LEARNING WEB DESIGN *Sixth Edition*
INSTRUCTOR'S GUIDE

by **Jennifer Niederst Robbins**

Learning Web Design, 6e provides a thorough foundation in HTML, CSS, JavaScript and image production. It requires no previous coding experience, yet by the end of the book, students will be able to create simple, static 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/6e/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** and **13** get readers comfortable writing style rules that apply to text and colors/backgrounds, respectively). Once the foundation is laid, ***Chapter 14*** introduces more selector types that will be utilized in exercises in the rest of the book. **Chapters 15** through **16** are related to page layout (the box model, floating/positioning, Flexbox, and CSS Grid). **Chapter 19, Responsive Web Design** ties many of these lessons together into a technique for designing web pages that adapt to screen size. **Chapter 20** introduces interactive properties for transitions and animation. **Chapter 21** is a grab bag of common CSS-related techniques that don't quite fit neatly into the prior chapter topics, but are worth knowing.

Part IV: JavaScript for Behavior

The four chapters in **Part IV** (written by JavaScript expert, Aaron Gustafson) provide the fundamentals of how JavaScript. **Chapters 22** through **24** are lessons in basic JavaScript syntax (statements, variables, conditional statements, arrays, and loops) as well as how to use features of the DOM and JavaScript events to manipulate the content of a page. There are numerous exercises that give students hands-on experience along the way that also address important principles such as accessibility and performance. **Chapter 25** is a big-picture overview of how JavaScript is used in the modern development workflow, including components, libraries, and platforms. These chapters are not a complete course in JavaScript or programming, which is beyond the scope of this overview book; however, resources for further learning are provided.

TIP: The site ***JavaScript.info*** is a free online resource with additional detailed tutorials.

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 (Scalable 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.

WHAT'S NEW IN THE SIXTH EDITION

NOTES

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

All content and resources have been updated to reflect the current specs and best practices. The following topics are new or have been expanded in the fifth edition:

- Long and unwieldy URLs in the book have been shortened using O'Reilly's URL shortener (***https://orei.ly/***). The new **Appendix D** lists the full URLs for those that have been shortened.
- **Chapter 1** is now titled **Teams and Tools**, and includes information related to job-finding in each discipline.
- **Chapter 2, How the Web Works**, and the rest of the book reflect HTTPS as the default protocol for web pages.
- **Chapter 13, Colors and Backgrounds** has a number of changes:
  - A new section on color spaces provides context for the new ways to specify color in CSS Color Module Level 4.
  - A new section on the OKLCH color mode
  - A new sidebar discussing the `object-fit` and `object-position` properties, which bring background image-like features to images placed in the document with the `img` element.
  - Sections on pseudo-class and -element selectors and external style sheets have been moved to **Chapter 14**.
- **Chapter 14, Selectors (Plus External Style Sheets)** is a new chapter with all of the selectors in one place, including new exercises related to using pseudo-class selectors. It also covers external style sheets.
- The fifth edition's **Chapter 16** has been split into two chapters: **Chapter 17, Getting in Line with Flexbox** and **Chapter 18, Grid Layout**.
- **Chapter 18, Grid Layout** has been significantly reorganized to introduce implicit grids and default grid behavior first, then moves on to exploring explicit, templated grid features.
- **Chapter 19, Responsive Web Design** includes a new section on container queries that base image and font sizes on the width of a containing element rather than the width of the entire viewport. There is a new multi-page sidebar introducing design systems, which have become an essential part of web production. The chapter also includes a new section on responsive and fluid typography techniques.
- **Chapter 21, More CSS Techniques** has new sections on custom properties (a.k.a. CSS variables), CSS Nesting, CSS Layers, and an introduction to CSS Frameworks.

NOTES

- **Part IV, JavaScript for Behavior** has been rewritten from scratch by JavaScript expert, Aaron Gustafson, with the intention of providing a more thorough introduction to writing code:
  - **Chapter 22, Getting Started with JavaScript** introduces basic syntax features and concepts including: statements, dot notation, working with numbers and strings, and declaring variables. Students learn about the JavaScript console in their browsers that can be used to troubleshoot code.
  - **Chapter 23, Functions, Conditionals, and Loops** builds on that foundation, introducing features that reduce redundancy in code, such as functions, `if...else` conditional statements, arrays, and loops.
  - In **Chapter 24, Working with the DOM and Events**, students learn about using the DOM and events by adding an interactive theme toggle button to a page.
  - **Chapter 25, Next-Level JavaScript** describes the JavaScript development environment including JavaScript libraries, web components, frameworks, progressive web applications, and how JavaScript is used outside of the browser in tools like Node.js.
- Topics in **Chapters 26, Web Image Basics** and **Chapter 27, Image Production Techniques** have been reorganized since the last edition.
  - **Chapter 26** begins with information on where to source images. Its primary focus is introducing the various web-appropriate image formats, including an updated and expanded WebP information, and new sections on AVIF and JPEG-XL. In addition, the chapter discusses screen resolution and reference pixels, transparency, and FavIcon production.
  - **Chapter 27, Image Production Techniques** uses a strategy for image production as the starting point for discussing optimization, responsive image set production, and automation.
- **Chapter 28, SVG** includes new exercises for working with SVGs including adding elements, background images, animation effects, and scaling.

No Longer in the Book

The following topics have been removed from the book due to space concerns and or because the techniques have become outdated. Most of the removed sections are now available online as articles at ***learningwebdesign.com/articles/***.

- In **Chapter 7, Adding Images**, the section "Responsive Image Markup" has been moved to an online article.
- In **Chapter 13, Colors and Backgrounds**, the section on adding tiling background colors has been condensed, and the full section from the fifth edition (with exercises) has been moved to an article.
- The fifth edition **Chapter 20, Modern Web Development Tools**, has been removed from this edition due to out of date information, but the sections

"Introduction to the Command Line" and "Version Control with Git" are available as articles.

- In **Chapter 21, More CSS Techniques**, the section "Styling Tables" has been moved to an online article. The "Image Replacement Techniques" and "CSS Sprites" sections have been removed because the techniques are no longer commonly used.
- The fifth edition **Appendix D, From HTML+ to HTML5** has been moved to an article.

NOTES

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.

- 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 HTML 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.
- The complete list of CSS Selectors, Level 4 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.
- 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. A code editor may be used, but is not necessary (I like Visual Studio Code, available for free at ***code.visualstudio.com***). The disadvantage to code editors is that they autocomplete code and autocorrect mistakes, and making mistakes is helpful in the learning process.

NOTES

- Use an up-to-date browser such as Chrome, Safari, Firefox or Microsoft Edge. Avoid Internet Explorer, if possible because many CSS features in the book are not supported. 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). Other options are listed in **Chapters 1** and **26**.
- The source materials provided for the exercises require 22 MB of space and are available at ***learningwebdesign.com/6e/materials***. Students may produce several more MB of files, but in general, the file size of 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).

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.

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**) (see **Note**).

   Activity: 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).

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

   When covering image markup, you could make a detour to briefly discuss images so they are prepared with assets when it is time to create their own sites. **Chapter 26** discusses sources for images and the image formats appropri-

ate for web pages. You may also introduce the SVG format (**Chapter 28**) and SVG markup (the second part of **Chapter 7**) at this time. **Chapter 27** is a deeper dive into web image production and optimization which is appropriate as a resource when students begin creating their own sites and images.

Activity: If students have sites they are working on in class (beyond the exercises in the book), they may begin gathering images and saving them in appropriate web formats. They can resize and optimize them once their page design is finalized.

NOTES

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 15**).

   In addition, I would introduce the CSS reset technique (**Chapter 21**) at this point. Browser defaults can create unexpected results, and removing them will help beginners see accurate results of the style sheets they write.

   Activity: 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.

5. **Add more selector options.**

   Once they are comfortable with writing basic style rules, introduce more selectors as outlined in **Chapter 14**.

6. **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 16**). Chapter 17, Flexbox and Chapter 18, Grid Layout present the primary methods for arranging elements in rows and columns. The code in these chapters gets more complex because the specifications provide so many options, but I've done my best to present the material in as clear a manner as possible.

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

7. **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 19** 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 19-1**.

   This is also an opportunity to cover responsive image markup, which is briefly mentioned in **Chapters 7** and **27**, but is covered in more detailed in the article "Responsive Image Markup" provided online at ***learningwebdesign.com/articles***. **Chapter 28** explains how to make SVGs responsive.

NOTES

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

   Chapters 22 through **24** introduce the building blocks of JavaScript, so students should be able to generally understand what a script is doing when they read through it. The exercises aim to give them a taste of working with scripts and applying them to the parts of a web page.

   Teaching how to write JavaScript independently is beyond the scope of the chapters in this book, so if scripting is a major part of your course, you will likely need additional resources. In other words, although Aaron provides a good introduction to JavaScript syntax and how it is used, it is unlikely students will have the skills necessary to add interactivity to their own projects without guidance.

9. **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 embellishments after the basics are mastered.

   — Tables, forms, and embedded media (**Chapters 8**, **9**, and **10**).

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

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

   — Git and GitHub are useful tools for web professionals. (Supplemental article available at ***learningwebdesign.com/articles***).

CLASS ASSIGNMENT IDEAS

The following class assignments utilize the lessons covered in the book.

TIP: The W3Schools "How To" page provides code for many common page elements that may not have been covered in the chapters, including useful JavaScript examples: ***www.w3schools.com/howto/***

HTML/CSS Projects

- **Personal Portfolio Website:** Students build a simple site to showcase their projects, resume, and a short bio.
- **Landing Page for a Fictional Product or Service:** Focuses on layout, calls to action, and basic visual design.
- **One-Page Résumé or CV Site:** Emphasizes structured content and clean typography

NOTES

Responsive Design Projects

- **Restaurant or Café Website:** Includes a homepage, menu, about page, and contact form.
- **Event Invitation Site:** A responsive digital invitation for a concert, wedding, or meetup

CSS Layout Focus

- **Photo Gallery or Art Portfolio:** Using CSS Grid/Flexbox to arrange images with hover effects.
- **Blog Layout Template:** Focus on typography, margins, and consistent layout.

UX/UI Design Projects

- **Redesign an Existing Website:** Students analyze usability and propose an updated, more modern design.
- **Wireframe to Web Page:** Start with a Figma/mockup and implement it with HTML/CSS.

JavaScript and Interactivity

Because the book is only able to provide a brief introduction to JavaScript syntax, additional study will be required to create fully interactive sites. However, there are many projects available online that can be easily adapted into assignments:

- **Geeks for Geeks "95 JavaScript Projects":** Many projects listed in this article are appropriate for beginners: ***www.geeksforgeeks.org/top-javascript-projects/***
- **Create an interactive to-do list:** Follow this video tutorial on the How to Become a Developer channel: ***www.youtube.com/watch?v=3OqWCGVaOkA***

SAMPLE 15-WEEK SEMESTER PLAN

There are many ways to organize a beginning web design class, so consider this just a starting point for developing a course appropriate for your students and time frame. (*Disclaimer: I have not personally tested this course plan.*)

Weeks 1–2: Intro to HTML & Basic Structure and Image Production

Mini Project 1: "About Me" Page: Create a simple single-page personal profile using semantic HTML and images.

Skills: HTML structure, headings, lists, links, images.

Weeks 3–4: Styling with CSS

Mini Project 2: Personal Portfolio Homepage: Style the previous project with external CSS. Focus on layout and visual design.

Skills: Selectors, fonts, colors, box model, Flexbox basics.

Weeks 5–6: Page Layout & Navigation

Mini Project 3: Multi-Page Portfolio Site: Expand to a 3–4 page website with consistent navigation and footer.

Skills: Reusable CSS, layout systems, linking, site structure.

Week 7: Responsive Design

Mini Project 4: Responsive Café/Restaurant Site: Build a small business website that adapts to mobile and desktop screens.

Skills: Media queries, mobile-first design, viewport, Flexbox/Grid.

Week 8: Midterm Assessment

Project Check-in / Peer Review: Mid-semester project review and code walkthrough with feedback.

Weeks 9–10: Forms & Accessibility

Mini Project 5: Accessible Contact Form or Survey: Create a form that meets basic accessibility standards.

Skills: Labels, tab order, form elements, ARIA, contrast.

Weeks 11–12: Intro to JavaScript & Interactivity

Mini Project 6: Customize the toggle exercise in **Chapter 24** and add it to a page —or— Create a to do list (***www.youtube.com/watch?v=3OqWCGVaOkA***)

Skills: DOM manipulation, events

Week 13: UX/UI & Wireframing

Design Exercise: Wireframe & Redesign Task: Choose a poorly designed site and create a wireframe for a redesign.

Skills: Layout planning, hierarchy, color/contrast, feedback forms.

Week 14: Final Project Build

Final Project (Part 1): Students build a full website based on wireframes/mockups. Choices could include:

— Portfolio site
— Business site
— Event page
— Blog template

Week 15: Final Project Presentation

Final Project (Part 2): Present and submit final site with:

— Accessibility check
— Mobile responsiveness
— Clean design and semantic HTML
— Source code submission with brief reflection

NOTES

CHAPTER REVIEW

NOTES

The chapter summaries that follow provide an overview of the topics covered in each chapter. My goal in this section is to give you a way to get very familiar with the book's content, how it is organized, and key takeaways without needing to read all 900 pages. (You're welcome!)

PART I: GETTING STARTED

Chapter 1: Getting Started in Web Design

Summary

This chapter starts with a description of the various disciplines and roles included under the wider umbrella of "web design" or "web development." 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 various design disciplines, frontend and backend development, content management, 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, browsers, and file management and transfer tools.

Key terms in Chapter 1

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.

NOTES

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.

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.

W3C (World Wide Web Consortium)
: The organization that oversees the development of web-related technologies.

WHATWG (Web Hypertext Application Technology Working Group)
: The organization that took over the development and maintenance of the HTML living standard in 2019.

presentation
: The way the page looks when it is displayed.

JavaScript
: The scripting language used to add interactivity to web pages. It has grown in popularity for all sorts of programming outside the browser as well.

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.

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.

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

Git
: A version control program widely used in web development that makes it easy for teams to collaborate on code.

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

NOTES

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 (`https://`), the domain name (with optional host name), and the pathname to the resource.'
- `https://` indicates that the site is using HTTPS, the secure version of HTTP, that encrypts data between the server and the browser. Sites with the original `http://` protocol are considered insecure by modern browsers and may trigger a warning.
- 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.
- 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 inserted into the page 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 HTTPS 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
: Security 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.

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 and WHATWG) 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** (first 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 or other factors, enabling the layout and its components to adapt to the screen.
- 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. It is critical (and may be required depending on the site) to employ **accessibility** features.
- 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.

Key terms in Chapter 3

NOTES

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

mobile devices
: Smaller devices such as smartphones and tablets, generally with touch screens.

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.

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.

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.

inclusive design
: A design methodology that considers the needs of all users

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.

Exercises in Chapter 3

None.

NOTES

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 HTML 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 **text-level** (previously called **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
- Validating your HTML document

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).

void element (previously "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.

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.

NOTES

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. These 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

Summary

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

Takeaways

NOTES

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`)
- Text-level (a.k.a. inline or phrasing) 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 text-level 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.

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

definition list
: A list 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.

NOTES

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

global attributes
: Attributes that can be used with any HTML element (for example, `id` and `class`). The full list of global attributes is listed in **Appendix B.**

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—for example `$#8212;` for an em dash (—).

Exercises in Chapter 5

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 text-level 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

NOTES

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
- Email (`mailto:`) and telephone (`tel:`) 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

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 directory*
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.

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 using the picture element to specify multiple image options. The `img` element discussion is a necessary component of basic web

NOTES

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**.

design training, and SVG options should be addressed as well, given their rising popularity.

The chapter ends with a brief introduction to responsive image markup (methods for specifying a number of image options for use in responsive layouts). For a more complete explanation, see the article "Responsive Image Markup" available online at ***learningwebdesign.com/articles/***. A logical point to swing back to responsive image markup is when discussing responsive web design (**Chapter 19**).

NOTES

NOTE

*While talking about adding images to a web page, you may also spend time with **Chapters 26**, **27**, and **28** regarding image production and optimization.*

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, WebP or SVG formats to be appropriate for web pages. Cutting-edge formats AVIF and JPEG-XL offer file savings but are less well supported.
- The `img` element and its attributes.
- The purpose and importance of alternative text (using the `alt` attribute).
- How to improve performance with lazy loading (`loading` attribute)

SVG images

- Basic familiarity with the SVG image format as an XML text document.
- Three methods for adding an SVG to the HTML document: `img` element, `object` element, and inline with the `svg` element. (Note that SVGs may also be used as CSS background images.)

The picture element

- The `picture` element allows authors to specify several image options in various formats to take advantage of file savings. The browser uses the first format it supports from a list of options provided by `source` elements. `picture` is an example of a responsive image technique.
- Other responsive image scenarios include:
  - — *Width-based responsive images:* Providing a set of images at different dimensions for use on different screen sizes using the `srcset` attribute with a **w-descriptor** and the `sizes` attribute in the `img` element.
  - — *Screen resolution-based responsive images:* Providing alternative images for use on high-density displays using the `srcset` attribute in the `img` element with an **x-descriptor**.
  - — *Art direction:* Providing slightly different versions of the image with the `picture` element. For example, the image may be cropped differently or omit typography for small screens.

Key terms in Chapter 7

NOTES

bitmap 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 directly embedded into an HTML document with the **`svg`** root element.

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

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.

Chapter 8: Table Markup

Summary

This straightforward chapter covers the markup for structuring "tabular material" (tables).

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`) and 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.

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

NOTES

NOTE

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.

NOTE

*For CSS properties specific to styling table elements, see the article "***Styling Tables***" available at learningwebdesign.com/articles.*

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.

NOTES

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.

NOTE

This chapter covers only the HTML elements related to form inputs. It does not cover form processing code or JavaScript usability enhancements which are beyond the scope of this beginner book.

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 (see **Note**).
- 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 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)
  - Multi-line text-entry field (`textarea`)
  - Special `input` types (`search`, `email`, `tel`, and `url`)
  - Buttons, including submit, 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
- How to add an audio player to a page using the `audio` element
- 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.

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)

NOTES

- 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**: 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**
- An introduction to the extremely useful developer tools built into major browsers

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

NOTES

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 descendants)

descendants
: 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.

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.

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 descendant, ID, and class 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-stretch`, `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.

NOTES

NOTES

- New selector types: **descendant** selectors (targeting elements only when they are contained within certain elements), **ID**, **class**, **universal**, and **combinators** (child, next-sibling, and subsequent-sibling).
- 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 oblique).

combinator (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

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 "Julius Sans 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 descendant, 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 line-height, alignment, capitalization, shadow, color, as well as some new selector types.

Chapter 13: Colors and Backgrounds

Summary

The chapter begins with an explanation of color spaces and color models to provide context for specifying color in CSS. Next, it looks at the various options for providing color values: by standard color name or numerically, using RGB, HSL, and OKLCH color values. The color discussion ends with the properties for adding color to the foreground and background of elements.

The remainder of the chapter includes an introduction to tiling background images (see **Note**) and gradient fills.

Takeaways

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

- How colors are communicated using color spaces and color models.
- 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(% % %)` **Note:** Older browsers require comma-separated values: `rgb(#, #, #)`
- How to specify RGB values with hexadecimal values (`#RRGGBB`)
- Specifying HSL color values: `hsl(#, % %)`

NOTES

NOTE

*See the article: "**Tiling Background Images**" (**learningwebdesign.com/articles/**) for a more in-depth discussion of the various background image properties. This article also includes exercises.*

NOTES

- Specifying OKLCH values for lightness, chroma, and hue: `oklch(% # #)`;
- Adding transparency to RGB and HSL values by including a slash and a fourth alpha value: `rgba(# # # / #)`, `hsla(# % % / #)`, `oklch(% # # / #)`
  Note: Older browsers require specilized `rgba()` and `hsla()` notation for alpha transparency.
- Using the `opacity` property to apply a level of transparency to an element
- 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 by 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

color space
: A way to define and reproduce colors, whether analog (like a collection of paint swatches) or numerically.

color model
: A system for describing colors numerically, such as RGB or CMYK.

gamut
: The range of colors a color model can represent

sRGB (Standard RGB)
: The RGB color space used by the web.

RGB color model
: 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).

OKLCH
: A color model with a wider gamut than sRGB that allows authors to access brighter colors available on high dynamic range (HDR) displays such as Display P3. OKLCH colors are specified using values for perceived lightness, chroma (vividness), and hue.

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.

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.

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

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.

Exercises in Chapter 13

***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 few background images*
A background image is added to the background of the whole page when it is applied to the `body` element.

NOTES

Chapter 14: Selectors
(plus External Style Sheets)

NOTES

Summary

NOTE

This is a new chapter in the Sixth Edition.

This chapter handles the best-supported CSS selectors all in one place. It begins with an overview of the selector types covered in earlier chapters (element, grouped, descendant and other combinators, ID, class, and universal). Then, it covers attribute selectors, pseudo-class, and pseudo-element selectors. The chapter ends with ways to link and import external style sheet documents (*.css*).

Takeaways

- Students should feel comfortable using the selector types listed in the "Selectors You've Learned So Far" section, based on exercises in the previous chapters.
- Elements can be selected by the attributes they contain as well as the values for those attributes. You can even select elements by just part of an attribute value (a **substring**) such as a string of characters at the beginning or end of the value.
- A **pseudo-class selector** selects elements when they are in a certain state or within a particular context.
- **Location pseudo-classes** are related to links. The most common of these selectors are `:link` and `:visited`, pertaining to whether an a element has been accessed.
- **User action pseudo-classes** select elements based on user actions. The most commonly used are `:focus, :hover`, and `:action`.
- These **pseudo-class selectors** are often used together to provide feedback when a user clicks or taps a link.
- **Tree-structural pseudo-classes** target elements based on where they appear in the document tree, or DOM. They include the root selector, child-indexed selectors, and typed child-indexed selectors.
- The `:root` selector applies to the html element in HTML documents
- **Child-indexed selectors** target elements based on their position within their parent element: `:nth-child()`, `:nth-last-child()`, `:first-child`, `:last-child`, and `:only-child`.
- **Typed child-indexed selectors** select elements based on both their position relative to their parent element and by element type: `:nth-of-type()`, `:nth-last-of-type()`, `:first-of-type`, `:last-of-type`, and `:only-of-type`.
- The `:not()` selector is used to select elements that do not match a provided selector. For beginners, it is the most useful of the logical pseudo-class selectors.

NOTES

- **Pseudo-element selectors** can be used to select elements without markup, such as the first letter (`::first-letter`) or first line (`::first-line`) of an element as it displays in the viewport.
- The other type of pseudo-elements select generated content, which is content that browsers apply before (`::before`) or after (`::after`) an element when it is rendered.
- An external style sheet is a plain-text document that includes only style rules and (optionally) comments and a character set (`@charset`) rule
- External style sheets are the most common way to apply style rules because they have the advantages of keeping the style rules in one place and make changing styles across a site more efficient.
- External style sheets use the *.css* file suffix.
- Style sheets can be applied to an HTML document by importing (using an @import rule in the style element or within another style sheet) or linking (using the link element in the head of the document).
- Modular style sheets are a technique for keeping related styles in separate style sheets and combining them in a mix-and-match fashion using `@import.`

Exercises in Chapter 14

***14-1**: Attribute selectors*
Readers will style input elements using a variety of attribute selectors.

***14-2**: Pseudo-class selectors for links*
This exercise gives students an opportunity to apply pseudo-classes to a list of links to improve usability.

***14-3**: Child-indexed pseudo-class selectors*
In this exercise, child-indexed selectors are used to target specific elements within a table for styling.

***14-4**: Typed structural pseudo-classes*
Here typed child pseudo-classes are used to target only the `h3` elements on a page.

***14-5**: The :not pseudo-class*
This quick exercise makes style changes to particular parts of a table using the `:not` logical pseudo-class.

***14-6**: Playing with pseudo-elements*
This provides an opportunity to play with generated content using `::before` and `::after` pseudo-element selectors.

***14-7**: Making an external style sheet*
In this simple exercise, students create an external style sheet and move the styles they've written so far into it.

Chapter 15: Thinking Inside the Box
(The CSS Box Model)

NOTES

Summary

Chapter 15 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

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

Takeaways

In **Chapter 15**, 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 new **logical properties and values** that are based on the writing direction of the language in which the document is written (e.g. `block-size` and `inline-size` which are equivalent to `width` and `height`, respectively)
- 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 15

NOTES

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

logical (flow-relative) properties and values
: Properties and values that are contextually based on the writing direction of the document. The keyword "inline" refers to the writing direction and "block" refers to the direction that is perpendicular to inline.

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 type
: 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 15

15-1: *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 (see **Note**). 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.

15-2: *Border tricks*
This exercise lets readers try out borders and border-radius properties on the bakery page.

15-3: *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 the exercises in **Chapters 16**, **17**, *and* **19**.

Chapter 16: 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.

NOTE

Floating and positioning were once the only options for achieving columned page layouts; however, that approach is obsolete now that Flexbox and Grid Layout are widely supported.

Takeaways

In **Chapter 16**, 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). See **Note**.

NOTE

*See the article "CSS Shapes" for additional details on wrapping options (**learningwebdesign.com/articles/**).*

NOTES

- 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 16

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

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. Modern browsers support `display: flow-root` for float containment, but older browsers must use the "clearfix" workaround.

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 16

16-1: *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.

16-2: *Adding shapes around floats*
Students are given the opportunity to add shaped wraps around the bread and muffin images.

16-3: *Absolute positioning*
This exercise uses absolute positioning to add an award graphic to the top of the home page.

16-4: *Fixed positioning*
The positioning type for the award graphic is changed from absolute to fixed, and the different behavior can be observed.

Chapter 17: Getting in Line with Flexbox

Summary

Chapter 17 covers the CSS Flexbox display type for arranging elements into rows or columns and specifies how and whether they are allowed to "flex" to fill the available space in the layout.

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 menu bars, complex header components, and gallery-like spaces where multiple images or product "cards" flow into a content area.

Takeaways

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

- 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

NOTES

NOTE

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

NOTES

- 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** (including **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

Key terms in Chapter 17

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

NOTES

Exercises in Chapter 17

17-1: *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.

17-2: *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.

17-3: *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.

Chapter 18: Grid Layout

Summary

Chapter 17 covers CSS Grid Layout, which 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.

Both Flexbox and Grid offer ways to allow content to expand or shrink into available spaces, making them excellent solutions for responsive layouts.

NOTE

Make it clear that Grid Layout 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 techniques.

Takeaways

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

- 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**)
- The default behavior of elements with their `display` set to `grid`

- Browsers generate enough rows or columns to accommodate all of the grid items in the grid container element by default. The grid that results from this default behavior is called an **implicit grid.**
- Using `grid-auto-rows`, `grid-auto-columns`, and `grid-auto-flow` properties to set parameters for implicit grid behavior
- By contrast, an **explicit grid** is one in which you specify exactly how many rows and columns the grid should contain, and (optionally) where each grid item should be placed within that grid.
- How to define explicit 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`)
- The `grid` shortcut property for explicit and implicit grids
- Alignment properties for aligning grid tracks in a larger container (`justify-content`, `align-content`)
- Alignment properties for aligning grid item elements within grid cells when there is extra room (`justify-items`, `align-items`, `justify-self`, `align-self`)
- Adding space between cells with `gap`

Key terms in Chapter 18

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.

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.

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 Layout 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.

NOTES

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).

implicit grid
: A grid resulting from 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

explicit grid
: A grid defined with a specific number of rows and columns

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

subgrid
: A display mode that allows a nested grid element to inherit the grid column and row widths from its parent

Exercises in Chapter 18

18-1: *Working with an implicit grid*
Students will observe the default grid behavior by applying basic grid properties to a list of terms and definitions.

18-2: *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).

18-3: *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.

18-4: *Playing with alignment*
In this exercise, students are given some starter documents and suggestions for trying out the various alignment properties. Unlike other exercises, this one is about experimentation and getting a feel for the properties without an explicit goal.

18-5: *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 19: 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 or containing elements, enabling layouts to adapt to the device on which they display. Today, it is the default approach for building websites.

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 19**, students will learn the following:

- The core principle of responsive web design: 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 fluid layout, 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.
- Fluid layout approaches allow the page to resize to fit the available width of the viewport. Fixed-width layouts remain at a specified pixel width.
- 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 the "**Responsive Image Markup**" online article) 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 */ }`.
- How to use **container queries** (`@container`) to apply styles based on the width of an element's **containing element** (rather than the entire viewport).
- **Container query units** are similar to viewport units, but they are based on the size of a containing element. For example, 1 `cqw` equals 1% of the query container's width.
- 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.

- How a design system ensures consistency across a site's design and code
- 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 19

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.

intrinsic design
: An updated term for responsive design that acknowledges newer CSS layout features such as Flexbox, Grid Layout, container queries, and more.

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

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

container query
: Applies styles based on the size of an element's containing element

containing context
: Created when you apply the `container-type` property to an element, making it the element tested by a container query

container query length units
: Units that specify size based on the size of the containing element. 1 container query unit equals 1% of the containing context.

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

NOTES

design system
: A repository of reusable page components as well as guidelines for their use as well as (optionally) other documentation pertaining to brand guidelines and aspects of a site

component library
: A collection of UI components, including the code for implementing them. They are the core feature of design systems.

style guide
: Documents the rules, standards, and assets that support an established brand identity

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

conditional loading
: A technique that delivers the most important information to small-screen users with links to additional content

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.

Responsive typography
: Changes font sizes at specified breakpoints

Fluid typography
: Font size is based on the size of the viewport or containing element, allowing it to resize smoothly and proportionally at sizes between breakpoints

device lab
: A collection of devices used for testing

virtual device
: Software that mimics the features and behaviors of a mobile device or browser

Exercises in Chapter 19

19-1: *Making sure images fit*

Here students will try setting images to `max-width: 100%` to see how it affects the image in the layout.

19-2: *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 17**. 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.

NOTES

19-3: *Container queries*

This exercise provides experience changing font size and flex direction based on the size of a containing `div`.

Chapter 20: Transitions, Transforms, and Animation

Summary

Chapter 20 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 keyframe animation.

Takeaways

In **Chapter 20**, 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, resize, and skew an element using the `rotate()`, `translate()`, `scale()`, and `skew()` function values, respectively.
- The newer dedicated transformation properties: `rotate`, `translate`, and `scale`.
- How to combine `transition` and `transform` to smooth out transformations between states.

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 20

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.

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 20

20-1: *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.

20-2: *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 21: More CSS Techniques

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 21**, students will learn the following:

- Tips and tricks for styling form elements, including adding styles to a sample sneaker order form.
- Using a **CSS reset** or **normalizer** to clear out browser default styles, creating a "clean slate" for our own style sheets.
- **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.
- **Vendor prefixing**, an obsolete method for testing new CSS features that may be useful if there is a valid need to support certain properties and values in old browsers
- **CSS custom properties** (a.k.a. CSS variables) that remove redundancy in code and make updates more efficient
- A quick introduction to **CSS processors**, Sass and Less (Variables were first introduced in Sass and were standardized due to their popularity.)

NOTES

- **CSS Nesting**, which allows you to write nested style rules that match the structure of the document. This alleviates the need to repeat selectors multiple times.
- **CSS Layers**, which is a tool for explicitly declaring which sets of rules should have higher priority than others in order avoid conflicts.

Key Terms in Chapter 21

user agent style sheet
: The default style sheet used by the browser

CSS reset
: A collection of style rules that override all of the user agent styles and sets a neutral starting point.

normalizer
: A type of reset that preserves some basic styles (such as space around headings and paragraphs) with the aim to create consistency between browsers. It is not as drastic as a CSS reset.

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.

feature flag
: A browser mechanism that allows developers to toggle on access to features that are not yet ready for public release

vendor prefix
: An obsolete method browsers used to start experimenting with newer CSS properties. The browser-specific prefix indicated that the implementation was proprietary and a work in progress.

CSS custom properties (variables)
: A syntax for using variables in CSS. Like variables in traditional programming, you declare a custom property and give it a value. When that value is needed throughout the style sheet, you can refer to it by the variable name.

CSS nesting
: A syntax for nesting style rules in a way that mimics the HTML structure, with the goal of reducing redundant selectors

CSS Layers
: A system that allows authors to give names to sets of rules and assign their priority, resulting in better control over the cascade.

NOTES

Exercises in Chapter 21

None.

Part IV: JAVASCRIPT FOR BEHAVIOR

Chapter 22: Getting Started with JavaScript (Statements and Variables)

Summary

This chapter is a gentle introduction to writing JavaScript code. It assumes that the reader has no coding experience, and introduces basic concepts and terminology such as statements, functions, dot notation, data types, and comments. Students will get familiar with using numbers and string values in scripts and will learn how to declare and use variables. The chapter also looks at ways to add JavaScript to a page (embedded and external) and closes out with an introduction to the JavaScript console for checking your code.

Takeaways

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

- What JavaScript is and what it's used for
- An introduction to the components of a simple script, including statements, comments, operators and functions
- How **dot notation** is used to attach **properties** (attributes) and **methods** (functionality) to JavaScript objects.
- The camel case capitalization convention used in JavaScript in which the first letter of every word is capitalized, with the exception of the first word (for example, `querySelector()`)
- Conventions for writing statements
- How to handle number values, including **math operators**, **comparison operators**, and **assignment operators**
- How to work with text strings, including multiline strings, strings with quotes and concatenating (combining) text strings
- Strings must always be written on one code line
- A look at data type **coercion**, in which one value type is turned into another in certain situations, such as when concatenating a string and a number (the number is "coerced" to a string)

NOTES

- An introduction to using and declaring variables with `let` and `const`
- Variables can be given a value when they are declared or be left **undefined**
- Some tips on how to name variables
- Adding external *.js* files with the script element, including why it's important to add JavaScript after the elements in the document that the script affects.
- An introduction to the **JavaScript console** in browsers' developer tools. It is a useful tool for typing in code and seeing how the JavaScript engine responds, which makes it great for troubleshooting.

Key Terms in Chapter 22

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

line comment
: A comment indicated by two forward slashes (`//`) which is used for short comments that stay on one line

block comment
: A comment that spans multiple lines, indicated by `/*` at the beginning and `*/` at the end of the comment

camel case (or camelCase)
: A capitalization convention in which the first word of every word, except for the first word, is capitalized; for example: `innerWidth` and `querySelectorAll()`

dot notation
: A shorthand syntax for accessing the properties and methods available for an object that uses periods (a.k.a. dots) to separate nested features

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

property
: An attribute or quality of an object, such as the `length` of a text string or array

method
: Functionality associated with an object, such as the `querySelector()` method that is available for the `document` object (`document.querySelector()`).

math operator
: A character that indicates a mathematical operation, such as `+` for addition, `*` for multiplication and `/` for division

comparison operator
: A character that compares one value to another, such as `<` for "less than" and `!==` for "not equal to"

NOTES

assignment operator
: A character that assigns a value. The most simple is = that assigns a value to a variable. Others provide a shorthand for math operations used to update a variable's value; for example: `+=` that adds a numeric value to the current numeric value of a variable.

incrementing operator
: Adds 1 to the current value (indicated by `++`)

decreasing operator
: Subtracts 1 from the current value (indicated by `--`)

string
: Text content, usually wrapped in quotation marks (or "quotes" as they are generally referred to in this context). Strings can include any characters, including letters, numbers, punctuation, hidden characters, and spaces

escape
: A way of representing a character with code

concatenation
: Combining two strings with the + operator

coercion
: The conversion of a value from one data type to another, such as a number being treated as a string when it is concatenated to a string value

newline character
: A hidden character that introduces a line break where it is inserted in a text string. It is indicated by the `/n` character escape, and is useful for introducing a shift to a new line in long string values.

refactor
: Rewriting code in order to improve some aspect of it

variable
: A named object used to store, retrieve, and manipulate values in code

value
: The data associated with a particular variable

variable declaration
: A statement used to create a new variable. Declarations are indicated by the `let`, `const`, and `var` keywords.

initializer
: The (optional) portion of a variable declaration that includes the equals sign (`=`) and the assigned value.

undefined variable
: A variable that has been declared in order to reserve the name, but that has not been assigned a value

mutable variables
: Variables for which the value can be changed (such as those created with `let` or `var`)

immutable variables
: Variables for which the value cannot be changed (such as those created with `const`)

embedded scripts
: Scripts that are written directly in the `script` element

external scripts
: Scripts written in a separate document with the *.js* element that are loaded into the page with the `script` element

console
: A command line interface in your browser's developer tools that can execute snippets of code. It is useful for debugging.

NOTES

Exercises in Chapter 22

22-1: *Your first script*
This one-statement exercise changes the background color of the page. It serves as the basis for upcoming exercises.

22-2: *Paint by numbers*
This exercises uses number values and comparison operators to apply background colors based on the width of the browser window.

22-3: *Combining strings*
This refactor of the script in EXERCISE 22-2 uses concatenation to string together a style declaration. It also provides a first glimpse of variables and conditional statements.

22-4: *Everything counts*
Here students will get experience with declaring and calling variables in a slightly more complicated script that calculates reading time and appends a phrase to the paragraph on the page.

22-5: *Coding outside the page*
The script from EXERCISE 22-4 is moved to an external *.js* file so it can be used in multiple HTML documents.

22-6: *Using the console*
This exercise provides a little experience typing directly into the JavaScript console in the browser.

22-6: *Rise to the challenge*
Two challenges are given that test readers' knowledge of the concepts in the chapter. Like the title says, these are more challenging and provide fewer instructions.

Chapter 23: Functions, Conditionals, and Loops

NOTES

Summary

This chapter builds on the foundation of JavaScript syntax basics with a collection of JavaScript concepts that reduce redundancy in code and result in more functionality in fewer lines: functions, conditionals, and loops

It starts with how to declare, name, and call functions, including how to pass argument values to functions and return values from them. The concept of function "scope" is introduced, including an introduction to namespacing. The "Conditional Statements" section covers the logic of `if...else` statements for basing functionality on test conditions, and introduces the Boolean (`true`/`false`) value type, logical operators for testing multiple conditions, and the concept of "truthy" and "falsy" values. Finally, arrays (lists of values) are introduced as a foundation to looping using the basic `for` loop.

Takeaways

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

Functions

- How to **declare a function** with the `function` keyword, both with and without a parameter (a placeholder for an expected value)
- **Calling a function** from within a script
- A more detailed explanation of **methods**, which are functions that work only in the context of the object they are attached to.
- Passing values (called arguments) to a function
- Variable scope (the context in which a variable and its value are declared and available to the script), including **function scope, block scope**, and **global scope**
- Using values **returned** by a function

Conditional Statements

- `if` statements run code only if a certain test criteria is met (i.e. returns the Boolean value `true`). If the test does not pass, nothing happens.
- `if...else` statements run one bit of code if the test returns `true` and runs another block of code if the code does not pass (returns `false` or a "falsy" value).
- How to test for multiple conditions using the **logical AND** (`&&`) and **OR** (`||`) **operators**
- How to test for the opposite of a statement with the **NOT operator** (`!`)
- An introduction to the JavaScript concepts of **truthy** and **falsy**, values that are not explicitly true or false, but can be interpreted that way in certain contexts

- How to use conditionals to write programs **defensively,** in a manner that prevents errors

Arrays and loops

- Using arrays for lists of values, including how to access, change, and add items in the array by using their **index** numbers
- How to use a `for` loop to run a block of code multiple times, such as to perform the same function on every item in an array or `NodeList`

Key Terms in Chapter 23

DRY (Don't Repeat Yourself)
: An approach to programming that avoids repetition in the code

parameter
: A placeholder in a function declaration indicating the function requires information when it runs

argument
: The value that gets passed to a function

call (a function)
: The common term for running a function by writing its name in a statement

scope
: The context in which a variable or function is declared and is available

function scope
: Variables declared within a function exist only within the context of that function

block scope
: Variables declared within a block statement exist only within that block (the exception is variables declared with the `var` keyword)

global scope
: Variables declared outside of blocks and functions are available everywhere

namespace
: An way to organize code to avoid naming collisions. It involves creating a global object that acts as a container for related functions, variables, and other objects.

conditional statement
: Allows code to be run only if a test condition is met

Boolean values
: A data type that represents one of two values: `true` or `false`

logic gate
: Another term for conditional statements

NOTES

NOTE

*There is a supplemental article, "***Web Storage***," for this chapter available online (**learningwebdesign.com/articles/***). It adds web storage features to the theme toggle button so that the user's theme preference is stored between sessions.*

NOTES

logical operators
: Operators used to make logic decisions based on multiple conditions. They include AND (`&&`), OR (`||`), and NOT (`!`)

falsy
: A value that is not false but can be interpreted that way in certain contexts such as conditional statements. The number 0, the `null` value, and empty string (""), and an `undefined` value are all considered "falsy".

generalizing
: Writing a script in a way that is reusable and able to handle a wider range of use cases

array
: A JavaScript object used to store a collection of ordered data values.

index
: The numeric position of an array item in the list, starting with 0 (zero)

bracket notation
: A way to access object properties using square brackets `[]`. In the context of arrays, it selects an item in the array by its index; for example: `myArray[0]` accesses the first item in an array called `myArray`)

loop
: A type of statement that allows code to be executed repeatedly, which is useful for repetitive tasks

Exercises in Chapter 23

The exercises in **Chapter 23** build on the script started in **Chapter 22** for calculating the time it takes to read a paragraph and appending "(Reading time: x minutes)" to the end of the paragraph. The code will be refactored to take advantage of functions, conditionals, and loops.

23-1: *Functional programming*
The statements written in EXERCISE 22-4 get converted to a function so it can be called whenever it's needed.

23-2: *Passing values around*
Here, the function from EXERCISE 23-1 is broken down into smaller, single-task functions. It provides experience declaring functions, calling them, and using the value they return in other variables and functions.

23-3: *Making sure the argument is an element*
This exercise uses an `if...else` statement and logical operators to determine whether the argument is a paragraph before running the `insertReadingTime()` function. Note that it begins by getting the code caught up with changes made in the "Flexible Functions" section.

23-4: *Loop the loop*
Students get the chance to use a for loop to run `insertReadingTime()` on every paragraph on a page (not just the first one).

23-5: *Challenge yourself!*
This exercise includes two more advanced challenges. In the first, readers are asked to use the console to compare the number of p and div elements on a page and log the results. The second uses arrays, loops, and concatenation to display a list of things in the console with a * character inserted before each item.

NOTES

Chapter 24: Working with the DOM and Events

Summary

With the basics of the JavaScript syntax itself established, this chapter focuses on tasks common to web development: manipulating page content by using features in the DOM, and using JavaScript events to trigger scripts. Students will use each new skill in the chapter to create a toggle button that switches between light and dark visual themes. The intent of the chapter is to give readers a general idea of how JavaScript is used to create UI elements and interactivity. It also addresses the concepts of progressive enhancement and accessibility as they pertain to JavaScript code.

Takeaways

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

- A refresher on **progressive enhancement** (a development technique that starts with a baseline experience that works for everyone) in the context of JavaScript
- An introduction to the **DOM (Document Object Model)** as an interface for accessing the elements, attributes, and content of a document. It makes markup programmable.
- A closer look at the document object, which represents the current page
- How to create (`createElement()`), add (`appendChild()` and `insertBefore()`), and remove (`removeChild()`) **element nodes**
- How to create **text nodes** in a more precise way using `createTextNode()`
- The difference between `innerHTML` and `innerText` for adding text content to an element
- Creating, updating, and removing **attribute nodes**
- Efficient methods for handling the `class` attribute (`classList.add()`, `classList.contains()`, `classList.remove()`, and `classList.toggle()`)
- Scripts can be triggered to run by user actions called **events**, such as clicking a button, scrolling, hovering the pointer, or typing.
- Events can be handled with **event properties** (such as `onclick`) or by using the `addEventListener()` method

- How to use ARIA (Accessible Rich Internet Applications) attributes to make interactive elements accessible. For example, the `aria-pressed` attribute indicates whether an element like a button has been pressed.

NOTES

Key Terms in Chapter 24

progressive enhancement
: An approach that prioritizes building a baseline experience that works for all users, then enhancing the experience with markup, CSS, and JavaScript when chosen features are available

static content
: HTML content that stays the same for all users. By contrast, dynamic content is generated on the fly by the server or JavaScript in the browser

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
: Components in a web page as seen by the DOM, including elements, their content, attributes, and comments. The relationship of elements forms a "tree" with branches of nested nodes.

event
: A user action or occurrence in the browser itself that can be used as a trigger for running JavaScript code

event handler
: Code that runs when a specific event fires

event propagation
: What happens behind the scenes when an event fires. The three phases are **capture**, **target**, and **bubble**.

event delegation
: An approach that "listens" for events higher in the DOM tree, resulting in more efficient and scalable code

anonymous function
: A function that has not been given a name (commonly used as event handlers)

named function
: A function that has been given an a name when it is declared

persist
: Information from a previous session that remains available when a user returns to a web page

Exercises in Chapter 24

NOTES

24-1: *Building a button*
This is the first step in creating the theme toggle button in which the element is created and inserted into the page.

24-2: *Adding attributes*
Here we add attributes to the button element we created in EXERCISE 24-1 that will be used to style the button like a switch control.

24-3: *Now it clicks!*
This exercise adds an event listener to the button that toggles between light and dark themes by adding or removing a `dark` class value in the `html` element. The page now switches between the dark and light themes, but the label on the switch does not change.

24-4: *Accessibility refactor*
A new function is added that maintains the proper state of the toggle button and applies an appropriate label for users with screen readers.

24-5: *Challenge yourself*
This challenge tests students' ability to create elements and add them to a page.

Chapter 25: Next-Level JavaScript

Summary

Chapter 25 is a collection of topics related to the JavaScript ecosystem that aspiring web developers should be familiar with. It begins with a look at JavaScript libraries—collections of code (typically functions) that can be reused to make development more efficient. Next, it looks at web components that allow you to create custom, reusable elements for building pages and interfaces. Professional web developers will inevitably be required to use one of the popular JavaScript frameworks, such as React or Angular, and this chapter provides an introduction to what they are and how they differ. It also looks at some ways JavaScript is used in the production workflow for tasks beyond the browser, such as Node.js. The chapter closes with an introduction to progressive web apps (PWA) that are built using web technologies but function more like standalone applications.

Takeaways

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

JavaScript Libraries

- **JavaScript libraries** are collections of reusable code (usually functions) that can be reused for common tasks
- Libraries often include code that fills in gaps in standard browser functionality or find ways to make certain tasks more efficient

NOTES

- You can create your own library or use one of the countless libraries made available
- Ways to implement the code from a library
- An introduction to popular JavaScript libraries, including jQuery, GSAP (for animation), and more
- **JavaScript modules**, which let you manage code in separate files and pull it together when you build your program using the `export` and `import` keywords.

Web components

- **Web components** use a combination of JavaScript and HTML to build custom, reusable elements
- Code for components is fully encapsulated in a portable JavaScript file.

JavaScript frameworks

- JavaScript framework is like a toolkit of prebuilt code libraries that can make building a site more efficient
- They are typically made up of **components** that can be assembled to create pages. More complex components are created out of simpler ones.
- Frameworks generally use JavaScript to generate the HTML and CSS for a site, in addition to functionality
- **TypeScript** is a version of JavaScript that is more strict about data types, which can help to avoid errors.
- The pros and cons of using a framework
- Introductions, with code examples, of the most popular frameworks as of this writing: React, Angular, Vue, and Svelte

JavaScript beyond the browser

- Programming environments such as Node.js and Deno use JavaScript for tasks beyond simply web content in the browser
- They are examples of JavaScript runtimes, which is an environment that allows JavaScript code to run. Different runtimes offer different functionality. The browser is one example of a runtime that allows access to the DOM.
- Node.js is a runtime that provides the ability to access the computer's file system, for example to update and create directories. It can also be used to create programs that run in the terminal (command-line) window or on the server.
- Deno addresses some limitations of Node.js, including limiting access to the machine the code is running on for security reasons

Progressive web apps (PWAs)

NOTES

- Progressive web apps provide functionality like traditional apps, but they are created with web technologies and live on the web at a URL.
- Examples of the capabilities of PWAs
- The core technologies of PWAs are:
  - — HTTPS for security,
  - — A web app manifest written in JSON that stores data about the app, and
  - — Service Worker, a script that runs in the background and manages network processes such as requests, caching, offline access, and push notifications.

Key Terms in Chapter 25

library
: A collection of code—typically functions—that can be reused for common tasks

JavaScript module
: A file that contains JavaScript code that can be imported into other files or modules, used to help organize and maintain code

web components
: A collection of technologies that allow you to create custom, reusable, single-purpose elements for building web pages and interfaces

custom element
: An HTML element that you define and provide functionality to with JavaScript

JavaScript class
: A template for creating a JavaScript object that encapsulates data and behavior

routing
: What JavaScript frameworks call navigating between pages

framework
: A coding environment that works as a toolkit for building sites and interfaces from pre-built components. They can make site development more consistent and efficient.

component (in a framework)
: A discreet piece of reusable code that produces HTML and may also include its relevant CSS and JavaScript

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.

strictly typed language
: A language for which you need to declare the type of value you are assigning. Typescript is an example of a strictly typed language.

virtual DOM (in React)
: A copy of the DOM that React stores in memory that limits calls to the actual DOM (thus improving performance)

decorators (in Angular)
: Hints in the code that tell Angular the role and characteristics of classes,so it knows whether to treat them as components, modules, etc.

template property (In Angular)
: Similar to a variable that stands in for a bit of dynamic content, indicated by curly braces (`{{ }}`), commonly often referred to as mustache syntax

compiler
: A program that converts one programming language to another; for example, code written with the Svelte syntax into standard HTML, CSS, and JavaScript

JavaScript runtime
: An environment that is able to run JavaScript that includes the engine with additional tools and functionality

Exercises in Chapter 25

(None)

Part V: WEB IMAGES

Chapter 26: Web Image Basics

Summary

Chapter 26 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, WebP, and GIF) as well as up-and-comers, AVIF and JPEG-XL (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 resolution, image resolution, and reference pixels. There is also a section about working with transparent images. Finally, it takes a look at favicons (the little icon that shows up in the browser window) and how to create them.

NOTES

NOTE

Chapter 26 *focuses on bitmap formats. SVG is handled separately in* ***Chapter 28, SVG****.*

Takeaways

NOTES

In **Chapter 26**, 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, using an AI image generator, or hiring a professional.
- A comparison of JPEG, PNG, WebP, GIF, AVIF and JPEG-XL formats
- JPEGs are **Truecolor images** with a **lossy compression** scheme that throws out data in order to make images as small as possible. Because JPEG compression 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 newer **WebP** format offers lossless or lossy compression and alpha transparency at smaller file sizes. It is a good alternative to JPEG for photos and to PNG-24 for images with transparency.
- The original web image format, **GIF**, is an indexed, 8-bit format that is also capable of animation. GIF has lost ground to PNG-8 and WebP formats.
- The **AVIF** image format uses compression scheme from the AV1 video codec and results in images that are 50% smaller than JPEG. They excel at preserving sharp edges and fine details.
- **JPEG-XL** is a versatile format that offers lossy or lossless compression, support for wide color gamuts, alpha transparency, animation, and better handling of sharp edges than JPEG. It is poorly supported as of this writing.
- How to choose the best file format for different image types.
- A refresher on using the `picture` element to serve multiple versions of an image in different file formats
- How bitmapped images are displayed on screens, and how the screen's pixel density affects the display of images.
- High-density displays use a unit of measurement called a **reference pixel** for layout so measurements are independent of the pixel density of the display.
- The difference between **binary (1-bit) transparency** and **alpha transparency**
- How to create favicons, both the traditional way and as a larger icon set for use across different device types.

Key Terms in Chapter 26

NOTES

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

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

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

Truecolor (also 24-bit 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 (although alpha is poorly supported by image tools)

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.

WebP
: An open source format with lossless or lossy compression and alpha transparency that generally compresses photographic images smaller than JPEG

NOTES

AVIF
: A newer image format that can compress photographic images smaller than JPEG and WebP while maintaining sharp edges and details. It also features alpha transparency.

JPEG-XL
: A newer, multi-feature image format that compress photographic images smaller than JPEG and WebP while maintaining sharp edges and details. It supports alpha transparency, animation, and HDR support. It can also be used for high-quality image editing and printing.

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

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

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

pixel density
: The resolution (ppi) of a screen

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

NOTES

device pixel ratio (DPR)
: The ratio of hardware pixels to reference pixels, for example, a 2x display has two hardware pixels in the space of one reference pixel

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

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.

touch icon
: An icon used to represent sites installed as applications (Progressive Web Apps) on iOS and Android

web manifest
: A document that stores data in the JSON (JavaScript Object Notation) format

Exercises in Chapter 26

26-1: *Comparing JPEG and PNG formats*
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.

26-2: *Take WebP for a spin*
The same two images are saved in WebP format at various compression levels and compared to the resulting JPEG and PNG images from EXERCISE 26-1.

26-3: *Creating a transparent image*
In this exercise, students create an image with a soft edge and save it with the transparency preserved as PNG-24 (alpha transparency), PNG-8 (binary), and GIF (binary). The resulting images are compared by placing them on a web page against a colored background.

Chapter 27: Image Production Techniques

NOTES

Summary

The chapter starts with an image production strategy flowchart that walks designers through a strategy to follow when producing images for responsive web sites. It serves as the jumping-off point for discussions on optimization, responsive image production, efficient production techniques, and automated image services.

Takeaways

In **Chapter 27**, students will learn the following:

- 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 optimize images:
  - — During the design and export process
  - — Using post-export compression tools
  - — Using automated services
- Special tips for optimizing JPEG, PNG-24, PNG-8, GIF, and WebP by taking advantage of their compression schemes
- An overview of optimization tools that can make files exported from image editing programs even smaller
- Strategies for creating sets of images for use in responsive layouts, including a brief overview of the responsive image HTML markup that serves the appropriate image from the set based on the user's viewing environment
- General tips for creating web images, including:
  - — Using a vector images whenever possible
  - — Starting with a high-quality original image
  - — Embedding large-scale bitmaps into the design tool used for page layout design
  - — Exporting multiple image sizes at once
  - — Using AI to upscale images
- An overview of image automation services (Image CDNs) that handle image set generation and serving the appropriate image

Key Terms in Chapter 27

NOTES

bit depth
: The number of bits used to store color in an image, which determines the number of colors the image can contain

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

upscaling
: Taking a bitmapped image and resizing it larger (usually resulting in blurry or lower image quality). There are now AI tools that can upscale images with more accuracy.

content delivery network (CDN)
: A globally distributed network of servers that cache your content so it has less distance to travel, resulting in increased page speed and less vulnerability to downtime or attacks.

Image CDN
: Provides specialized image optimization, manipulation, and delivery services

Exercises in Chapter 27

27-1: *Optimize some images*
Two types of images are compressed using the free ***squoosh.app*** and the "Save for Web" feature in Photoshop. The resulting file sizes and image quality are then compared.

Chapter 28: SVG
(Scalable Vector Graphics)

Summary

This chapter introduces the SVG image format which has become a critical tool for responsive design because, as vectors, they can scale without loss of quality. Readers get an introduction to the SVG markup language (an application of XML), 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 28**, 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.

NOTES

- 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 bitmap 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.
- 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 and CSS can be used to add interactivity and animation to SVG images.
- Tools for creating SVGs including vector illustration tools, user-interface design tools (like Figma), and SVG-specific tools.
- 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 28

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 bitmap 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.

user space
: See **viewbox**.

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 (also "responsive 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.

NOTES

Exercises in Chapter 28

NOTES

28-1: *Working with SVG shapes*
This exercise provides an opportunity to draw shapes by writing out the code in an *.svg* document. Students will draw eyes, a nose, and mouth for "Bobby the Robot."

28-2: *Adding an oval background image*
Here we add a circuit-board background image to the SVG and clip it to an oval shape.

28-3: *Adding a glow effect*
A filter (blur) is used to give Bobby a glowing light.

28-4: *Using symbols*
Bobby's eyes are converted to symbols using symbol and use, to show how editing a symbol affects all of its instances.

28-5: *Adding a little animation*
Here keyframe animation is used to make the light pulsate on and off.

28-6: *Scaling SVGs*
The steps in this exercise lead students through experimentation with scaling the SVG.