Design

NOTES

Summary

At this point, your students will have covered all the properties required to style text and text boxes, including layout tools such as floating, positioning, Flexbox, and Grid. With that foundation, it is time to put these skills together into this chapter on Responsive Web Design (RWD). RWD is a technique that applies different styles based on the width of the viewport, enabling layouts to adapt to the device on which they display. RWD is the preferred method for creating web pages that work on mobile devices. For example, Google search gives responsive sites a boost in listings.

NOTE

In Ethan Marcotte's original recipe for Responsive Web Design, percentage values are used to size elements so they adapt to various screen widths proportionally. Today, we have Flexbox and Grid that offer more sophisticated ways to make elements flexible, as I present in this chapter. This updated approach to responsive sites has recently been given the name "Intrinsic Web Design" by Jen Simmons.

Takeaways

In **Chapter 17**, students will learn the following:

- The core principle of Responsive Web Design (RWD): all devices get the same HTML source, located at the same URL, but different styles are applied based on the viewport size. Compare this to **m-dot** sites, which are entirely separate sites served to mobile devices only.
- The components of RWD are a flexible grid, flexible images, CSS media queries, and the viewport `meta` element.
- Using the viewport `meta` element to make the size of the initial **viewport** (the canvas the page is drawn on) the same size as the physical screen of the device.
- Making images change size to fit their containers by setting `img {max-width: 100%;}`. You may also choose to use a responsive image technique (outlined in **Chapter 7**) to avoid serving unnecessarily large images to smaller devices.
- How to use **media queries** to test for certain browser features and serve styles based on the query matches (example: `@media screen and (min-width: 40em) { /* style rules */ }`.
- Choosing **breakpoints** (style changes) for individual components at different screen sizes.
- General guidelines for designing responsively with regard to content, layout, typography, navigation, images, and special content.
- Familiarity with various layout patterns identified and named by Luke Wroblewski (mostly fluid, column drop, layout shifter, tiny tweaks, off canvas).
- Familiarity with various navigation approaches, such as top navigation, "Priority +", a select menu, link to footer menu, accordion sub-navigation, push and overlay toggles, and off-canvas/fly-in.

- Options for testing responsive sites: real devices (the best, but most expensive option), emulators, and third-party services.

NOTES

Key Terms in Chapter 17

viewport
: The canvas on which the page is rendered.

device width
: The width of the screen on the device. Small devices may use a viewport (canvas) that is similar to the pixel dimensions of a desktop browser window then shrink that down to fit the screen width.

fluid layout
: The page grid resizes proportionally to fill the available width of the browser window.

fixed-width layouts
: Web pages designed to always display at a specific pixel width.

media queries
: A rule in a style sheet that applies styles based on whether the browser meets certain criteria, such as screen width and orientation.

breakpoint
: The point at which a media query is used to introduce a style change.

content parity
: An approach that makes sure the same content is accessible regardless of the device used to access the site.

line length
: The number of characters in a line of text. Ideally, there should be 45 to 75 characters per line, so if there are fewer or more, it may be time to introduce a new breakpoint in the layout.

Exercises in Chapter 17

17-1: *Making the bakery home page responsive*
In this exercise, we give the Black Goose Bakery a responsive layout that goes from one column to two columns based on screen width. Note that the *bakery.html* file (provided with the chapter materials) does not start exactly where it left off in **Chapter 16**. I made a few tweaks (as noted in the exercise) to make it appropriate for the small-screen experience as a starting point. This exercise focuses on giving students practice at adding media queries to a familiar document and to get a feel for the type of design elements you might change at various breakpoints.

Chapter 18: Transitions, Transforms, and Animation

NOTES

Summary

Chapter 18 introduces CSS properties for moving and distorting elements and adding time-based effects and animation. It is divided into three parts. First, it looks at the transition properties that smooth out style changes from one state to another (such as a hover effect). Next, it runs through all the transformation properties for moving, scaling, rotating, and skewing an element. Transformation effects can be combined with transitions, for example, to make an element smoothly move from one position to another. The chapter closes with a brief introduction to CSS animation.

Takeaways

In **Chapter 18**, students will learn the following:

Transitions

- Transitions and animation, when used thoughtfully, can make interfaces more intuitive and enhance brand personality.
- Transitions require a beginning state and an end state. The beginning state is how the element displays when it first loads and the end state is triggered by a state change (such as `:hover`, `:focus`, or `:active`).
- The `transition-property` property identifies the CSS property to which the transition will be applied.
- You can specify how long the transition should take (`transition-duration`) and how long to wait before the transition starts (`transition-delay`). Both values are provided in seconds (`s`) or microseconds (`ms`).
- Timing functions (`transition-timing-function`) describe the way the transition behaves over time; for example, starting out slowly, then speeding up toward the end. The timing function can be plotted and customized with a Bezier curve.

Transformations

- Using the `transform` property to **rotate**, relocate (translate), resize (**scale**), and **skew** an element.
- How to combine `transition` and `transform` to smooth out transformations between states.
- An introduction to 3-D transformation, a set of properties that makes elements look like cards floating in space.

Keyframe animation

- How to set up the timing of a keyframe animation using the `@keyframes` rule that defines a series of keyframes and the property and value for each state.

- Adding the `@keyframes` rule to the element to be animated with the animation properties (`animation-name`, `animation-duration`, `animation-timing-function`, `animation-delay`, and other properties).

NOTES

Key Terms in Chapter 18

tweening
: The filling in of frames between two end state frames, common in animation.

timing function
: The speed and manner in which the transition rolls out over time.

Cubic Bezier curve
: A line that illustrates the change of a transition over time. Steep parts of a curve indicate a fast rate of change, and the flat parts indicate a slow rate of change. You can create custom timing functions by editing the curve and providing coordinates in the `cubic-bezier()` notation.

bounding box
: The element box from border edge to border edge. Used to calculate percentage length values.

keyframe animation
: An animation that transitions through a series of keyframe states, enabling more sophisticated control of the action.

keyframes
: Frames in an animation that define the beginning or end of an animation segment.

explicit animation
: Animation affects that are programmed by an author, such as keyframe animation.

implicit animation
: Animation that gets triggered automatically, such as CSS transitions.

animation inspector
: Tools built into Firefox and Chrome browsers to inspect and modify web animations.

Exercises in Chapter 18

18-1: *Trying out transitions*
Students are walked through the process of adding transition effects to a set of menu buttons, including trying out a variety of values that have a dramatic impact on usability.

18-2: *Transitioning transforms*
In this exercise, elements are transformed (made larger and rotated) when the user hovers over them, and a transition is also applied to make the transformation smooth for a nice animated effect.

Chapter 19: More CSS Techniques

NOTES

Summary

This chapter is a collection of CSS techniques that are good to know but that didn't quite fit into the previous chapter topics. For beginners, it may be especially helpful to cover CSS resets and Normalize.css for clearing out browser default styles before writing the styles for their own web pages. It is confusing when writing a style rule doesn't produce the expected result, and browser defaults are often the culprit. This is the last chapter in the book related to style sheets.

Takeaways

In **Chapter 19**, students will learn the following:

- Tips and tricks for styling form elements, including adding styles to a sample sneaker order form.
- Some table-specific properties that affect the spacing between cells (`border-collapse` and `border-spacing`), how the width is calculated (`table-layout`), where the caption appears (`caption-side`), and how empty cells are handled (`empty-cells`).
- Using a CSS reset and normalizer to clear out browser default styles, creating a "clean slate" for our own style sheets.
- Some image-replacement techniques for using images in place of text in a way that is accessible to screen readers and search engines.
- A technique for combining images into one image file (a sprite) to reduce the number of HTTP requests to the server in order to improve performance.
- Two methods (CSS Feature Queries and Modernizr) for testing whether a browser supports a newer CSS property so fallbacks can be provided.
- CSS feature queries that test browser support for properties and values using an `@supports` rule. If the browser passes the test, the styles in the brackets are applied.
- Modernizr, a JavaScript library that tests for HTML5 and CSS3 features when the page is loaded. It is more widely supported than feature queries, but will not work at all if JavaScript is not enabled on the browser.

Key Terms in Chapter 19

cell padding
: The amount of space within a cell, between the content and the cell edge.

cell spacing
: The amount of space between cells.

CSS reset
: A collection of style rules that overrides all user agent styles and creates a neutral starting point.

NOTES

image replacement technique
: A method for using an image in place of text (such as a logo) in a way that keeps the text accessible to screen readers and search engines. The image is added as a CSS background on the text element, and the text itself is shifted out of the way with a margin so it is not rendered on the page.

sprite
: Combines multiple smaller images into a single image, resulting in fewer server requests and better performance. CSS width, height, and background position properties are used to position the sprite so only one portion is visible at a time.

feature detection
: Testing for browser support for a specific CSS property or value. Supporting browsers are given the desired styles, and a fallback is provided to non-supporting browsers.

feature query
: The `@support` rule that tests browser support for a particular property and value declaration (similar to a media query).

operators
: Refines the feature query test by adding `not`, `and`, or `or` to the declaration or multiple declarations.

Modernizr
: A lightweight JavaScript library that tests for a variety of HTML5 and CSS features when the page is loaded in the browser.

Exercises in Chapter 19

None.

Chapter 20: Modern Web Development Tools

Summary

Chapter 20 introduces a number of tools that are central to the web developer toolkit. They help make development work more efficient, code more robust, and teamwork easier. Because many developer tools work only via the command line, the chapter begins with an introduction to the command-line interface. From there we look at CSS preprocessors and postprocessors for writing CSS efficiently and accurately. We'll also see tools like Grunt and Gulp that automate common repetitive tasks in the production and publishing process. Finally, no discussion of the modern developer workflow would be complete without an introduction to Git, a tool for version control that enables a team to collaborate on a code project.

Takeaways

NOTES

In **Chapter 20**, students will learn the following:

Command-line interface

- The fundamentals of the **command-line interface (CLI)**, including identifying the prompt and basic commands for navigating the file system.

Pre- and postprocessors for CSS

- **Preprocessing languages** (such as Sass, LESS, and Stylus) that provide a programming-like syntax (such as using variables and "mixins") to make writing styles more efficient. Files written in a preprocessor language must be compiled into standard CSS before being sent to a browser.
- **Postprocessor** tools take standard CSS files and perform optimization functions such as checking for errors, improving cross-browser consistency, reducing file size, and more.
- Over time, pre- and postprocessors have begun handling a lot of the same tasks, so their roles may overlap.

Build tools/task runners

- Most professional web developers use some sort of **build tool** or **task runner** (like Grunt or Gulp) to automate the many repetitive tasks associated with creating web sites.
- Common tasks for automation include concatenation, compression, checking for errors, optimizing images, pushing to Git, and more.
- Many sites are created with templates for each page type and content data that gets poured in on the fly.

Git

- **Version control systems (VCS)** keep track of versions of work and allow creators to go back to earlier versions and share code source. with a team.
- The most popular VCS in the web world is **Git**.
- **GitHub** is a service that hosts Git repositories and provides a helpful user interface to the Git command line functions.
- Basic Git terminology such as: repository (repo), commit, working directory, staging, branch, master, fork, merging, remotes, cloning, pushing, and pulling.

Key Terms in Chapter 20

command-line interface (CLI)
: A way to interact with programs on the computer by typing commands into a terminal application.

shell
: The program that interprets the commands you type into a CLI. There are many shell programs, including bash, the shell that comes on Mac and Linux computers.

NOTES

terminal
: The program you use to type commands into. Terminal (with a capital T) is the name of the command-line tool on macOS.

prompt
: A series of characters indicating the computer is ready to receive a command.

flag
: When added to a command, it changes how the utility operates, like an option or a preference.

argument
: Provides specific information required for a function to run.

dotfile
: Files (beginning with a dot or period character) that are part of an operating system and are kept hidden from view.

preprocessor
: A programming-like syntax for writing CSS such as Sass, LESS, and Stylus. Files in preprocessor languages must be converted (compiled) into standard CSS for the browser.

postprocessor
: A CSS optimization tool that takes a standard CSS file and optimizes it. Postprocessor tools are available for reducing file size, supporting old browsers, checking for errors, and much more.

compile
: In preprocessors, compiling is the translation process that converts specialized syntax to standard CSS rules.

variable
: A value you can define once, then use multiple times throughout a document.

mixin
: In the Sass preprocessor, a set of styles that are saved and named as a group so they can be used throughout the style sheet.

build tool/task runner
: A tool that automates repetitive production and publishing tasks, such as Grunt and Gulp. Build tools are used through a command-line interface.

version control system (VCS)
: Software that keeps track of versions of work, allowing you to go back to earlier iterations and also share files.

Git
: A distributed version control system commonly used in web production.

repository
: Where past versions and a change history of the project are stored in Git.

NOTES

commit
: A change to a file that is logged in with a unique ID number in Git.

distributed version control system
: A system that allows users to make copies of files to their own computers and work on or offline.

working directory
: The directory of files on your computer where you do your actual work.

staging
: Adding a file to Git so it can be tracked. Staged files are stored in a local index of files that will eventually be committed to the repository.

branch
: A sequential series of commits (also called a stack of commits) that represents a thread of development.

master
: The primary or default branch, which may represent the official version of the project.

merge
: Combining commits from one branch into another or from different versions of the same branch.

conflict
: Two different changes in the same line of code typically uncovered when branches are merged.

remote
: A repository on a computer other than your own.

clone
: An exact replica of a repository and everything it contains. It is common to clone a repo from a remote computer to your own before working on it.

push
: The process of moving data from your local repository to a remote repository.

pull
: Moving data from the remote repository to your local computer in order to update it.

fork
: Making a copy of a GitHub repository to your GitHub account so you have your own copy to edit.

pull request
: Asking an owner of a repository to pull your changes into the master.

Exercises in Chapter 20

None.

NOTES

Chapter 21: Introduction to JavaScript

Summary

This chapter provides an introduction to the building blocks of JavaScript and serves as a good starting point for further exploration. Complete training in scripting is beyond the scope and scale of an overview book like this one, so the goal here is to become familiar with JavaScript syntax—variables, functions, operators, loops, and events—so students can look at existing scripts and have a general understanding of what they do. There are also examples that demonstrate the types of things JavaScript can be used for. Many web developers I know have learned JavaScript by adapting existing scripts to meet their needs, then expanding their knowledge from there as needed. Resources for further learning are provided at the end of the chapter.

Takeaways

In **Chapter 21**, students will learn the following:

- That JavaScript is a **client-side scripting** language, which means it runs on the user's machine and is dependent on the capabilities of the browser.
- That JavaScript is said to add a behavioral layer to the HTML structural layer. All the elements, attributes, styles, and text on a web page can be accessed and manipulated with JavaScript.
- The types of things JavaScript can do, such as validate forms, show and hide content based on clicks, test for browser features, and more.
- Embedding a script directly in the source document by putting it in a `script` element.
- Linking a standalone *.js* file to the document with the `src` attribute in the `script` element.
- That scripts are composed of a series of **statements** (a command that tells the browser what to do).
- About **variables**, containers for information that you can reuse throughout a script by calling them by name. Variables are defined with the `var` keyword. You can change the value of a variable at any time in a script.
- The **data types** of variables including undefined, null, numbers, text strings, and Boolean (true/false).
- About **arrays**, lists of values provided for a single variable.
- **If/else statements**, a way to ask a true/false question and execute commands based on the results. They form the foundation of the logic that can be written with JavaScript.

NOTES

- **Loops** that allow scripts to run through every item in an array sequentially without needing to type out each one individually.
- A **function** is code for performing a task that is set up but doesn't run until it is called or referenced. It is like a variable that has executable code rather than a static value. Functions allow code to be reused as needed in the script without writing it multiple times.
- Some functions are built in to JavaScript (for example, `alert()`); others are custom functions that you write yourself using the `function` keyword.
- Paying attention to whether variables are **globally scoped** (available to all the scripts on the page) or **locally scoped** (available only to its parent function).
- Accessing and manipulating the browser itself (known as the window object).
- JavaScript listens for **events,** actions that can be detected such as loading the page or hovering over an element with the pointer. Events can be used as triggers functions. Examples of events in JavaScript include `onload`, `onmouse-over`, and `onclick`.

Key Terms in Chapter 21

client-side scripting
: Scripts that are processed on the user's machine, and not the server.

dynamic programming language
: Scripts can be delivered and interpreted directly by the browser without needing to be compiled as some programming languages require.

loosely typed language
: A language for which you do not need to specify what type of information will be stored in a variable in advance. JavaScript assigns a type to a variable based on what kind of information you provide for it. For example, the variable value 5 is interpreted as a number type.

case-sensitive
: Capitalization matters, so `myVariable`, `myvariable`, and `MYVariable` would be treated as three different objects.

statement
: A command within a script that tells the browser what to do.

variable
: A variable is used to store, retrieve, and manipulate values in code.

data types
: Describes the type of variable value.

array
: A group of multiple values (called **members**) that can be assigned to a single variable.

indexed values
: Values in an array that can be referred to by number according to the order in which they apper in the list.

comparison operator
: Special characters that evaluate and compare values (for example, the == operator means "is equal to").

loop
: Syntax for instructing a script to run through all the values in an array sequentially.

function
: A reusable bit of code that is defined once and only runs when it is called by name.

argument
: A value or data that a function uses when it runs. You can pass arguments to functions on the fly so the function can be used with different data throughout the script.

scope
: Refers to the availability of a variable in the script.

globally scoped
: A variable that can be used by any of the scripts on your page.

locally scoped
: A variable that's available only within its parent function.

event
: An action that can be detected with JavaScript, such as when the document loads or someone clicks an element.

Exercises in Chapter 21

21-1: *English-to-JavaScript translation*
This exercise walks students through the steps for writing a script that adds a title with a counter to the browser window tab.

Chapter 22: Using JavaScript
and the Document Object Model

Summary

This chapter applies what we've learned about JavaScript to manipulate the objects in a web page. Students are introduced to the Document Object Model and some methods available for accessing, adding, deleting, and changing various aspects of the document. It also addresses JavaScript used to patch browser behavior (**polyfills**) as well as JavaScript libraries that make using JavaScript easier and more efficient.

NOTES

Takeaways

NOTES

In **Chapter 22**, students will learn the following:

- The **DOM (Document Object Model)** as an interface for accessing the elements, attributes, and content of a document. It makes markup programmable.
- How the DOM sees an HTML document as a tree-like structure of nodes representing elements, attributes, and text.
- DOM methods and functions that can be used to interact with the document nodes.
- Several examples of how to access nodes within a document, including
  - — By element name: `getElementsByTagName()`
  - — By `id` attribute value: (`getElementById()`
  - — By `class` value: `getElementsByClassName()`
  - — Via a CSS-style selector: `querySelectorAll()`
  - — By attribute value: `getAttribute()`
- Examples of what you can do with nodes once you've accessed them, such as setting an attribute value, changing content within an element, and modifying the style applied to the element.
- Examples of how to add and remove entire elements from the DOM.
- Using **polyfills** to make older browsers support new features; for example, making them recognize new HTML5 elements and CSS3 selectors.
- How a JavaScript library can help you add interactivity to your site with preexisting scripts that can be adapted for your uses.
- jQuery, an open source and freely available JavaScript library. There are versions targeted to building user interfaces (UI) as well as evening out mobile browser inconsistencies (jQuery Mobile).
- How to implement jQuery: download *jquery.js* and save it to your main site directory, add it to your document with the `script` element, set up a "ready event" that tells the browser when to run it, then write your own scripts using jQuery for shortcuts.

Key Terms in Chapter 22

Document Object Model (DOM)
: A programming interface (API) for HTML and XML pages that gives us a way to access and manipulate the contents of a document.

node
: Each element in a web page as seen by the DOM. The relationship of elements forms a "tree" with branches of nested nodes (elements and content).

polyfill
: A script that provides older browsers with modern features.

JavaScript library
: A collection of prewritten functions and methods that you can use in your scripts to accomplish common tasks or simplify complex ones.

ready event
: A statement in a script that checks the document and waits until it is ready to be manipulated before running the script itself.

NOTES

Exercises in Chapter 22

None.

Part V: WEB IMAGES

Chapter 23: Web Image Basics

Summary

Chapter 23 provides essential background information that will help budding designers and developers create images that are appropriate for the web. It starts by discussing several options for finding images with a mind toward copyright issues. A majority of the chapter introduces the image file formats that are most popular on the web (JPEG, PNG, GIF) as well as an up-and-comer, WebP (see **Note**).

Understanding what's happening under the hood for each format helps designers choose the best format for the job. The chapter provides a primer on screen and image resolution, including how to accommodate high-density displays. There is also a flow chart that walks designers through a strategy to follow when producing images for responsive web sites. Finally, it takes a look at favicons (the little icon that shows up in the browser window) and how to create them.

NOTE

Chapter 23 *focuses on bitmapped formats. SVG is handled separately in* ***Chapter 25*** *because it is a vector format written in an XML file.*

Takeaways

In **Chapter 23**, students will learn the following:

- Where to get images for sites, including creating your own, using stock images (either **rights-managed** or **royalty-free**), finding clip art online, or hiring a professional.
- JPEGs are **Truecolor images** with a **lossy compression** scheme that throws out data in order to make images as small as possible. Because JPEG com-

NOTES

pression is optimized for smooth color transitions, JPEG is the best choice for photographs.

- The PNG format comes in several bit depths. **PNG-24** is a lossless Truecolor format that is generally avoided for web images due to large file sizes; however, it is useful if variable transparency is required. **PNG-8** is an indexed, 8-bit format that works best on images with areas of flat color. PNG-8 can do binary (on/off) transparency.
- The original web image format, **GIF**, is an indexed, 8-bit format that is also capable of animation.
- **WebP**, from Google, can handle all types of image data and promises much smaller file sizes. It is slowly growing in browser support.
- How to choose the best file format for different image types.
- How bitmapped images are displayed on screens, and how the screen's pixel density affects the display of images.
- A strategy for producing image assets for a site with the goal of keeping file sizes as small as possible, minimizing the number of HTTP requests, not downloading more data than is needed, and sending high-quality images to high-density displays.
- How to create favicons, both the traditional way and as a larger icon set for use across different device types.

Key Terms in Chapter 23

rights-managed images
: Images for which the copyright holder (or company representing them) controls who may reproduce the image. Rights-managed images are generally available for a specific use and for a fee.

royalty-free images
: Images for which you do not need to pay a licensing fee.

Creative Commons License
: A license by which artists may release their artwork for free with certain restrictions.

JPEG (Joint Photographic Experts Group)
: An image format that is well-suited to photos because it can contain 24-bit color images and uses a lossy compression scheme to keep file sizes small.

lossy compression
: A compression scheme that permanently throws out data in order to reduce the size of the file.

lossless compression
: A compression scheme that retains all the original data when the file is compressed.

NOTES

Truecolor (also RGB color)
: A palette of millions of colors made up of 256 levels each of red, blue, and green light.

PNG (Portable Network Graphics)
: A robust file format designed to contain both 24-bit color and 8-bit indexed color images (as well as gray scale). 8-bit PNGs are the best choice for graphics with hard edges and flat areas of color.

PNG-24
: A 24-bit PNG capable of storing millions of colors and multiple levels of transparency (alpha transparency).

PNG-8
: An 8-bit PNG capable of storing 256 colors total and both binary and alpha transparency.

GIF (Graphic Interchange Format)
: The first image format supported in web pages, it is an 8-bit indexed format that is also capable of binary transparency and animation.

8-bit
: With regard to image formats, 8-bits can describe up to 256 colors (2^8=256).

indexed color
: A color model that stores color information in a color table that is referenced by each pixel in the image. It can store a maximum of 256 colors.

palette
: The collection of colors in an image.

color table (also color map)
: Where colors in an indexed color image are stored. The color table for indexed images can be accessed in image editing tools.

quantization
: The conversion from 24-bit RGB color to an indexed 8-bit format.

binary transparency
: A model in which pixels are either fully transparent or fully opaque.

alpha transparency
: A model in which pixels may be any of 256 levels of opaqueness.

alpha channel
: A fourth channel (in RGB images) that stores transparency information as a gradient.

gamma
: The brightness setting on a monitor.

screen designer
: A new term for designers who design products intended to be viewed on screens (such as web sites, applications, and video games).

ppi
: Stands for "pixels per inch," and is a measure resolution.

resolution
: The number of pixels per inch in a digital image or in a screen display.

pixel density
: The resolution (ppi) of a screen.

high-density display
: A screen with a pixel density that is 1.5 to 4 times higher than traditional screens.

reference pixel
: A unit of measurement used by high-density devices for purposes of layout independent of the resolution of the physical pixels in the screen.

point (PT)
: What Apple calls its reference pixel, equal to 1 standard device pixel.

device-independent pixel (DP)
: What Android calls its reference pixel, equal to one pixel at 160ppi.

CSS pixel
: The pixel unit used in CSS length measurements. The term is used interchangeably with device reference pixels because they map one-to-one.

favicon
: The little icon that shows up in the browser tab and in bookmark lists. It helps users find your site in a lineup of tabs or bookmarks and can strengthen a brand.

Exercises in Chapter 23

None.

Chapter 24: Image Asset Production

Summary

With the background information about web graphic files out of the way, this chapter focuses on image production. We start by simply exporting and saving in a variety of web-appropriate formats, including an exercise that demonstrates how the format affects file size. The next section is a deep dive on binary and alpha-layer transparency. We take a look at the ways modern image editing tools support making images for responsive sites and for high-density displays. Finally, and most importantly, we look at measures every designer and developer can take to optimize each image type so they have the smallest file size as possible.

Takeaways

In **Chapter 24**, students will learn the following:

NOTES

NOTE

All examples of saving images are demonstrated in Adobe Photoshop CC and GIMP because they are both cross-platform and available for free (a trial version for Photoshop and an open source license for GIMP).

NOTES

- How to export and save in web formats in Adobe Photoshop CC and GIMP, including what options are available.
- That file format and compression scheme has a big impact on the file size of the image asset.
- How **binary**, **alpha**, and **alpha-palette transparency** work.
- How to avoid a "halo" of mismatching pixels around a transparent PNG or GIF.
- How to create images with transparent areas in Photoshop and GIMP.
- Using modern image editing tools to create images for use in responsive layouts and on high-density displays.
- The importance of optimizing images to improve performance.
- General guidelines for optimizing all images and special tips for optimizing JPEGs, PNGs, and GIFs.
- An overview of optimization tools that can make files exported from image editing programs even smaller.

Key Terms in Chapter 24

binary transparency
: Pixels are either entirely transparent or entirely opaque, like an on/off switch.

alpha transparency
: A pixel may be one of 256 levels of transparency from entirely opaque to entirely transparent. Only PNG, WebP, and JPEG 2000 support alpha transparency.

halo
: The term used for the fringe of pixels around the edges of the image that do not blend in with the background color.

alpha channel
: A fourth channel in RGB images that stores transparency information as a gradient from black to white, where white areas are opaque, black areas are transparent, and grays are a scale in between.

alpha-palette PNG (or PNG8+alpha)
: An 8-bit, indexed color PNG file that stores transparency levels in its color map.

upscaling
: Taking a bitmapped image and resizing it larger (usually resulting in blurry or lower image quality).

dithering
: A speckle pattern that results when colors from a palette are mixed to simulate an unavailable color.

Exercises in Chapter 24

NOTES

24-1: *Formats and file size*
Two different types of images are saved in various file formats and compression rates then the resulting quality and file sizes are compared. The lesson is that JPEG is most efficient for photos and PNG is the best for flat colors.

24-2: *Creating transparent images*
In this exercise, students create an image with a soft edge and save it with the transparency preserved as binary, alpha, and PNG-8+alpha. The resulting images are compared by placing them on a web page against a colored background.

24-3: *Optimize some images*
The best images from **Exercise 24-1** are run through an online optimizer to see if they can be compressed down even smaller. (Spoiler: they can!)

Chapter 25: SVG
(Scalable Vector Graphics)

Summary

In the past few years, SVG graphics have gone from a cutting-edge format with spotty browser support to a best practice for adding scalable images to responsive web pages. In this edition, SVG gets an entire chapter describing the SVG markup language, its remarkable features, how they can be used to add interactivity to a page, as well as tools and tips for creating them. The chapter closes with instructions for how to make an SVG scale proportionally in a responsive layout.

Takeaways

In **Chapter 25**, students will learn the following:

- That SVG images are made of **vectors** (not a grid of pixels), so they can scale up and down with no loss of quality.
- Those vectors are defined using the **SVG markup language**. A simple example shows how to establish the viewport dimensions, the use of shape elements (`circle`, `rect`, and `path`), a gradient fill, and text.
- More about **XML, eXtensible Markup Language**, a set of rules for creating other markup languages.
- That SVGs can contain embedded bitmapped images which can be clipped, masked, and altered with Photoshop-like filters.
- How to use the `defs` and `symbol` elements to create objects and effects that can be reused in the document, thus cutting down on redundant code and keeping the file size in check.
- Because SVG images are text documents with elements and attributes, you can access and manipulate their components with styles and scripts.

NOTES

- Options for styling SVG elements:
  - — Presentation attributes from the SVG language
  - — Inline styles with the **`style`** attribute
  - — An embedded style sheet in the SVG itself using the **`style`** element
  - — A style sheet in the HTML document where the SVG is embedded with the **`svg`** element.
- JavaScript can be used to add interactivity and animation to SVG images. SVGs may also be animated with effects from the SVG spec and CSS animation properties, although not as reliably.
- Some tools for creating SVGs include Adobe Illustrator, Inkscape (free), Boxy, and SVG-Edit. User interface tools such as Sketch, AdobeXD, and Affinity Designer are vector-based and can also save in SVG format.
- Best practices for optimizing SVG code, including measures you can take in image editing tools prior to export as well as using the optimization tool SVGO to further reduce the code they produce.
- How to add an SVG to a page in a way that it resizes proportionally in a responsive layout using **`img`**, **`object`**, and inline with the **`svg`** element.

Key Terms in Chapter 25

SVG (Scalable Vector Graphics)
: An XML markup language used to describe 2-D vector images. It also refers to the file format of the images written in that language.

XML (eXtensible Markup Language)
: A meta-language (set of rules) for creating other markup languages. SVG, MathML, and XHTML are examples of XML languages.

clipping
: Using a vector shape to cut out a shape out of the area below it.

masking
: Using a gradient or bitmapped image to affect the transparency of the image area below it.

filter primitive
: A very specific image effect (such as blur or saturate) that can be combined with other effects.

DRY (Don't Repeat Yourself)
: A coding approach that aims to be as efficient as possible and avoids redundant code.

SVG sprite
: A technique in which multiple SVG drawings are defined in one SVG. The **`use`** element pulls a particular symbol within the sprite onto the page.

XML Character Data Block
: In XML documents, a method for wrapping executable code in the `script` element so the `<`, `>`, and `&` symbols are parsed correctly:

```
<![CDATA[
   //script here
]]>
```

SMIL (Synchronized Multimedia Integration Language)
: An XML language for creating synchronized audio, video, and animated elements. It is not well supported.

aspect ratio
: The ratio of width to height.

viewport
: Defined by the `width` and `height` attributes on the `svg` element, it forms a sort of window through which you see the drawing.

viewbox
: The canvas on which the SVG is drawn (also called the user space), with its own coordinate system. The viewbox may or may not be the same as the viewport that contains it.

viewport coordinate system
: The set of coordinates used by the viewport, with 0 starting in the top-left corner and increasing to the right and downward.

user coordinate system
: The set of coordinates used by the drawing space (user space) that is independent of the viewport coordinates.

adaptive icons
: Icons that dynamically change design based on their size, with very simplified versions used at small sizes and more detailed versions for larger instances.

Exercises in Chapter 25

None.

NOTES