The HTML-CSS-JavaScript Self-Mastery Workbook

Table of Contents

🖥️ Prerequisites

Before starting this workbook, you only need some **basic computer knowledge** and a few essential tools. Don't worry — nothing advanced is required.

✅ What You Should Already Know

- How to create, save, and open files/folders on your computer.
- How to use a web browser (Chrome, Firefox, or Edge).
- How to copy and paste text.

✅ What You Need Installed

- A **code editor** like **Visual Studio Code (VSCode)** (free).
- A modern **web browser** (Chrome, Firefox, or Edge).

✅ Helpful Skills (You'll Use Them Often Here)

- Open browser **DevTools** (right-click → Inspect).
- Type simple searches into Google (e.g., *"HTML table example"*).
- Read answers on MDN Web Docs or Stack Overflow.
- Use ChatGPT (or another AI) to ask for explanations/examples.

How to Use This Workbook

This document is **not a textbook**. It will not hand you the answers.
Instead, it gives you the **right questions to ask yourself** — questions every developer must be able to answer to master the topic at a global standard.

Here's how to use it effectively:

1. **Ask Yourself First**
    - Before looking things up, try to explain the answer in your own words.
    - If you can't, that's fine — it means you found a gap in your knowledge.
    - If a **new question pops up in your own mind** that's not in here, that's your curiosity leading you deeper — write it down and explore it.
2. **Leverage All Resources**
    - Use **Google**, **Stack Overflow**, and **ChatGPT** to research.
    - Read documentation, articles, and examples.
    - Find a way to practice and produce results.
3. **Learn by Doing**
    - Each section has **project exercises**.
    - Completing these exercises forces you to practice and discover the answers naturally.
    - Don't skip them — doing is how you'll turn "theory" into mastery.
4. **Reflect and Explain**
    - After finding an answer, try teaching it back:
        - Explain to a friend, or a fellow developer.
        - Write notes.
        - Or even record yourself explaining.
    - **If you can explain clearly, you've truly learned it.**
5. **Iterate and Improve**
    - Revisit questions regularly.
    - As you grow, your answers will become deeper and more precise.

🌱 Philosophy Behind This Workbook

This is a **"find the answer within yourself" document — the programming version.**

- The **questions** represent the knowledge every web developer must internalize.
- **Be curious** → always ask "why does this work this way?"
- The **resources** (Google, Stack Overflow, ChatGPT) are your tools — but the true goal is that **the understanding lives inside you**, not just in your search history.
- The **exercises** are opportunities to struggle, explore, and discover.
- **Expect mistakes** → debugging is how you learn.
- **Reflect** → explain new concepts in your own words.

By the time you've asked and answered everything here — and built the exercises — you won't just "know HTML, CSS, and JavaScript." You'll **understand them so deeply that you can build, debug, and explain any project with confidence**.

HTML

1. Basics & Structure

- What is HTML, and what problem does it solve?
- What does the <!DOCTYPE html> declaration mean?
- What are the main parts of an HTML document (<html>, <head>, <body>)?
- What's the difference between block-level and inline elements?
- How do opening and closing tags work? Are all tags paired?
- What are **void elements** (self-closing tags), and can you name some examples?
- What are <template> and <slot> elements, and when do you use them?

```
Exercise: Build a "Hello World" webpage.
•    Create a valid HTML document with <!DOCTYPE html>.
•    Use <html>, <head>, <body>.
•    Add at least one block element (<div>) and one inline
     element (<span>).
•    Use a void element (<br> or <img>).
•    Experiment with <template> and explain why it doesn't
     show until cloned with JS.
```

2. Text & Content

- How do you create headings (<h1>–<h6>), and when should each be used?
- What's the difference between <p>, , and ?
- What are semantic text tags like , , <mark>, <abbr>?
- How do you create lists (<ul>, <ol>, <li>) and when to use each?
- How do you add line breaks (
) and horizontal rules (<hr>)?
- How do you add comments in HTML, and why are they useful?
- What are <dialog> and <details> elements, and how do they improve UX?

```
Exercise: Create a Blog Article Page.
•    Add a title with <h1> and multiple subheadings (<h2>–
     <h3>).
•    Write paragraphs with <p>.
•    Emphasize text with <strong> and <em>.
•    Use <abbr> for an abbreviation.
•    Add an ordered list of "Top 3 Tips" and an unordered
     list of "Related Topics."
•    Insert <hr> for separation.
•    Use <dialog> for a "Read More" popup.
```

3. Links & Media

- How do you create a hyperlink (<a>), and what attributes does it need?
- What is the difference between absolute and relative URLs?

- How do you make a link open in a new tab (target="_blank") safely?
- How do you insert an image (<img>)? What are the required attributes?
- What is the importance of the alt attribute in images?
- How do you embed audio and video in HTML? (<audio>, <video>)
- How do you embed external content (like YouTube) using <iframe>?
- How do you use <picture> and <source> for responsive images?

```
Exercise: Build a Personal Profile Page.
•    Add navigation links to “Home,” “About,” and
     “Contact.”
•    Use absolute and relative links.
•    Make one link open in a new tab (target="_blank").
•    Insert an image with an alt attribute.
•    Embed a video with <video> and an external YouTube
     video with <iframe>.
•    Add a responsive image using <picture> and <source>.
```

4. Attributes & Metadata

- What are HTML attributes, and how do they work?
- What’s the difference between global attributes (id, class, style) and element-specific attributes?
- What is the title attribute, and when should you use it?
- What are data attributes (data-*), and how are they useful?
- What goes inside the <head> tag? (title, meta, link, script, etc.)
- What’s the purpose of <meta charset="UTF-8">?

- What are viewport settings (<meta name="viewport">), and why are they important for mobile devices?
- What are Progressive Web App (PWA) meta tags, and how do they enable installable apps?

```
Exercise: Create a “Portfolio Landing Page.”
•    Add a <title>, <meta charset="UTF-8">, and viewport
     meta tag.
•    Add id and class attributes for styling.
•    Use data-* attributes to store extra info.
•    Add a favicon using <link>.
•    Add meta tags for description and keywords.
•    Add a PWA manifest link (even if empty).
```

5. Forms & User Input

- How do you create a form (<form>), and what attributes does it need?
- What’s the difference between method="GET" and method="POST"?
- What are the common form input types (text, password, email, number, checkbox, radio, file)?
- How do you create labels for inputs, and why are they important for accessibility?
- How do you group inputs using <fieldset> and <legend>?
- What’s the purpose of the <select> and <option> elements?

- What is the difference between name, value, and placeholder attributes in inputs?
- How do you make a form input required (required)?
- How do you associate a label with an input using for and id?
- What are new input types (date, datetime-local, range, color) and their use cases?

```
Exercise: Build a Sign-Up Form.
•    Add inputs: text, email, password, number, date,
     color.
•    Group related inputs with <fieldset> and <legend>.
•    Add a <select> with multiple <option>s.
•    Add required attributes and placeholders.
•    Create labels with for attributes linked to inputs.
```

6. Tables

- How do you create a table in HTML (<table>, <tr>, <td>, <th>)?
- What is the difference between <thead>, <tbody>, and <tfoot>?
- What is the difference between <td> and <th>?
- What are colspan and rowspan, and when do you use them?
- Why are semantic tables important for accessibility?
- How do you make tables responsive on small screens?

```
Exercise: Build a Student Grades Table.
•    Create a table with <thead>, <tbody>, and <tfoot>.
•    Use <th> for headers, <td> for data.
•    Add rowspan and colspan.
```

7. Semantic HTML

- What is semantic HTML, and why is it important?
- What's the difference between and <section>?
- What are <header>, <nav>, <main>, <article>, <section>, <aside>, <footer>?
- How does semantic HTML help with SEO and accessibility?
- When should you use <figure> and <figcaption>?
- What is the role of **landmark elements** in accessibility?

```
Exercise: Build a News Article Layout.
•    Use <header> for site title.
•    <nav> for navigation menu.
•    <main> with an <article>.
•    Add a <section> inside article for "Highlights."
•    Add an <aside> with related links.
•    Use <footer> for copyright.
•    Wrap an image in <figure> with <figcaption>.
```

8. Accessibility (a11y)

- What is accessibility in HTML, and why does it matter?
- Why must every image have an alt attribute?

- How do headings (h1–h6) help screen readers?
- How do label elements help with accessible forms?
- What is the purpose of ARIA attributes (role, aria-label, etc.)?
- Why is tab order important for navigation?
- What are aria-live regions, and when are they useful?

```
Exercise: Make the Sign-Up Form Accessible.
•    Ensure all images have alt.
•    Use proper heading order.
•    Add aria-label to buttons/icons.
•    Test keyboard tab order.
•    Add a live region (aria-live) that displays form
     validation errors dynamically.
```

CSS

🟢 Beginner: Foundations

1. What is CSS, and what problem does it solve?
2. What is the difference between **inline, internal, and external** CSS?
3. What does **specificity** mean in CSS? How is it calculated?
4. What is the difference between **id, class, and element selectors**?
5. What are **pseudo-classes** (e.g., :hover, :nth-child) and **pseudo-elements** (e.g., ::before, ::after)?
6. What's the difference between **relative units** (em, rem, %) and **absolute units** (px, cm)?
7. How do you **reset or normalize CSS styles** across browsers?
8. What is the difference between **inline elements** and **block elements**?
9. What are CSS logical properties (margin-inline, padding-block)?

```
Exercise: Style the “Hello World” webpage (from HTML 1).
•    Apply inline, internal, and external CSS.
•    Use element, class, and id selectors.
•    Apply pseudo-classes (:hover, :first-child) and
     pseudo-elements (::before, ::after).
•    Use relative (em, rem, %) vs absolute (px) units.
•    Apply a CSS reset or normalize.
•    Style inline vs block elements differently.
•    Add logical properties (margin-inline, padding-
     block).
```

🟡 Intermediate: Styling & Layout

1. How do the **box model properties** (content, padding, border, margin) affect layout?
2. What’s the difference between relative, absolute, fixed, and sticky **positioning**?
3. How do **float** and **clear** work? When should you avoid floats?
4. How do you create a **two-column or three-column layout** with modern CSS?
5. What’s the difference between **inline-block vs. flex vs. grid** for layout?
6. How does **z-index** work, and what is a **stacking context**?
7. What’s the difference between **min-width, max-width** and **min-height, max-height**?
8. How do **transitions** differ from **animations** in CSS?

9. What is the difference between opacity and visibility?

10. What is aspect-ratio, and how does it help responsive design?

11. How do CSS functions like clamp(), min(), and max() work?

```
Exercise: Style the Blog Article (from HTML 2).
•    Apply margins, padding, borders (box model demo).
•    Position elements using relative, absolute, fixed,
     and sticky.
•    Create a floated image with text wrapping, then clear
     it.
•    Make a 2-column layout with Flexbox and a 3-column
     layout with Grid.
•    Demonstrate inline-block vs flex.
•    Use z-index layering.
•    Apply min-width and max-width.
•    Add a simple hover transition and a keyframe
     animation.
•    Toggle between opacity: 0 and visibility: hidden.
•    Use aspect-ratio for images or videos.
•    Resize text using clamp().
```

🔵 Advanced: Responsive & Performance

1. How do **media queries** work, and what's the difference between max-width vs. min-width queries?

2. What are **responsive units** (vw, vh, vmin, vmax)?

3. How do you use **CSS Grid** vs. **Flexbox** effectively? When is one better than the other?

4. What are the performance implications of using **CSS selectors like div div div** vs. a class?
5. What's the difference between **inline SVG styling** and **CSS styling for background images**?
6. What are **hardware-accelerated properties** (like transform: translateZ(0)) and when should they be used?
7. How do you implement **dark mode / theme switching** using CSS variables (custom properties)?
8. How do you use **clipping and masking** in CSS (clip-path, mask-image)?
9. What's the difference between **absolute units (px)** vs. **relative units (rem, %, vh, vw) in responsive design**?
10. What are container queries (@container) and how do they differ from media queries?
11. What is subgrid in CSS Grid Level 2, and why is it powerful?
12. What are prefers-color-scheme and prefers-reduced-motion media queries?

```
Exercise: Make the Portfolio Page (from HTML 4)
responsive.
•    Add media queries with min-width and max-width.
•    Use responsive units (vw, vh, vmin, vmax).
•    Switch layouts between Flexbox and Grid depending on
     screen size.
•    Compare performance of .class div p vs .class
     selectors.
•    Apply inline SVG as icon vs background-image icon.
•    Use transform: translateZ(0) for hardware
     acceleration.
•    Implement dark mode with prefers-color-scheme and CSS
     variables.
•    Mask part of an image using clip-path.
•    Compare px vs rem in scaling.
•    Add a component that resizes with container queries
     (@container).
•    Use subgrid inside a larger grid.
•    Honor prefers-reduced-motion for accessibility.
```

🔴 Expert: Architecture & Edge Cases

1. How do you manage **CSS at scale** in large projects (BEM, OOCSS, SMACSS, ITCSS, utility-first like Tailwind)?
2. What are the trade-offs between **global stylesheets vs. CSS-in-JS vs. utility frameworks**?
3. How do **cascading and inheritance** interact when multiple rules apply to the same element?

4. What are **render-blocking CSS** issues and how do you optimize for **performance**?
5. How do you implement **print stylesheets** (@media print)?
6. What are **custom properties (CSS variables)**? How do they differ from **preprocessor variables** (like in SASS/LESS)?
7. How do you implement **fluid typography** that adapts to screen size?
8. What are **accessibility considerations** for CSS (e.g., prefers-reduced-motion, color contrast, focus states)?
9. How do you debug CSS issues effectively (Chrome DevTools, Firefox inspector)?
10. How do you avoid **CSS specificity wars** in large codebases?
11. How do CSS cascade layers (@layer) work, and why are they important?
12. What are new CSS color spaces (lab(), lch(), color-mix())?
13. How do you design accessible focus states with :focus-visible?

Exercise: Refactor the News Article Layout (from HTML 7).

- Apply a naming convention (BEM or utility-first with Tailwind).
- Compare global stylesheet vs inline CSS vs CSS-in-JS (write one button 3 ways).
- Create cascading conflicts, then resolve them with specificity & cascade layers (@layer).
- Optimize render-blocking CSS by deferring styles.
- Add a print stylesheet.
- Use CSS variables vs SCSS variables.
- Implement fluid typography.
- Ensure color contrast meets accessibility standards (WCAG).
- Debug styles with DevTools.
- Design focus states with :focus-visible.
- Use new CSS color spaces (lab(), lch(), color-mix()).

Javascript

1. Introduction, Variables & Data Types

- What is Javascript, and what problem does it solve?
- What is a variable, and how is it used?
- What is the difference between let, const, and var?
- What are the primitive data types in JavaScript?
- What is the difference between primitive and reference data types?
- How do you check the type of a variable (typeof, instanceof)?
- What is type coercion? Can you give an example?
- What's the difference between == and ===?
- What is BigInt, and when should you use it?
- What is the difference between undefined, null, and void 0?

```
Exercise: Build a "Data Inspector."
•    Declare variables with let, const, var.
•    Print all primitive types (string, number, boolean,
     null, undefined, BigInt, symbol).
•    Show reference types (objects, arrays).
•    Check types using typeof and instanceof.
•    Demonstrate type coercion ("5" + 1 vs "5" - 1).
•    Compare == vs ===.
•    Show difference between null, undefined, and void 0.
```

2. Operators & Expressions

- What are arithmetic operators (+, -, *, /, %)?
- What is the difference between ++i and i++?
- What are comparison operators, and how are they used?
- What are logical operators (&&, ||, !)?
- How does the ternary (? :) operator work?
- What are the ?? (nullish coalescing) and ?. (optional chaining) operators?

```
Exercise: Calculator Upgrade.
•    Implement addition, subtraction, multiplication,
     division, modulus.
•    Show difference between ++i and i++.
•    Use comparison operators (>, <, >=, <=).
•    Use logical operators (&&, ||, !).
•    Use ternary operator for null variables.
•    Use nullish coalescing (??) and optional chaining
     (?.).
```

3. Control Flow

- How do if, else if, and else statements work?
- What is a switch statement, and when is it useful?
- What are loops (for, while, do...while)?
- What's the difference between break and continue?
- What is the difference between for...in and for…of?

- What are for await...of loops, and when do you use them?

```
Exercise: Build a “Student Grading Program.”
•    Use if, else if, else to assign letter grades.
•    Implement a switch version.
•    Loop through students with for, while, do...while.
•    Demonstrate break and continue.
•    Use for...in for objects, for...of for arrays.
•    Use for await...of with async data fetching.
```

4. Functions

- What is a function, and why do we use it?
- What is the difference between a function declaration and a function expression?
- What is an arrow function?
- What are parameters vs arguments?
- What does it mean for a function to return a value?
- What are higher-order functions? Can you explain map, filter, and reduce?
- What are default parameters?
- What is recursion? Can you explain with an example?
- What are async functions, and how do they differ from normal ones?
- What are generators (function*) and yield?

```
Exercise: Build a Math Utility Library.
•    Create normal functions (add, subtract).
•    Create arrow functions (multiply).
•    Use parameters and arguments.
•    Return values from functions.
•    Use higher-order functions (map, filter, reduce).
•    Add default parameters.
•    Implement recursion (factorial).
•    Add an async function (fetchNumber).
•    Demonstrate generators (function* and yield).
```

5. Objects & Arrays

- What is an object? What are keys and values?
- How do you access properties with dot vs bracket notation?
- How do you add, update, or delete properties in an object?
- What is object destructuring?
- What is an array? How do you add, remove, or update elements?
- How do you loop through an array?
- What are array methods like map, filter, reduce, find, some, every?
- How do you copy arrays/objects (shallow vs deep copy)?
- How do Object.entries(), Object.values(), and Object.fromEntries() work?
- What is the difference between shallow copy vs deep copy using structuredClone()?

```
Exercise: Build a Shopping Cart.
•    Store products in objects ({ name, price }).
•    Use dot and bracket notation.
•    Add, update, delete properties.
•    Destructure product objects.
•    Store multiple products in an array.
•    Use array methods (map, filter, reduce).
•    Copy arrays shallow vs deep (slice, structuredClone).
•    Use Object.entries(), Object.values(), and
     Object.fromEntries().
```

6. Scope & Closures

- What is the difference between global, function, and block scope?
- What is hoisting? Which declarations are hoisted?
- What is a closure, and why is it useful?
- How do closures help with data privacy?
- How do ES modules affect scope compared to scripts?

```
Exercise: Build a Private Counter.
•    Create a counter variable inside a function.
•    Demonstrate block vs function vs global scope.
•    Show hoisting differences between var, let, const.
•    Return inner functions that access private variables
     (closure).
•    Show how modules isolate scope compared to scripts.
```

7. Asynchronous JavaScript & APIs

- What is the difference between synchronous and asynchronous code?
- What is a callback function?

- What are APIs, and how do they work?
- What are API Endpoints?
- How do you use the fetch API to make HTTP requests?
- What are Promises? How do then, catch, and finally work?
- What is async/await, and why is it useful?
- How do Promise.allSettled(), Promise.any(), and Promise.race() differ?

```
Exercise: “User Data Fetcher”
Practice callbacks, Promises, and async/await while
keeping it simple.
•    Write a function that fetches data from https://
     jsonplaceholder.typicode.com/users using a callback.
•    Display the results.
•    Rewrite the fetch function using Promises
     (.then, .catch).
•    Rewrite it again with async/await and proper
     try...catch.
•    Use Promise.all to load users and posts together.
•    Show an error message if the network call fails.
```

8. DOM and Browser Session

- What is the DOM?
- How do you select elements (querySelector, getElementById, etc.)?
- How do you change text, styles, or attributes of an element?
- How do you create and append new DOM elements?
- How do you add an event listener?

- What is event bubbling vs capturing?
- What is event delegation, and why is it powerful?
- What is localStorage and sessionStorage, and how do you use them?

```
Exercise: “Mini Contact Manager”
Build a small app that manipulates the DOM and uses
browser storage.
•    Create an input field and an Add Contact button.
•    When the button is clicked:
     •    Create a new list item with the contact name.
     •    Append it to a list (<ul>).
•    Add a Delete button to each item (practice event
     delegation).
•    Save the contacts in localStorage.
•    On page load, read from storage and display the list.
•    (Optional) Add a “Clear All” button.
```

9. Advanced Concepts

- What is “this” in JavaScript, and how does its meaning change in different contexts?
- What is prototypal inheritance?
- What is the difference between a class and a constructor function?
- What are ES6+ features (template literals, spread/rest, destructuring, optional chaining)?
- What are modules in JavaScript (import / export)?
- What is error handling (try...catch, throwing errors)?

- What is garbage collection?
- How do memory leaks happen in JavaScript?
- How do you write code that's easy to test?
- How do you structure a project into multiple files/modules?
- How do you use Intl for localization and formatting?

```
Exercise: "Notes App with Modules"
Put together advanced syntax and code organization.
•    Create a simple Notes App:
          - A text area to write notes.
          - A "Save Note" button.
          - A list that displays saved notes.
•    Use classes (or constructor functions) to represent a
     Note.
•    Use ES6+ features:
          - Template literals for rendering HTML.
          - Destructuring for pulling fields from objects.
          - Spread/rest operators when adding or deleting
     notes.
•    Split the code into modules (app.js, note.js,
     storage.js) and use import/export.
•    Add basic error handling (e.g., prevent empty notes).
•    (Optional) Format note timestamps using
     Intl.DateTimeFormat.
```