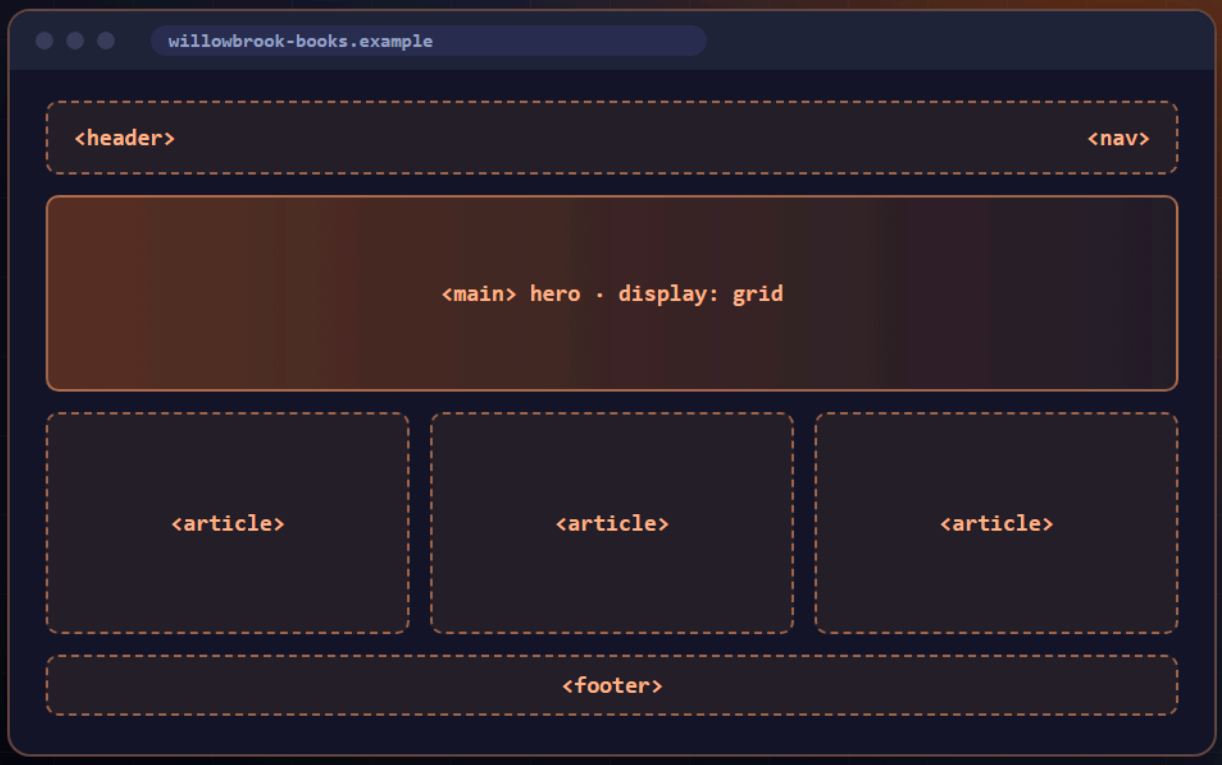


THE COMPLETE PRACTICAL GUIDE

HTML and CSS from Scratch

Build Modern, Responsive, Accessible Websites

Semantic HTML · Flexbox · Grid · Responsive Design · Accessibility · Modern CSS · Publishing

28 CHAPTERS · 4 APPENDICES · ZERO TO LIVE SITE

THENEXBOOKS

HTML and CSS from Scratch

The Complete Practical Guide to Building Modern, Responsive, Accessible Websites

From Your First Tag to a Published Website

First Edition

THENEXBOOKS

HTML and CSS from Scratch
The Complete Practical Guide to Building Modern, Responsive, Accessible Websites

First Edition, 2026

Copyright © 2026 thenexbooks (thenexbooks.gumroad.com). All rights reserved.

No part of this publication may be reproduced, distributed, or transmitted in any form or by any means without the prior written permission of the publisher, except for brief quotations in reviews and certain other non-commercial uses permitted by copyright law. Code examples may be used freely in your own projects.

The information in this book is provided "as is" without warranty of any kind. While every precaution has been taken in its preparation, the publisher assumes no responsibility for errors or omissions, or for damages resulting from the use of the information contained herein. Web standards, browser support, and hosting services change over time; always check current documentation such as MDN Web Docs.

GitHub, Netlify, Cloudflare, Visual Studio Code, Chrome, Firefox, Safari, Edge, and other product names are trademarks of their respective owners. Their use in this book is for identification only and does not imply endorsement.

Willowbrook Books, the running example in this book, is fictional. Any resemblance to real businesses is coincidental.

Preface

Every website you have ever visited — a news site, an online store, a portfolio, a web app — is built on the same two languages. **HTML** describes what the content is: headings, paragraphs, images, links, forms. **CSS** describes how it looks: colors, fonts, spacing, and layout on every screen size. Frameworks, site builders, and design tools come and go, but underneath them all is HTML and CSS.

Learning them well is one of the most useful skills you can have on the web. It lets you build your own site without monthly builder fees, customize the tools you already use, collaborate with designers and developers, and take your first real step into web development. And modern CSS is more capable — and more pleasant — than ever: layouts that used to require hacks now take a few lines of Grid or Flexbox.

This book starts from zero. You do not need any prior coding experience. We explain how the web works, write our first page, and then build up — structure, accessibility, styling, layout, responsive design, modern CSS features, and reusable components — until you have designed, built, tested, and published a complete multi-page website.

Who This Book Is For

- **Complete beginners** who want to build websites and understand how they work.
- **Students and career changers** starting a path into web development or UI design.
- **Designers, marketers, and content creators** who want to build and adjust pages themselves.
- **Small-business owners** who want full control of their own website.
- **Self-taught developers** who learned in fragments and want a solid, modern foundation.

What You Will Learn

- How browsers, servers, HTML, CSS, and JavaScript fit together.
- Semantic HTML for text, links, images, lists, tables, forms, and media.
- Accessibility fundamentals so your sites work for everyone.
- CSS selectors, the cascade, specificity, the box model, units, colors, and typography.
- Layout with normal flow, positioning, Flexbox, and Grid.
- Responsive design with media queries, fluid sizing, `clamp()`, and container queries.
- Custom properties, dark mode, transitions, animations, and modern selectors like `:has()`.
- Reusable components, CSS organization, performance, SEO basics, testing, and publishing.

How This Book Is Organized

Part I — Getting Started (Chapters 1–4) sets up your tools and your first page. **Part II — Structuring Content with HTML** (Chapters 5–9) covers semantic HTML, forms, media, and accessibility. **Part III — CSS Foundations** (Chapters 10–14) teaches how CSS works. **Part IV — Layout** (Chapters 15–19) masters positioning, Flexbox, Grid, and responsive design. **Part V — Modern CSS and Polish** (Chapters 20–24) adds theming, motion, modern selectors, components, and organization. **Part VI — Projects and Launch** (Chapters 25–28) builds and publishes the complete Willowbrook Books website. The appendices provide cheat sheets, a launch checklist, and a glossary. Each chapter ends with a **Key Takeaways** box.

The Running Example: Willowbrook Books

Throughout the book we build the website for **Willowbrook Books**, a fictional independent bookstore and café: a home page with featured books and upcoming events, a catalog with book cards, event listings, a blog, and a contact page with a form and opening hours. Every technique you learn is applied to this site, so by the end you will have a complete, responsive, accessible website you understand line by line.

Conventions Used in This Book

> **TIP**
>
> A recommended practice or a technique that makes your work easier.

> **NOTE**
>
> Background information, browser support details, or a useful aside.

> **WARNING**
>
> A common mistake that breaks layouts, accessibility, or compatibility.

Code appears in `monospace`, with the file name above each example. Newer CSS features are labeled with their support status; check MDN Web Docs or caniuse.com for the latest details.

Contents

Preface

PART I — GETTING STARTED

1 How the Web Works: Browsers, Servers, HTML, CSS, and JavaScript

2 Setting Up Your Tools: Code Editor, Browser DevTools, and Your First Page

3 HTML Fundamentals: Elements, Attributes, Nesting, and Document Structure

4 Text, Links, and Images

PART II — STRUCTURING CONTENT WITH HTML

5 Semantic HTML and Page Structure

6 Lists, Tables, and Structured Content

7 Forms and Inputs

8 Media: Responsive Images, Video, Audio, and SVG

9 Accessibility Fundamentals

PART III — CSS FOUNDATIONS

10 CSS Basics: Selectors, Properties, Values, and Linking Stylesheets

11 The Cascade, Specificity, Inheritance, and Cascade Layers

12 The Box Model, Sizing, and CSS Units

13 Colors, Backgrounds, Borders, Gradients, and Shadows

14 Typography and Web Fonts

PART IV — LAYOUT

15 Normal Flow, Display, Positioning, and Stacking

16 Flexbox in Depth

17 CSS Grid in Depth

18 Responsive Design: Media Queries, Fluid Layouts, and Mobile-First

19 Modern Responsive Techniques: clamp(), Container Queries, and Intrinsic Layouts

PART V — MODERN CSS AND POLISH

20 Custom Properties, Theming, and Dark Mode

21 Transitions, Transforms, and Animations

22 Modern Selectors and Features: :is(), :where(), :has(), and Nesting

23 Building Reusable Components: Buttons, Cards, Navigation, and Modals

24 Organizing CSS: Architecture, Naming Conventions, and Maintainability

PART VI — PROJECTS AND LAUNCH

25 Project: The Willowbrook Books Home Page and Site Layout

26 Project: Catalog, Events, Blog, and Contact Pages

27 Performance, SEO Basics, Validation, and Cross-Browser Testing

28 Publishing Your Website: Hosting, Domains, and Next Steps

APPENDICES

A HTML Elements Cheat Sheet

B CSS Properties and Layout Reference

C Launch Checklist

D Glossary

PART I

Getting Started

How the web works, the tools you need, and your very first web page — built by hand and understood line by line.

1 · How the Web Works

2 · Setting Up Your Tools

3 · HTML Fundamentals

4 · Text, Links, and Images

CHAPTER 1 · PART I — GETTING STARTED

How the Web Works: Browsers, Servers, HTML, CSS, and JavaScript

Before we write a single line of code, we will follow a web page on its journey from a computer somewhere in the world to the screen in front of you. Once you understand that journey, every tag and every style rule in the rest of this book will have a clear place to live.

Why Start with the Big Picture?

Many beginners jump straight into typing tags and quickly feel lost. They copy a snippet, it works, and they have no idea why. Then a second snippet does not work, and they have no idea why either. The cure is a simple mental model of what actually happens when someone visits a website. You do not need to become a network engineer. You only need to know who the main players are, what each one is responsible for, and how they talk to one another.

In this chapter we will meet those players: the **browser** on your device, the **server** that stores a website's files, the **internet** that connects them, and the three languages that browsers understand — **HTML**, **CSS**, and **JavaScript**. We will also introduce the project you will build throughout this book: the website for Willowbrook Books, a small independent bookstore and café. By the end of the chapter you will be able to explain, in plain words, how a page reaches a visitor and what job each language does on that page.

The Two Sides of Every Website: Clients and Servers

Every time you look at a web page, two computers are involved. The first is yours: a laptop, a phone, a tablet, or even a television. The program you use to view websites on it — Chrome, Firefox, Safari, Edge, or another — is called a **web browser**. In technical language, the browser is a **client**, because it asks for things.

The second computer is a **server**. A server is simply a computer whose job is to wait for requests and answer them. It is usually located in a data center, runs all day and night, and has no screen attached. When a browser asks for a page, the server finds the right file (or builds one on the spot) and sends it back. The word "server" describes the role, not a special kind of hardware. During development, your own laptop can act as a server too, as you will see in Chapter 2.

This arrangement is called the **client–server model**. The client makes a **request**; the server sends a **response**. The whole web is built from billions of these small conversations. Here is the idea as a simple diagram:



Notice that the server does not "push" pages to you unprompted. Nothing happens until the browser asks. Keep this in mind: when you edit a file on your computer and nothing changes in the browser, it is almost always because the browser has not asked for the new version yet. Refreshing the page sends a fresh request.

Addresses on the Web: URLs

To ask for something, the browser needs to know where it lives. Every resource on the web — a page, an image, a stylesheet, a video — has an address called a **URL** (Uniform Resource Locator). You type URLs into the address bar, click them as links, and, as a web developer, you will write them into your HTML constantly. Let us take one apart. Imagine the Willowbrook Books website is published at `willowbrookbooks.example` (the `.example` ending is reserved for documentation, so we can use it safely without pointing at a real company):

```
https://www.willowbrookbooks.example:443/events/index.html?month=june#poetry-night
\___/   \_____________________________/\__/\_____________/\_________/\__________/
scheme             host            port       path          query     fragment
```

Part	Example	What it means
Scheme	`https`	The protocol the browser should use to talk to the server.
Host	`www.willowbrookbooks.example`	The name of the server (the domain name).
Port	`443`	A numbered "door" on the server. Usually left out, because browsers know the defaults.
Path	`/events/index.html`	Which resource on that server we want, much like a folder and file name.
Query	`?month=june`	Extra information sent to the server, written as name=value pairs.
Fragment	`#poetry-night`	A spot inside the page. The browser handles this itself and scrolls to it.

Most of the time you will see much shorter URLs such as `https://willowbrookbooks.example/events.html`. The port is left out because `https` uses port 443 by default, and there is no query or fragment. You will meet the fragment again in Chapter 4, where we

use it to link directly to a section of a page.

Domain names and DNS

Computers on the internet actually find each other using numeric **IP addresses**, such as `192.0.2.10` (another address range reserved for documentation). Numbers like these are hard for people to remember, so we use **domain names** instead. The **Domain Name System**, or **DNS**, is the internet's address book: it translates a name like `willowbrookbooks.example` into the IP address of the server that hosts it.

When you type a URL, your browser first checks whether it already knows the IP address from a recent visit. If not, it asks a DNS resolver — usually run by your internet provider or a public DNS service — to look the name up. The answer is cached for a while, so the next visit is faster. You will deal with DNS directly only when you buy a domain name and connect it to your hosting, which we cover in Chapter 28. For now it is enough to know that a name is turned into a number before any page can be requested.

The Conversation: HTTP and HTTPS

Once the browser knows where the server is, it opens a connection and speaks a protocol called **HTTP**, the Hypertext Transfer Protocol. HTTP is a set of rules for how requests and responses are formatted. It is a surprisingly readable, text-based conversation. A simplified request from the browser looks like this:

```
GET /events.html HTTP/1.1
Host: willowbrookbooks.example
Accept: text/html
Accept-Language: en-US
```

The first line says what the browser wants to do (`GET` means "please give me"), which resource it wants (`/events.html`), and which version of the protocol it speaks. The lines after it are **headers**: small labeled pieces of extra information, such as which host we are talking to and which languages the visitor prefers.

The server replies with a response. Again simplified:

```
HTTP/1.1 200 OK
Content-Type: text/html; charset=utf-8
Content-Length: 5120

<!DOCTYPE html>
<html lang="en">
  ...the rest of the page...
</html>
```

The first line contains a **status code**. `200 OK` means everything went well. The headers tell the browser what kind of content is coming — here, HTML text encoded as UTF-8 — and after a blank line comes the **body**: the HTML document itself. Modern browsers and servers often use newer versions of HTTP that pack this information more efficiently, but the ideas of method, path, headers, status, and body stay the same, and that is all you need as an HTML and CSS developer.

Status codes you will actually meet

Status codes are three-digit numbers grouped by their first digit. You do not need to memorize the full list, but a handful appear so often that recognizing them will save you time when something goes wrong.

Code	Name	What it usually means for you
`200`	OK	The file was found and sent. All is well.
`301`	Moved Permanently	The page has a new address; the browser follows it.
`304`	Not Modified	The browser's saved copy is still fresh, so nothing new was sent.
`404`	Not Found	No file exists at that path. Often a typo in a link or file name.
`500`	Internal Server Error	Something broke on the server itself, not in your HTML.

The one you will see most while learning is `404`. If your stylesheet does not load, or an image shows as a broken icon, the cause is very often a path that does not match the real file name. In Chapter 2 we will open the browser's Network panel and watch these codes appear for every file a page requests.

What the "S" in HTTPS adds

HTTPS is HTTP with encryption. The browser and server agree on secret keys before exchanging any data, so nobody in between — on a café's Wi-Fi network, for example — can read or quietly change the pages, form entries, or passwords traveling between them. The server proves its identity with a **certificate** issued by a trusted authority, which is why browsers show a warning when a certificate is missing or invalid.

Today HTTPS is the normal, expected way to serve every website, not only shops and banks. Browsers label plain HTTP pages as "Not secure," and several modern web features only work on secure pages. The good news is that most hosting services set up HTTPS for you automatically. We will make sure the Willowbrook Books site uses it when we publish in Chapter 28.

> **NOTE**
>
> You do not need a server or HTTPS to start learning. For most of this book you will open files straight from your own computer or use a tiny local server that comes with your editor. The request-and-response model still applies; the "server" just happens to be your own machine.

One Page, Many Requests

Here is something that surprises many beginners: a web page is rarely a single file. The first response contains the HTML document, but that document usually refers to other files — a stylesheet, a few images, perhaps some fonts and a script. As the browser reads the HTML and discovers each reference, it sends a new request for that file. A simple Willowbrook Books home page might trigger a sequence like this:

```
1. GET /index.html                   -> 200  (the HTML document)
2. GET /css/styles.css               -> 200  (the stylesheet)
3. GET /images/storefront.jpg        -> 200  (hero photo)
4. GET /images/book-the-quiet-river.jpg -> 200  (featured book cover)
5. GET /js/menu.js                   -> 200  (small script for the mobile menu)
6. GET /favicon.ico                  -> 404  (no icon file yet!)
```

Each line is a separate round trip between browser and server. This explains several things you will notice as a developer. It is why a page can appear in stages, with text first and images a moment later. It is why a single misspelled file name breaks only one part of the page rather than the whole thing. And it is why performance work, which we discuss in Chapter 27, often focuses on sending fewer and smaller files.

Browsers also keep copies of files they have already downloaded in a **cache**. If you visit the page again, the browser may reuse its saved copy of `styles.css` instead of downloading it again. This is great for visitors but occasionally confusing for developers: you change your CSS, refresh, and see the old styles. A "hard refresh" — usually `Ctrl` + `Shift` + `R` on Windows and Linux, or `Cmd` + `Shift` + `R` on macOS — tells the browser to ignore the cache and fetch everything again.

The Three Languages of the Web

Once files arrive, the browser needs to understand them. Browsers speak three core languages, and each has a clear, separate job. A helpful way to picture them is to think about building the Willowbrook Books shop itself.

- **HTML is the structure** — the walls, rooms, shelves, and signs. It says "this is a heading," "this is a paragraph," "this is a list of books," "this is a link to the events page."
- **CSS is the presentation** — the paint, lighting, furniture arrangement, and window display. It says "headings are deep green and use a serif font," "book cards sit in a grid of three columns on wide screens."
- **JavaScript is the behavior** — the automatic door, the till, the bell that rings when someone enters. It makes things happen in response to what the visitor does, such as opening a menu when a button is pressed.

Keeping these jobs separate is one of the most important habits in web development. Your content remains meaningful even if the styles fail to load, the design can change without rewriting the content, and scripts add extras on top of a page that already works. This book focuses on HTML and CSS, the two languages every website depends on. We will only touch JavaScript lightly, but you will understand exactly where it fits.

HTML: HyperText Markup Language

HTML is a **markup language**, not a programming language. It does not calculate anything or make decisions. Instead, it *marks up* content: you wrap pieces of text in **tags** that describe what each piece is. The "hypertext" part refers to links — text that takes you to another document when you click it. Linking documents together is the idea that made the web a web.

Here is a tiny but complete HTML document for Willowbrook Books. Do not worry about every detail yet; Chapter 3 explains each line carefully.

index.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>Willowbrook Books — Independent Bookstore and Café</title>
    <link rel="stylesheet" href="css/styles.css">
  </head>
  <body>
    <h1>Willowbrook Books</h1>
    <p>Your neighborhood bookstore and café since the day we opened our doors.</p>
    <p><a href="events.html">See our upcoming events</a></p>
    <button type="button" id="hours-button">Show opening hours</button>
    <p id="hours" hidden>Open daily, 9 am to 7 pm.</p>
    <script src="js/hours.js"></script>
  </body>
</html>
```

Even without knowing the rules, you can probably guess what most of this does. The text between `<h1>` and `</h1>` is the main heading. The `<p>` tags mark paragraphs. The `<a>` tag creates a link to `events.html`. The `<link>` line inside the head points to a stylesheet, and the `<script>` line near the end points to a JavaScript file. Every one of those references becomes an extra request, exactly as we described in the previous section.

CSS: Cascading Style Sheets

CSS controls how HTML looks: colors, fonts, spacing, sizes, and layout. A CSS file contains **rules**. Each rule has a **selector** that picks which elements to style and a set of **declarations** that say how to style them. Here is a first taste of the Willowbrook Books stylesheet, using the brand's cream background, deep green headings, and terracotta accent:

css/styles.css

```
body {
  margin: 0;
  padding: 1.5rem;
  background-color: #fbf6ec;   /* warm cream */
  color: #2b2b2b;
  font-family: "Helvetica Neue", Arial, sans-serif;
  line-height: 1.6;
}

h1 {
  color: #1f4d3a;              /* deep green */
  font-family: Georgia, "Times New Roman", serif;
}

a {
  color: #b5542d;              /* terracotta */
}
```

Read the first rule as a sentence: "For the `body` element, remove the default margin, add some padding, set a cream background and dark gray text, use a sans-serif font, and space the lines comfortably." With this file linked, the same page now has a warm cream background, a green serif heading, and terracotta links. The HTML did not change at all — only its presentation.

The word "cascading" refers to the rules that decide what happens when several CSS rules try to style the same element. That is a big topic, and we give it a whole chapter (Chapter 11). For now, just remember that CSS describes appearance, and that one stylesheet can style every page on a site.

JavaScript: behavior and interactivity

JavaScript is a full programming language. It can react to clicks and key presses, change the page after it has loaded, fetch new data, and much more. In our example, a few lines are enough to reveal the opening hours when the button is pressed:

js/hours.js

```
const button = document.getElementById("hours-button");
const hours = document.getElementById("hours");

button.addEventListener("click", function () {
  hours.hidden = !hours.hidden;
});
```

When a visitor clicks the button, the paragraph with the opening hours appears; click again, and it hides. You do not need to understand this code — this is an HTML and CSS book — but notice that the script finds elements using the `id` values we gave them in the HTML. Well-structured HTML makes JavaScript easier to write, and CSS easier too.

> **TIP**
>
> A good rule of thumb is to reach for HTML first, CSS second, and JavaScript last. Many effects that once needed scripts — expanding sections, smooth color changes, responsive layouts, even simple dialogs — can now be done with HTML and CSS alone. Pages built this way tend to be faster, sturdier, and more accessible.

How the Browser Turns Code into Pixels

We have followed files from the server to the browser. The final part of the journey happens inside the browser, where text files become the colorful, clickable page you see. This process is often called the **rendering pipeline**. Knowing its broad steps will help you understand many CSS behaviors later in the book.

```
HTML text ---> parse ---> DOM (tree of elements)
                                \
                                 +--> render tree --> layout --> paint --> screen
                                /
CSS text  ---> parse ---> CSSOM (tree of style rules)
```

1. **Parsing HTML.** The browser reads the HTML from top to bottom and builds the **DOM**, the Document Object Model. The DOM is a tree: the `html` element at the root, `head` and `body` as its branches, and every heading, paragraph, and link as smaller branches and leaves. JavaScript works with this tree, and the browser's developer tools show it to you.
2. **Parsing CSS.** The browser reads every stylesheet and builds a similar structure of style rules, often called the **CSSOM**.
3. **Matching styles.** The browser combines the two, working out the final style of every visible element — its color, font, size, and so on. Elements that are hidden are left out of what gets drawn.
4. **Layout.** The browser calculates the exact position and size of every box on the page, based on the screen width, the content, and the CSS. This is sometimes called **reflow**.
5. **Paint and composite.** Finally, the browser fills in the pixels — text, colors, borders, images, shadows — and assembles the layers into the image on your screen.

The browser repeats parts of this process whenever something changes: when you resize the window, when an image finishes downloading and needs space, or when JavaScript alters the page. That is why a layout can "jump" when an image loads late, and why, in Chapter 4, we will give images a width and height so the browser can reserve their space in advance.

Browsers are forgiving

One remarkable feature of HTML parsing is that browsers are extremely tolerant of mistakes. If you forget to close a paragraph or misspell a tag, the browser does not show an error message; it makes its best guess and carries on. CSS works similarly: if the browser does not understand a declaration, it

simply skips it and applies the rest.

> **WARNING**
> "It looks right in my browser" is not the same as "it is correct." Always check your pages with the HTML validator, test them at different screen widths, and try using them with only the keyboard. We will show you how to do each of these as the book progresses.

Static Sites and Dynamic Sites

So far we have described a server that simply finds a file and sends it back unchanged. A website built this way is called a **static site**. Every visitor receives the same HTML, CSS, and images. Static sites are fast, secure, cheap or free to host, and perfect for a great many purposes: portfolios, small business sites, documentation, event pages, and personal blogs. The Willowbrook Books site we build in this book is a static site.

A **dynamic site** builds its pages on the server when they are requested. A program — written in a language such as PHP, Python, JavaScript on the server, Ruby, Java, or C# — reads data from a database, fills in a template, and sends the result. Online shops, social networks, and web email are dynamic because each visitor sees different content.

Here is the key point for you: **no matter how a page is produced, what reaches the browser is still HTML, CSS, and JavaScript.** A dynamic online shop and a hand-written static page both end up as the same kinds of files in the browser. That is why HTML and CSS are the foundation for every web developer, whether they later specialize in design, front-end frameworks, content management systems, or back-end programming. The skills in this book carry over to all of them.

Who Decides How HTML and CSS Work?

HTML and CSS are not owned by any single company. They are open **web standards**, written and maintained by standards groups in collaboration with browser makers, companies, and individual experts. The HTML standard is maintained by the WHATWG (the Web Hypertext Application Technology Working Group) as a "living standard" that is updated continually rather than released in big numbered versions. CSS is developed by the W3C (World Wide Web Consortium) CSS Working Group and is split into many separate modules — one for colors, one for grid layout, one for fonts, and so on — each progressing at its own pace.

Checking browser support

Because features arrive gradually, you will sometimes want to know whether a particular property or element works in the browsers your visitors use. Two free resources answer that question:

- **MDN Web Docs** (developer.mozilla.org) — detailed, trustworthy documentation for every HTML element and CSS property, with examples and a compatibility table at the bottom of each page.

- **Can I use** (caniuse.com) — a site dedicated to browser support, showing at a glance which browsers support a feature.

Throughout this book, when we introduce a newer feature we will tell you whether it is widely supported or still settling in, and we will encourage you to check these sites before relying on it in a real project. Support changes over time, almost always for the better, so a quick check is always worthwhile.

A Web for Everyone, on Every Screen

Two ideas shape every page we build in this book, so it is worth introducing them now rather than bolting them on later.

The first is **accessibility**: making websites usable by people with disabilities. Some visitors are blind and use a **screen reader**, software that reads the page aloud or sends it to a braille display. Some cannot use a mouse and navigate entirely with a keyboard or a switch device. Some have low vision and zoom in heavily; some are color-blind; some have cognitive disabilities and benefit from clear, predictable layouts. Accessibility also helps everyone else: captions help people in noisy cafés, good contrast helps people reading in bright sunshine, and clear structure helps everyone scan a page quickly.

The wonderful news is that HTML is accessible by default when you use it correctly. A real `<button>` can be reached with the keyboard and announced as a button by screen readers without any extra work. A proper heading structure lets screen reader users jump from section to section. Much of accessibility comes down to choosing the right HTML element for the job, which is exactly what the chapters in Part II teach. Chapter 9 then brings the ideas together.

The second idea is **responsive design**: pages that adapt to any screen size. Visitors may view Willowbrook Books on a small phone on the bus, a tablet on the sofa, or a wide monitor at work. Rather than building separate sites for each, we build one site whose layout flexes. The `meta viewport` line you saw in the example page is the first small step toward that goal, and Part IV of this book covers responsive layout in depth.

Meet Willowbrook Books: Your Project for This Book

Learning is far more effective when you build something real, so instead of isolated exercises we will build a complete website together. Our client is **Willowbrook Books**, a fictional independent bookstore with a small café in the corner. The owners want a site that helps neighbors discover new books, find out about readings and book clubs, and get in touch. Their brand is warm and welcoming: a cream background, deep green as the main color, and a terracotta accent, with a classic serif font for headings and a clean sans-serif font for body text.

Here are the pages we will create, and the chapters where they take shape:

File	Purpose
`index.html`	Home page: welcome hero, featured books, upcoming events, newsletter signup.
`books.html`	Catalog of books displayed as a grid of cards.
`book.html`	A single book's details page.
`events.html`	Readings, signings, and book clubs, with dates and times.
`blog.html` and `post.html`	Staff picks and news articles.
`about.html`	The story of the shop and its team.
`contact.html`	Contact form, opening hours table, and a map placeholder.

All pages share a stylesheet in a `css` folder, and all pictures live in an `images` folder. Later, in Chapter 24, we will split the single stylesheet into several focused files as the project grows. The finished folder structure will look like this:

```
willowbrook-books/
  index.html
  books.html
  book.html
  events.html
  blog.html
  post.html
  about.html
  contact.html
  css/
    styles.css
  images/
    storefront.jpg
    book-the-quiet-river.jpg
    ...
  js/
    menu.js
```

Putting It All Together: The Journey of a Page

Let us finish by walking through the entire journey once more, now that you know each step. A visitor on their phone taps a link to `https://willowbrookbooks.example/events.html`.

1. The browser looks up `willowbrookbooks.example` using DNS and learns the server's IP address.
2. It opens a secure HTTPS connection to that server, which proves its identity with a certificate.
3. It sends an HTTP `GET` request for `/events.html`.
4. The server finds the file and replies with status `200 OK` and the HTML in the body.
5. The browser starts parsing the HTML and building the DOM. It finds the stylesheet link, an image or two, and a script, and requests each one.

6. As the CSS arrives, the browser matches styles to elements, calculates the layout for the phone's narrow screen, and paints the page.
7. The script runs and makes the menu button work. The visitor scrolls, reads about next week's poetry night, and taps another link — and the whole cycle begins again, often faster, thanks to the cache.

All of this usually happens in a fraction of a second. As a web developer, you control the files at the center of it: the HTML that gives the page its structure and meaning, the CSS that makes it look and feel like Willowbrook Books, and, occasionally, the JavaScript that adds a little interactivity. In the next chapter we set up the tools you need to create those files and to see exactly what the browser does with them.

Key Takeaways

- The web works through requests and responses: a browser (the client) asks, and a server answers with files.
- A URL is made of a scheme, host, optional port, path, optional query, and optional fragment; DNS turns the host name into an IP address.
- HTTP defines the conversation; HTTPS encrypts it and is the expected standard for every modern website.
- A single page triggers many requests — HTML first, then stylesheets, images, fonts, and scripts — and status codes such as 200 and 404 tell you how each one went.
- HTML provides structure and meaning, CSS provides presentation, and JavaScript provides behavior; keep these jobs separate.
- Browsers build a DOM from HTML, combine it with CSS, calculate layout, and paint pixels; they forgive errors silently, so validate your code.
- Whether a site is static or dynamic, what reaches the browser is always HTML, CSS, and JavaScript, which makes these skills universal.
- Accessibility and responsive design are built in from the start, and throughout the book we will build the Willowbrook Books website step by step.