HTML & CSS

A hands-on guide to building clear, attractive pages that work on every screen

LEARNSAPPEAL | ORIGINAL COURSE

Structure first. Style with purpose. Make every page usable.

Create web pages with semantic HTML and modern CSS, then bring the pieces together in a small portfolio project.

What you need

A computer, a text editor, and a web browser. No coding experience is required. Save your work as plain text files, open the HTML file in a browser, and refresh after each change. Start with small edits and compare what you expected with what the browser shows.

Course map: Page structure (Modules 1-4), then layout, responsive design, accessibility, and complete projects (Modules 5-12). Includes examples, practice questions with answers, and a glossary.

How to use this course

Create a project folder with an index.html file and a styles.css file. Type each example rather than only reading it. Change one property at a time and observe the result. Use the browser inspector to see which rule applies when something looks different from what you expected.

By the end, you can: structure a page with HTML; use meaningful headings, links, and forms; style text and color; arrange layouts with Flexbox and Grid; adapt to narrow and wide screens; and check a page with keyboard navigation and browser zoom.

Course map

• 1. How a web page is made
• 2. Create a first HTML document
• 3. Structure content with semantic HTML
• 4. Links, images, and accessible text
• 5. Meet CSS: selectors and declarations
• 6. Color, type, spacing, and the box model
• 7. Layout with Flexbox and Grid
• 8. Responsive design for different screens
• 9. Forms, focus, and accessible styling
• 10. Organize and debug your project
• 11. Capstone: a LearnsAppeal course page
• 12. Publish, review, and keep improving

PAGE STRUCTURE

1. How a web page is made

A browser reads HTML to understand the page content and structure. CSS adds visual presentation such as colors, spacing, and layout. A browser can open local files directly, so you can begin without a server. A web page is made from plain text files and other assets such as images. File names and folder paths must match exactly, including capitalization on many servers.

Example

```
my-first-site/
  index.html
  styles.css
  images/
    banner.jpg
```

Check your understanding

- What does HTML describe?
- What does CSS control?
- Why should file names match exactly?

Suggested answers

1. The content and structure of the page.
2. How page content looks and is arranged.
3. A link that works on one system may fail on another if the path differs.

2. Create a first HTML document

HTML uses elements written with tags. Many elements have an opening tag, content, and a closing tag. The document begins with a doctype and contains a head for page information and a body for visible content. Set the language, character encoding, and viewport so browsers can handle the page properly.

Example

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <title>My First Page</title>
</head>
<body>
  <h1>Hello, web!</h1>
  <p>This is my first page.</p>
</body>
</html>
```

Check your understanding

- Where does visible page content go?
- What does the title element do?
- What does the viewport setting help with?

Suggested answers

1. Inside the body element.
2. Names the page in the browser tab and other contexts.
3. Helps the layout use the device screen width on phones.

3. Structure content with semantic HTML

Choose elements by meaning. Headings describe sections; paragraphs hold text; lists group related items; navigation holds major links; main marks the primary content; and article can represent a self-contained item. Use one clear page heading and keep heading levels in a logical order. Semantic structure helps people and assistive technology understand a page.

Example

```
<header>
  <h1>Explore courses</h1>
</header>
<main>
  <section aria-labelledby="popular-title">
    <h2 id="popular-title">Popular topics</h2>
    <ul><li>Python</li><li>Design</li></ul>
  </section>
</main>
```

Check your understanding

• What is main used for?

• Why use headings in order?

• When is a list useful?

Suggested answers

1. The primary content of the page.
2. They show how the content is organized.
3. For a group of related items.

4. Links, images, and accessible text

Use links to go to another page or resource. Link wording should explain where the link leads. Add images with the img element and write alternative text that describes useful information in the image. If an image is purely decorative, empty alt text lets assistive technology skip it. Use descriptive file names and check that paths work.

Example

```
<a href="courses/python.html">Explore the Python course</a>

<img src="images/course-map.png"
     alt="Diagram showing the course modules">

<img src="images/decoration.svg" alt="">
```

Check your understanding

• When should you use a link?

• What should useful alternative text explain?

• Why can “click here” be unclear?

Suggested answers

1. When someone should navigate to a destination.
2. The important information or purpose of the image.
3. It does not explain the destination when read on its own.

STYLE, LAYOUT, AND COMPLETE PAGES

5. Meet CSS: selectors and declarations

A CSS rule has a selector and a group of declarations. A selector chooses which elements to style; each declaration sets a property to a value. You can select an element by tag, class, or id. Classes are reusable and are usually the best choice for visual patterns. Keep the stylesheet organized and use readable names.

Example

```
<link rel="stylesheet" href="styles.css">

.course-card {
  background-color: #ffffff;
  color: #243447;
  border-radius: 12px;
}
```

Check your understanding

• What does a selector choose?

• How is a CSS class written in a selector?

• Why prefer classes for reusable styles?

Suggested answers

1. The elements a rule will style.
2. With a dot before the class name, such as .course-card.
3. The same style can be applied to many elements.

6. Color, type, spacing, and the box model

Readable design begins with enough contrast, comfortable text sizes, and consistent spacing. Each element is a box: content, padding, border, and margin. Padding is inside the border; margin is outside. Setting box-sizing to border-box makes a chosen width include padding and border, which simplifies layout calculations.

Example

```
* {
  box-sizing: border-box;
}

body {
  margin: 0;
  font-family: Arial, sans-serif;
  color: #243447;
  line-height: 1.6;
}

.card { padding: 1rem; margin: 1rem 0; }
```

Check your understanding

• Which space is inside the border?

• Which space is outside it?

• Why set line-height?

Suggested answers

1. Padding.
2. Margin.
3. It gives lines of text room and improves readability.

7. Layout with Flexbox and Grid

Flexbox arranges items along a row or column, making it useful for navigation and aligned controls. CSS Grid organizes items in rows and columns, which suits card collections and page sections. Both can use gaps for consistent space. Avoid positioning every item by hand; let layout adapt to its available space.

Example

```
.navigation {
  display: flex;
  align-items: center;
  gap: 1rem;
}

.course-grid {
  display: grid;
  grid-template-columns: repeat(3, 1fr);
  gap: 1.25rem;
}
```

Check your understanding

• When is Flexbox a good fit?

• When might Grid be useful?

• What does gap do?

Suggested answers

1. Arranging items in one main direction.
2. Building a two-dimensional row and column layout.
3. Adds consistent spacing between layout items.

8. Responsive design for different screens

A responsive layout adapts to the space available. Use flexible widths, wrapping, and a viewport setting rather than fixed page widths. A media query can adjust the layout when there is enough room for more columns. Test at narrow widths, zoomed text, and in both portrait and landscape views.

Example

```
.course-grid {
  display: grid;
  grid-template-columns: 1fr;
  gap: 1rem;
}

@media (min-width: 700px) {
  .course-grid {
    grid-template-columns: repeat(2, 1fr);
  }
}
```

Check your understanding

- Why begin with a one-column layout?
- What does a media query let CSS do?
- What should you test besides desktop size?

Suggested answers

1. It works in a narrow space and can expand for larger screens.
2. Apply styles only when a screen or device condition matches.
3. Phone widths, zoom, keyboard use, and orientation.

9. Forms, focus, and accessible styling

Forms collect information. Connect every input to a visible label and use an appropriate input type. Make focus visible so keyboard users can see which control is active. Do not communicate errors by color alone; pair color with text or an icon and explain how to correct the problem. Keep controls large enough to use comfortably.

Example

```
<form>
  <label for="email">Email address</label>
  <input id="email" name="email"
         type="email" autocomplete="email">
  <button type="submit">Join</button>
</form>

:focus-visible { outline: 3px solid #087e8b; }
```

Check your understanding

- Why connect a label to an input?
- What does visible focus help with?
- Why not rely only on red to show an error?

Suggested answers

1. It tells people what information belongs in the field.
2. It shows keyboard users which control is selected.
3. Some users may not perceive the color difference.

10. Organize and debug your project

Keep HTML in .html files and styles in a linked .css file. Group images in a folder and use relative paths. When a style does not appear, check the stylesheet link, file name, selector, braces, and whether a later rule overrides it. Browser developer tools show computed styles and help identify the source of a rule.

Example

```
project/
  index.html
  styles.css
  images/

/* Use clear class names */
.course-card__title {
  margin-block: 0 0.5rem;
}
```

Check your understanding

- What should you check if CSS is missing?
- What can computed styles show?
- Why use clear class names?

Suggested answers

1. The href path, exact file name, selector, syntax, and overrides.
2. Which values the browser applies and where a rule came from.
3. They make a project easier to understand and maintain.

11. Capstone: a LearnsAppeal course page

Build a page for one course. Use a header, main content, a course description, a topic list, a cover image, and a clear PDF link. Style a course card and layout with Grid or Flexbox. Add a responsive rule. Check heading order, contrast, keyboard focus, alternative text, and that every path works.

Example

```
<!-- Course card content -->
<article class="course-card">
  <img src="images/python-cover.png"
       alt="Python Programming course cover">
  <h2>Python Programming</h2>
  <p>Practice building useful programs.</p>
  <a href="courses/python.pdf">Open course PDF</a>
</article>
```

Check your understanding

- Why is an article suitable for a course card?
- Name two checks before publishing.
- Why describe the PDF link clearly?

Suggested answers

1. It can stand as a self-contained content item.
2. Check the file path, keyboard use, mobile layout, contrast, or alt text.
3. People should understand what the link opens.

12. Publish, review, and keep improving

A site host serves your HTML, CSS, and images. Before publishing, check every link, title, image path, screen width, and keyboard interaction. Ask another person to review the page. Keep a copy of the last working version and make small updates so problems are easier to find and fix.

Example

```
Final review:
[ ] page title is clear
[ ] links and images load
[ ] layout works at phone width
[ ] keyboard focus is visible
[ ] text remains readable when zoomed
[ ] course PDF opens correctly
```

Check your understanding

- What should you verify after publishing?
- Why keep a known working version?
- How can another person help?

Suggested answers

1. Links, images, layout, readability, keyboard use, and file behavior.
2. It provides a recovery point if a change causes a problem.
3. They may notice confusing wording or usability issues you missed.

Practice projects

1. Personal profile page

Create a profile with a clear heading, short introduction, useful links, and one image. Use semantic HTML, a readable type scale, and a layout that works on a phone.

2. Course catalog

Make a page with three course cards. Use CSS Grid on wide screens and a single column on narrow screens. Add clear links and meaningful cover-image descriptions.

3. Contact form design

Create a static contact form with labeled name, email, and message fields. Style focus, helper text, and an error example. This visual exercise does not submit data to a server.

Project checklist
Does every page have a clear title? Are headings meaningful? Do links say where they go? Is the text readable? Does the layout fit narrow screens? Can you use every control by keyboard?

Glossary

Term	Meaning
Attribute	Extra information in an HTML element, such as href or alt.
Class	A reusable HTML label selected in CSS with a dot prefix.
CSS	Rules for presentation and layout.
Element	A building block of an HTML document.
Flexbox	A CSS layout system for arranging items along one main direction.
Grid	A CSS layout system for rows and columns.
HTML	Markup that describes page content and structure.
Media query	A CSS condition for applying styles in particular contexts.
Semantic HTML	Elements chosen to communicate their meaning and role.
Selector	The part of a CSS rule that chooses which elements receive styles.
Responsive design	A layout that adapts to different available screen sizes.
Alternative text	A text description that conveys an image’s useful information.

Keep learning

These free, maintained references explain HTML, CSS, and accessibility in more detail. Use them to look up elements and properties as your projects grow.

• MDN Web Docs. *Learn web development.* https://developer.mozilla.org/en-US/docs/Learn_web_development
• WHATWG. *HTML Living Standard.* https://html.spec.whatwg.org/
• W3C. *Web Content Accessibility Guidelines (WCAG) 2.2.* https://www.w3.org/TR/WCAG22/

Course completion checklist
I can create a structured HTML page; link a stylesheet; style text, color, and spacing; arrange content with Flexbox or Grid; adapt a page to phones; add useful alternative text and visible focus; and review a page before publishing.