WEB DEVELOPMENT

A practical guide from your first page to a connected, accessible website

LEARNSAPPEAL | ORIGINAL COURSE

Learn the pieces, then bring them together.
Start with the browser, HTML, CSS, and JavaScript. Continue into forms, web APIs, project structure, security, testing, and publishing a real site.

Before you start
Basic computer skills and a code editor are enough. The first examples are small and explained step by step. Later modules use a few HTML, CSS, and JavaScript files. Keep a folder for your work and change one thing at a time.

Course map: Core building blocks (Modules 1-5), followed by building and connecting a site (Modules 6-12). Includes examples, self-checks with answers, projects, and a glossary.

How to use this course

Follow the modules in order. Type the examples, then change the text, colors, and behavior. Open the page in a browser and use developer tools to inspect errors and layout. When something goes wrong, change one thing at a time and read the error message.

By the end, you can: create pages with meaningful HTML; style them for different screens; add interactions with JavaScript; connect a page to an API; organize a small project; check keyboard access and security basics; and publish a site with a plan for updates.

Small steps work well. You do not need a large framework to make a useful website. Learn the browser basics first. Frameworks can help organize bigger projects, but the skills in this course still apply when you use one.

Table of contents

How to use this course 2
1. What happens when you open a website? 3
2. Build a clear page with HTML 3
3. Style pages with CSS 4
4. Make layouts work on phones and support all users 4
5. Add behavior with JavaScript 5
6. Organize files and reusable page pieces 5
7. Forms and browser validation 6
8. Load and send data with web APIs 6
9. Understand the server and data behind a site 7
10. Security basics for websites 7
11. Test, debug, and improve the site 8
12. Publish a site and complete the LearnsAppeal project 8
Guided lab: create a course card 9
Project 1: responsive course catalog 9
Project 2: enrollment form 9
Project 3: resource search page 10
Glossary 10
Keep learning 10

CORE BUILDING BLOCKS

1. What happens when you open a website?

The browser requests a page

When you enter a web address, the browser finds the site, connects securely, and asks a server for files. The server sends a response that may contain HTML, CSS, JavaScript, images, or data. The browser reads these files and displays the page.

Three building blocks

HTML gives content its structure and meaning. CSS controls appearance and layout. JavaScript adds behavior, such as opening a menu or loading new results.

Your project folder

A small site might contain index.html, styles.css, script.js, and an images folder. Use descriptive names and keep related work together. A local server is useful when modules or network requests are involved.

Developer tools

The Elements panel shows page structure. The Console reports JavaScript errors. The Network panel shows files and API requests. The device toolbar previews screen widths.

Check your understanding

- What does HTML provide?
- Which panel helps find JavaScript errors?
- What can the Network panel show?

Suggested answers

1. Page structure and meaning.

2. The Console.

3. Files and requests sent to servers.

2. Build a clear page with HTML

Elements and nesting

An HTML element usually has an opening tag, content, and a closing tag. Elements can sit inside other elements. Indentation helps show which items belong together.

Choose elements by meaning

Use headings for sections, paragraphs for text, lists for groups, links for navigation, and buttons for actions. Semantic elements help browsers and assistive technologies understand the content.

Links, images, and page titles

Give each page a descriptive title. Link text should explain where it goes. Add useful alternative text to informative images; use empty alternative text for decorative images. Keep headings in a sensible order.

Check your understanding

- When should you use a link rather than a button?
- Why is a page title useful?
- What should alternative text do?

Suggested answers

1. Use a link to go somewhere; use a button to perform an action.

2. It identifies the page in browser tabs, bookmarks, and assistive technology.

3. Describe useful image information, or be empty when decorative.

3. Style pages with CSS

Rules and selectors

A CSS rule selects elements and applies declarations such as color, spacing, or font size. A class is a reusable label you add to HTML. Keep selectors simple.

The box model

Every element has content, padding, border, and margin. Padding adds space inside the border; margin adds space outside it. border-box sizing makes width calculations easier.

Layout with Flexbox and Grid

Flexbox arranges items in one direction, such as a navigation bar. Grid is useful for rows and columns, such as a card layout. Use flexible sizes and gaps.

Readable design

Choose a small set of colors, font sizes, and spacing values. Keep enough contrast and show visible focus for keyboard users.

Check your understanding

- What is the difference between padding and margin?
- When might CSS Grid help?
- Why should focus be visible?

Suggested answers

1. Padding is inside an element border; margin is outside it.

2. When arranging items in rows and columns.

3. Keyboard users need to see which control is active.

4. Make layouts work on phones and support all users

Responsive layout

A responsive page adapts to its available width. Start with a layout that works on a narrow screen, then add changes for wider screens. Flexible widths, wrapping, and media queries can help.

Touch and readable controls

Buttons and links need enough space to tap. Do not make meaning depend on color alone. Avoid tiny text and horizontal scrolling for ordinary content. Respect preferences such as reduced motion.

Keyboard and assistive technology

Try the page using Tab, Shift+Tab, Enter, and Space. Keep focus order logical and visible. Use labels with form controls.

Check the result

Automated tools catch some common problems, but do not understand every interaction. Combine a scan with keyboard use, browser zoom, and manual review. WCAG offers testable recommendations.

Check your understanding

- What should you test beyond a wide desktop layout?
- Why is color alone a poor way to show an error?
- Do automated scans find every accessibility issue?

Suggested answers

1. Narrow screens, zoomed text, keyboard use, and touch controls.

2. Some people cannot distinguish the colors, and color may not be announced by assistive technology.

3. No. Manual review and real interaction checks are also needed.

5. Add behavior with JavaScript

Values and variables

JavaScript stores text, numbers, true/false values, lists, and objects. Use const when a name will not be reassigned and let when it will.

Functions and decisions

A function groups steps under a name. Conditions choose which steps to run. Loops repeat work. Small functions are easier to test and change.

Events and the page

A browser event can happen when someone clicks, types, submits a form, or resizes a window. JavaScript can listen and update the page. Use textContent to insert ordinary text.

Debugging

Read the first useful error in the Console. Check spelling, brackets, and whether an element exists. Use a temporary console.log to inspect a value.

Check your understanding

- What is a function for?
- Name two events JavaScript can respond to.
- Which is safer for plain text: innerHTML or textContent?

Suggested answers

1. It groups reusable steps under a meaningful name.

2. Click and form submission; typing and resizing are also events.

3. textContent.

BUILD AND CONNECT

6. Organize files and reusable page pieces

Separate the parts

Keep structure in HTML, appearance in CSS, and behavior in JavaScript files. Use clear folders as a project grows.

Reusable pieces

A navigation header, course card, or footer may appear on several pages. Shared components or templates keep repeated patterns consistent.

When to use a framework

A framework can help manage complex screens, components, and routing. It also adds concepts and build steps. Learn browser basics first, then choose based on project size and maintenance.

Track changes with Git

Git records changes over time. Make small commits with messages that explain what changed. Review the diff before saving a commit.

Check your understanding

- Why keep HTML, CSS, and JavaScript roles clear?
- Why use reusable components?
- What does a Git commit record?

Suggested answers

1. It makes the project easier to read, debug, and change.
2. To keep repeated patterns consistent and update them in one place.
3. A snapshot of focused changes with a message.

7. Forms and browser validation

Use a real form

Forms let people send information. Use labels, helpful input types, buttons, and group related controls together.

Validate on the server too

Browser validation can be bypassed. A server must check every value it receives, including type, length, and allowed choices.

Accessible feedback

Tell people what happened after submission. Identify fields with problems and explain how to fix them. Do not rely on color alone.

Protect personal data

Only ask for information the site needs. Explain why sensitive information is collected and who can access it.

Check your understanding

- Can browser validation replace server validation?
- What makes an error message helpful?
- Why collect only necessary personal data?

Suggested answers

1. No. A request can be sent without using the browser form.
2. It identifies the field and explains a correction.
3. It reduces exposure and limits data the site must protect.

8. Load and send data with web APIs

What is an API?

An API is a defined way for one program to ask another for data or an action. A web API often uses HTTP requests and returns JSON.

Use Fetch

Fetch sends a request and returns a promise. A response with an error status such as 404 still counts as a response, so check response.ok. Handle network errors too.

Loading, empty, and error states

Show a loading state while waiting, an empty state when there are no results, and an error state when the request fails.

Avoid stale results

If someone searches quickly, an older request may finish after a newer one. Cancel outdated work or ignore its result.

Check your understanding

- What does response.ok help you check?
- Name three states a data page should handle.
- Why cancel an old search request?

Suggested answers

1. Whether the HTTP response status indicates success.
2. Loading, empty, and error.
3. To avoid wasted work and stale results.

9. Understand the server and data behind a site

Client and server

The browser is often called the client. It asks a server for pages or data. The server may check permissions, apply rules, save information, and respond.

Routes and resources

A route connects a URL and HTTP method to behavior. A GET route can return courses; a POST route can submit enrollment. Use clear status codes.

Databases

A database stores information for later retrieval and updates. Tables organize related records. Use constraints to prevent invalid data and parameterized queries to keep user text separate from instructions.

Do not expose secrets

Passwords and database credentials belong on a protected server, not in JavaScript sent to every browser. Use server-side access checks for private information.

Check your understanding

- Where should important permission checks happen?
- What does a database do?
- Why use parameterized queries?

Suggested answers

1. On the server for each protected action or resource.
2. It stores and organizes information.
3. They keep user-provided values separate from query instructions.

10. Security basics for websites

Treat input as untrusted

A visitor can change any value sent by the browser. Validate data on the server and escape text for the place it will be displayed. Avoid inserting untrusted strings as HTML.

Common risks

Cross-site scripting can run untrusted content as code in a page. Cross-site request forgery can misuse a signed-in browser to submit an unwanted request. Use platform protections, secure cookies, and server-side authorization.

Authentication and authorization

Authentication checks who is signed in. Authorization checks whether that person can perform an action or see a record.

A simple security habit

Before launch, ask what data is private, who can change it, and what an attacker might try. Use a trusted library for login.

Check your understanding

- Why is text from a visitor untrusted?
- What is the difference between authentication and authorization?
- Why use a trusted login library?

Suggested answers

1. It can be changed to contain invalid or harmful content.
2. Authentication establishes identity; authorization checks permission.
3. Login has subtle security requirements and mistakes can expose accounts.

11. Test, debug, and improve the site

Test useful behavior

Check links, forms, navigation, empty results, small screens, and error states. Try keyboard-only use and browser zoom.

Fix errors in order

Reproduce the issue, inspect Console and Network panels, make one small change, then check again. Compare the diff if the issue started after a change.

Performance basics

Large images and unnecessary scripts slow pages. Resize images for their display size and load only what a page needs.

After publishing

Watch for broken links, failed requests, and user reports. Keep a copy of the last working release and know how to restore it.

Check your understanding

- Name two useful manual checks before launch.
- What should you do after reproducing a bug?
- Name one simple way to improve speed.

Suggested answers

1. Keyboard navigation and narrow-screen layout; links and forms are useful too.
2. Inspect the error, change one thing, and test again.
3. Resize images or remove scripts the page does not need.

12. Publish a site and complete the LearnsAppeal project

Build and hosting

A website host makes files or an application available through a domain. Static sites serve HTML, CSS, JavaScript, images, and PDFs. A site with accounts or saved data also needs a backend and data protections.

Plan an update

Keep course title, topic, summary, and file path consistent. Test every link after adding a PDF. If a file is missing, show a useful message.

Capstone: course catalog

Create a catalog with topic filters, course cards, and PDF links. Add search and an empty state. Make it usable on a phone, by keyboard, and with browser zoom.

Capstone checklist

Use meaningful headings and page titles; label controls; show focus; check every PDF link; avoid publishing private information; test missing-file behavior; and record how to update the catalog.

Check your understanding

- What extra pieces does a site with accounts need?
- What should you test after adding a course PDF?
- Name one useful empty state.

Suggested answers

1. A backend, data storage, authentication, authorization, and privacy/security plan.
2. The link, file name, open behavior, and missing-file state.
3. A note that no courses matched the selected topic or search.

Guided lab: create a course card

Save these files in one folder. The HTML uses an article, heading, description, and link. Add a second card, then change the CSS to see what each rule affects.

index.html

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <title>LearnsAppeal Courses</title>
  <link rel="stylesheet" href="styles.css">
</head>
<body>
  <main>
    <h1>Explore a course</h1>
    <article class="course-card">
      <h2>Python Foundations</h2>
      <p>Practice writing small programs and useful scripts.</p>
      <a href="courses/python.pdf">Open course PDF</a>
    </article>
  </main>
</body>
</html>
```

styles.css

```
* { box-sizing: border-box; }
body { margin: 0; padding: 2rem; color: #193047; font: 1rem/1.5 system-ui, sans-serif; }
.course-card { max-width: 34rem; padding: 1.25rem; border: 1px solid #cbd5df; border-radius: 0.75rem; }
a:focus-visible { outline: 3px solid #087e8b; outline-offset: 3px; }
```

Try next: Add an image with useful alternative text. Create another card. Use a narrow browser window and confirm the card still fits. Keep the course link usable even if JavaScript fails.

Project 1: responsive course catalog

Build a list of courses from a small set of data. Add a topic filter and search. Show a helpful empty state, keep filters usable by keyboard, and ensure every PDF link points to a real file. Write a checklist for adding a course.

Project 2: enrollment form

Create a form for a learner to request enrollment. Add labels and helpful required fields, show confirmation and error messages, and validate on the server if you connect a backend. Do not collect information the course does not need.

Project 3: resource search page

Fetch a small JSON list of resources and display matching results. Add loading, error, and no-results states. Prevent old searches from replacing new ones. Include a check for a slow connection and an empty response.

Glossary

Term	Meaning
API	A defined way for programs to exchange requests and results.
CSS	Rules that control the appearance and layout of a web page.
DOM	The browser object model for the current HTML document.
HTML	Markup that gives page content structure and meaning.
HTTP	The request and response protocol commonly used by browsers and servers.
JavaScript	A programming language used for browser behavior and server applications.
Responsive layout	A layout that adapts to the available screen or container size.
Semantic element	An HTML element whose name describes its role.
Server	A program or computer that responds to requests.
URL	An address that identifies a web resource.
Validation	Checking that data follows expected rules.
WCAG	W3C Web Content Accessibility Guidelines.

Keep learning

These maintained references offer a next step for detailed HTML behavior, accessibility guidance, and practical browser examples. Check current documentation when using evolving APIs.

- 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/
- MDN Web Docs. Making network requests with JavaScript. https://developer.mozilla.org/en-US/docs/Learn_web_development/Core/Scripting/Network_requests
- OWASP Cheat Sheet Series. Cross Site Scripting Prevention. https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html

You have completed this course when you can:
Create a page with meaningful HTML; style it for different screens; add an interaction; connect to a data source; test links, forms, and keyboard access; explain server and security responsibilities; and publish with a clear way to update course content.