Web Programming

HTML & CSS

Fenerbahce University

HTML Foundations: Key ideas

- HTML gives structure and meaning to page content.
- Browsers read elements such as headings, paragraphs, links, and lists.
- HTML is not a programming language and does not control page styling.
- A good HTML file starts with a valid document structure.

HTML Foundations: Practical notes

- Think in terms of content: title, section, paragraph, action, media.
- Browsers ignore extra whitespace but still require logical nesting.
- Opening and closing tags must match to create predictable structure.
- The browser can render plain HTML before any CSS is added.

Minimal HTML page

HTML Unit 01 • Example

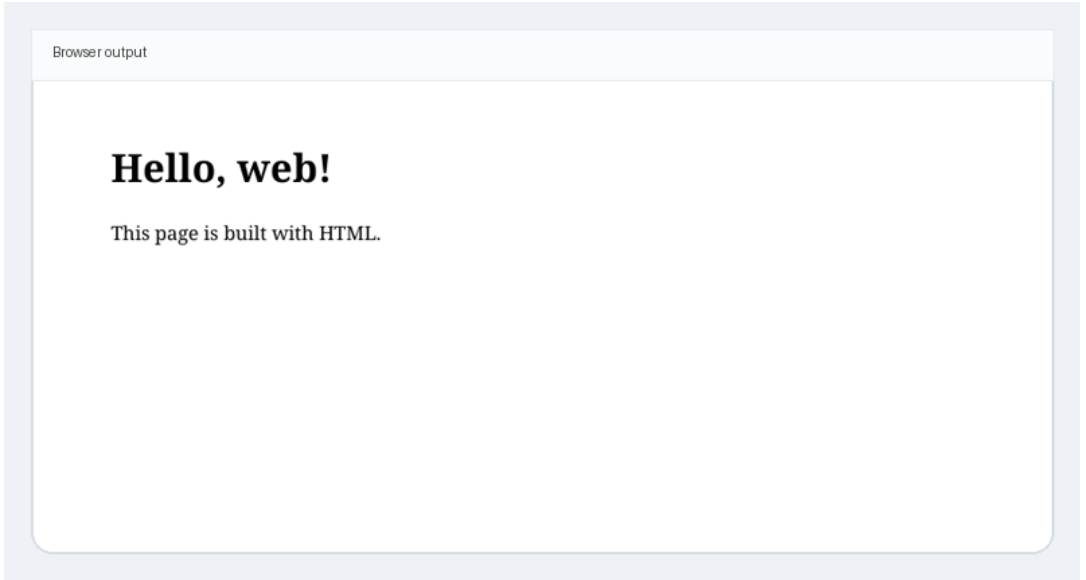
What to notice: The browser tab uses the <title> text from the head. • Visible page content lives inside the <body> element.

Code

HTML

```html
<!doctype html>
<html>
  <head>
    <title>My First Page</title>
  </head>
  <body>
    <h1>Hello, web!</h1>
    <p>This page is built with HTML.</p>
  </body>
</html>
```

Rendered output



Headings and a short intro

HTML Unit 01 • Example

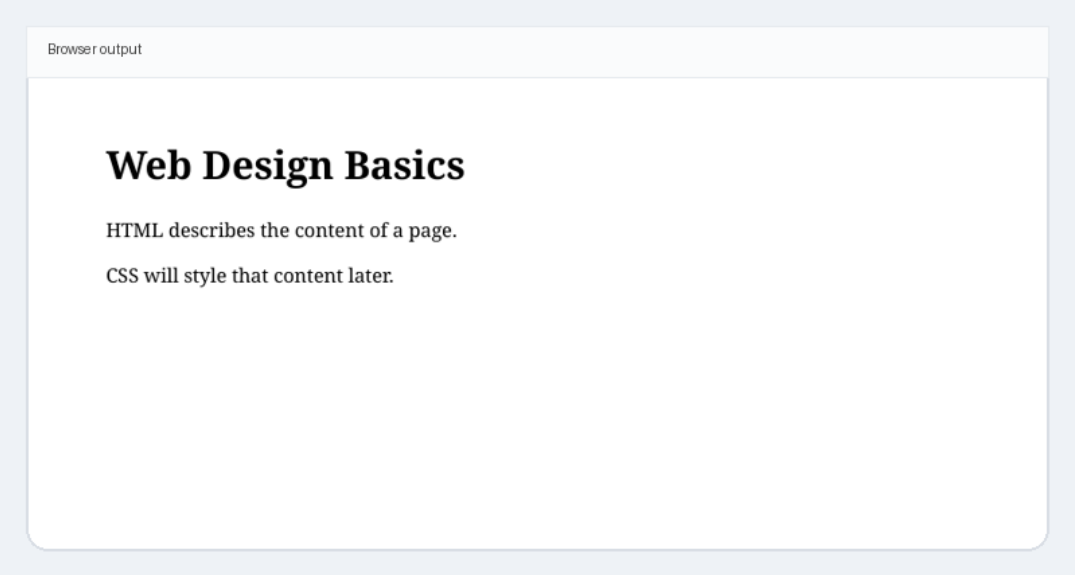
What to notice: Headings introduce the topic of the page or section. • Paragraphs break ideas into readable blocks.

Code

HTML

```
1  <h1>Web Design Basics</h1>
2  <p>HTML describes the content of a page.</p>
3  <p>CSS will style that content later.</p>
```

Rendered output



HTML ignores extra spaces

HTML Unit 01 • Example

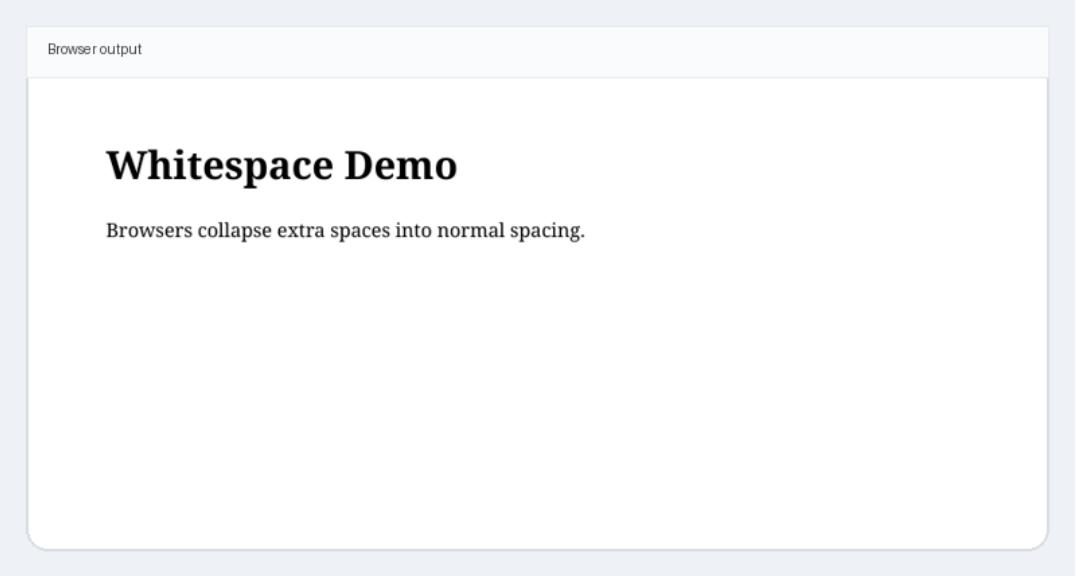
What to notice: Many spaces or line breaks are collapsed in normal text. • Structure matters more than spacing inside the source code.

Code

HTML

```
<h1>Whitespace Demo</h1>
<p>Browsers          collapse
extra     spaces
into normal spacing.</p>
```

Rendered output



Comments in source code

HTML Unit 01 • Example

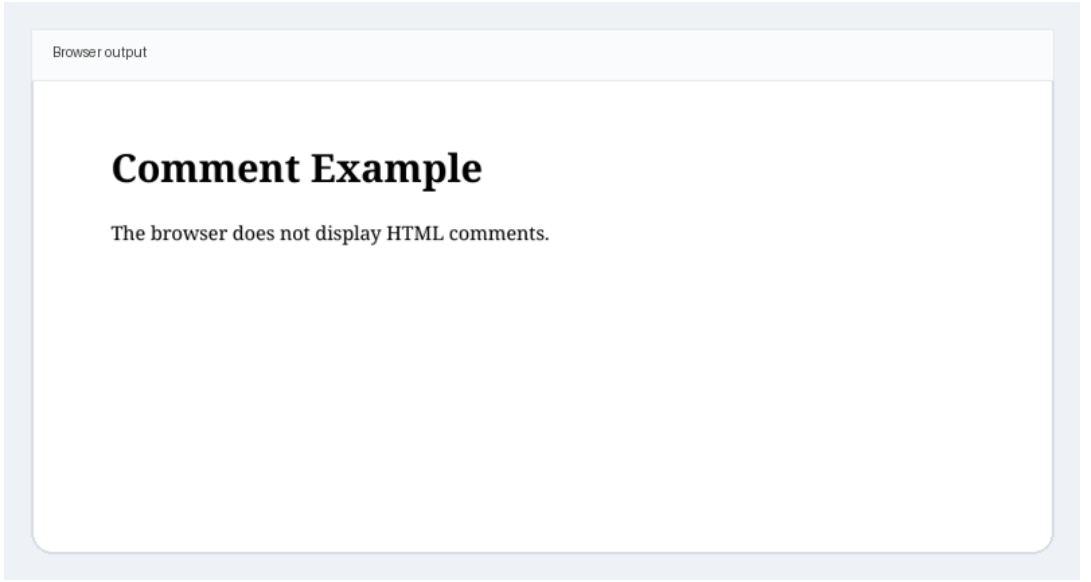
What to notice: Comments help developers explain or temporarily hide code. • Comments do not appear as page content.

Code

HTML

```
 1  <!-- This note is visible in the source, not on the page
-->
 2  <h1>Comment Example</h1>
 3  <p>The browser does not display HTML comments.</p>
```

Rendered output



A simple page with multiple elements

HTML Unit 01 • Example

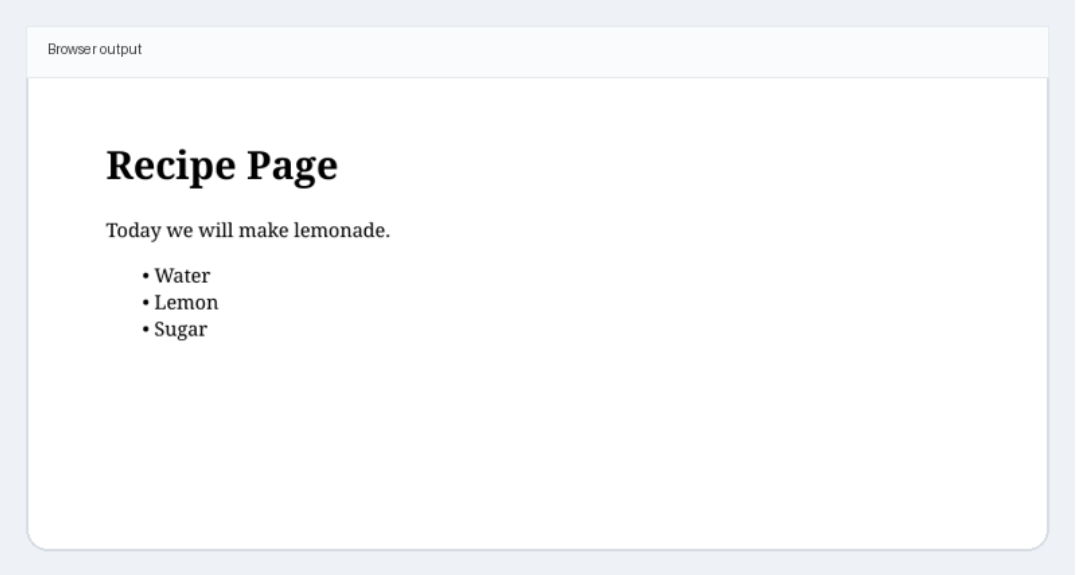
What to notice: One small HTML file can combine headings, text, and lists. • Default browser styles already create readable separation.

Code

HTML

```html
<h1>Recipe Page</h1>
<p>Today we will make lemonade.</p>
<ul>
  <li>Water</li>
  <li>Lemon</li>
  <li>Sugar</li>
</ul>
```

Rendered output



HTML Foundations: Common mistakes

- Do not place visible page content inside the <head>.
- Do not forget closing tags when the element requires one.
- Do not use HTML to “draw” layout with repeated line breaks.
- Use meaningful elements instead of a page made of plain text only.

HTML Foundations: Recap

- HTML describes meaning and structure before presentation.
- The basic document has doctype, html, head, and body.
- Readable source code helps debugging and teamwork.
- Default browser rendering is enough for early prototypes.

Web Programming

HTML Unit 02

Document Structure

- Every page should begin with <!doctype html> to trigger standards mode.
- The <html> element wraps the entire document.

Fenerbahce University

Document Structure: Key ideas

- Every page should begin with <!doctype html> to trigger standards mode.
- The <html> element wraps the entire document.
- The <head> stores metadata; the <body> stores visible content.
- Language and charset information improve accessibility and compatibility.

Document Structure: Practical notes

- A clean skeleton becomes the starting point for almost every page.
- Meta tags do not usually appear visually, but they still matter.
- The lang attribute helps screen readers and search engines.
- Nested sections inside the body create an easier-to-read outline.

Standard HTML skeleton

HTML Unit 02 • Example

What to notice: This skeleton is a safe starting point for new projects. • The lang and charset settings belong in the document root and head.

Code

HTML

```html
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>Course Page</title>
  </head>
  <body>
    <h1>HTML Structure</h1>
  </body>
</html>
```

Rendered output

Head and body working together

HTML Unit 02 • Example

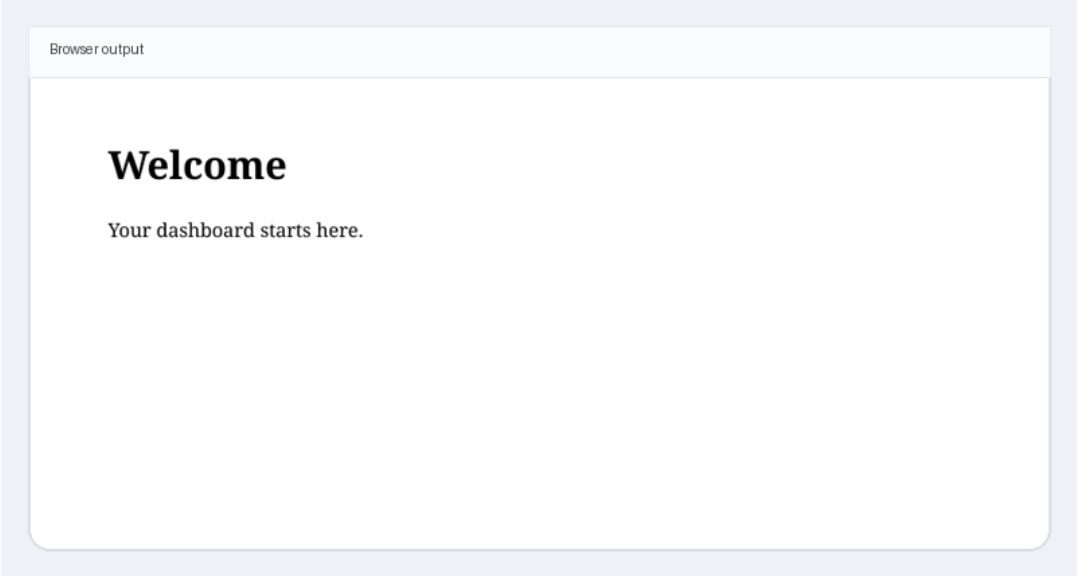
What to notice: The title appears in the browser tab, not inside the page. • The heading and paragraph appear in the body area.

Code

HTML

```html
<!doctype html>
<html lang="en">
  <head>
    <title>Student Portal</title>
  </head>
  <body>
    <h1>Welcome</h1>
    <p>Your dashboard starts here.</p>
  </body>
</html>
```

Rendered output



Using the lang attribute

HTML Unit 02 • Example

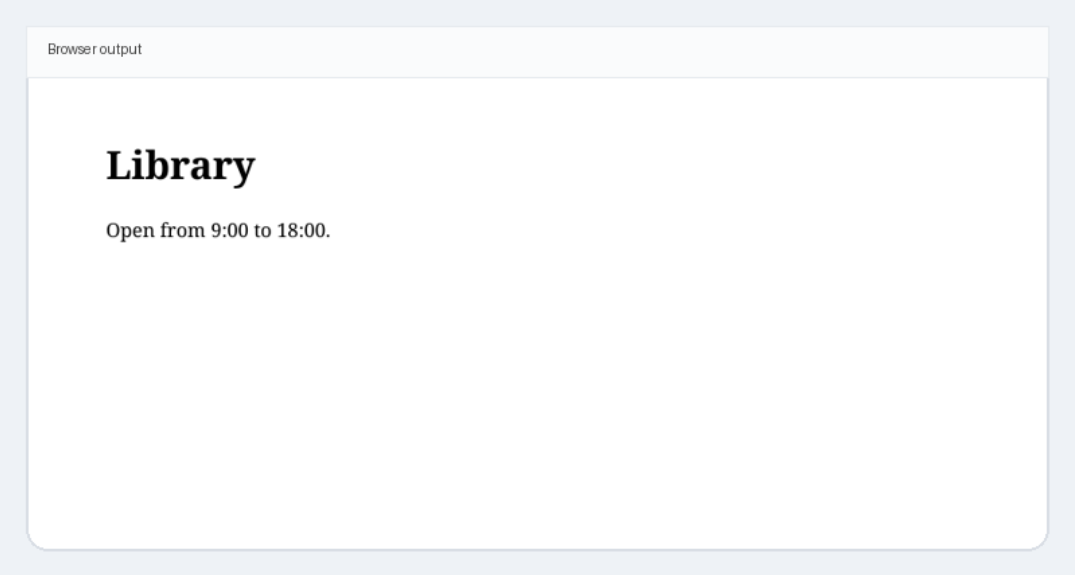
What to notice: Language information supports pronunciation and search relevance. • Set the main page language once on the html element.

Code

HTML

```
<!doctype html>
<html lang="en">
  <body>
    <h1>Library</h1>
    <p>Open from 9:00 to 18:00.</p>
  </body>
</html>
```

Rendered output



A body with semantic sections

HTML Unit 02 • Example

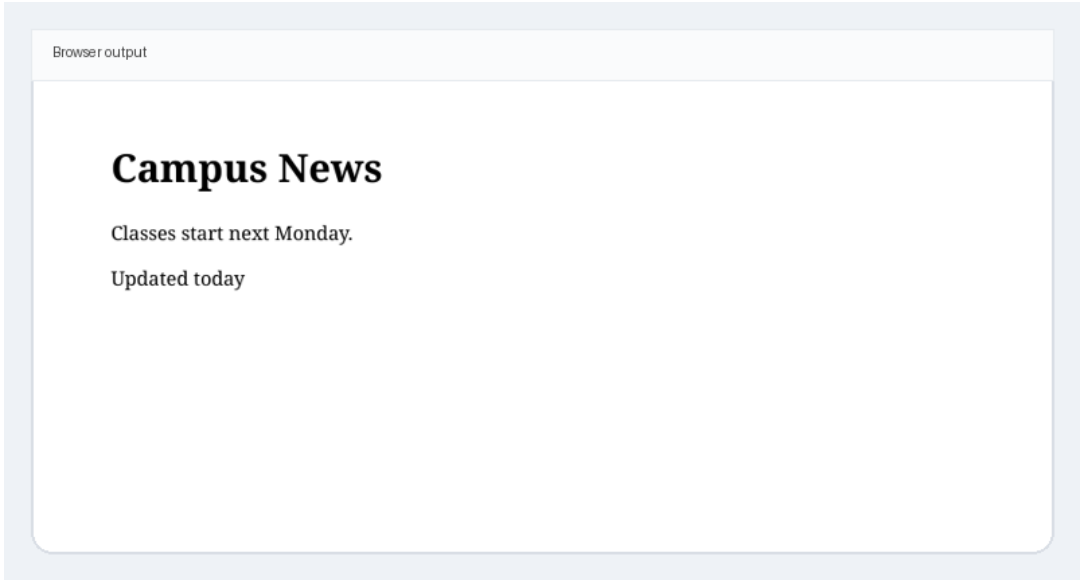
What to notice: Even simple pages benefit from header, main, and footer structure. • Semantic sections become more powerful as pages grow.

Code

HTML

```html
<body>
  <header><h1>Campus News</h1></header>
  <main><p>Classes start next Monday.</p></main>
  <footer><p>Updated today</p></footer>
</body>
```

Rendered output



A readable indentation style

HTML Unit 02 • Example

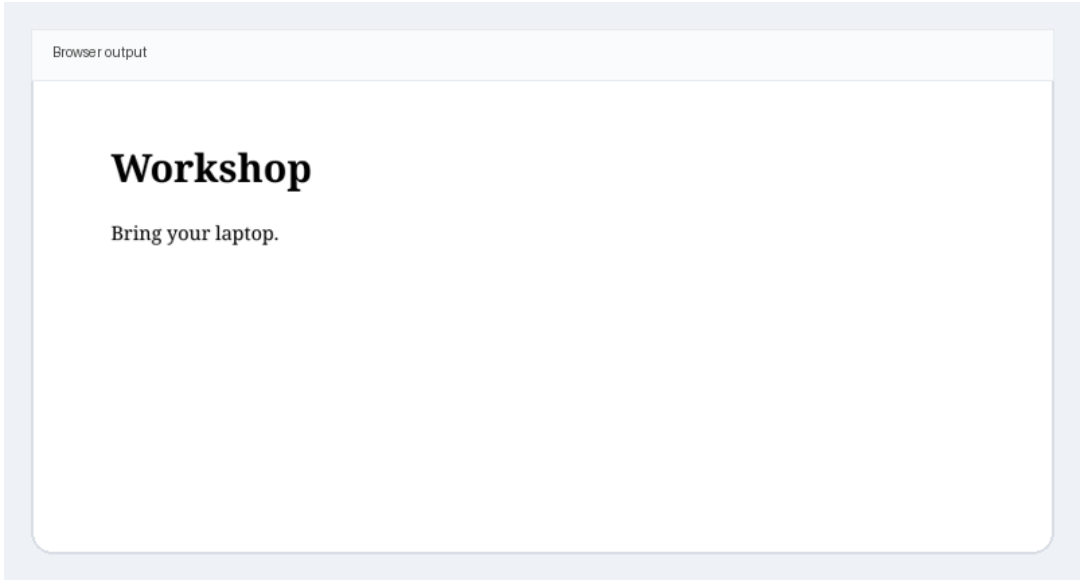
What to notice: Indentation reveals the parent–child relationship of elements. • Good formatting reduces structural mistakes.

Code

HTML

```html
<body>
  <main>
    <section>
      <h1>Workshop</h1>
      <p>Bring your laptop.</p>
    </section>
  </main>
</body>
```

Rendered output



Document Structure: Common mistakes

- Do not skip the doctype in modern documents.
- Do not put headings, images, or forms directly inside the head.
- Do not leave the page language undefined when you know it.
- Do not mix unrelated content at the same nesting level without structure.

Document Structure: Recap

- Structure starts before visible content appears.
- The document root, head, and body each have a clear responsibility.
- Language and metadata make the page more reliable.
- A neat skeleton saves time during every later lesson.

Web Programming

HTML Unit 03

Text Content Basics

- Headings create a content hierarchy from <h1> to <h6>.
- Paragraphs use <p>, not repeated line breaks.

Fenerbahce University

Text Content Basics: Key ideas

- Headings create a content hierarchy from <h1> to <h6>.
- Paragraphs use <p>, not repeated line breaks.
- Use
 only for intentional line breaks inside the same idea.
- <hr> marks a thematic break between topics.

Text Content Basics: Practical notes

- A single page should usually have one main <h1>.
- Do not jump heading levels only because a lower level looks smaller.
- Use paragraphs for normal prose and <pre> for preformatted text.
- Choose text elements by meaning, not visual size.

Heading hierarchy

HTML Unit 03 • Example

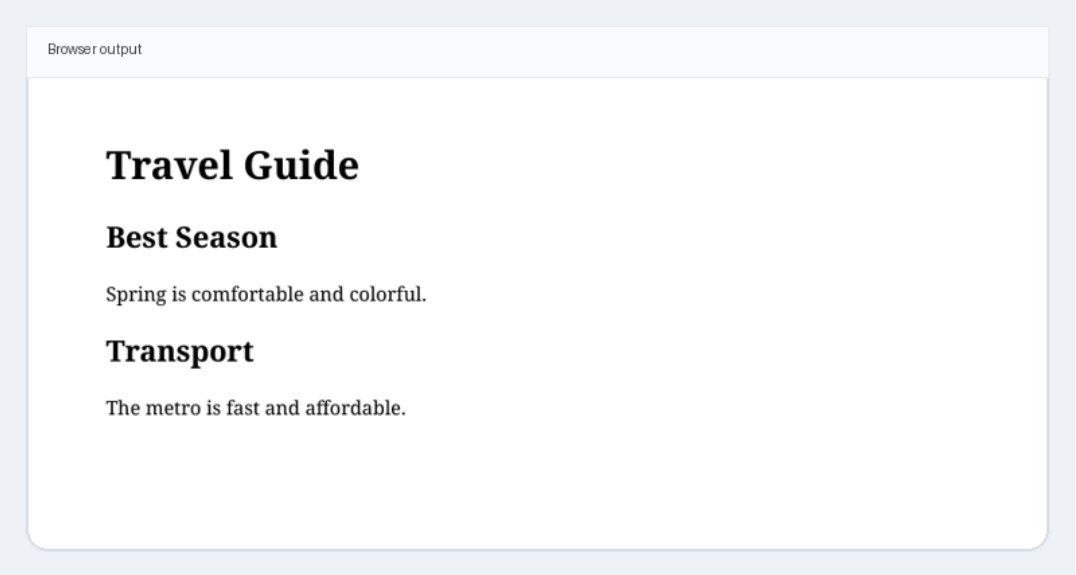
What to notice: Subheadings divide the page into related sections. • The structure should read like an outline.

Code

HTML

```
<h1>Travel Guide</h1>
<h2>Best Season</h2>
<p>Spring is comfortable and colorful.</p>
<h2>Transport</h2>
<p>The metro is fast and affordable.</p>
```

Rendered output



Paragraphs and a line break

HTML Unit 03 • Example

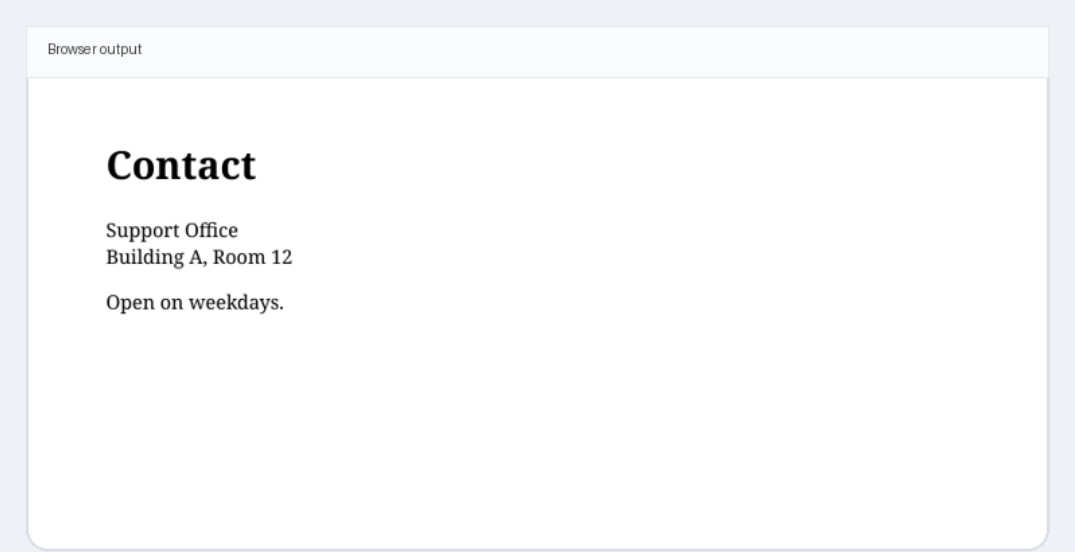
What to notice: Use
 for a line break within a single block of text. • Separate ideas still belong in separate paragraphs.

Code

HTML

```
<h1>Contact</h1>
<p>Support Office<br>Building A, Room 12</p>
<p>Open on weekdays.</p>
```

Rendered output



Thematic break with hr

HTML Unit 03 • Example

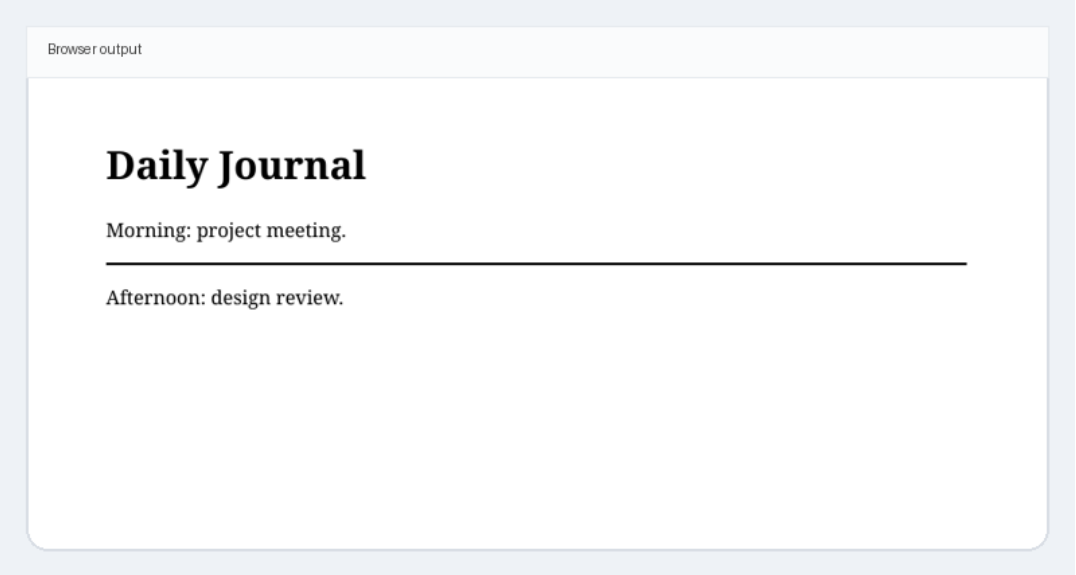
What to notice: <hr> signals a shift in topic or section. • It is stronger than simply adding blank space.

Code

HTML

```
1  <h1>Daily Journal</h1>
2  <p>Morning: project meeting.</p>
3  <hr>
4  <p>Afternoon: design review.</p>
```

Rendered output



Preformatted text

HTML Unit 03 • Example

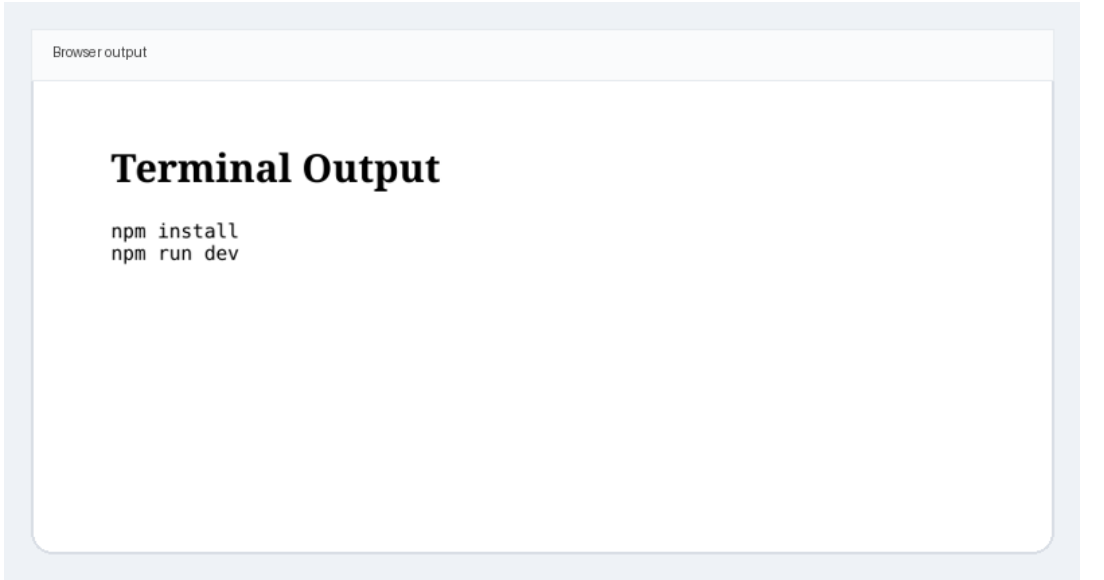
What to notice: <pre> keeps spaces and line breaks as written. • It is useful for code, commands, or formatted text blocks.

Code

HTML

```
<h1>Terminal Output</h1>
        <pre>
npm install
npm run dev
        </pre>
```

Rendered output



Blockquote for quoted text

HTML Unit 03 • Example

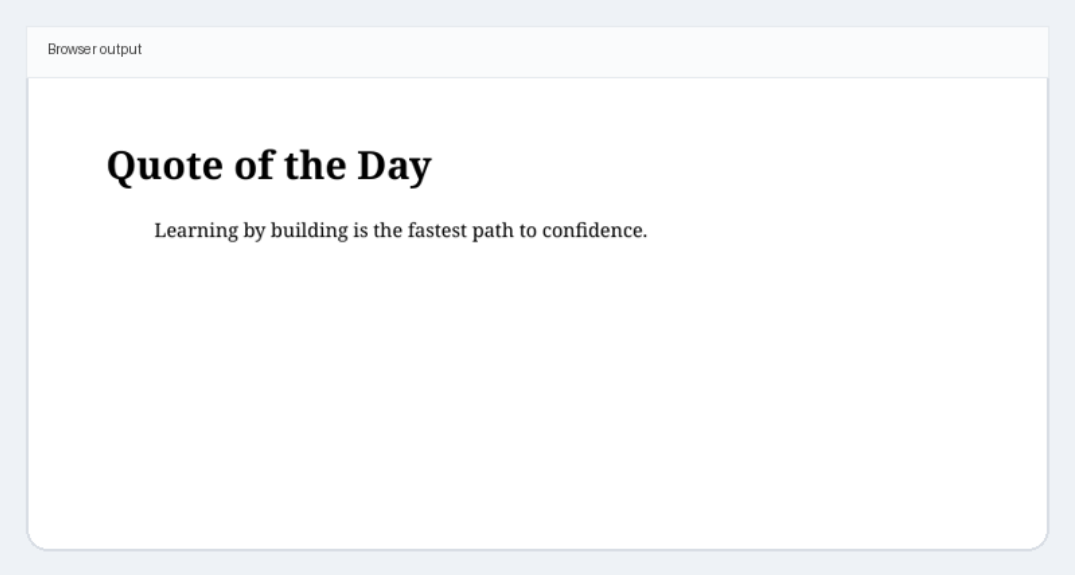
What to notice: Block quotes distinguish quoted material from regular paragraphs. • Quoted content gets its own semantic element.

Code

HTML

```
<h1>Quote of the Day</h1>
<blockquote>
  Learning by building is the fastest path to confidence.
</blockquote>
```

Rendered output



Text Content Basics: Common mistakes

- Do not use headings only to make text look bold or large.
- Do not replace paragraphs with many
 tags.
- Do not use <pre> for layout tricks.
- Keep heading order logical for readers and assistive tools.

Text Content Basics: Recap

- Text elements shape how content is read and understood.
- Headings, paragraphs, and thematic breaks each do different jobs.
- Small elements like
 should stay focused and limited.
- Meaningful text structure improves both clarity and accessibility.

Web Programming

HTML Unit 04

Inline Text Semantics

- Inline elements add meaning inside sentences without starting a new block.
- Use and for importance and emphasis.

Fenerbahce University

Inline Text Semantics: Key ideas

- Inline elements add meaning inside sentences without starting a new block.
- Use and for importance and emphasis.
- Other useful tags include <mark>, <small>, <sub>, <sup>, and <code>.
- Semantic inline tags give assistive technology richer context.

Inline Text Semantics: Practical notes

- Choose for importance, not just visual boldness.
- Choose when a different spoken stress changes meaning.
- <abbr> and <time> attach extra meaning to ordinary text.
- Inline semantic elements make content more precise and searchable.

Strong and emphasis

HTML Unit 04 • Example

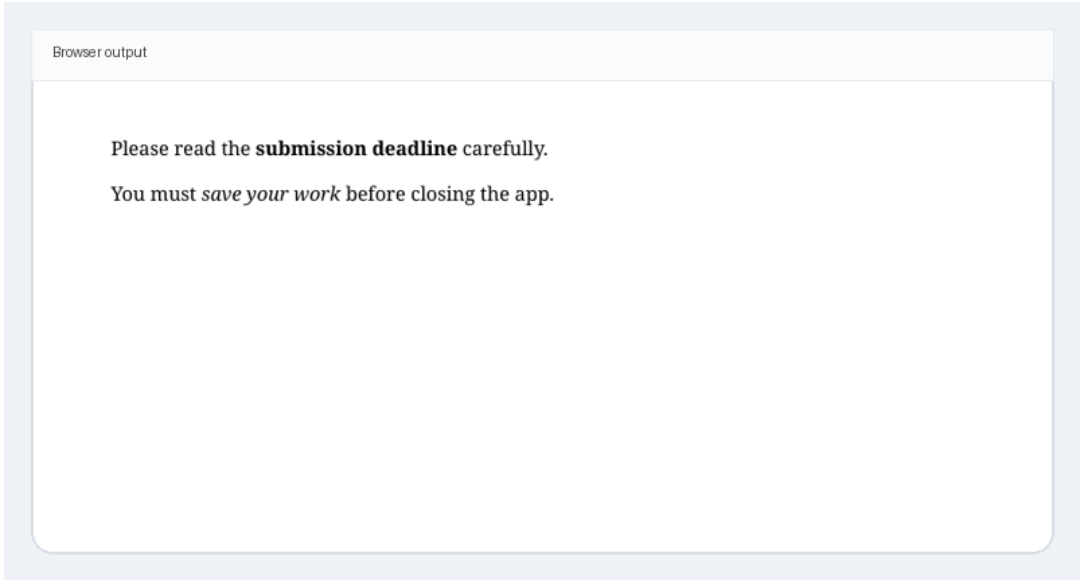
What to notice: Importance and emphasis are related, but not identical. • Browsers style them differently by default.

Code

HTML

```
 1  <p>Please read the <strong>submission deadline</strong>
carefully.</p>
 2  <p>You must <em>save your work</em> before closing the
app.</p>
```

Rendered output



Mark, small, and time

HTML Unit 04 • Example

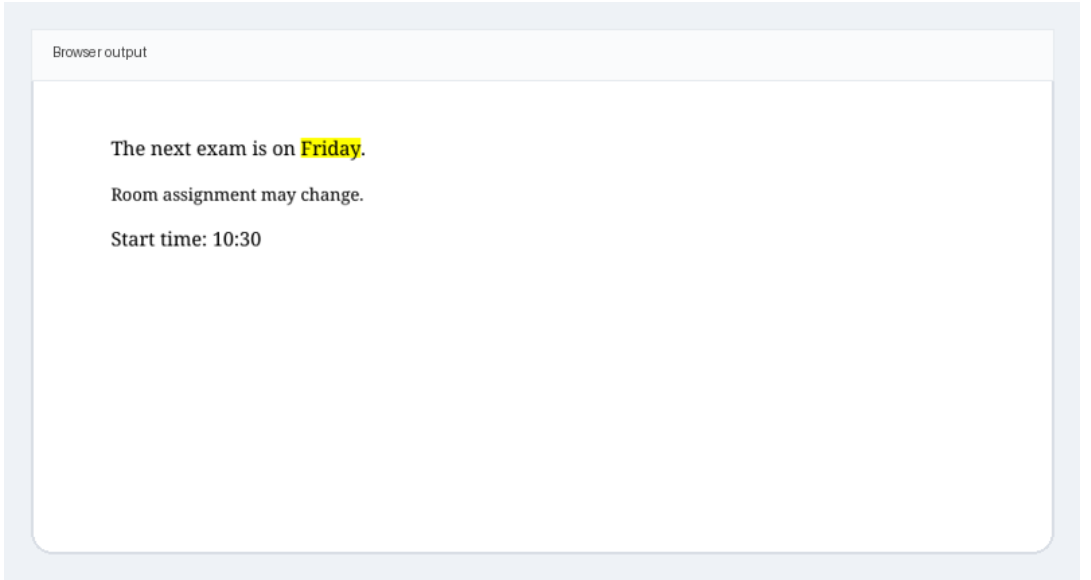
What to notice: Inline tags can highlight, qualify, or label details. • These elements still belong naturally inside paragraphs.

Code

HTML

```
1  <p>The next exam is on <mark>Friday</mark>.</p>
2  <p><small>Room assignment may change.</small></p>
3  <p>Start time: <time>10:30</time></p>
```

Rendered output



Subscript and superscript

HTML Unit 04 • Example

What to notice: Subscript and superscript communicate technical notation clearly. • They should be used for meaning, not for random decoration.

Code

HTML

```
1  <p>Water is written as H<sub>2</sub>O.</p>
2  <p>Area can be expressed as m<sup>2</sup>.</p>
```

Rendered output

Browser output

Water is written as H_2O.

Area can be expressed as m^2.

Code and keyboard input

HTML Unit 04 • Example

What to notice: Commands and keyboard keys deserve dedicated inline elements. • The default browser styles already hint at their special meaning.

Code

HTML

```
 1  <p>Use <code>npm start</code> to run the
project.</p>
 2  <p>Press <kbd>Ctrl</kbd> + <kbd>S</kbd> to
save.</p>
```

Rendered output

Abbreviation and inserted text

HTML Unit 04 • Example

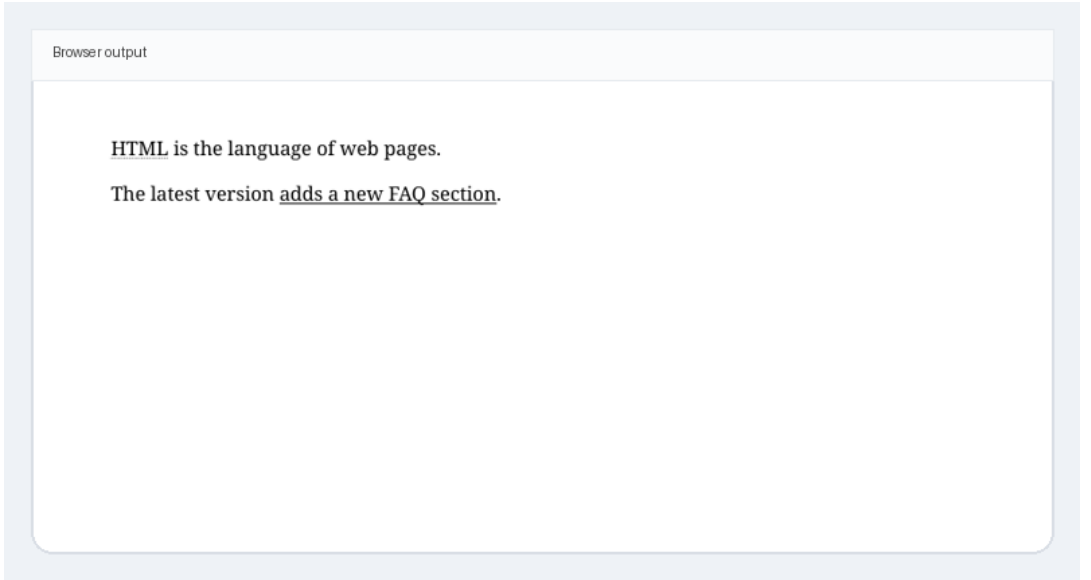
What to notice: Abbreviations can provide a full meaning through the title attribute. • Inserted text can mark newly added content in a revision.

Code

HTML

```
 1  <p><abbr title="HyperText Markup Language">HTML</abbr> is the
language of web pages.</p>
 2  <p>The latest version <ins>adds a new FAQ section</ins>.</p>
```

Rendered output



Inline Text Semantics: Common mistakes

- Do not replace semantic tags with plain bold or italic formatting habits.
- Do not overuse <mark> so often that nothing stands out anymore.
- Do not use <sub> or <sup> just to shrink or raise normal text.
- Keep inline elements inside meaningful sentence flow.

Inline Text Semantics: Recap

- Inline semantics enrich the meaning of ordinary sentences.
- Different tags serve emphasis, notation, quoting, code, and metadata.
- Choosing the right tag improves both clarity and machine interpretation.
- Small semantic details add up to higher-quality HTML.

Web Programming

HTML Unit 05

Lists

- Use unordered lists for items without ranking or sequence.
- Use ordered lists when order or steps matter.

Fenerbahce University

Lists: Key ideas

- Use unordered lists for items without ranking or sequence.
- Use ordered lists when order or steps matter.
- Use description lists for terms with explanations or values.
- Lists can be nested to express hierarchy.

Lists: Practical notes

- A list should group related items, not random page fragments.
- Each list item belongs inside an <li> element.
- Description lists use <dl>, <dt>, and <dd> instead of <li>.
- Nested lists are useful for menus, plans, and categories.

Unordered shopping list

HTML Unit 05 • Example

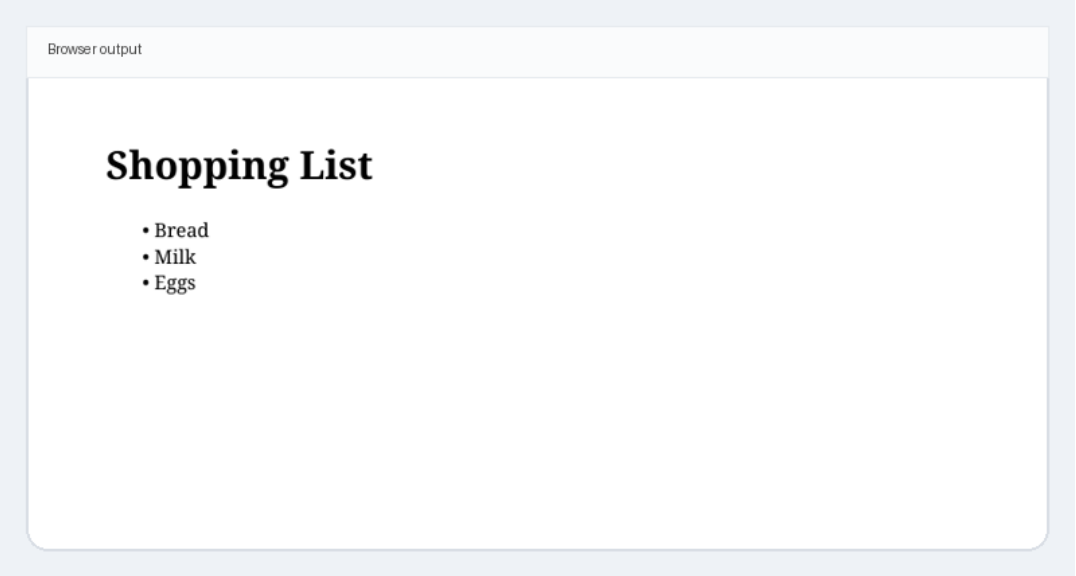
What to notice: Bullet lists work well when sequence does not matter. • Each item becomes a separate, scannable line.

Code

HTML

```
<h1>Shopping List</h1>
<ul>
  <li>Bread</li>
  <li>Milk</li>
  <li>Eggs</li>
</ul>
```

Rendered output



Ordered setup steps

HTML Unit 05 • Example

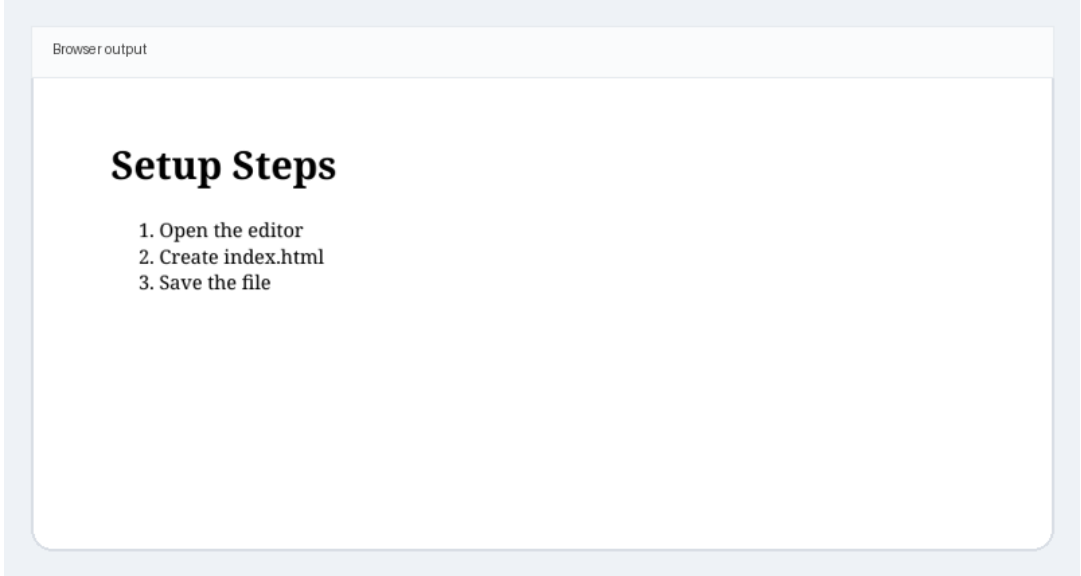
What to notice: Ordered lists communicate sequence and progression. • Changing the item order changes the meaning.

Code

HTML

```html
<h1>Setup Steps</h1>
<ol>
  <li>Open the editor</li>
  <li>Create index.html</li>
  <li>Save the file</li>
</ol>
```

Rendered output



Nested list structure

HTML Unit 05 • Example

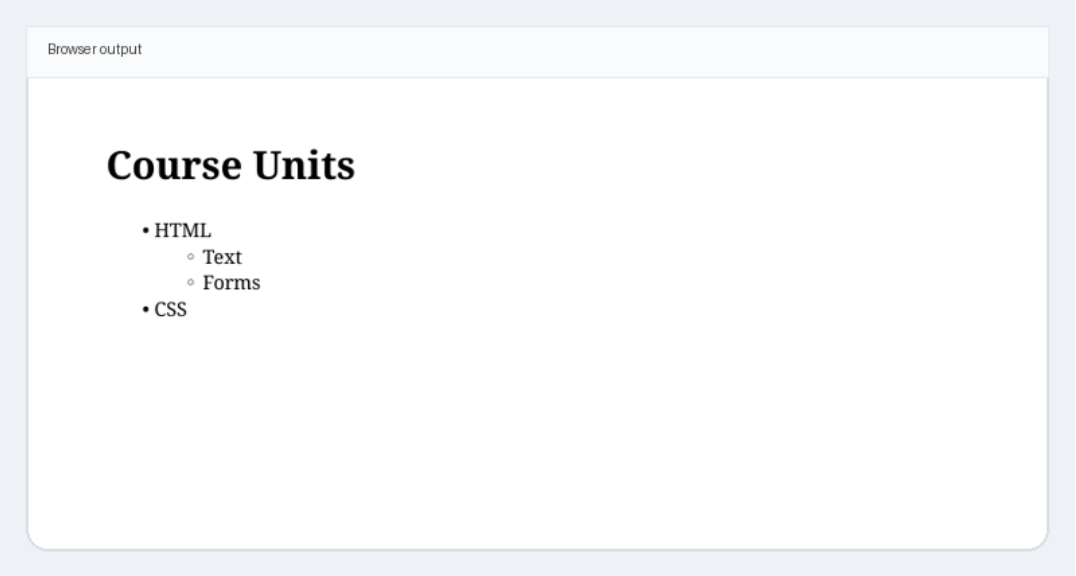
What to notice: A nested list shows categories and subcategories. • Indentation in the source matches the content hierarchy.

Code

HTML

```html
<h1>Course Units</h1>
<ul>
  <li>HTML
    <ul>
      <li>Text</li>
      <li>Forms</li>
    </ul>
  </li>
  <li>CSS</li>
</ul>
```

Rendered output



Description list

HTML Unit 05 • Example

What to notice: Description lists pair terms with explanations or values. • They are useful for glossaries, FAQs, and metadata.

Code

HTML

```html
<h1>Web Terms</h1>
<dl>
  <dt>HTML</dt>
  <dd>Defines structure and meaning.</dd>
  <dt>CSS</dt>
  <dd>Controls presentation and layout.</dd>
</dl>
```

Rendered output

Browser output

Web Terms

HTML
: Defines structure and meaning.

CSS
: Controls presentation and layout.

Ordered list with a custom start

HTML Unit 05 • Example

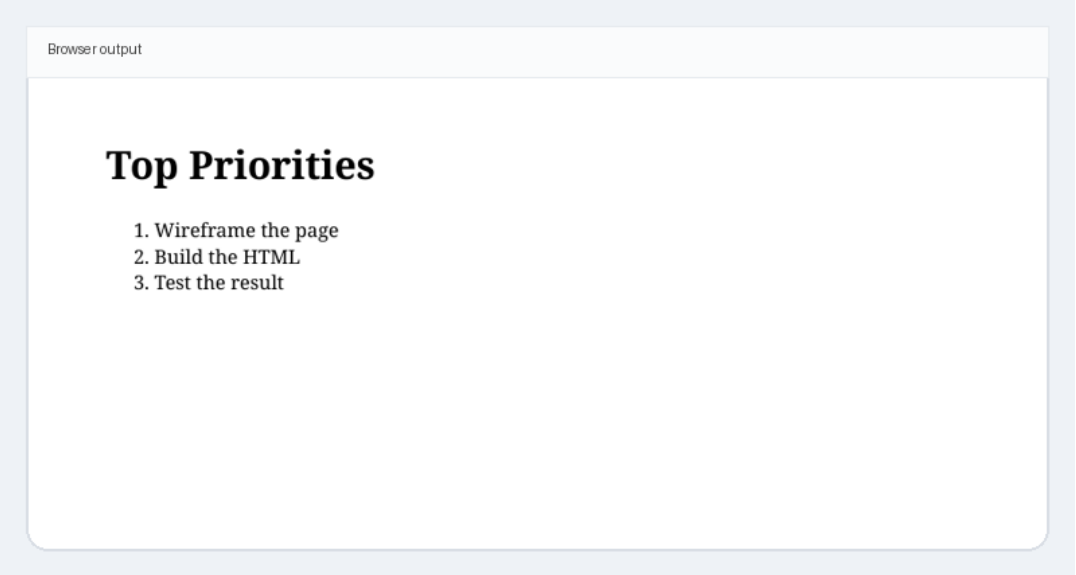
What to notice: The start attribute changes the visible numbering. • This is helpful when a list continues from an earlier section.

Code

HTML

```
<h1>Top Priorities</h1>
<ol start="3">
  <li>Wireframe the page</li>
  <li>Build the HTML</li>
  <li>Test the result</li>
</ol>
```

Rendered output



Lists: Common mistakes

- Do not fake a list by typing hyphens in plain paragraphs.
- Do not place text directly under <ul> or <ol> without <li>.
- Do not use ordered lists when sequence is irrelevant.
- Nested lists should stay simple enough to scan quickly.

Lists: Recap

- Lists make repeated content easier to read and compare.
- Use the list type that matches the meaning of the content.
- Nested lists express structure; description lists express relationships.
- Lists are one of the most common HTML building blocks.

Web Programming

HTML Unit 06

Links and Navigation

- The <a> element creates links to pages, sections, files, or actions.
- Links can be absolute, relative, or fragment-based.

Fenerbahce University

Links and Navigation: Key ideas

- The <a> element creates links to pages, sections, files, or actions.
- Links can be absolute, relative, or fragment-based.
- Good link text tells the user where the link goes.
- Navigation menus are often built from lists of links.

Links and Navigation: Practical notes

- Use href to define the destination.
- Use target="_blank" carefully and usually only for external destinations.
- Use #id links to jump to sections on the same page.
- mailto: and tel: can trigger useful actions on supported devices.

Simple navigation links

HTML Unit 06 • Example

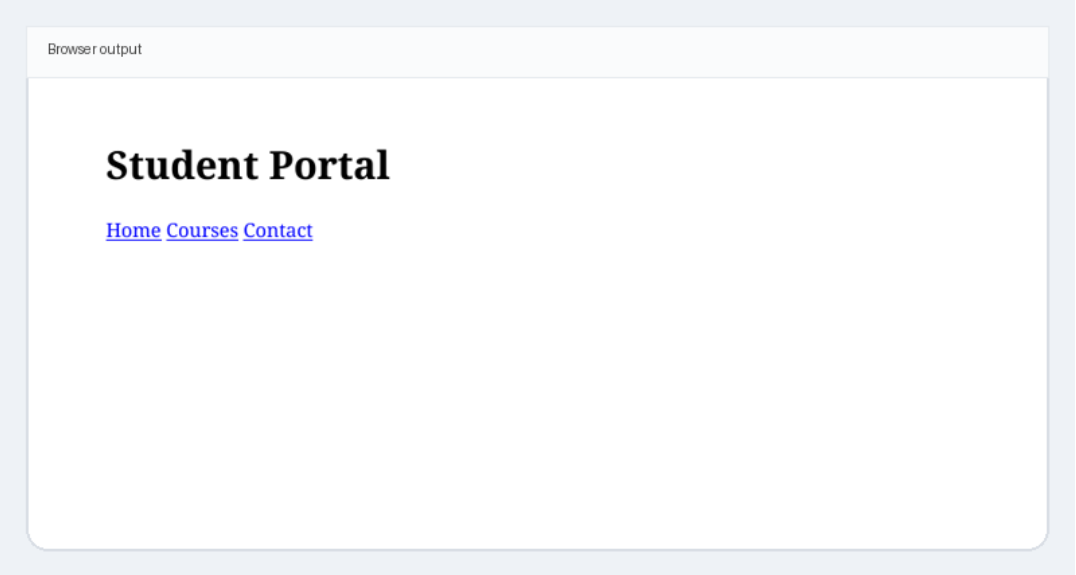
What to notice: A navigation area often starts as a row of plain links. • Relative paths point to files inside the same project.

Code

HTML

```html
<h1>Student Portal</h1>
<nav>
  <a href="index.html">Home</a>
  <a href="courses.html">Courses</a>
  <a href="contact.html">Contact</a>
</nav>
```

Rendered output



External link

HTML Unit 06 • Example

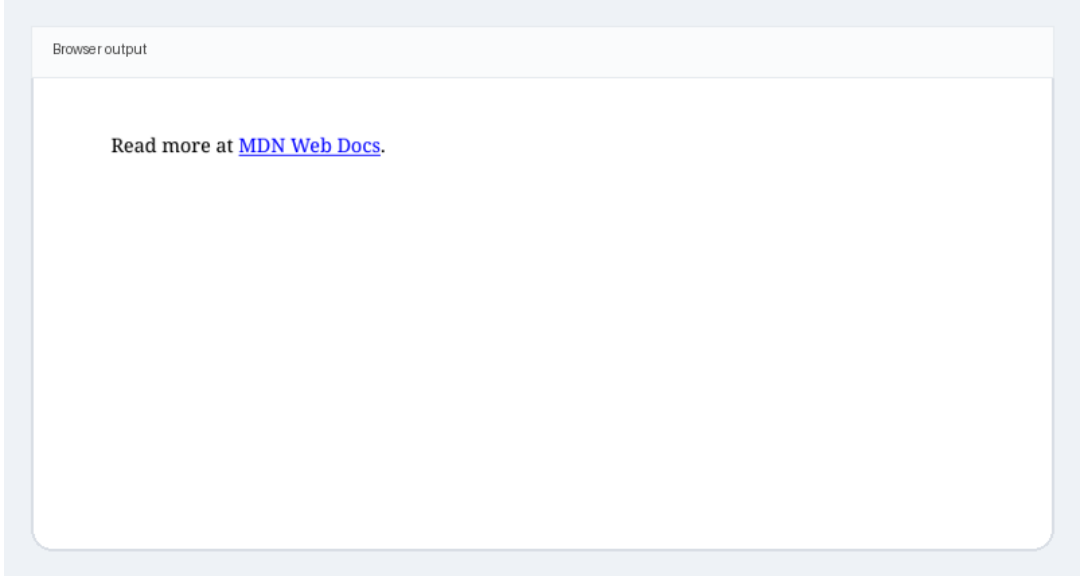
What to notice: Absolute URLs include the full web address. • External links usually connect to content outside your project.

Code

HTML

```
 1  <p>
 2    Read more at
 3    <a href="https://developer.mozilla.org/">MDN Web
Docs</a>.
 4  </p>
```

Rendered output



Open a new tab

HTML Unit 06 • Example

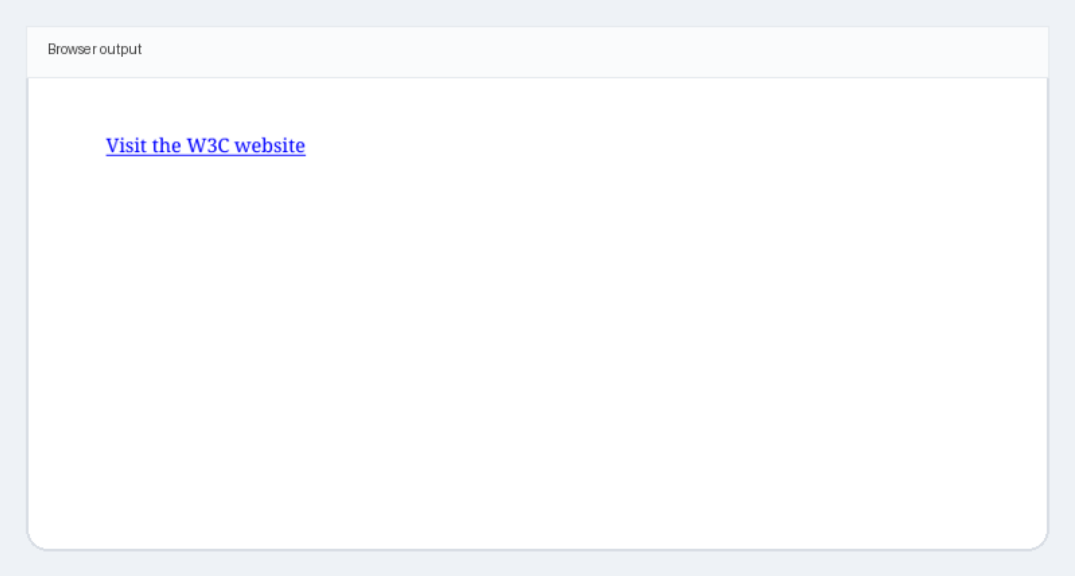
What to notice: New-tab behavior is common for external references. • rel="noopener" is a safe companion for target="_blank".

Code

HTML

```
 1  <p>
 2    <a href="https://www.w3.org/" target="_blank"
rel="noopener">
 3      Visit the W3C website
 4    </a>
 5  </p>
```

Rendered output



Same-page jump link

HTML Unit 06 • Example

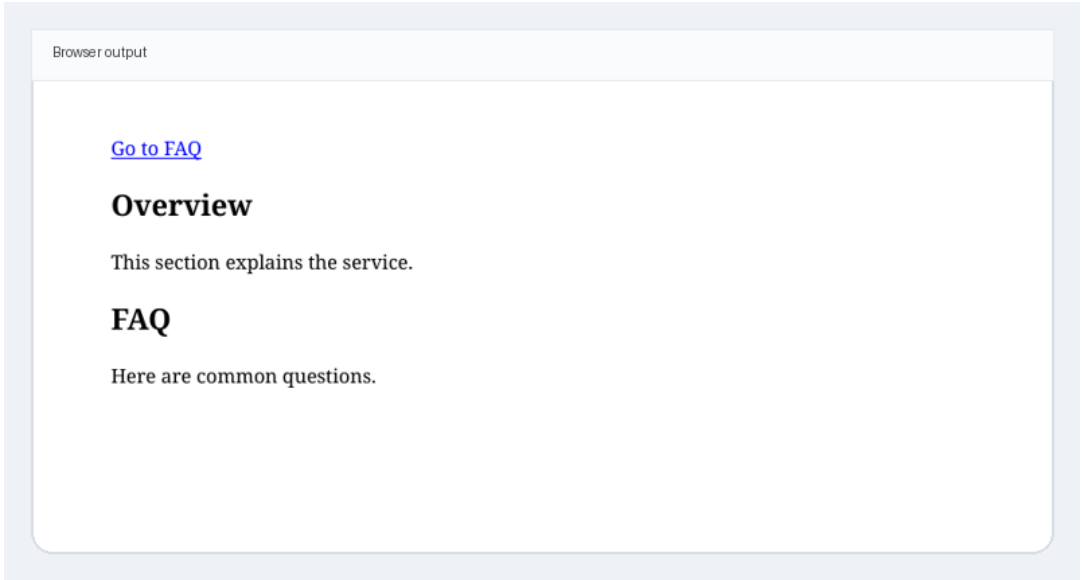
What to notice: Fragment links jump to an element with a matching id. • They are useful for long pages with internal navigation.

Code

HTML

```html
<p><a href="#faq">Go to FAQ</a></p>
<h2>Overview</h2>
<p>This section explains the service.</p>
<h2 id="faq">FAQ</h2>
<p>Here are common questions.</p>
```

Rendered output



Email and phone links

HTML Unit 06 • Example

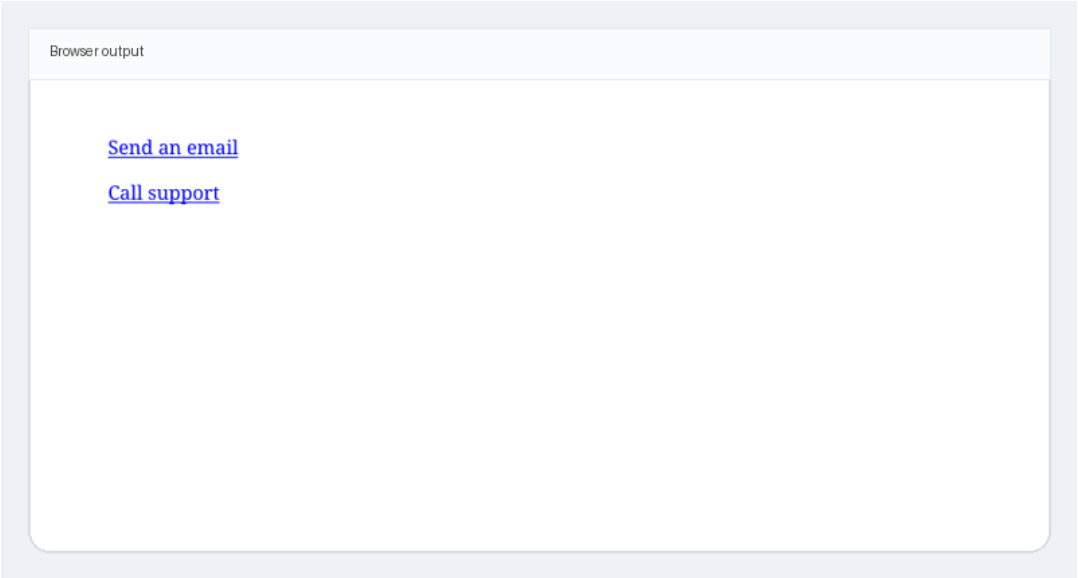
What to notice: Some links trigger actions instead of opening documents. • These formats are especially useful on mobile devices.

Code

HTML

```
 1  <p><a href="mailto:info@example.com">Send an
email</a></p>
 2  <p><a href="tel:+905551112233">Call support</a></p>
```

Rendered output



Links and Navigation: Common mistakes

- Do not use vague link text such as “click here” without context.
- Do not add links that go nowhere or use # as a fake destination.
- Do not open every link in a new tab by default.
- Keep navigation labels short, clear, and consistent.

Links and Navigation: Recap

- Links connect pages, sections, and actions.
- Relative paths support internal navigation; absolute URLs support external links.
- Fragment links and good link text improve usability.
- Navigation is one of the earliest structures users notice.

Web Programming

HTML Unit 07

Images and Figure Content

- Use <img> to display images inside a page.
- The src attribute points to the file and alt describes the image.

Fenerbahce University

Images and Figure Content: Key ideas

- Use <img> to display images inside a page.
- The src attribute points to the file and alt describes the image.
- Width and height help browsers reserve layout space.
- <figure> and <figcaption> group media with a caption.

Images and Figure Content: Practical notes

- Alt text should communicate the purpose of the image, not every pixel.
- Decorative images may use empty alt text when they add no information.
- Figures are useful for screenshots, charts, product photos, and diagrams.
- Images can also be wrapped in links.

Basic image element

HTML Unit 07 • Example

What to notice: The image file path and the alt text serve different purposes. • The width attribute gives the browser an initial size hint.

Code

HTML

```
 1  <h1>Campus Map</h1>
 2  <img src="map.png" alt="Map of the campus with building labels"
width="220">
```

Rendered output

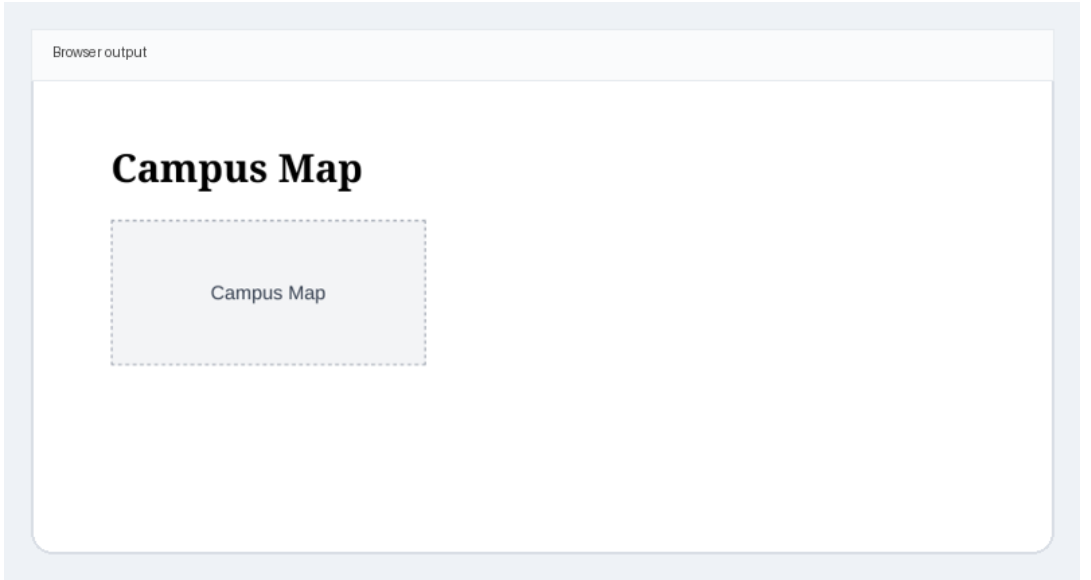


Figure with caption

HTML Unit 07 • Example

What to notice: Figure and figcaption keep media and its label together. • This pattern is useful for reports and tutorials.

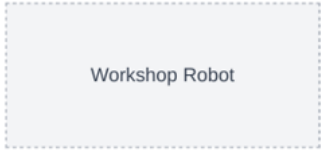
Code

HTML

```
 1  <figure>
 2    <img src="robot.jpg" alt="Student-built robot on a table"
width="220">
 3    <figcaption>Prototype robot from the workshop.</figcaption>
 4  </figure>
```

Rendered output

Browser output



Prototype robot from the workshop.

Linked image

HTML Unit 07 • Example

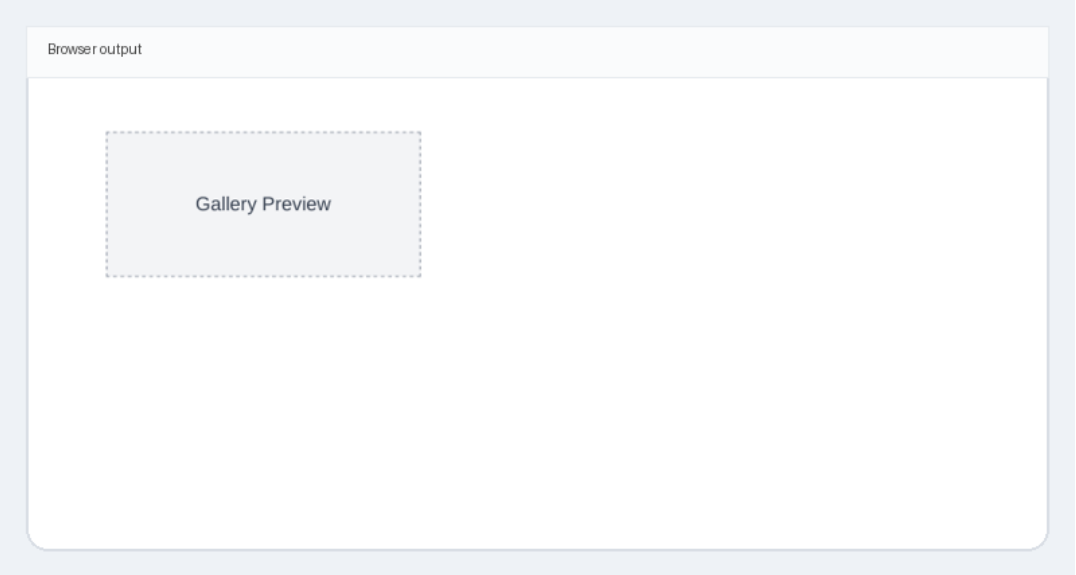
What to notice: An image can act as a link when wrapped in an anchor element. • The alt text should still describe the linked image meaningfully.

Code

HTML

```
 1  <a href="gallery.html">
 2    <img src="thumbnail.jpg" alt="Preview of the gallery"
width="180">
 3  </a>
```

Rendered output



Picture with multiple sources

HTML Unit 07 • Example

What to notice: <picture> lets the browser choose among sources. • It is useful for art direction and responsive image choices.

Code

HTML

```
 1  <picture>
 2    <source srcset="banner-large.jpg" media="(min-width:
800px)">
 3    <img src="banner-small.jpg" alt="Open day event
banner" width="260">
 4  </picture>
```

Rendered output

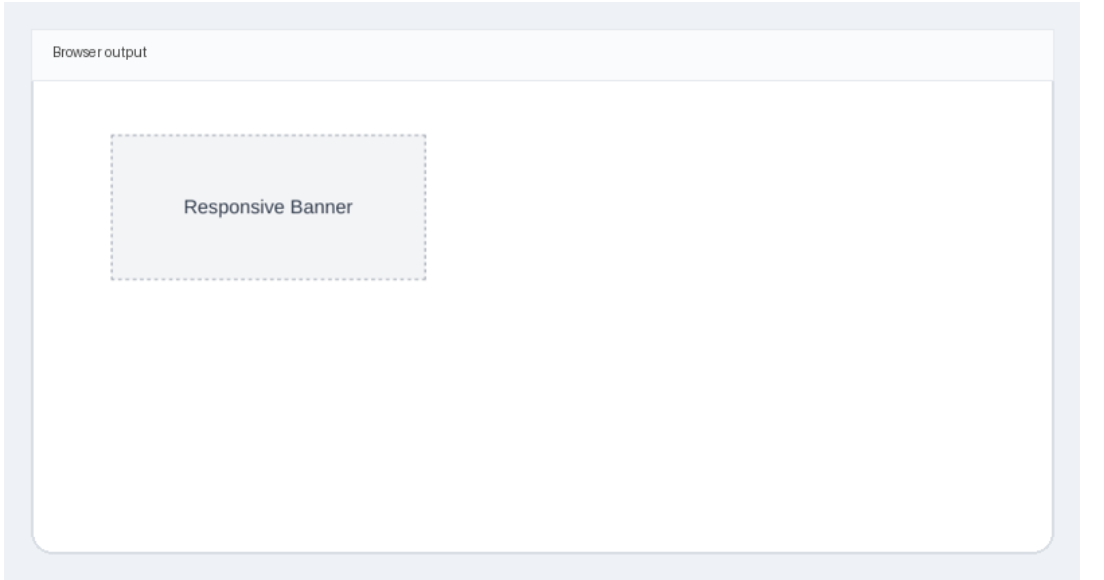


Image followed by text

HTML Unit 07 • Example

What to notice: Images and text often work together inside the same content section. • The browser stacks them naturally in the source order.

Code

HTML

```
 1  <h1>Photography Club</h1>
 2  <img src="club.jpg" alt="Students with cameras during a city
walk" width="220">
 3  <p>Meet every Wednesday at 16:00.</p>
```

Rendered output

Images and Figure Content: Common mistakes

- Do not skip alt text on informative images.
- Do not write alt text that starts with “image of” unless necessary.
- Do not use images instead of real text for ordinary headings.
- Keep file names and folders organized so image paths remain clear.

Images and Figure Content: Recap

- Images need both a source and a useful text alternative.
- Figures help connect media with captions.
- Images can link, illustrate, or support explanations.
- HTML gives the meaning; CSS will later control appearance.

Web Programming

HTML Unit 08

Containers, Attributes, IDs, Classes, and Data

- Global attributes can appear on many elements.
- Useful attributes include id, class, title, hidden, and data-*.

Fenerbahce University

Containers, Attributes, IDs, Classes, and Data: Key ideas

- Global attributes can appear on many elements.
- Useful attributes include id, class, title, hidden, and data-*.
- is a generic block container and is a generic inline container.
- Even before CSS, ids and classes help organize content and behavior.

Containers, Attributes, IDs, Classes, and Data: Practical notes

- Use id for a unique element and class for reusable grouping.
- Data attributes store custom information for scripts or UI logic.
- The title attribute can provide extra advisory text on hover in many browsers.
- Choose semantic elements first and generic containers only when needed.

Using a div as a generic section

HTML Unit 08 • Example

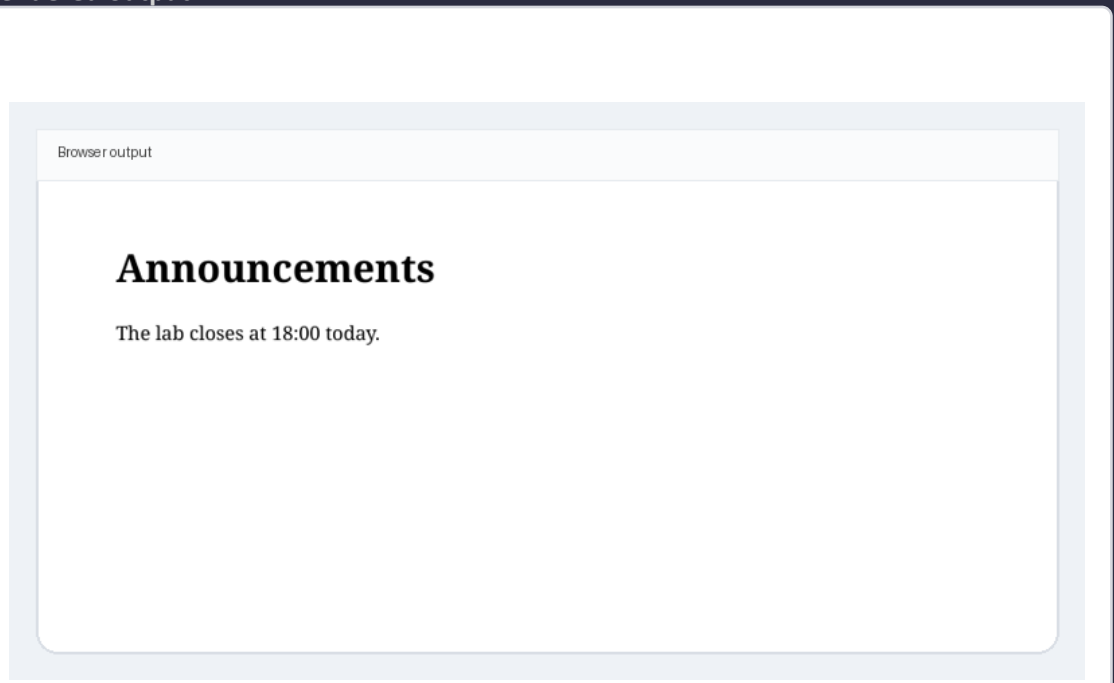
What to notice: creates a block container when no more specific element fits. • The meaning still comes from the content inside it.

Code

HTML

```
<div>
  <h1>Announcements</h1>
  <p>The lab closes at 18:00 today.</p>
</div>
```

Rendered output



Using a span inside text

HTML Unit 08 • Example

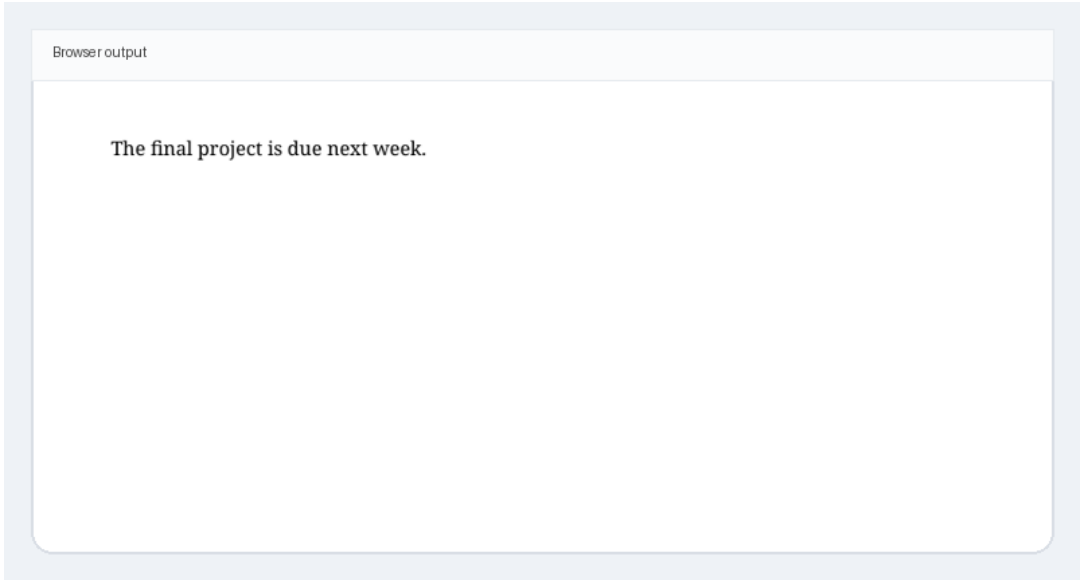
What to notice: marks an inline part of a sentence. • It becomes especially useful later with CSS and JavaScript.

Code

HTML

```
 1  <p>The <span>final project</span> is due next
week.</p>
```

Rendered output



Unique id for fragment links

HTML Unit 08 • Example

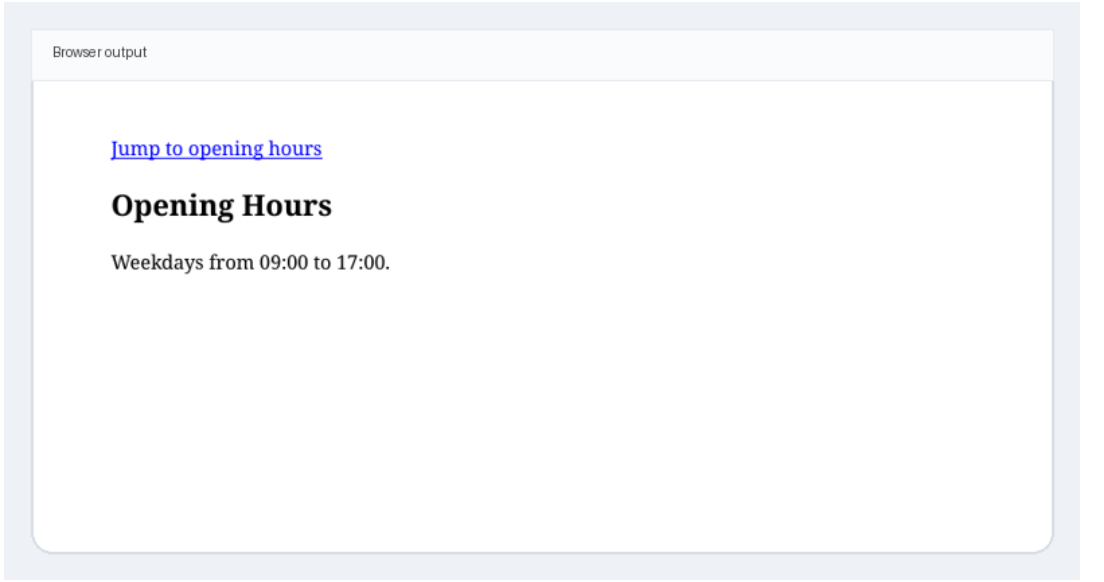
What to notice: The id attribute gives a unique target for links and scripts. • One id value should not be repeated elsewhere on the page.

Code

HTML

```
1  <p><a href="#hours">Jump to opening hours</a></p>
2  <h2 id="hours">Opening Hours</h2>
3  <p>Weekdays from 09:00 to 17:00.</p>
```

Rendered output



Classes on repeated items

HTML Unit 08 • Example

What to notice: Classes group similar items under one shared label. • The class attribute can contain more than one class name.

Code

HTML

```
<ul>
  <li class="featured">HTML Basics</li>
  <li class="featured">Forms Workshop</li>
  <li>Final Review</li>
</ul>
```

Rendered output

Browser output

- HTML Basics
- Forms Workshop
- Final Review

Custom data attributes

HTML Unit 08 • Example

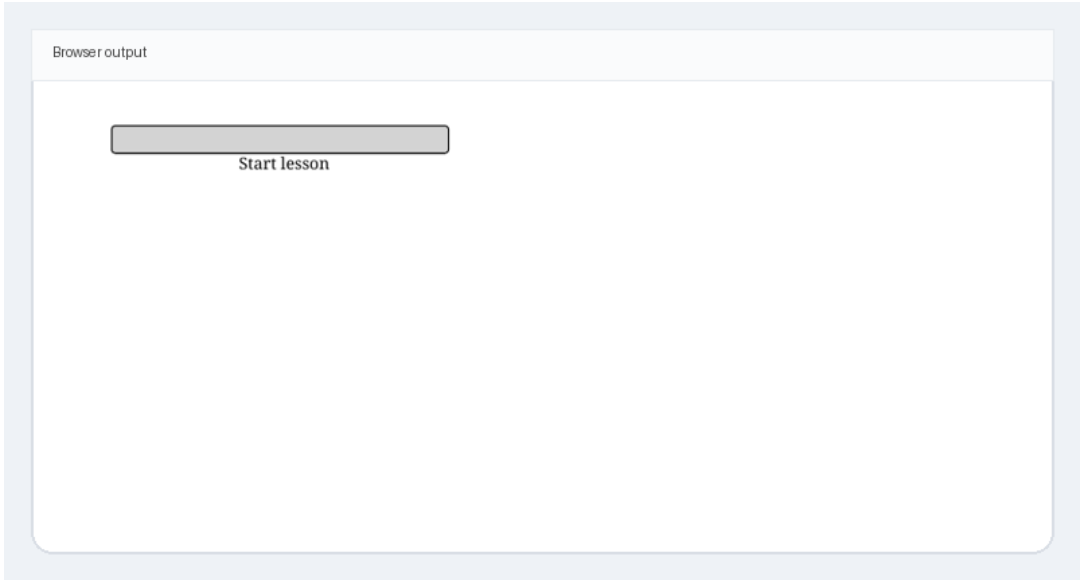
What to notice: data-* attributes store custom values in valid HTML. • They are useful for scripts without inventing non-standard attributes.

Code

HTML

```
<button data-topic="html" data-level="beginner">
  Start lesson
</button>
```

Rendered output



Containers, Attributes, IDs, Classes, and Data: Common mistakes

- Do not use div everywhere when a semantic element fits better.
- Do not duplicate the same id on multiple elements.
- Do not put important visible information only in the title attribute.
- Use data attributes for custom data, not for styling text content.

Containers, Attributes, IDs, Classes, and Data: Recap

- Attributes enrich elements with identity and extra meaning.
- Div and span are generic tools, not automatic first choices.
- Ids are unique; classes are reusable; data-* is custom but valid.
- Clean attribute usage makes later CSS and JavaScript easier.

Web Programming

HTML Unit 09

Semantic Page Layout

- Semantic layout elements describe the role of each page region.
- Common layout elements include header, nav, main, section, article, aside, and footer.

Fenerbahce University

Semantic Page Layout: Key ideas

- Semantic layout elements describe the role of each page region.
- Common layout elements include header, nav, main, section, article, aside, and footer.
- These elements help readers, developers, and assistive tools understand structure.
- Semantic tags are stronger than a page built entirely from generic divs.

Semantic Page Layout: Practical notes

- Use <main> once for the primary content of the page.
- Use <article> for self-contained items such as posts or cards.
- Use <section> for grouped content with a shared heading.
- Use <aside> for related but secondary information.

Simple semantic page shell

HTML Unit 09 • Example

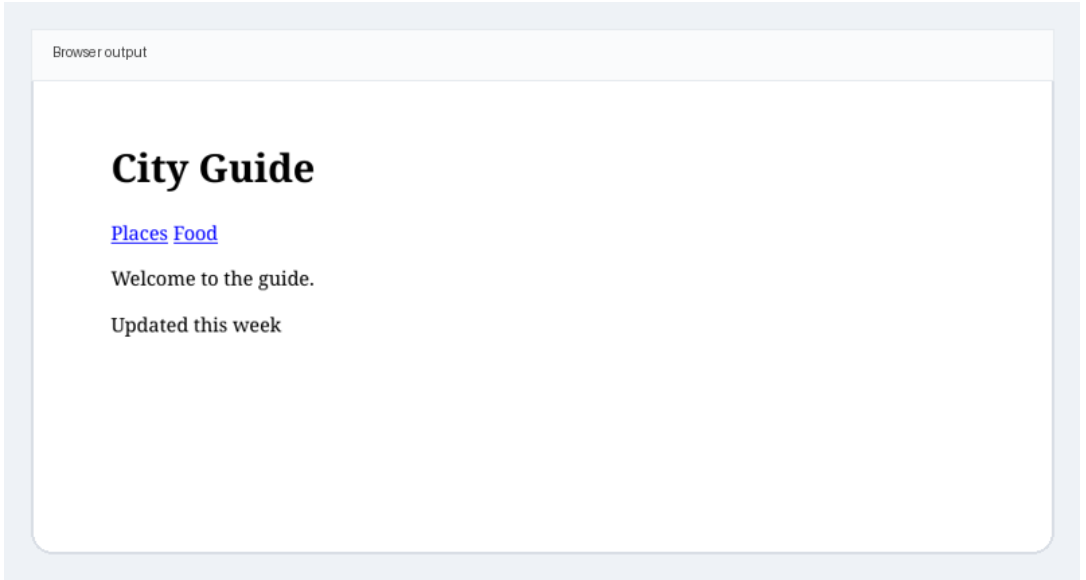
What to notice: This structure already communicates the main regions of the page. • The default flow follows the source order from top to bottom.

Code

HTML

```
 1  <header><h1>City Guide</h1></header>
 2  <nav><a href="#">Places</a> <a
href="#">Food</a></nav>
 3  <main><p>Welcome to the guide.</p></main>
 4  <footer><p>Updated this week</p></footer>
```

Rendered output



Article inside a section

HTML Unit 09 • Example

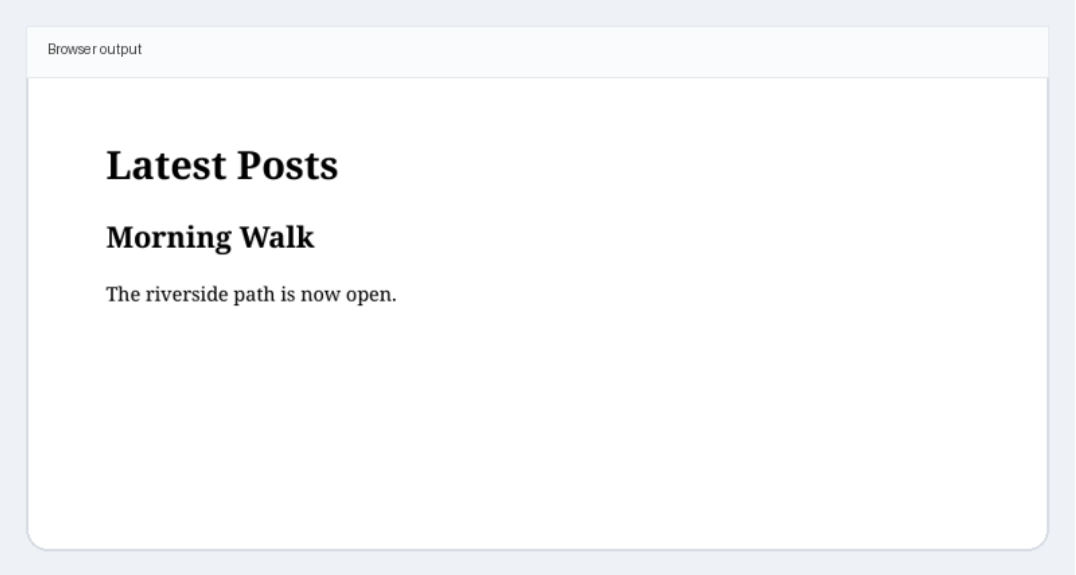
What to notice: Sections group related content; articles represent individual entries. • Nested semantics become useful on blogs and news pages.

Code

HTML

```html
<main>
  <section>
    <h1>Latest Posts</h1>
    <article>
      <h2>Morning Walk</h2>
      <p>The riverside path is now open.</p>
    </article>
  </section>
</main>
```

Rendered output



Using aside for a note

HTML Unit 09 • Example

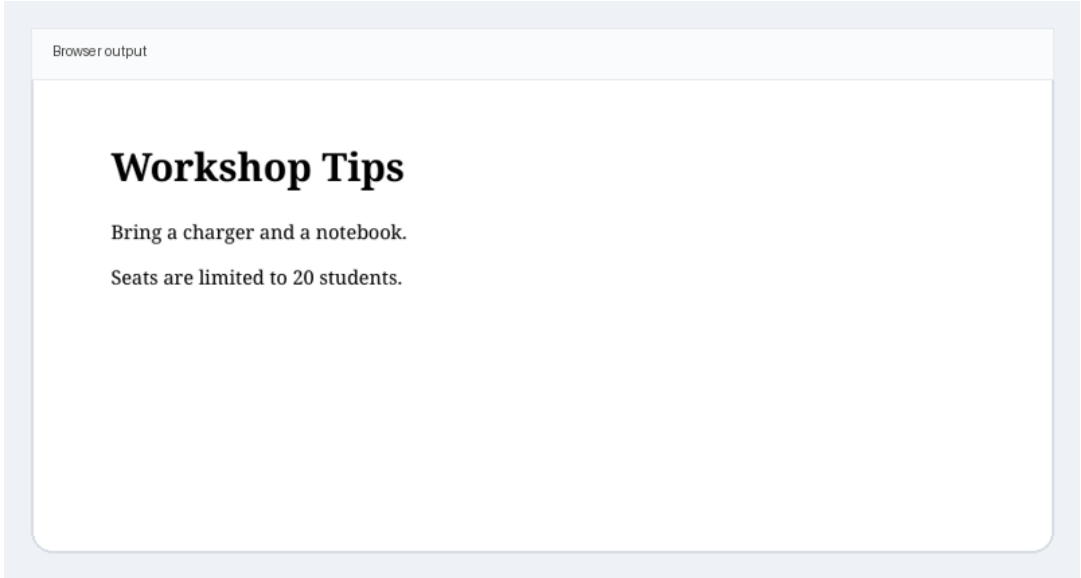
What to notice: Aside content is related, but not the main message. • It often holds tips, notices, ads, or supporting material.

Code

HTML

```
<article>
  <h1>Workshop Tips</h1>
  <p>Bring a charger and a notebook.</p>
  <aside><p>Seats are limited to 20 students.</p></aside>
</article>
```

Rendered output



Multiple articles on one page

HTML Unit 09 • Example

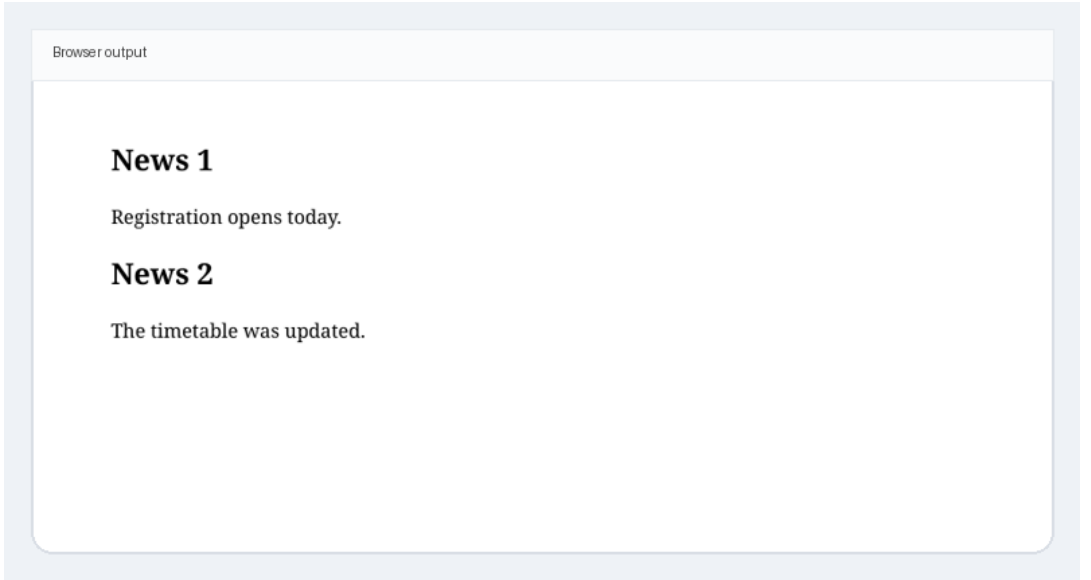
What to notice: Each article is a self-contained item with its own heading. • This is common in feeds, lists, and archive pages.

Code

HTML

```
 1  <main>
 2    <article><h2>News 1</h2><p>Registration opens
today.</p></article>
 3    <article><h2>News 2</h2><p>The timetable was
updated.</p></article>
 4  </main>
```

Rendered output



Nested sections with headings

HTML Unit 09 • Example

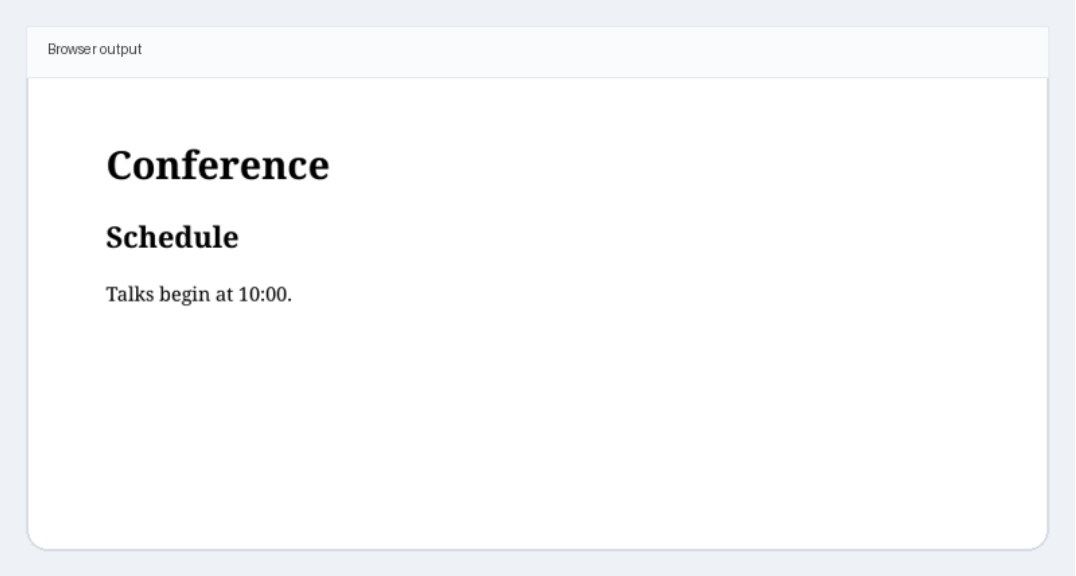
What to notice: Sections often make sense only when they have headings. • Heading levels should reflect the nesting structure.

Code

HTML

```
<main>
  <section>
    <h1>Conference</h1>
    <section>
      <h2>Schedule</h2>
      <p>Talks begin at 10:00.</p>
    </section>
  </section>
</main>
```

Rendered output



Semantic Page Layout: Common mistakes

- Do not use many section elements without headings.
- Do not wrap everything in article when the content is not self-contained.
- Do not create multiple main elements on one page.
- Keep the structure meaningful instead of merely decorative.

Semantic Page Layout: Recap

- Semantic layout elements turn structure into meaning.
- Header, nav, main, article, section, aside, and footer each have a role.
- Good semantics make pages easier to understand and maintain.
- These elements become the backbone of accessible web pages.

Web Programming

HTML Unit 10

Tables

- Tables represent data arranged in rows and columns.
- Core elements include table, caption, tr, th, and td.

Fenerbahce University

Tables: Key ideas

- Tables represent data arranged in rows and columns.
- Core elements include table, caption, tr, th, and td.
- thead, tbody, and tfoot improve structure in larger tables.
- Tables are for data, not for page layout.

Tables: Practical notes

- Header cells should use <th> instead of <td>.
- Captions help users understand the table purpose quickly.
- colspan and rowspan merge cells when the data model requires it.
- Scope attributes improve table accessibility.

Simple data table

HTML Unit 10 • Example

What to notice: This example uses a visible border only to make structure obvious. • The first row uses header cells to label each column.

Code

HTML

```html
<table border="1">
  <tr>
    <th>Course</th>
    <th>Room</th>
  </tr>
  <tr>
    <td>HTML</td>
    <td>B201</td>
  </tr>
</table>
```

Rendered output

Browser output

Course	Room
HTML	B201

Table with caption and body

HTML Unit 10 • Example

What to notice: Caption explains the table before the user reads every cell. • thead and tbody become useful as tables grow.

Code

HTML

```html
<table border="1">
  <caption>Weekly Lab Schedule</caption>
  <thead>
    <tr><th>Day</th><th>Topic</th></tr>
  </thead>
  <tbody>
    <tr><td>Mon</td><td>HTML</td></tr>
    <tr><td>Wed</td><td>Forms</td></tr>
  </tbody>
</table>
```

Rendered output

Browser output

Weekly Lab Schedule

Day	Topic
Mon	HTML
Wed	Forms

Using colspan

HTML Unit 10 • Example

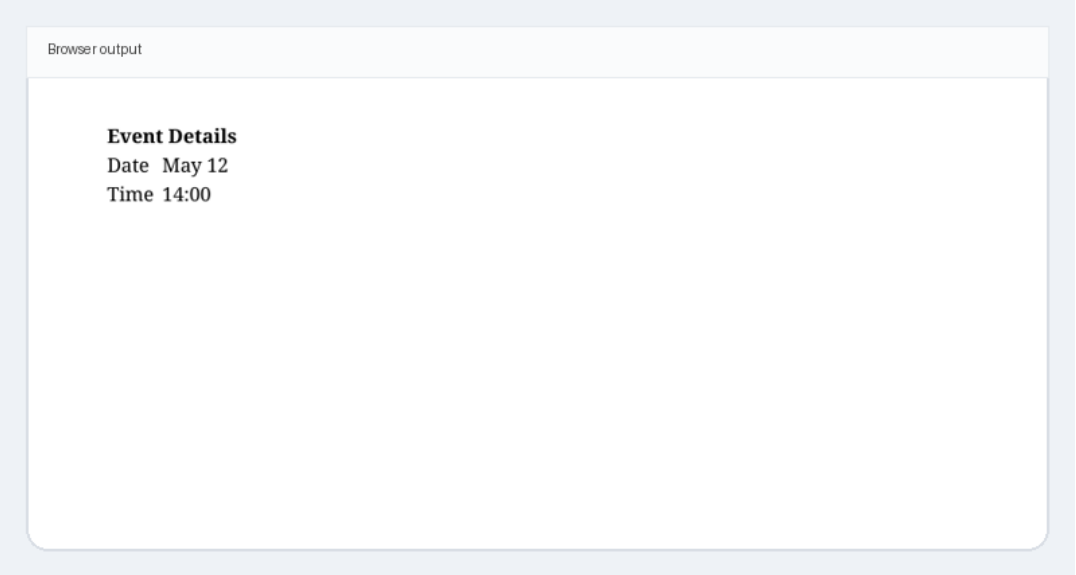
What to notice: A cell can span across multiple columns when needed. • Merged cells should still support a clear reading order.

Code

HTML

```
<table border="1">
  <tr><th colspan="2">Event Details</th></tr>
  <tr><td>Date</td><td>May 12</td></tr>
  <tr><td>Time</td><td>14:00</td></tr>
</table>
```

Rendered output



Using rowspan

HTML Unit 10 • Example

What to notice: rowspan lets one cell cover multiple rows. • Use it only when the data really shares the same heading.

Code

HTML

```
<table border="1">
  <tr><th>Team</th><th>Member</th></tr>
  <tr><td
rowspan="2">Design</td><td>Aylin</td></tr>
  <tr><td>Mert</td></tr>
</table>
```

Rendered output

Browser output

Team	Member
Design	Aylin
	Mert

Table with scope attributes

HTML Unit 10 • Example

What to notice: scope clarifies whether a header labels a row or a column. • This helps assistive technologies interpret the grid.

Code

HTML

```html
<table border="1">
  <tr>
    <th scope="col">Name</th>
    <th scope="col">Score</th>
  </tr>
  <tr>
    <td>Ece</td>
    <td>92</td>
  </tr>
</table>
```

Rendered output

Browser output

Name	Score
Ece	92

Tables: Common mistakes

- Do not use tables to force a page layout.
- Do not leave users guessing what the rows and columns represent.
- Do not overuse merged cells unless the data demands them.
- Keep tables concise and readable on the page.

Tables: Recap

- Tables are for structured data, not for general layout.
- Use captions and headers so users can scan quickly.
- Structural elements like thead and tbody improve clarity.
- Accessibility matters even in small data tables.

Web Programming

HTML Unit 11

Forms Overview

- Forms collect information from users and send it for processing.
- A form usually combines labels, controls, and buttons.

Fenerbahce University

Forms Overview: Key ideas

- Forms collect information from users and send it for processing.
- A form usually combines labels, controls, and buttons.
- The action attribute sets the destination and method sets the request type.
- Field grouping improves clarity when forms get longer.

Forms Overview: Practical notes

- Every meaningful control should have a label.
- The name attribute identifies the submitted value.
- Fieldset and legend organize related questions.
- Even simple forms should be easy to scan and complete.

Basic contact form

HTML Unit 11 • Example

What to notice: The form groups all related inputs into one submission area. • Labels explain what each field expects from the user.

Code

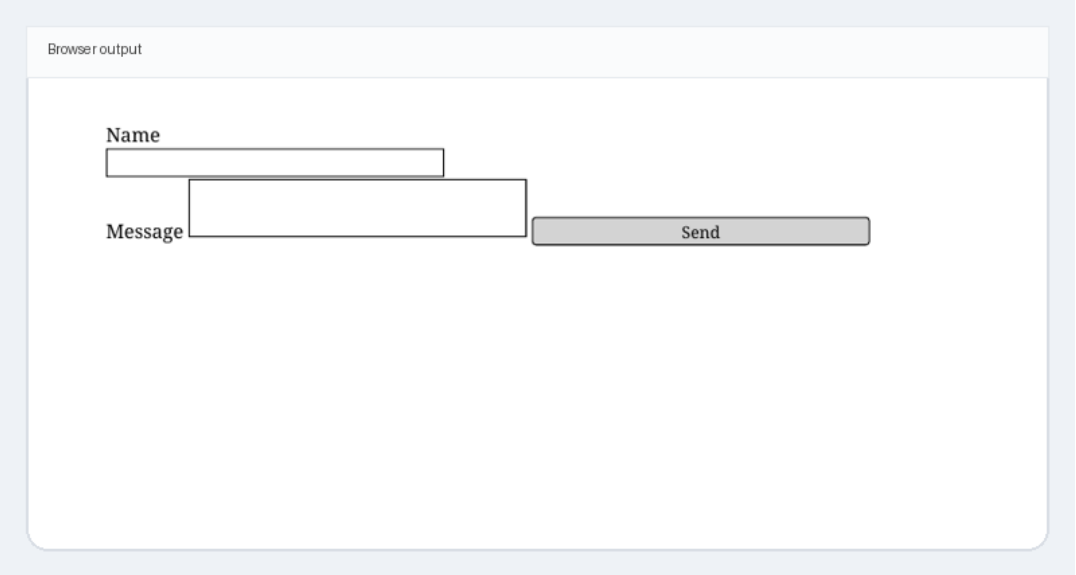
HTML

```
<form>
  <label for="name">Name</label>
  <input id="name" name="name" type="text">

  <label for="message">Message</label>
  <textarea id="message"
name="message"></textarea>

  <button type="submit">Send</button>
</form>
```

Rendered output



Form with action and method

HTML Unit 11 • Example

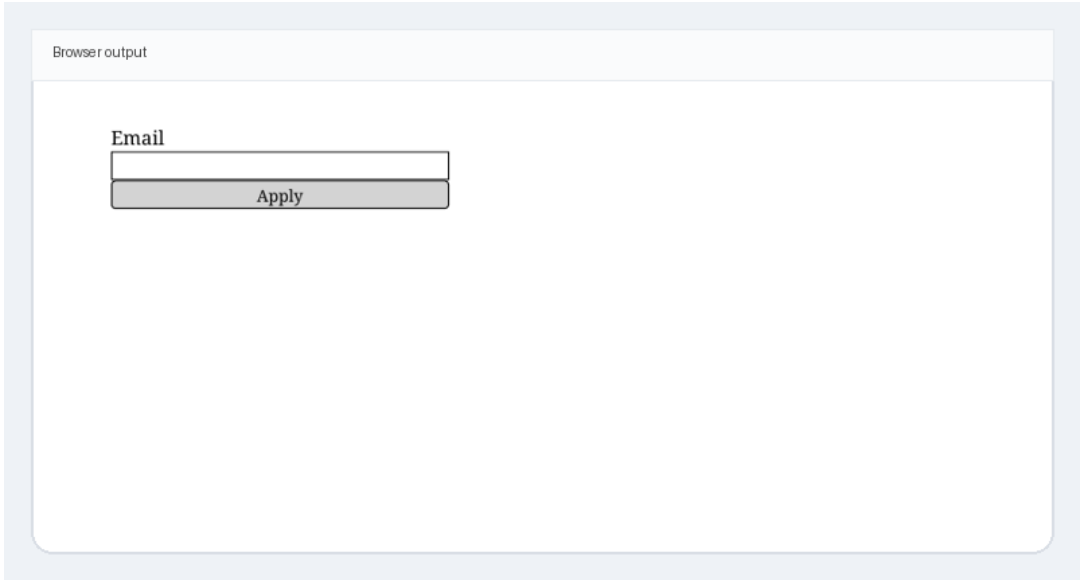
What to notice: action points to the processing endpoint. • method controls how the browser sends the form data.

Code

HTML

```html
<form action="/apply" method="post">
  <label for="email">Email</label>
  <input id="email" name="email" type="email">
  <button type="submit">Apply</button>
</form>
```

Rendered output



Fieldset and legend

HTML Unit 11 • Example

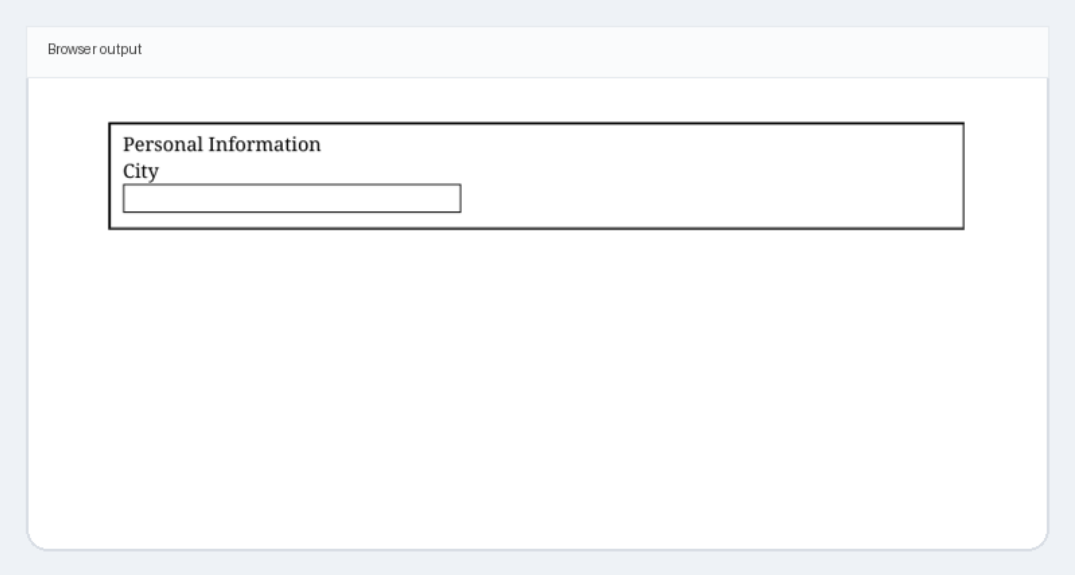
What to notice: fieldset creates a visible group of related controls. • legend names the group for users and assistive tools.

Code

HTML

```html
<form>
  <fieldset>
    <legend>Personal Information</legend>
    <label for="city">City</label>
    <input id="city" name="city" type="text">
  </fieldset>
</form>
```

Rendered output



Textarea for longer content

HTML Unit 11 • Example

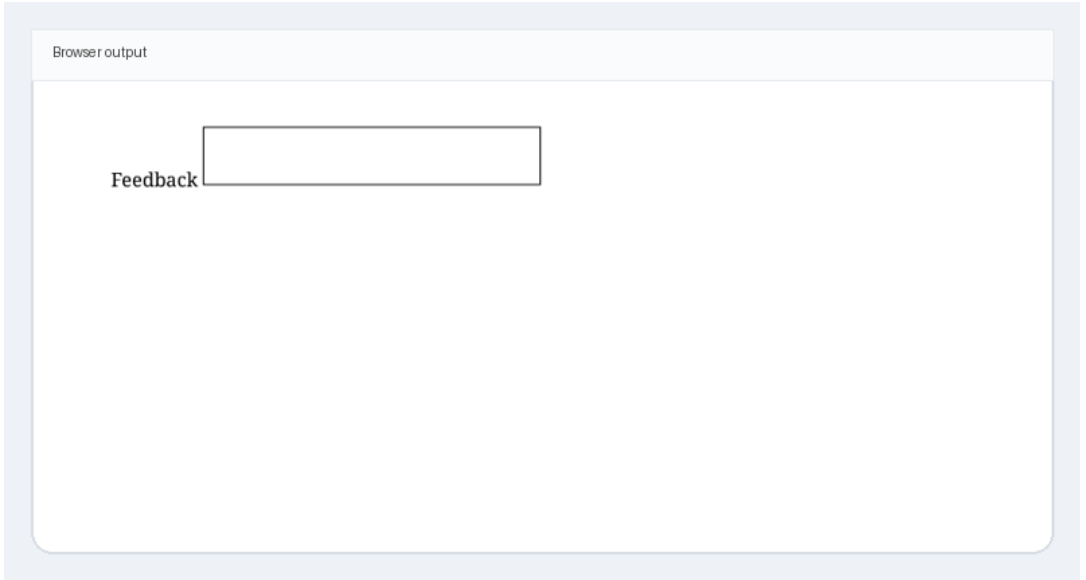
What to notice: Textarea is better than a single-line input for longer messages. • rows and cols give the browser a starting size.

Code

HTML

```
 1  <form>
 2    <label for="feedback">Feedback</label>
 3    <textarea id="feedback" name="feedback" rows="4"
cols="30"></textarea>
 4  </form>
```

Rendered output



Label wrapping the control

HTML Unit 11 • Example

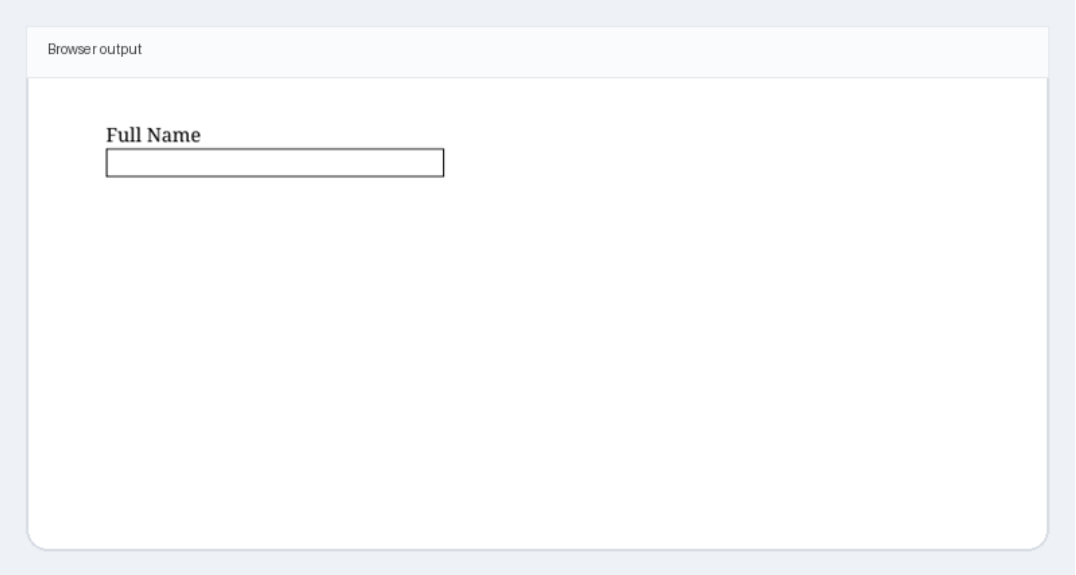
What to notice: A label can wrap a control instead of using for/id. • Both patterns are valid when implemented clearly.

Code

HTML

```
<form>
  <label>
    Full Name
    <input name="full-name" type="text">
  </label>
</form>
```

Rendered output



Forms Overview: Common mistakes

- Do not rely on placeholder text instead of a proper label.
- Do not omit the name attribute on fields that should be submitted.
- Do not mix unrelated questions without grouping.
- Place controls in a logical order that matches user flow.

Forms Overview: Recap

- Forms connect user intent to application logic.
- Labels, names, and buttons are the essential building blocks.
- Grouping related questions improves readability.
- Clear HTML form structure leads to better user experience later.

Web Programming

HTML Unit 12

Text Inputs and Common Attributes

- Different input types match different kinds of text entry.
- Common types include text, email, password, search, tel, url, number, and date.

Fenerbahce University

Text Inputs and Common Attributes: Key ideas

- Different input types match different kinds of text entry.
- Common types include text, email, password, search, tel, url, number, and date.
- Attributes such as placeholder, value, required, readonly, and disabled change behavior.
- Choose the input type that best matches the user task.

Text Inputs and Common Attributes: Practical notes

- Correct types can trigger better keyboards and browser validation.
- Placeholder text is a hint, not a replacement for a label.
- readonly keeps the value visible but not editable.
- disabled prevents editing and submission for that control.

Login fields

HTML Unit 12 • Example

What to notice: Different text-based input types support different user goals. • Password fields mask characters by default.

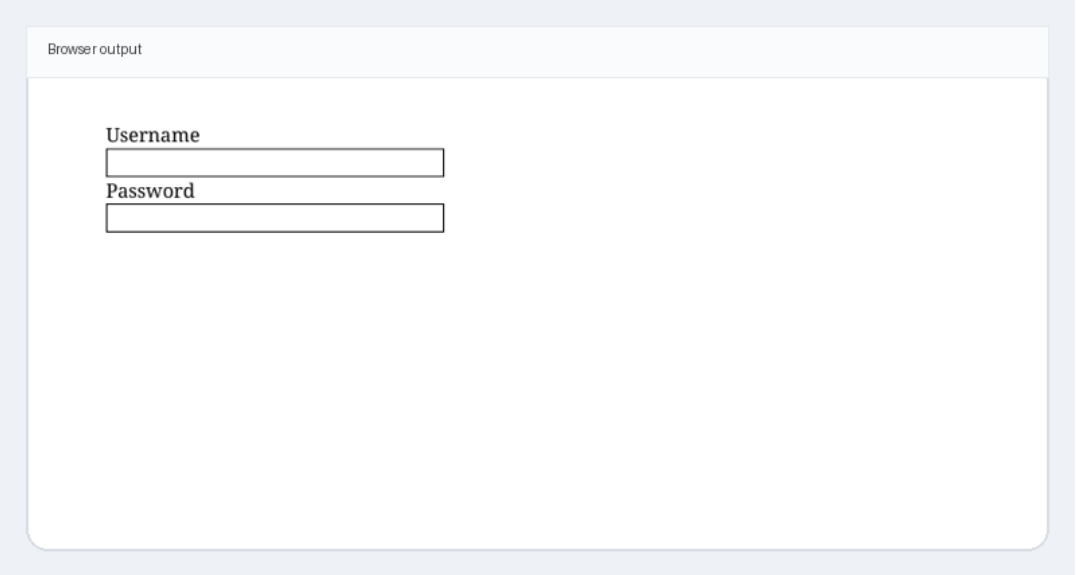
Code

HTML

```
<form>
  <label for="user">Username</label>
  <input id="user" name="user" type="text">

  <label for="pass">Password</label>
  <input id="pass" name="pass" type="password">
</form>
```

Rendered output



Email and search inputs

HTML Unit 12 • Example

What to notice: Email and search fields communicate intent to the browser. • Specialized inputs improve semantics and usability.

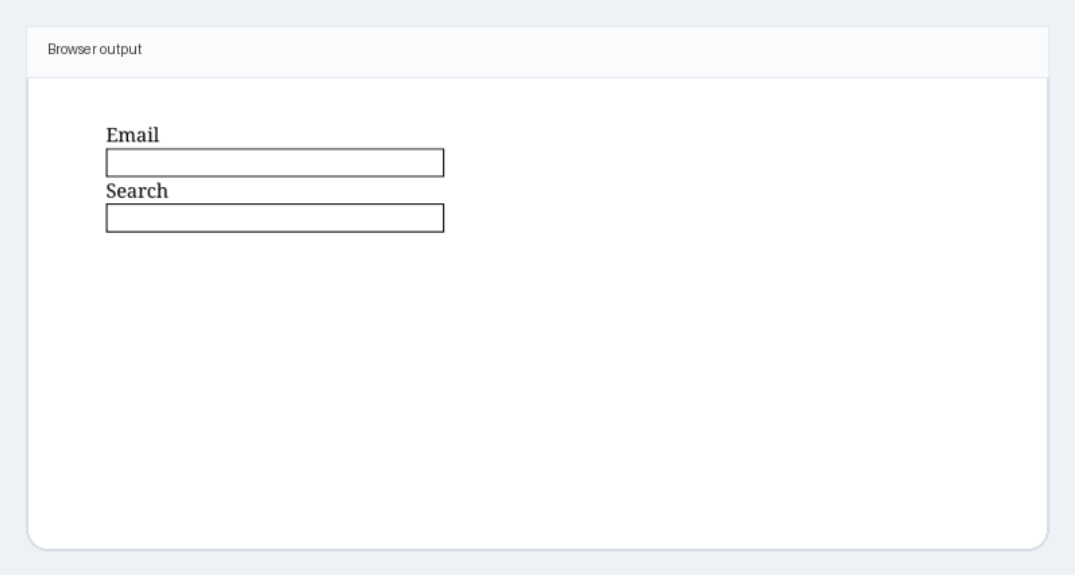
Code

HTML

```html
<form>
  <label for="email">Email</label>
  <input id="email" name="email" type="email">

  <label for="search">Search</label>
  <input id="search" name="search" type="search">
</form>
```

Rendered output



Number and date fields

HTML Unit 12 • Example

What to notice: Use a numeric field only when the value is truly numeric. • Date fields can trigger a date picker in many browsers.

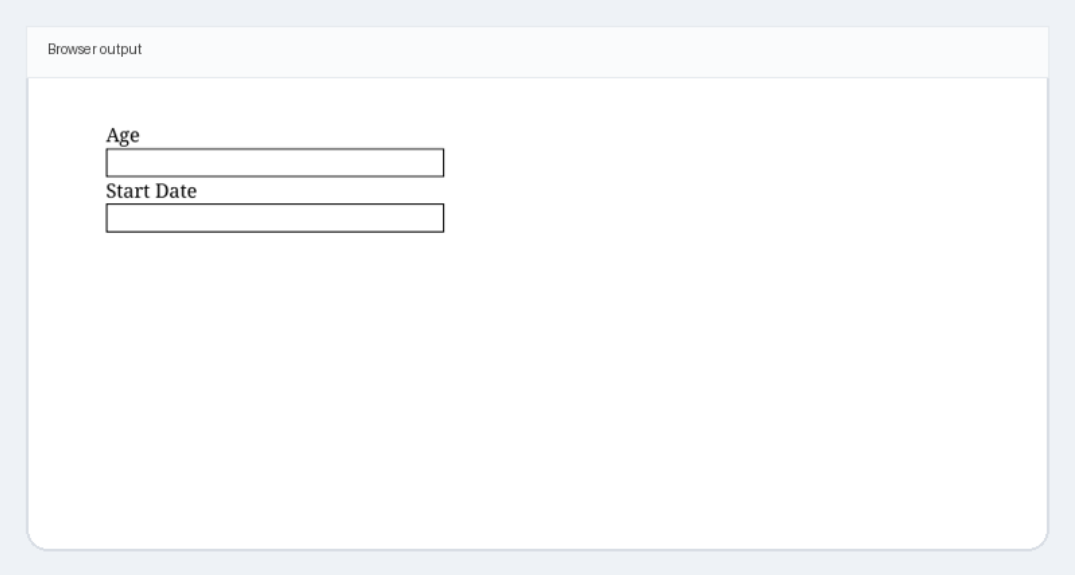
Code

HTML

```
<form>
  <label for="age">Age</label>
  <input id="age" name="age" type="number">

  <label for="date">Start Date</label>
  <input id="date" name="date" type="date">
</form>
```

Rendered output



Readonly and disabled

HTML Unit 12 • Example

What to notice: readonly keeps a value available but uneditable. • disabled removes the field from normal interaction and submission.

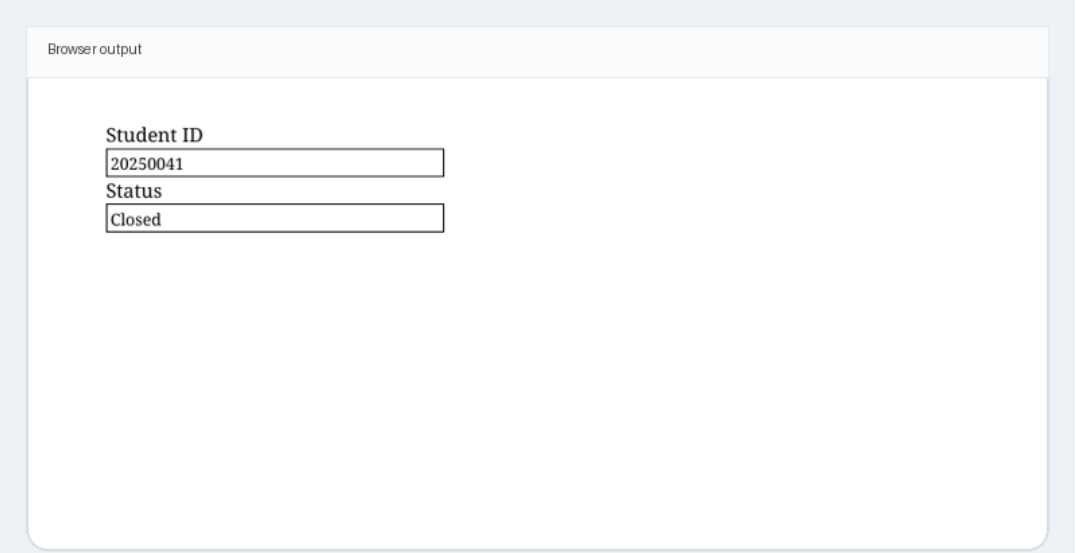
Code

HTML

```html
<form>
  <label for="student">Student ID</label>
  <input id="student" value="20250041" readonly>

  <label for="status">Status</label>
  <input id="status" value="Closed" disabled>
</form>
```

Rendered output



Placeholder and prefilled value

HTML Unit 12 • Example

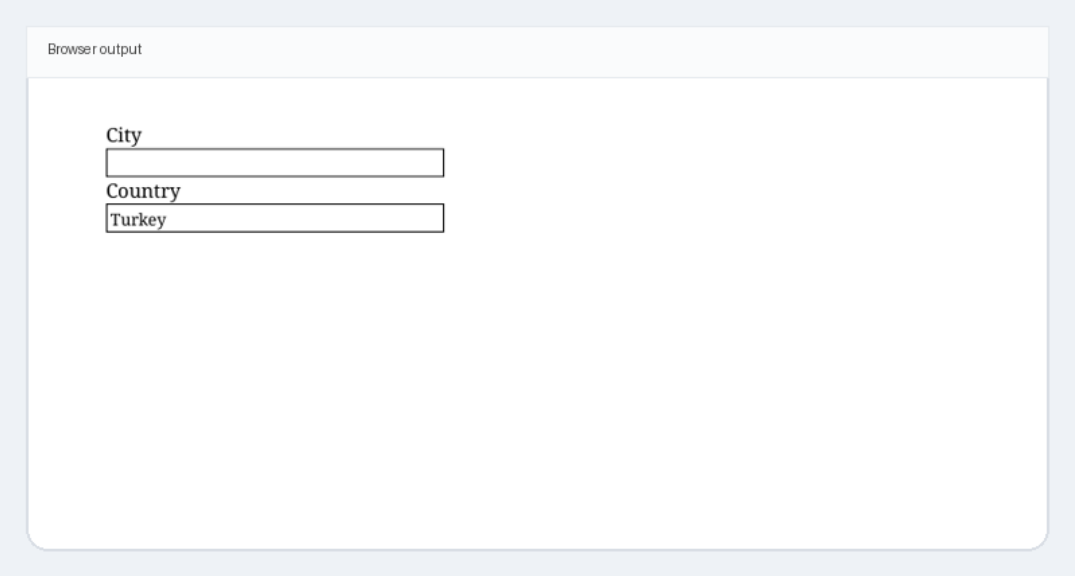
What to notice: placeholder suggests a format or example value. • value pre-fills the field with known data.

Code

HTML

```
 1  <form>
 2    <label for="city">City</label>
 3    <input id="city" name="city" type="text" placeholder="Enter
your city">
 4
 5    <label for="country">Country</label>
 6    <input id="country" name="country" type="text"
value="Turkey">
 7  </form>
```

Rendered output



Text Inputs and Common Attributes: Common mistakes

- Do not use number input for IDs, phone numbers, or postal codes blindly.
- Do not hide important instructions only in placeholder text.
- Do not disable fields when users still need to read or copy the value.
- Choose the narrowest input type that matches the data.

Text Inputs and Common Attributes: Recap

- Input types communicate intent before any styling is applied.
- Attributes fine-tune how fields behave in the browser.
- Text inputs should remain predictable and well labeled.
- Small HTML choices can prevent large UX problems later.

Web Programming

HTML Unit 13

Choice Inputs

- Choice controls help users select from predefined options.
- Common controls include checkbox, radio, select, option, and datalist.

Fenerbahce University

Choice Inputs: Key ideas

- Choice controls help users select from predefined options.
- Common controls include checkbox, radio, select, option, and datalist.
- Checkboxes support multiple selections; radios usually support one choice in a group.
- Labels are especially important for small click targets.

Choice Inputs: Practical notes

- Radio buttons are grouped by sharing the same name.
- A select menu saves space when many options exist.
- Datalist adds suggestions to an input without forcing them.
- Multiple-choice controls should present clear, mutually understandable labels.

Checkbox choices

HTML Unit 13 • Example

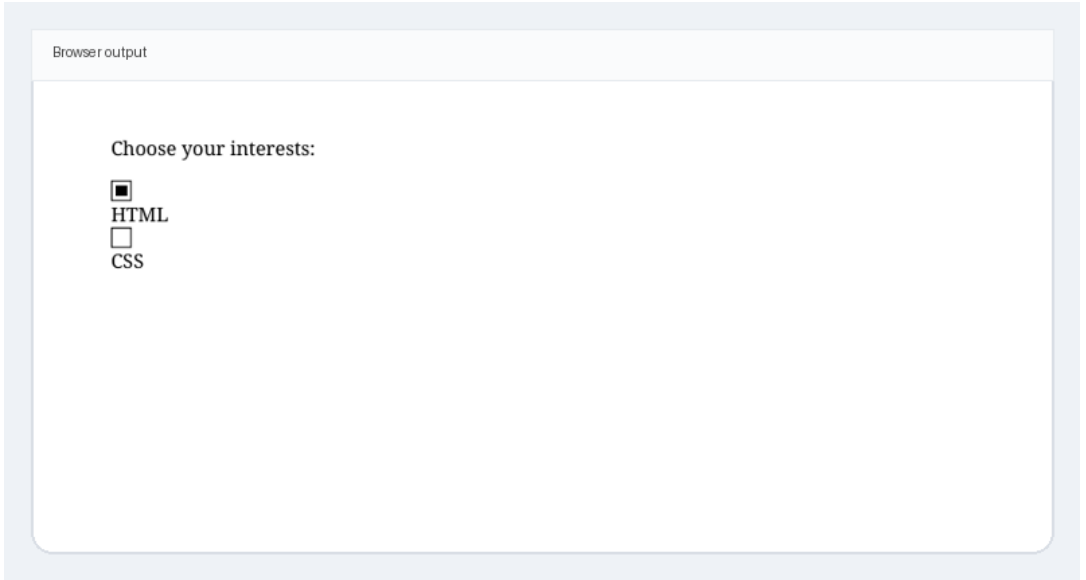
What to notice: Checkboxes are independent, so more than one can be selected. • Wrapping the input in the label makes clicking easier.

Code

HTML

```
 1  <form>
 2    <p>Choose your interests:</p>
 3    <label><input type="checkbox" name="topic" checked>
HTML</label>
 4    <label><input type="checkbox" name="topic">
CSS</label>
 5  </form>
```

Rendered output



Radio button group

HTML Unit 13 • Example

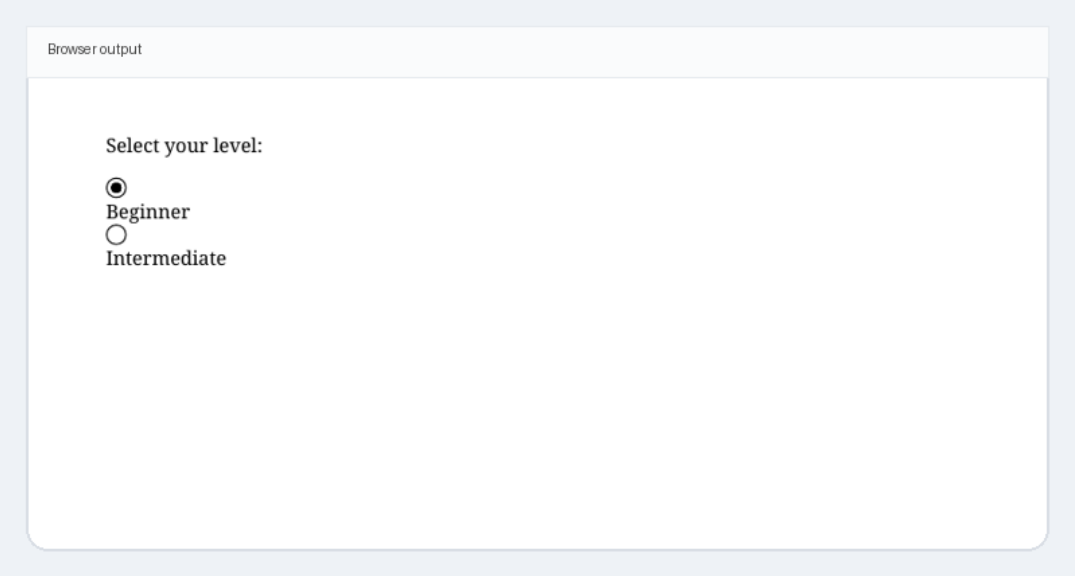
What to notice: Radio buttons become a group when they share a name value. • Only one choice is active at a time.

Code

HTML

```
 1  <form>
 2    <p>Select your level:</p>
 3    <label><input type="radio" name="level" checked>
Beginner</label>
 4    <label><input type="radio" name="level">
Intermediate</label>
 5  </form>
```

Rendered output



Basic select menu

HTML Unit 13 • Example

What to notice: Select menus keep the page compact when choices are known. • Options become the selectable values inside the control.

Code

HTML

```html
<form>
  <label for="city">City</label>
  <select id="city" name="city">
    <option>Istanbul</option>
    <option>Ankara</option>
    <option>Izmir</option>
  </select>
</form>
```

Rendered output

Select with optgroup

HTML Unit 13 • Example

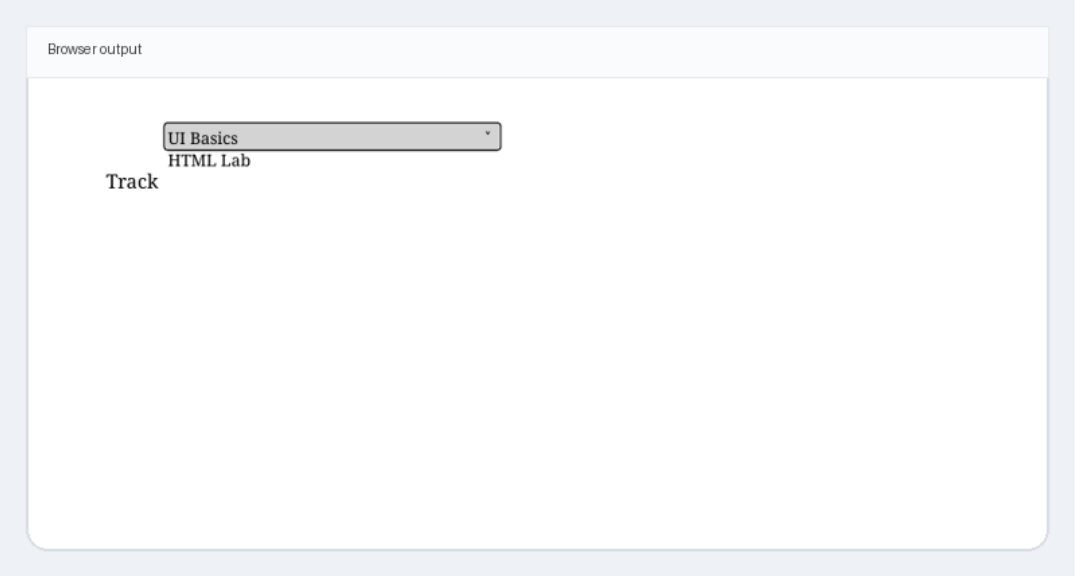
What to notice: optgroup organizes many options under a visible label. • Grouping makes long selects easier to scan.

Code

HTML

```html
<form>
  <label for="track">Track</label>
  <select id="track" name="track">
    <optgroup label="Design">
      <option>UI Basics</option>
    </optgroup>
    <optgroup label="Development">
      <option>HTML Lab</option>
    </optgroup>
  </select>
</form>
```

Rendered output



Input with datalist suggestions

HTML Unit 13 • Example

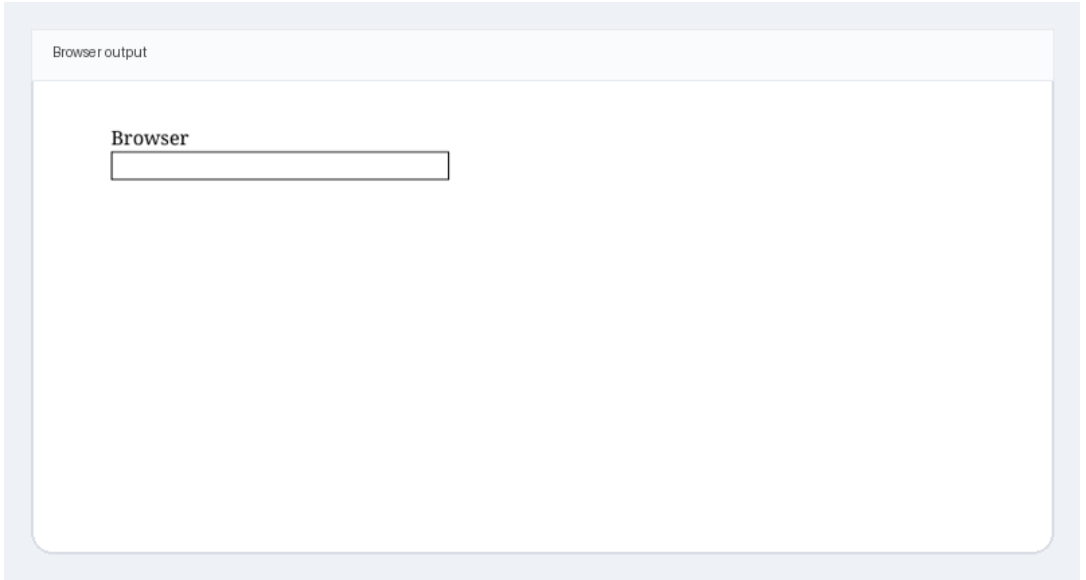
What to notice: datalist suggests values while still allowing custom text. • It is different from a strict drop-down select.

Code

HTML

```
 1  <form>
 2    <label for="browser">Browser</label>
 3    <input id="browser" list="browsers"
name="browser">
 4    <datalist id="browsers">
 5      <option value="Chrome">
 6      <option value="Firefox">
 7      <option value="Safari">
 8    </datalist>
 9  </form>
```

Rendered output



Choice Inputs: Common mistakes

- Do not use radio buttons when multiple answers should be possible.
- Do not create tiny unlabeled click targets for checkboxes and radios.
- Do not hide important options inside a long confusing select.
- Group related choices so the form stays understandable.

Choice Inputs: Recap

- Choice inputs reduce typing and improve consistency.
- Checkboxes, radios, selects, and datalists solve different problems.
- The name and label strategy matters as much as the control type.
- Good choice design prevents user hesitation.

Web Programming

HTML Unit 14

Advanced Form Controls

- HTML provides controls beyond plain text fields.
- Useful examples include range, color, file, hidden, and different button types.

Fenerbahce University

Advanced Form Controls: Key ideas

- HTML provides controls beyond plain text fields.
- Useful examples include range, color, file, hidden, and different button types.
- Buttons can submit, reset, or trigger custom behavior.
- Specialized controls make forms more expressive and efficient.

Advanced Form Controls: Practical notes

- Use hidden fields only for data the user does not need to edit.
- Use file inputs when the user must attach local content.
- Range inputs work well for approximate values like volume or rating.
- Choose the correct button type to avoid accidental submissions.

Range and color controls

HTML Unit 14 • Example

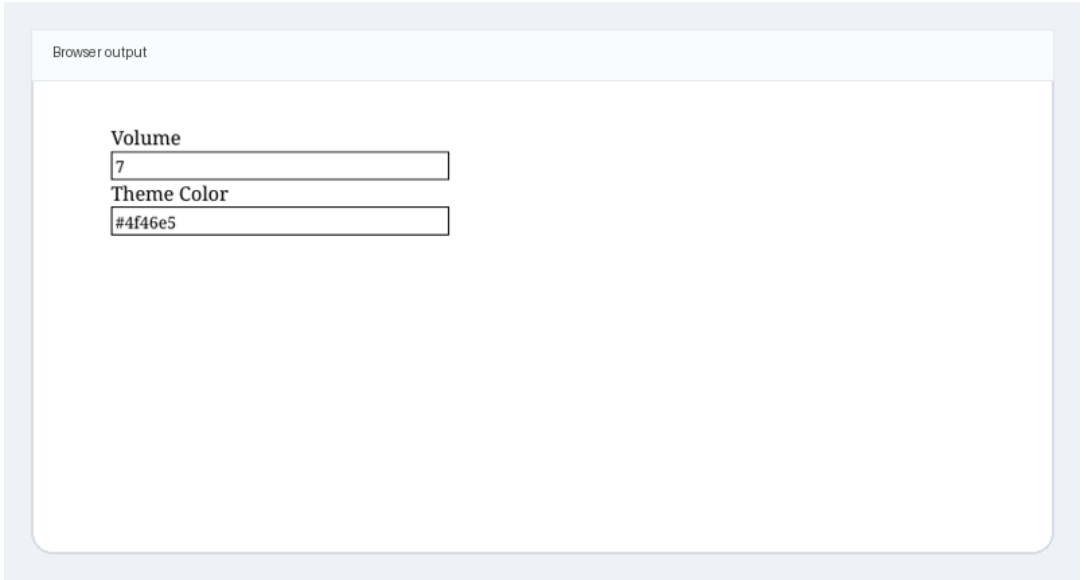
What to notice: Special controls can match the shape of the decision more naturally. • The browser renders each type with its own UI widget.

Code

HTML

```
 1  <form>
 2    <label for="volume">Volume</label>
 3    <input id="volume" type="range" min="0" max="10"
value="7">
 4
 5    <label for="theme">Theme Color</label>
 6    <input id="theme" type="color" value="#4f46e5">
 7  </form>
```

Rendered output



File input

HTML Unit 14 • Example

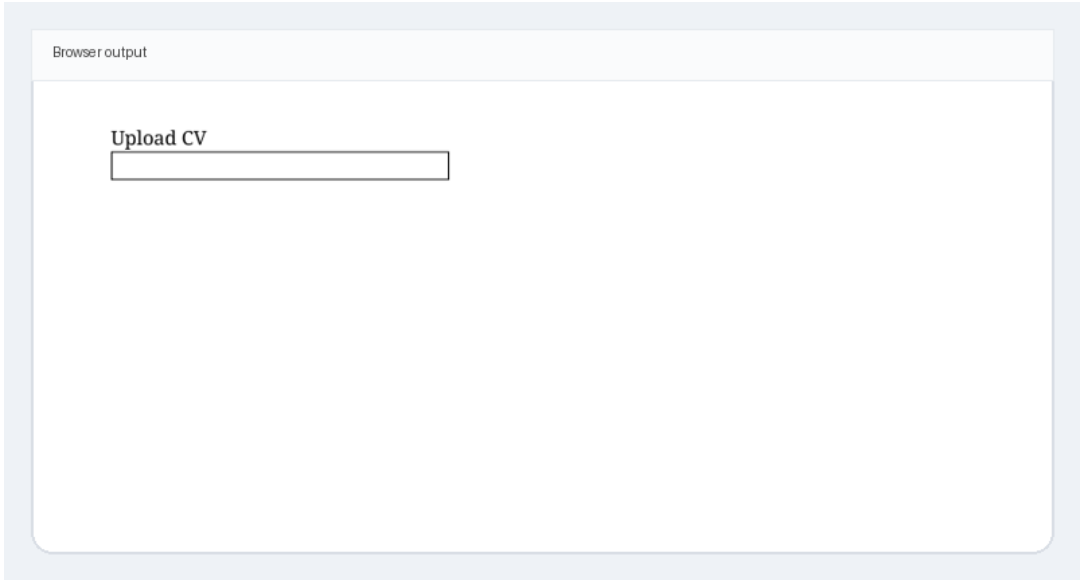
What to notice: File fields let the browser open a system file picker. • The selected file name appears beside the control.

Code

HTML

```html
<form>
  <label for="resume">Upload CV</label>
  <input id="resume" name="resume" type="file">
</form>
```

Rendered output



Hidden input with visible fields

HTML Unit 14 • Example

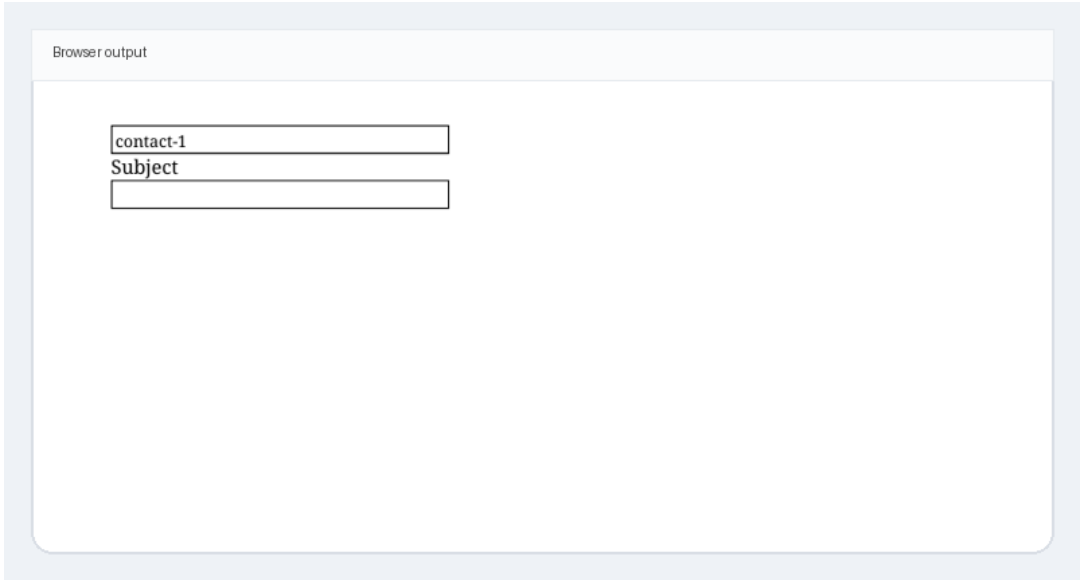
What to notice: Hidden fields can carry metadata without adding visible controls. • They should never replace visible confirmation or consent.

Code

HTML

```html
<form>
  <input type="hidden" name="form-id" value="contact-1">
  <label for="subject">Subject</label>
  <input id="subject" name="subject" type="text">
</form>
```

Rendered output



Different button types

HTML Unit 14 • Example

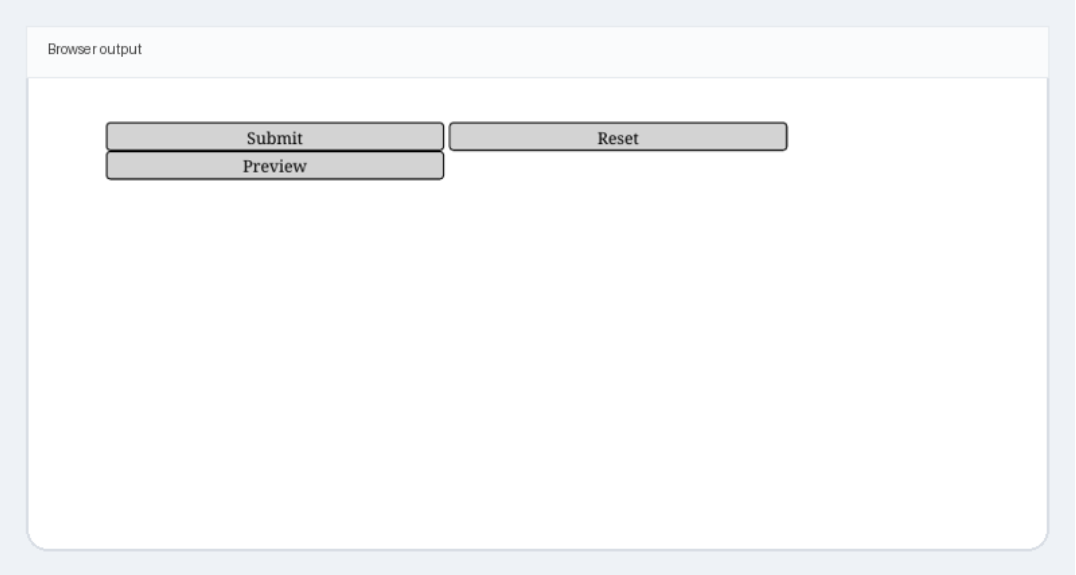
What to notice: Button type changes what happens when the button is pressed. • Explicit types prevent confusing default behavior.

Code

HTML

```html
<form>
  <button type="submit">Submit</button>
  <button type="reset">Reset</button>
  <button type="button">Preview</button>
</form>
```

Rendered output



Progress and meter

HTML Unit 14 • Example

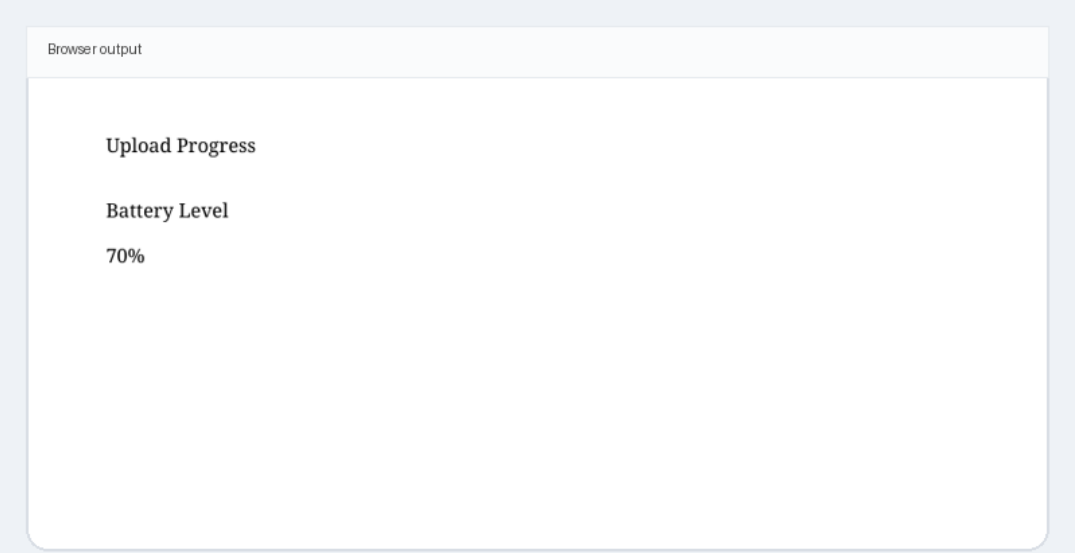
What to notice: Some HTML elements communicate measured progress or status. • They add meaning even before CSS improves their look.

Code

HTML

```
1  <p>Upload Progress</p>
2  <progress value="60" max="100"></progress>
3  <p>Battery Level</p>
4  <meter value="0.7">70%</meter>
```

Rendered output



Advanced Form Controls: Common mistakes

- Do not use hidden fields to store secret information.
- Do not assume users understand what a reset button will do.
- Do not use a range input when precise exact entry is required.
- Specialized controls should still have labels and context.

Advanced Form Controls: Recap

- Advanced controls reduce friction for specific tasks.
- Button type matters more than many beginners expect.
- HTML includes useful status elements such as progress and meter.
- Better control choice usually leads to better form UX.

Web Programming

HTML Unit 15

Validation and Form UX

- HTML includes built-in validation rules for many inputs.
- Useful attributes include required, minlength, maxlength, min, max, step, and pattern.

Fenerbahce University

Validation and Form UX: Key ideas

- HTML includes built-in validation rules for many inputs.
- Useful attributes include required, minlength, maxlength, min, max, step, and pattern.
- Validation should guide users, not surprise them.
- Clear forms combine labels, hints, grouping, and sensible defaults.

Validation and Form UX: Practical notes

- Browser validation helps catch common mistakes early.
- Pattern should be used only when a real rule exists.
- Helpful hint text belongs near the control, not hidden far away.
- Validation is strongest when the field type and attributes work together.

Required fields

HTML Unit 15 • Example

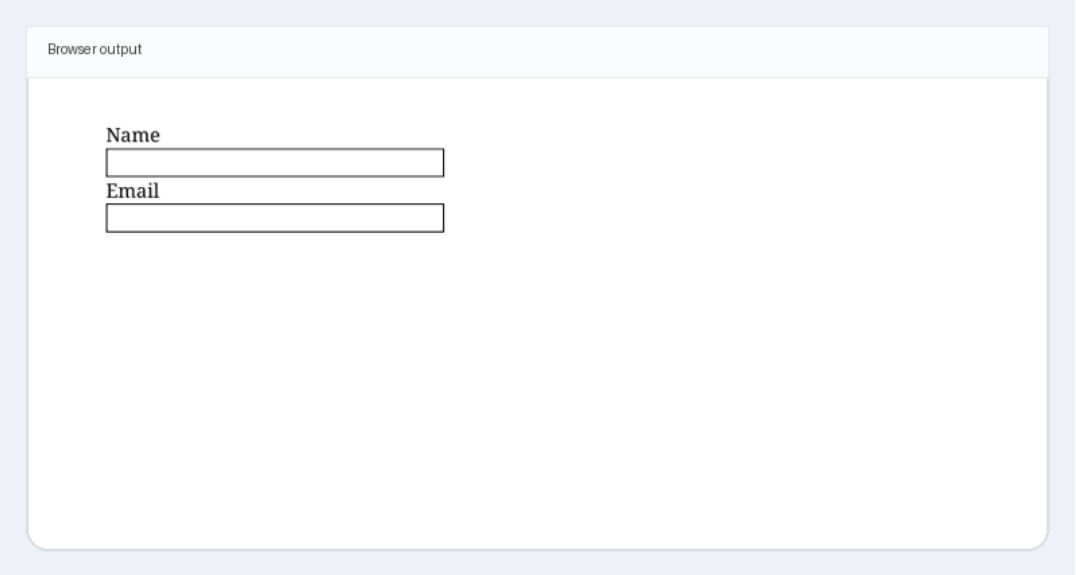
What to notice: required asks the browser to block empty submission. • The user gets immediate feedback before data is sent.

Code

HTML

```
 1  <form>
 2    <label for="name">Name</label>
 3    <input id="name" name="name" required>
 4
 5    <label for="email">Email</label>
 6    <input id="email" name="email" type="email"
required>
 7  </form>
```

Rendered output



Length rules

HTML Unit 15 • Example

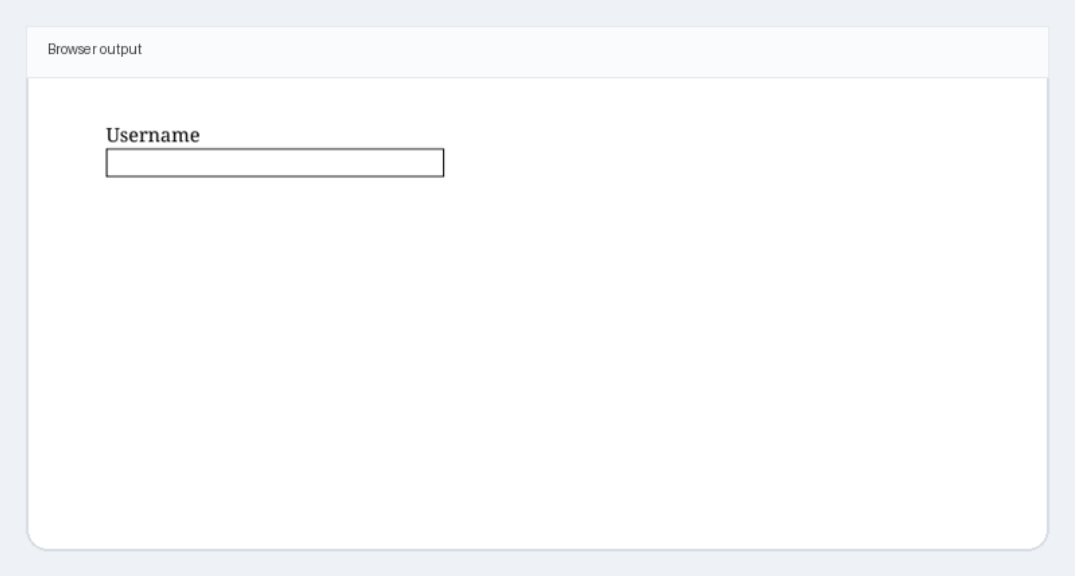
What to notice: minlength and maxlength define an allowed text range. • These constraints are useful for usernames and codes.

Code

HTML

```
 1  <form>
 2    <label for="user">Username</label>
 3    <input id="user" name="user" minlength="4"
maxlength="12">
 4  </form>
```

Rendered output



Numeric limits

HTML Unit 15 • Example

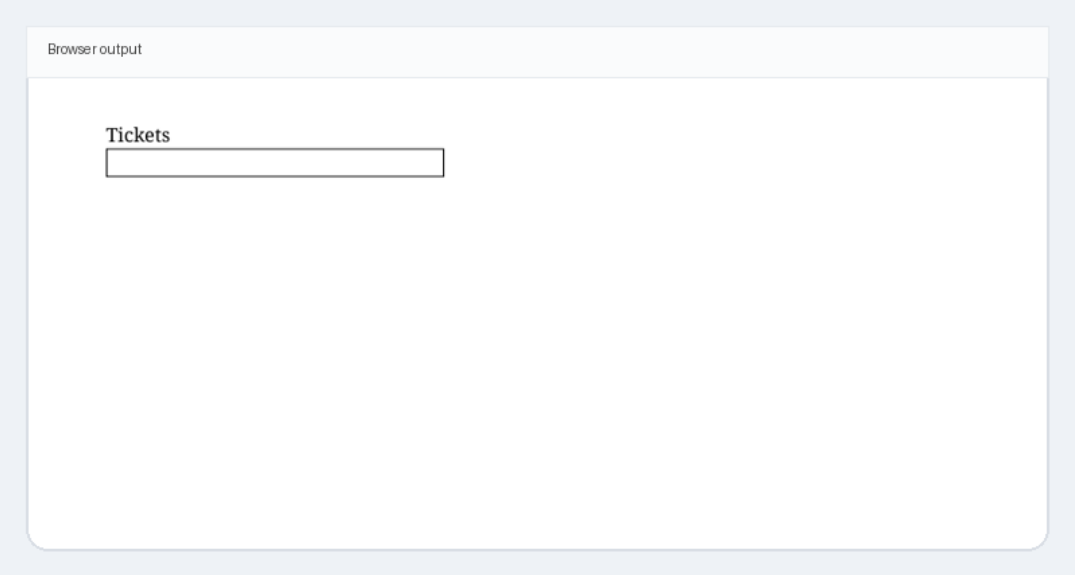
What to notice: min, max, and step guide acceptable numeric input. • The browser can prevent obviously invalid values.

Code

HTML

```
 1  <form>
 2    <label for="count">Tickets</label>
 3    <input id="count" name="count" type="number" min="1" max="5"
step="1">
 4  </form>
```

Rendered output



Pattern example

HTML Unit 15 • Example

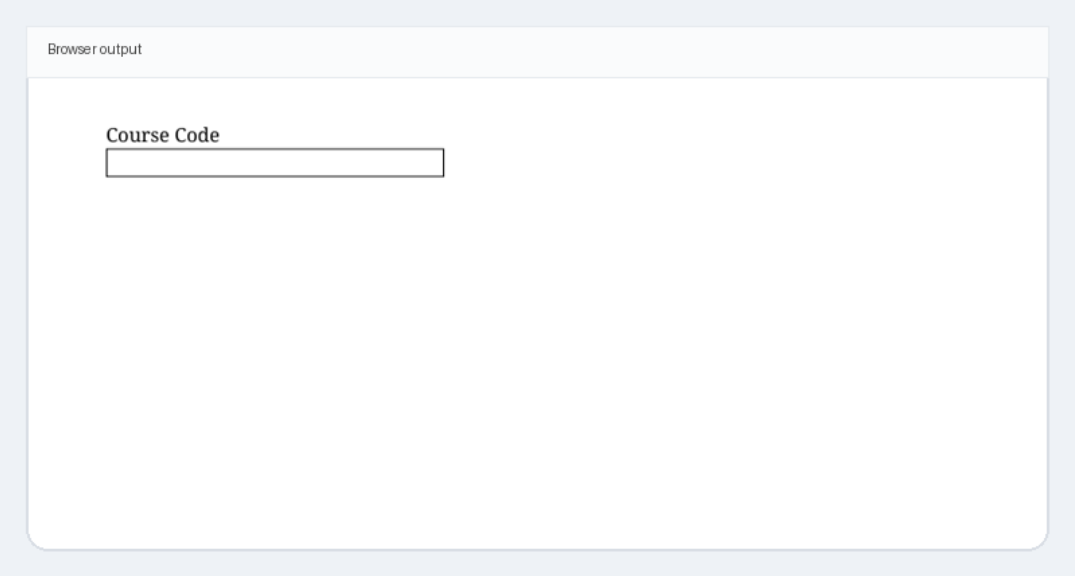
What to notice: pattern matches a required text format. • The placeholder can show an example, but the pattern enforces the rule.

Code

HTML

```
 1  <form>
 2    <label for="code">Course Code</label>
 3    <input id="code" name="code" pattern="[A-Z]{3}[0-9]{3}"
placeholder="WEB101">
 4  </form>
```

Rendered output



Hint text near a field

HTML Unit 15 • Example

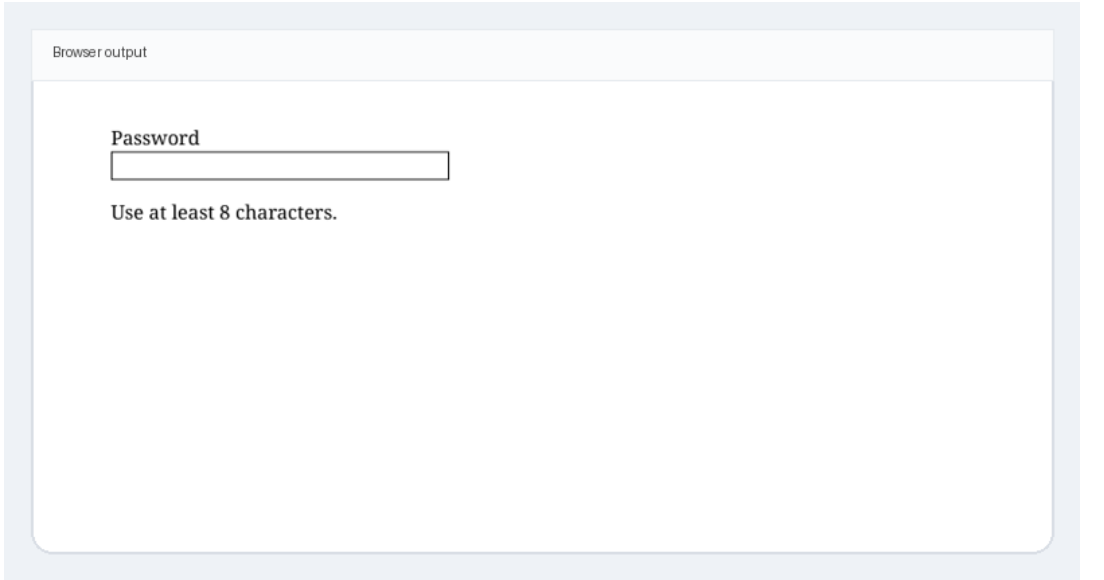
What to notice: Good UX explains the rule before the error happens. • Helpful guidance reduces frustration and failed submissions.

Code

HTML

```
 1  <form>
 2    <label for="password">Password</label>
 3    <input id="password" name="password" type="password"
minlength="8">
 4    <p>Use at least 8 characters.</p>
 5  </form>
```

Rendered output



Validation and Form UX: Common mistakes

- Do not create strict patterns users cannot easily understand.
- Do not hide requirements until after the user fails.
- Do not combine conflicting rules such as impossible min/max values.
- Validation should support user success, not punish mistakes.

Validation and Form UX: Recap

- HTML validation is practical and often enough for common cases.
- Attributes can express required format, size, and range.
- UX quality depends on labels and hints as much as validation rules.
- A good form feels predictable from the first click.

Web Programming

HTML Unit 16

Media, Embeds, and Interactive Details

- HTML can include audio, video, and embedded documents.
- Common elements include audio, video, source, track, iframe, details, and summary.

Fenerbahce University

Media, Embeds, and Interactive Details: Key ideas

- HTML can include audio, video, and embedded documents.
- Common elements include audio, video, source, track, iframe, details, and summary.
- Media elements often need controls so the user can interact with them.
- Embedded content should remain relevant and clearly labeled.

Media, Embeds, and Interactive Details: Practical notes

- Use fallback text when media cannot load.
- Details and summary create expandable content without JavaScript.
- Iframe can embed another page or a controlled mini document.
- Keep media purposeful so it supports the lesson rather than distracts.

Audio element

HTML Unit 16 • Example

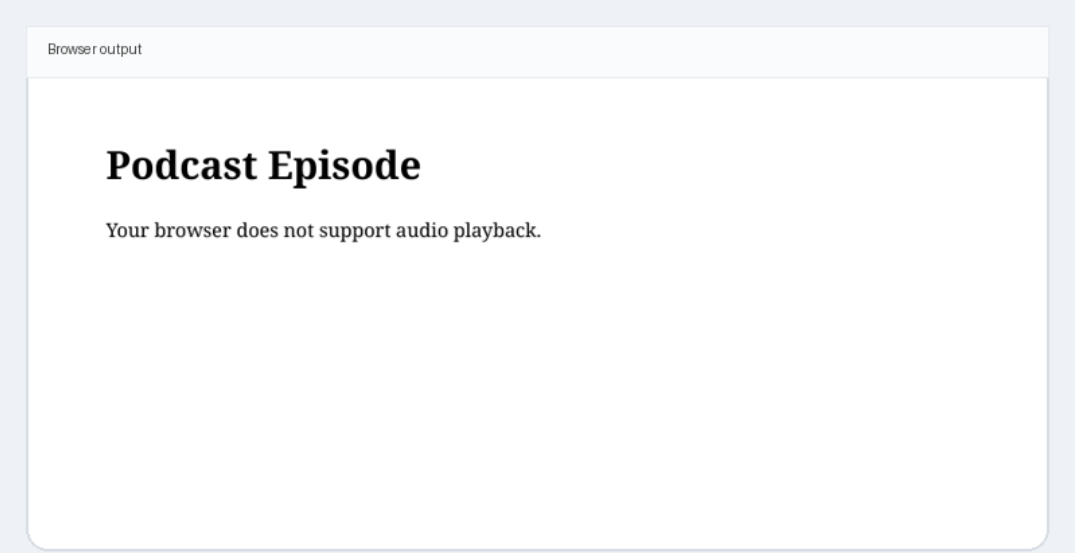
What to notice: The controls attribute asks the browser to show playback controls. • Fallback text appears if the media cannot be played.

Code

HTML

```
<h1>Podcast Episode</h1>
<audio controls>
  <source src="episode.mp3" type="audio/mpeg">
  Your browser does not support audio playback.
</audio>
```

Rendered output



Video element

HTML Unit 16 • Example

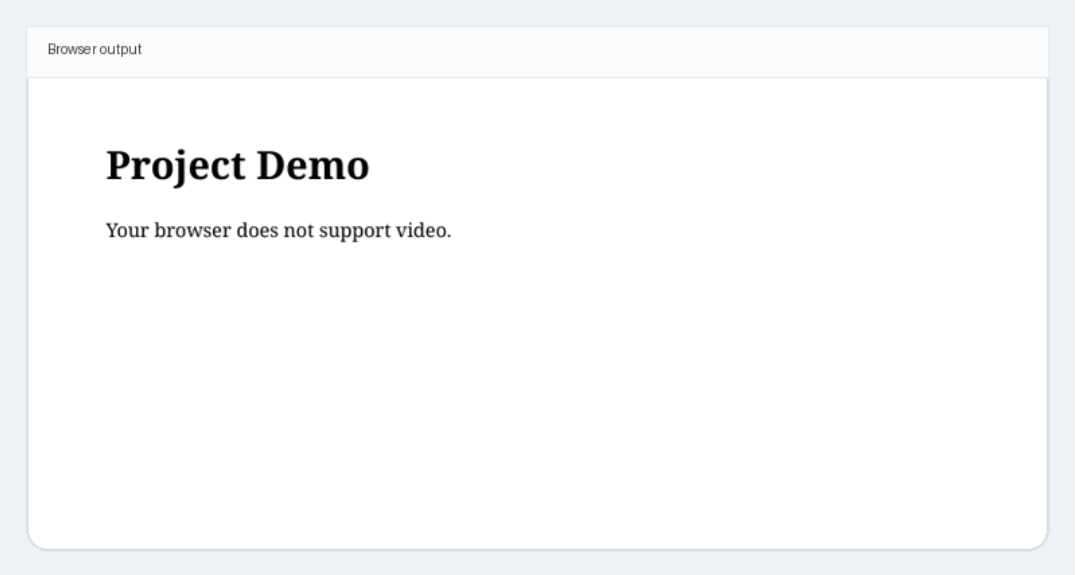
What to notice: Video behaves like audio but also reserves visible space on the page. • Additional source files can improve compatibility.

Code

HTML

```
<h1>Project Demo</h1>
<video controls width="280">
  <source src="demo.mp4" type="video/mp4">
  Your browser does not support video.
</video>
```

Rendered output



Iframe with srcdoc

HTML Unit 16 • Example

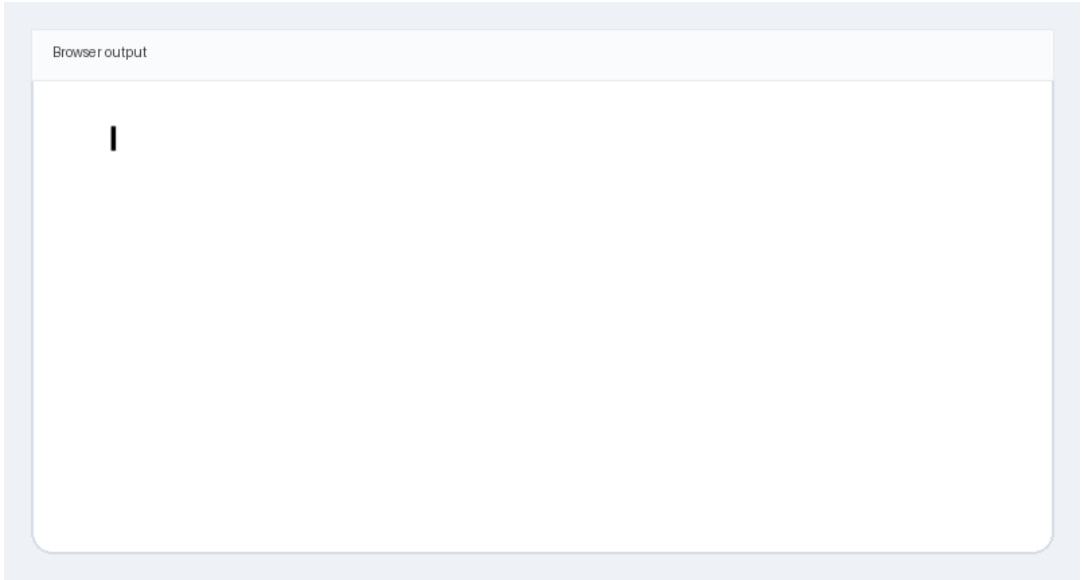
What to notice: An iframe creates a separate embedded browsing context. • The title attribute helps explain the embedded content.

Code

HTML

```
<iframe
  srcdoc="<h1>Mini Page</h1><p>This content lives inside an iframe.</p>"
  title="Mini example page">
</iframe>
```

Rendered output



Details and summary

HTML Unit 16 • Example

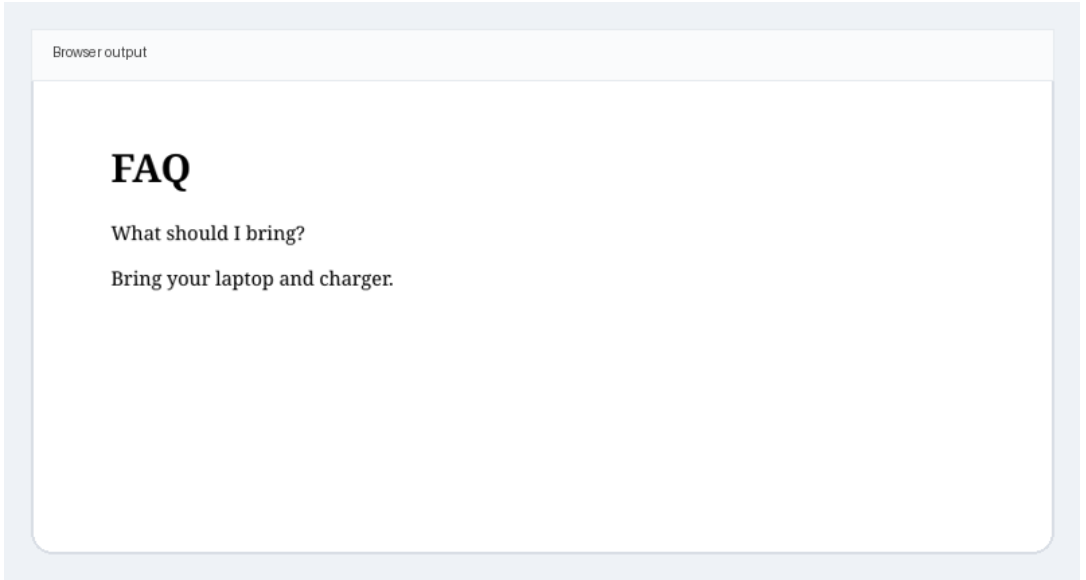
What to notice: This pattern builds collapsible disclosure content with plain HTML. • open makes the details visible by default.

Code

HTML

```html
<h1>FAQ</h1>
<details open>
  <summary>What should I bring?</summary>
  <p>Bring your laptop and charger.</p>
</details>
```

Rendered output



Video with captions track

HTML Unit 16 • Example

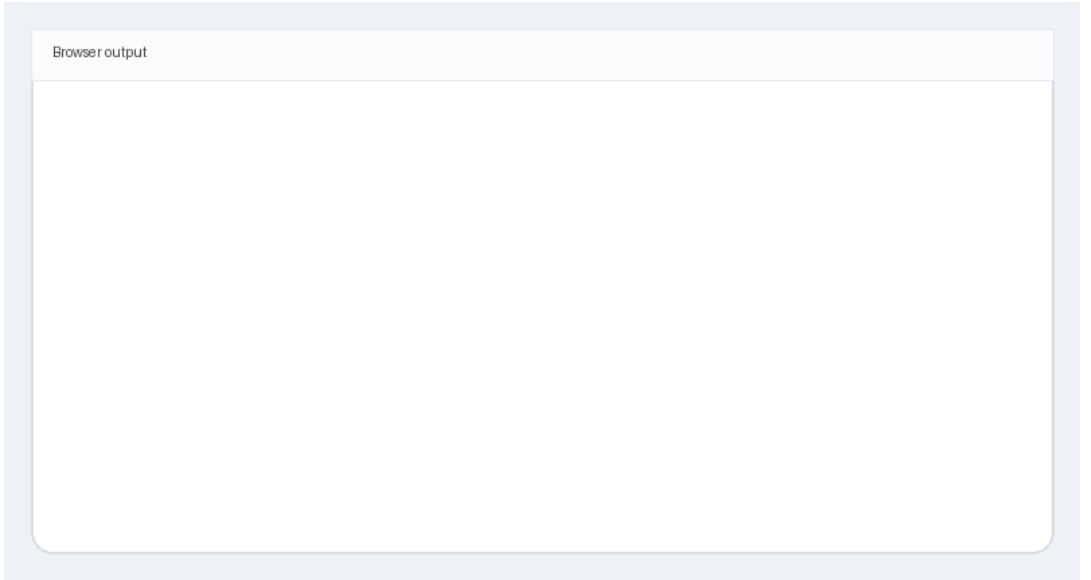
What to notice: Tracks attach captions or subtitles to the media element. • Caption support improves accessibility and comprehension.

Code

HTML

```
 1  <video controls width="280">
 2    <source src="lesson.mp4" type="video/mp4">
 3    <track kind="captions" src="lesson.vtt" srclang="en"
label="English">
 4  </video>
```

Rendered output



Media, Embeds, and Interactive Details: Common mistakes

- Do not autoplay media that interrupts the user unexpectedly.
- Do not embed unrelated content just because it is possible.
- Do not forget fallback text or accessible titles where needed.
- Expandable details work best when the summary text is specific.

Media, Embeds, and Interactive Details: Recap

- HTML media elements bring sound, video, and expandable content to the page.
- Controls, fallback text, and labels matter for usability.
- Iframes embed another document; details reveal content progressively.
- Media should clarify the lesson instead of competing with it.

Web Programming

HTML Unit 17

Head, Metadata, and Project Paths

- The head element contains document metadata, not visible page content.
- Common metadata includes title, charset, viewport, description, and linked assets.

Fenerbahce University

Head, Metadata, and Project Paths: Key ideas

- The head element contains document metadata, not visible page content.
- Common metadata includes title, charset, viewport, description, and linked assets.
- Correct file paths make images, pages, and scripts load reliably.
- A small folder structure becomes important very quickly in real projects.

Head, Metadata, and Project Paths: Practical notes

- Metadata supports search, sharing, compatibility, and document identity.
- Relative paths are usually easier than absolute paths inside one project.
- Use predictable folder names for images, pages, and scripts.
- The browser reads head information before showing the full page.

Metadata in the head

HTML Unit 17 • Example

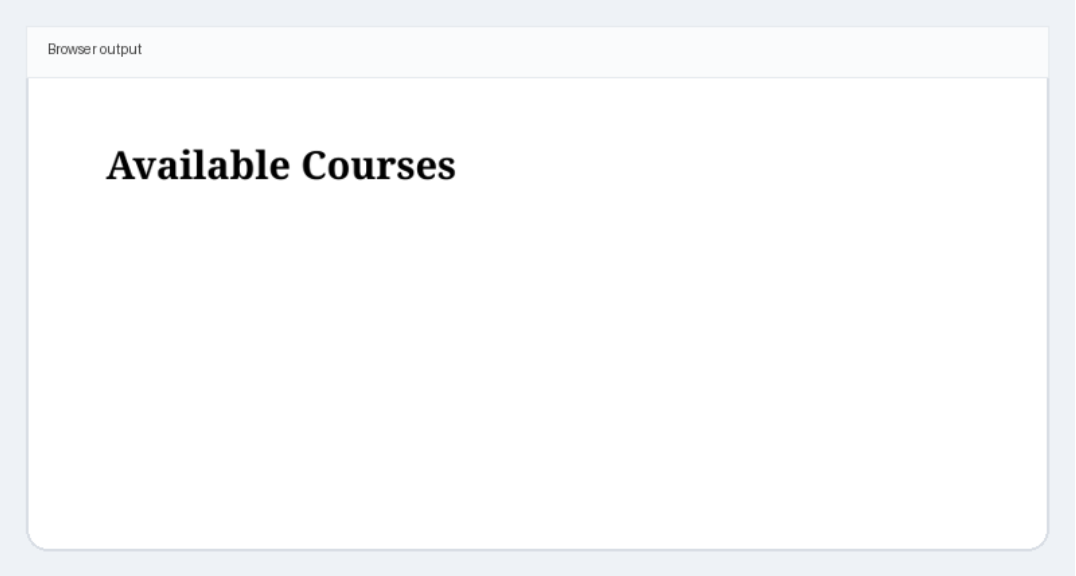
What to notice: The head stores technical and descriptive information about the page. • The viewport meta tag matters especially on mobile devices.

Code

HTML

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Course Catalog</title>
  </head>
  <body>
    <h1>Available Courses</h1>
  </body>
</html>
```

Rendered output



Description metadata

HTML Unit 17 • Example

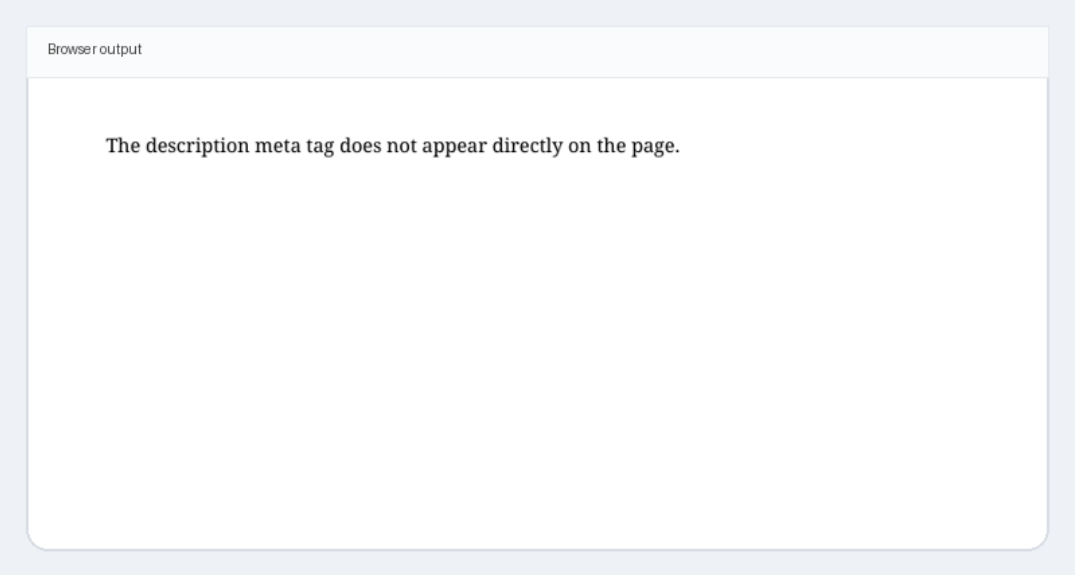
What to notice: Description metadata can support search previews and page summaries. • It belongs in the head, not the visible page body.

Code

HTML

```
 1  <head>
 2    <title>Portfolio</title>
 3    <meta name="description" content="Student portfolio with
projects and contact details.">
 4  </head>
```

Rendered output



Favicon link

HTML Unit 17 • Example

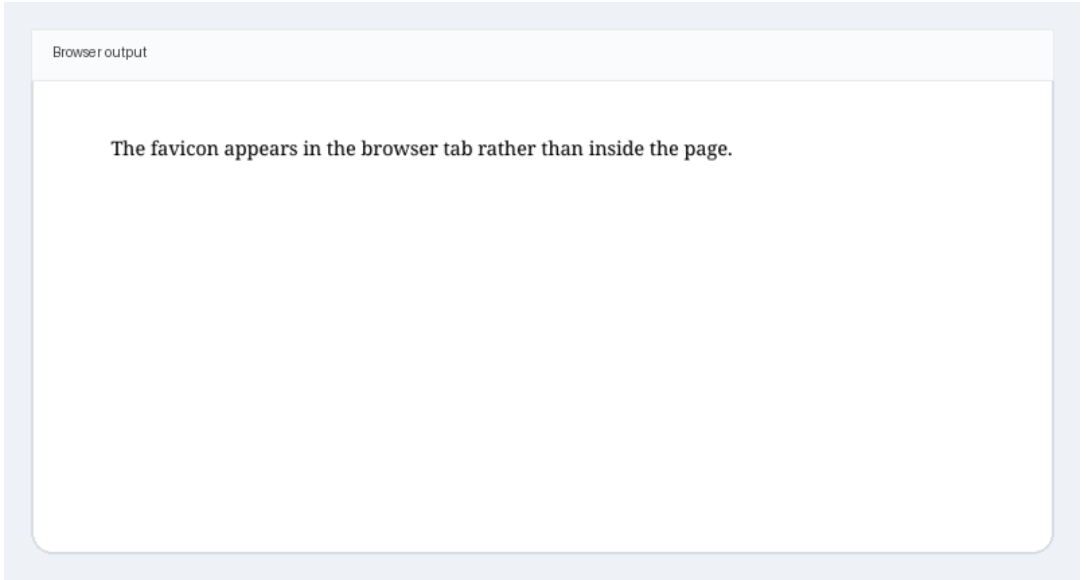
What to notice: A favicon is the small icon often shown in the browser tab. • The linked file path must be correct for the icon to load.

Code

HTML

```html
<head>
  <title>News Page</title>
  <link rel="icon" href="favicon.png">
</head>
```

Rendered output



Relative paths in a project

HTML Unit 17 • Example

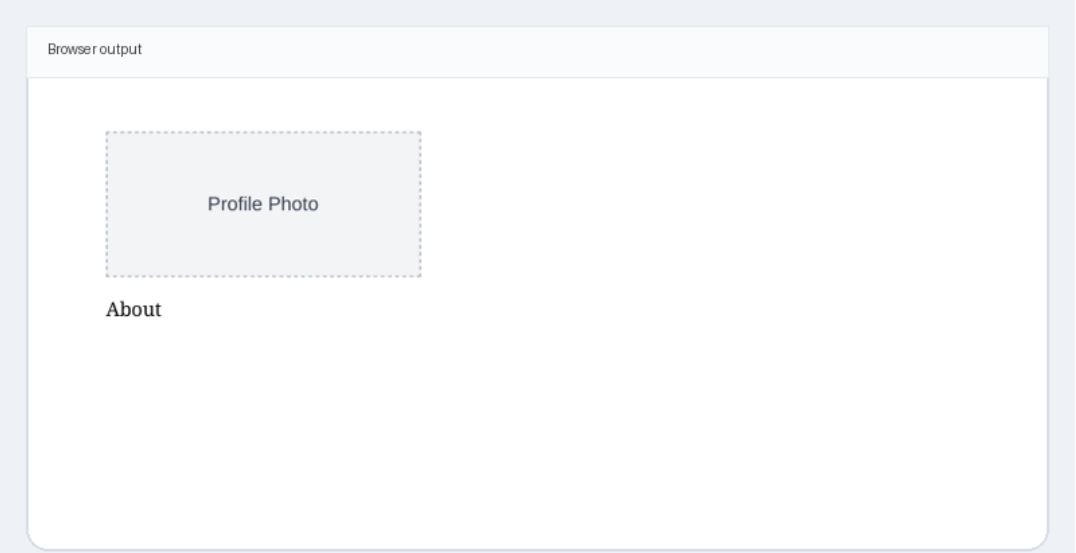
What to notice: Paths are read from the location of the current HTML file. • Clear folders help prevent broken links and missing assets.

Code

HTML

```
 1  <img src="images/profile.jpg" alt="Student profile
photo">
 2  <a href="pages/about.html">About</a>
 3  <script src="scripts/app.js"></script>
```

Rendered output



Parent folder path

HTML Unit 17 • Example

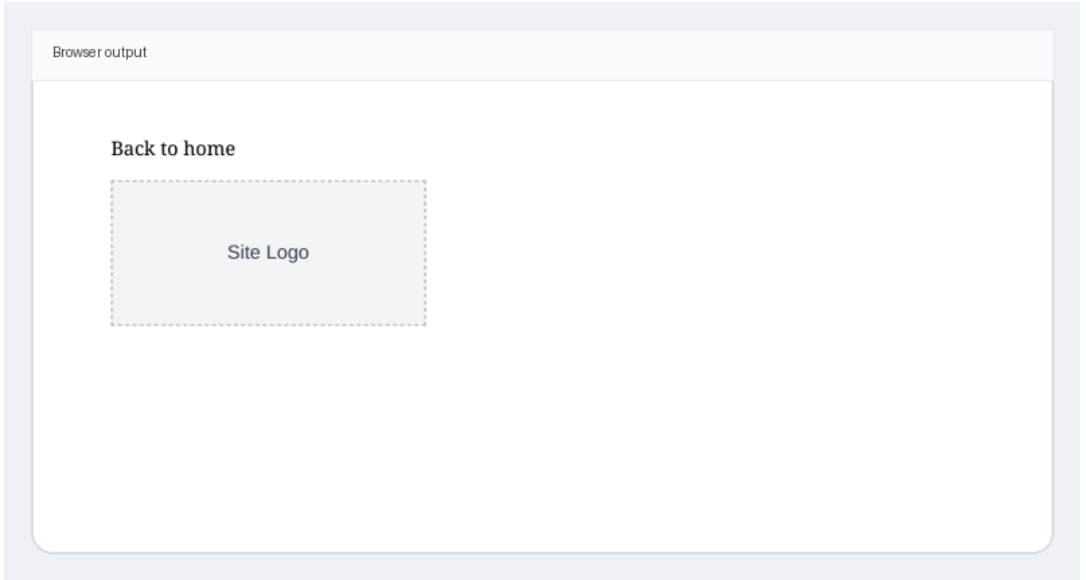
What to notice: .. moves one level up in the folder hierarchy. • Relative paths should mirror the real project structure carefully.

Code

HTML

```html
<a href="../index.html">Back to home</a>
<img src="../images/logo.png" alt="Site logo">
```

Rendered output



Head, Metadata, and Project Paths: Common mistakes

- Do not put headings or paragraphs directly in the head.
- Do not guess file paths; verify the project tree.
- Do not use vague or duplicate page titles across the site.
- Metadata is invisible, but it still shapes the page experience.

Head, Metadata, and Project Paths: Recap

- The head is the page’s control center for metadata.
- Titles, descriptions, and paths all matter in production work.
- Project organization prevents many broken-file errors.
- Strong structure outside the body is still part of good HTML.

Web Programming

HTML Unit 18

Entities, Language, Direction, and Comments

- HTML entities let you write reserved or special characters safely.
- Common entities include <, >, &, ©, and .

Fenerbahce University

Entities, Language, Direction, and Comments: Key ideas

- HTML entities let you write reserved or special characters safely.
- Common entities include <, >, &, ©, and .
- Language and direction attributes help multilingual content.
- Comments document the source without appearing on the page.

Entities, Language, Direction, and Comments: Practical notes

- Use entities when the character would otherwise be treated as markup.
- lang can be applied to the whole page or only to a phrase.
- dir is useful when content flows right to left.
- Comments should explain intent, not repeat the obvious.

Showing reserved characters

HTML Unit 18 • Example

What to notice: Entities make reserved markup characters visible as text. • Without them, the browser would try to parse a tag.

Code

HTML

```
 1  <p>To write a paragraph tag, type
&lt;p&gt;Hello&lt;/p&gt;.</p>
 2  <p>Fish &amp; Chips is written with an ampersand.</p>
```

Rendered output

Browser output

To write a paragraph tag, type Hello .

Fish & Chips is written with an ampersand.

Copyright and non-breaking space

HTML Unit 18 • Example

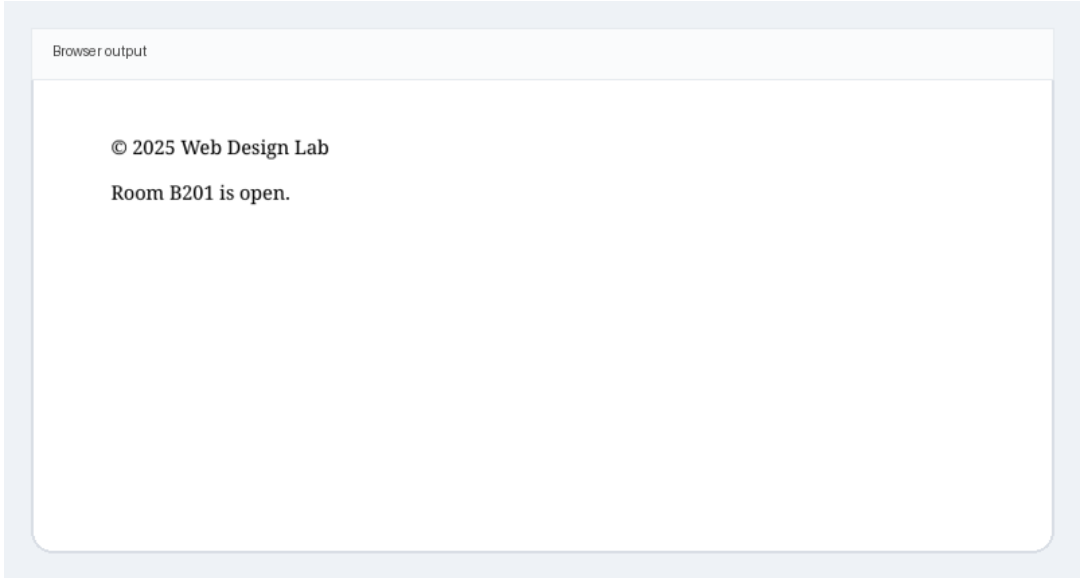
What to notice: Entities can express symbols and spacing rules. • Non-breaking spaces keep words together on one line when possible.

Code

HTML

```
1  <p>&copy; 2025 Web Design Lab</p>
2  <p>Room B201 is open.</p>
```

Rendered output



Language on part of a sentence

HTML Unit 18 • Example

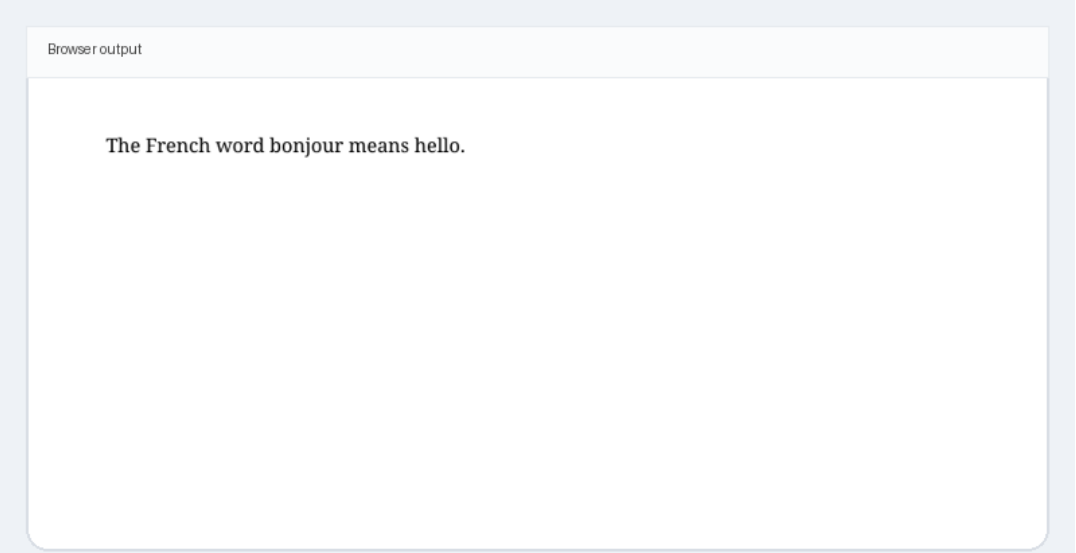
What to notice: lang can apply to one word, sentence, or full document. • This helps pronunciation and text processing tools.

Code

HTML

```
 1  <p>The French word <span lang="fr">bonjour</span> means
hello.</p>
```

Rendered output



Right-to-left direction

HTML Unit 18 • Example

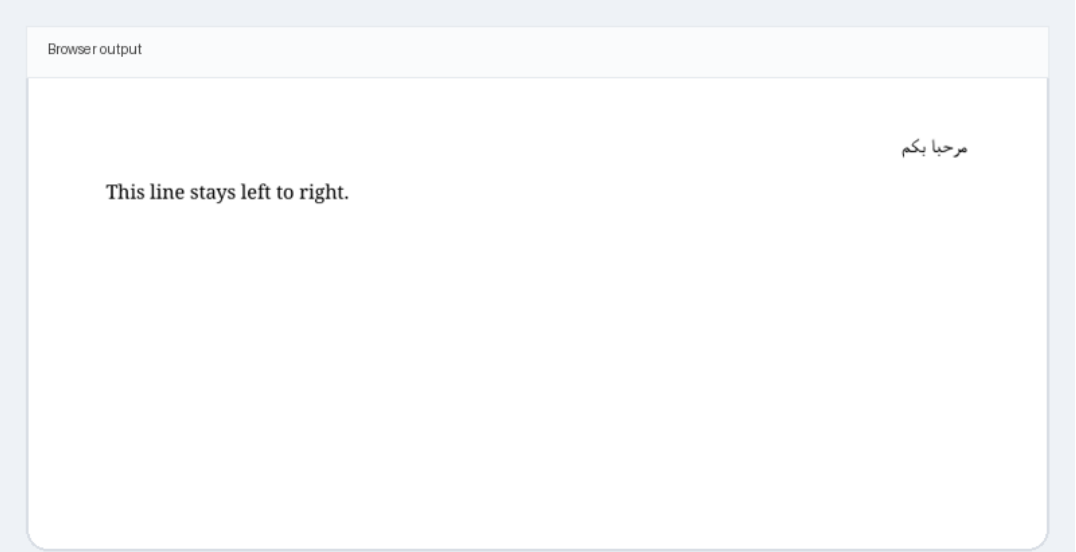
What to notice: Direction settings matter for Arabic, Hebrew, and mixed content. • HTML can express text direction semantically.

Code

HTML

```
1  <p dir="rtl">مرحبا بكم</p>
2  <p>This line stays left to right.</p>
```

Rendered output



Source comments

HTML Unit 18 • Example

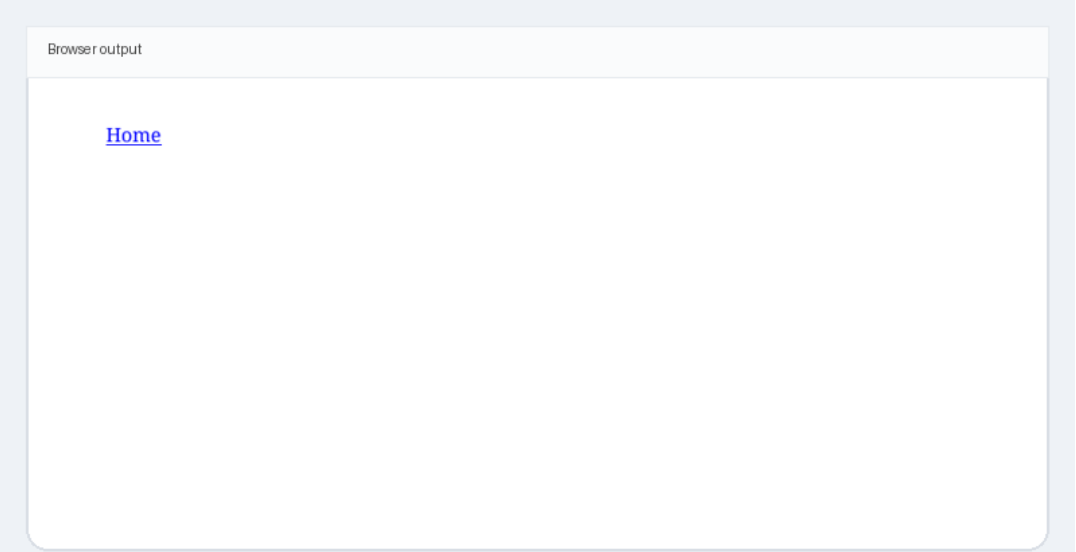
What to notice: Comments can mark regions or explain tricky decisions in the source. • They are for developers, not page readers.

Code

HTML

```
<!-- Navigation starts here -->
<nav>
  <a href="index.html">Home</a>
</nav>
```

Rendered output



Entities, Language, Direction, and Comments: Common mistakes

- Do not overuse   to fake layout spacing.
- Do not leave outdated comments that mislead other developers.
- Do not forget lang when mixing multiple languages in one page.
- Only use entities when they are needed or genuinely helpful.

Entities, Language, Direction, and Comments: Recap

- Entities let text stay safe and precise inside HTML.
- Language, direction, and comments all add context beyond visible words.
- These details often seem small but matter in real documents.
- Well-documented HTML is easier to maintain and translate.

Web Programming

HTML Unit 19

Accessibility Essentials

- Accessible HTML makes content usable for more people and devices.
- Semantics, labels, headings, alt text, and clear links are core foundations.

Fenerbahce University

Accessibility Essentials: Key ideas

- Accessible HTML makes content usable for more people and devices.
- Semantics, labels, headings, alt text, and clear links are core foundations.
- Accessibility should be built in from the first draft, not added at the end.
- Better accessibility usually also improves clarity for everyone.

Accessibility Essentials: Practical notes

- Use native HTML semantics before reaching for ARIA.
- Headings should create a logical outline.
- Buttons trigger actions; links move to destinations.
- Visible text, readable structure, and predictable controls all matter.

Accessible image

HTML Unit 19 • Example

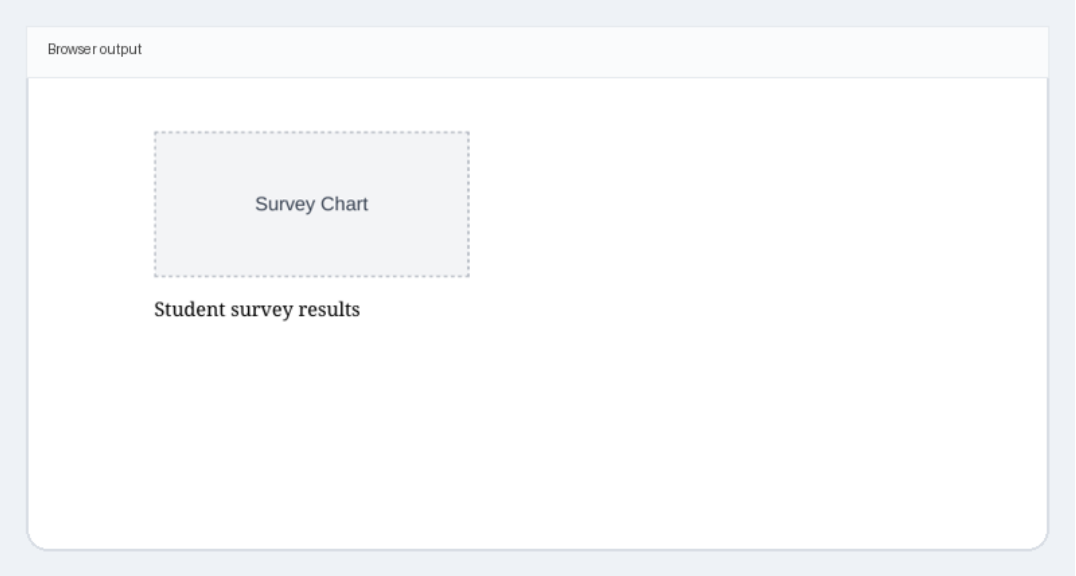
What to notice: The alt text should explain the useful message of the image. • Figure and caption add extra context around the media.

Code

HTML

```
<figure>
  <img src="chart.png" alt="Bar chart showing that HTML scored highest in the survey">
  <figcaption>Student survey results</figcaption>
</figure>
```

Rendered output



Well-labeled form control

HTML Unit 19 • Example

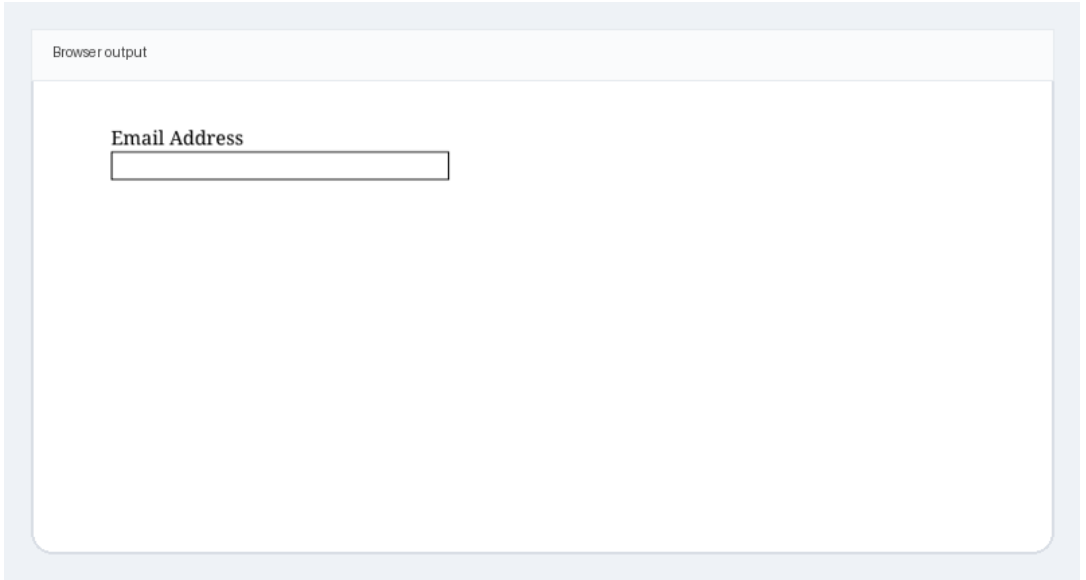
What to notice: A visible label helps both sighted users and screen reader users. • The required attribute sets a meaningful expectation.

Code

HTML

```
 1  <form>
 2    <label for="email">Email Address</label>
 3    <input id="email" name="email" type="email"
required>
 4  </form>
```

Rendered output



Descriptive link text

HTML Unit 19 • Example

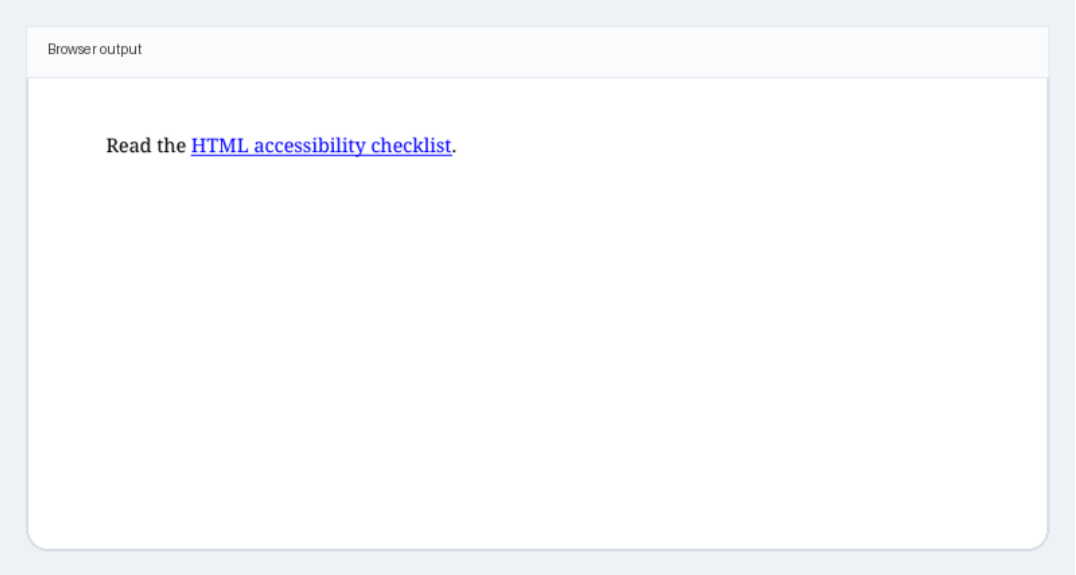
What to notice: The link explains the destination without extra surrounding context. • Users scanning links can predict where it leads.

Code

HTML

```
<p>
  Read the
  <a href="guide.html">HTML accessibility checklist</a>.
</p>
```

Rendered output



Button versus link

HTML Unit 19 • Example

What to notice: Choose the element that matches the user action. • This makes keyboard, screen reader, and browser behavior more predictable.

Code

HTML

```html
<button type="button">Open help panel</button>
<a href="help.html">Go to help page</a>
```

Rendered output

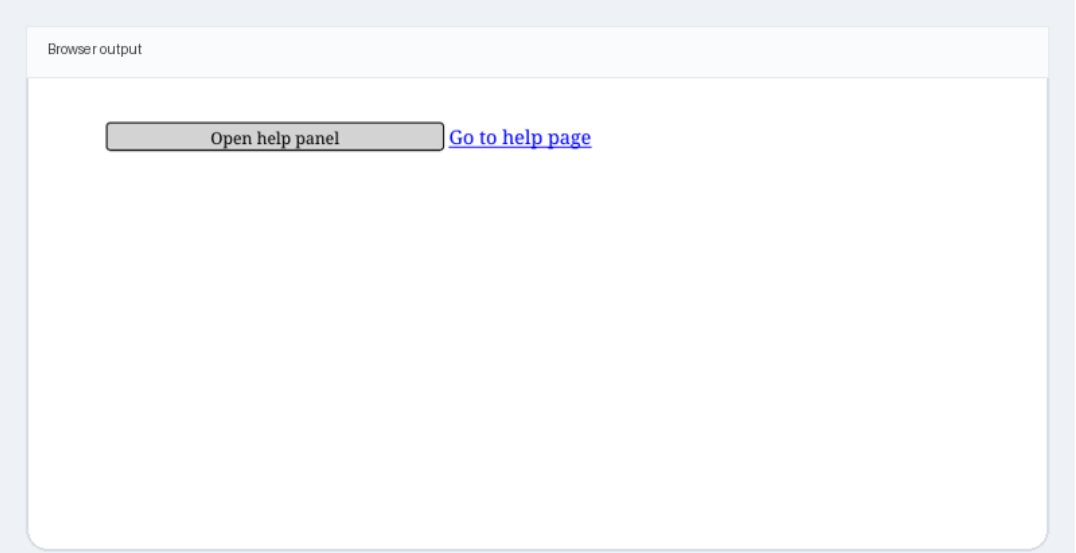


Table headers with scope

HTML Unit 19 • Example

What to notice: Accessible tables clearly label rows and columns. • Even simple tables benefit from proper header markup.

Code

HTML

```html
<table border="1">
  <tr>
    <th scope="col">Name</th>
    <th scope="col">Score</th>
  </tr>
  <tr>
    <td>Elif</td>
    <td>95</td>
  </tr>
</table>
```

Rendered output

Browser output

Name	Score
Elif	95

Accessibility Essentials: Common mistakes

- Do not use color alone to communicate important meaning.
- Do not skip visible labels because a placeholder “looks cleaner”.
- Do not use generic headings or repeated “Read more” links everywhere.
- Start with native HTML semantics before adding extra attributes.

Accessibility Essentials: Recap

- Accessible HTML is mostly good HTML done consistently.
- Semantics, labels, headings, and text alternatives carry most of the work.
- Users benefit when actions, structure, and relationships are clear.
- Accessibility is part of quality, not a separate optional layer.

Web Programming

HTML Unit 20

Mini Projects and Debugging

- Mini projects combine many small HTML ideas into one coherent page.
- Validation and debugging help catch structural mistakes early.

Fenerbahce University

Mini Projects and Debugging: Key ideas

- Mini projects combine many small HTML ideas into one coherent page.
- Validation and debugging help catch structural mistakes early.
- The browser developer tools and HTML validators are useful companions.
- Before CSS, a strong HTML page is already meaningful and usable.

Mini Projects and Debugging: Practical notes

- Test pages with real content, not only placeholder fragments.
- Indentation, semantics, and labels make bugs easier to spot.
- Broken paths, mismatched tags, and missing labels are common early issues.
- A final HTML review should check structure, meaning, and accessibility.

Mini project: profile page

HTML Unit 20 • Example

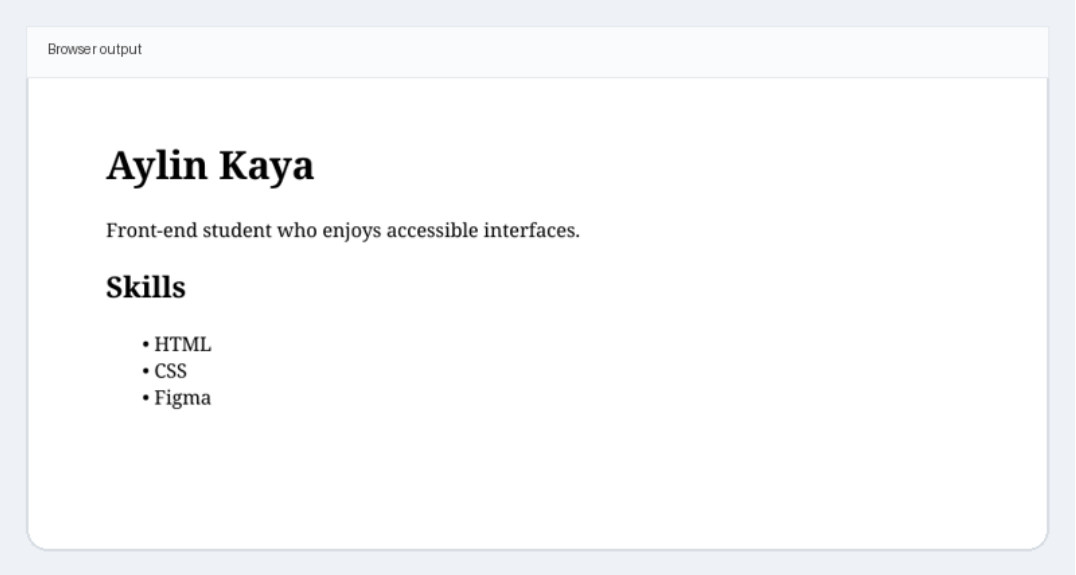
What to notice: A small profile page combines headings, paragraphs, and lists. • This is a strong practice project for early HTML lessons.

Code

HTML

```
<header><h1>Aylin Kaya</h1></header>
<main>
  <p>Front-end student who enjoys accessible
interfaces.</p>
  <h2>Skills</h2>
  <ul><li>HTML</li><li>CSS</li><li>Figma</li></ul>
</main>
```

Rendered output



Mini project: FAQ page

HTML Unit 20 • Example

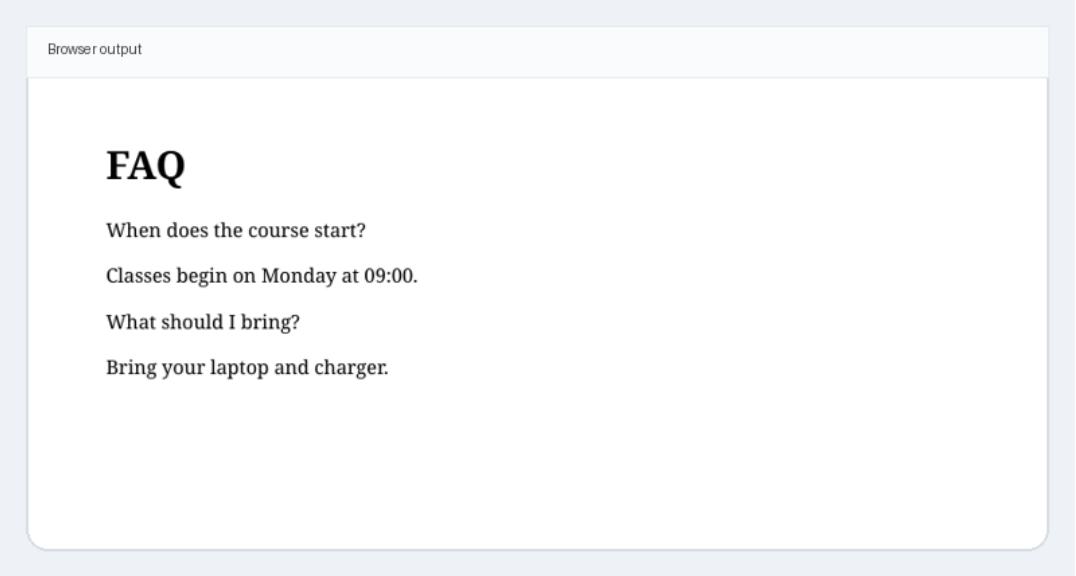
What to notice: The FAQ project mixes headings and interactive disclosure elements. • No CSS is required to make it understandable and usable.

Code

HTML

```html
<h1>FAQ</h1>
<details open>
  <summary>When does the course start?</summary>
  <p>Classes begin on Monday at 09:00.</p>
</details>
<details>
  <summary>What should I bring?</summary>
  <p>Bring your laptop and charger.</p>
</details>
```

Rendered output



Mini project: menu page

HTML Unit 20 • Example

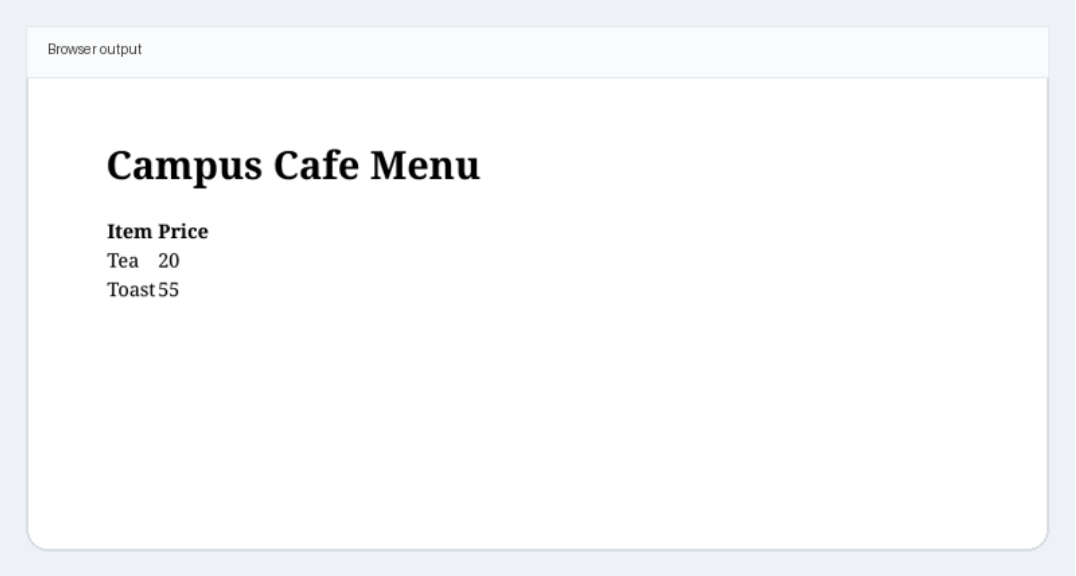
What to notice: A menu page is a practical place to use headings and data tables. • Simple HTML alone can already communicate the content clearly.

Code

HTML

```
<h1>Campus Cafe Menu</h1>
<table border="1">
  <tr><th>Item</th><th>Price</th></tr>
  <tr><td>Tea</td><td>20</td></tr>
  <tr><td>Toast</td><td>55</td></tr>
</table>
```

Rendered output



Mini project: registration form

HTML Unit 20 • Example

What to notice: This project combines text input, checkbox consent, and submission. • Validation attributes add useful behavior without scripts.

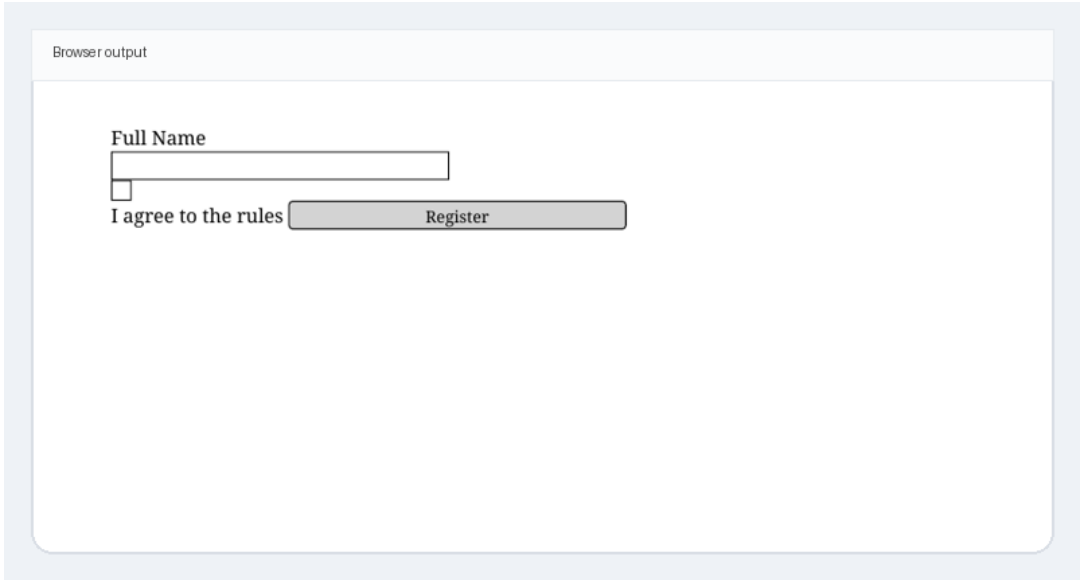
Code

HTML

```
<form>
  <label for="name">Full Name</label>
  <input id="name" name="name" required>

  <label><input type="checkbox" name="agree" required> I agree to the rules</label>
  <button type="submit">Register</button>
</form>
```

Rendered output



Debugging target: mismatched nesting

HTML Unit 20 • Example

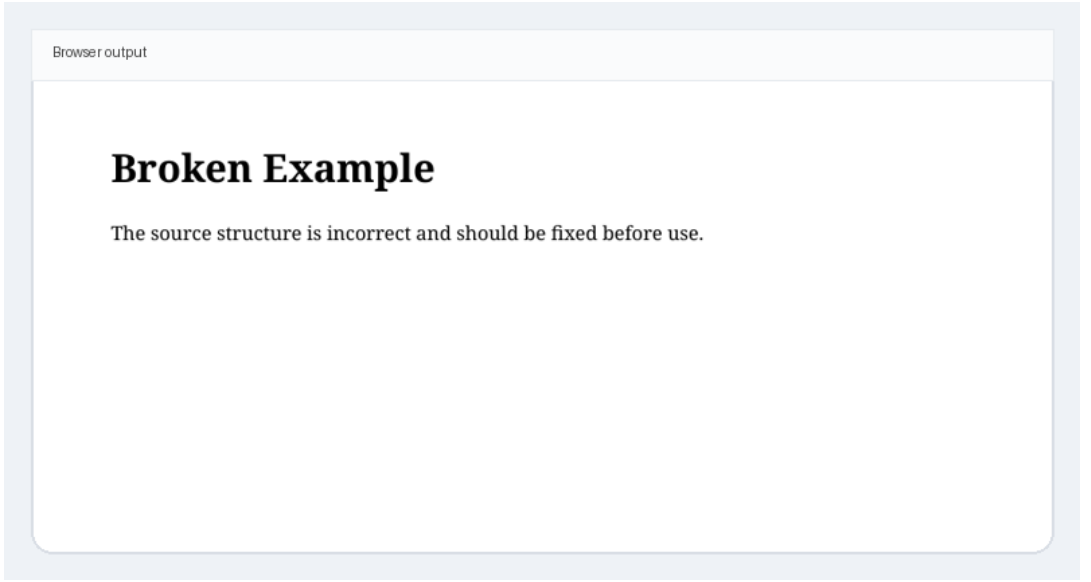
What to notice: This code is intentionally broken so students can diagnose the issue. • Debugging means checking both structure and element meaning.

Code

HTML

```
 1  <h1>Broken Example</h1>
 2  <p>This paragraph starts
 3  <ul>
 4    <li>but a list is incorrectly placed inside
it</li>
 5  </ul>
```

Rendered output



Mini Projects and Debugging: Common mistakes

- Validate the document when the page behaves unexpectedly.
- Read error messages carefully; many problems come from one missing tag.
- Check file paths and alt text during every review.
- Use mini projects to practice combining multiple HTML concepts naturally.

Mini Projects and Debugging: Recap

- HTML learning becomes real through complete mini pages.
- Debugging is easier when the source is semantic and well indented.
- By this point, the class has a strong HTML foundation without mixing CSS.
- The next stage is styling that structure with CSS.

Web Programming

Part 2 • CSS Unit 01

CSS Basics and Rule Syntax

- CSS controls presentation, while HTML controls structure and meaning.
- A CSS rule has a selector and a declaration block.

Fenerbahce University

CSS Basics and Rule Syntax: Key ideas

- CSS controls presentation, while HTML controls structure and meaning.
- A CSS rule has a selector and a declaration block.
- Declarations use property: value pairs.
- Styles can be inline, internal, or external, but external files scale best.

CSS Basics and Rule Syntax: Practical notes

- Start with clear selectors and a small set of properties.
- The cascade decides which rule wins when multiple rules match.
- Keep styles readable with one declaration per line.
- Class-based styling is usually more reusable than styling by id.

First CSS rule

CSS Unit 01 • Example

What to notice: The selector is h1 and the declaration changes its color. • Only matching elements are affected by the rule.

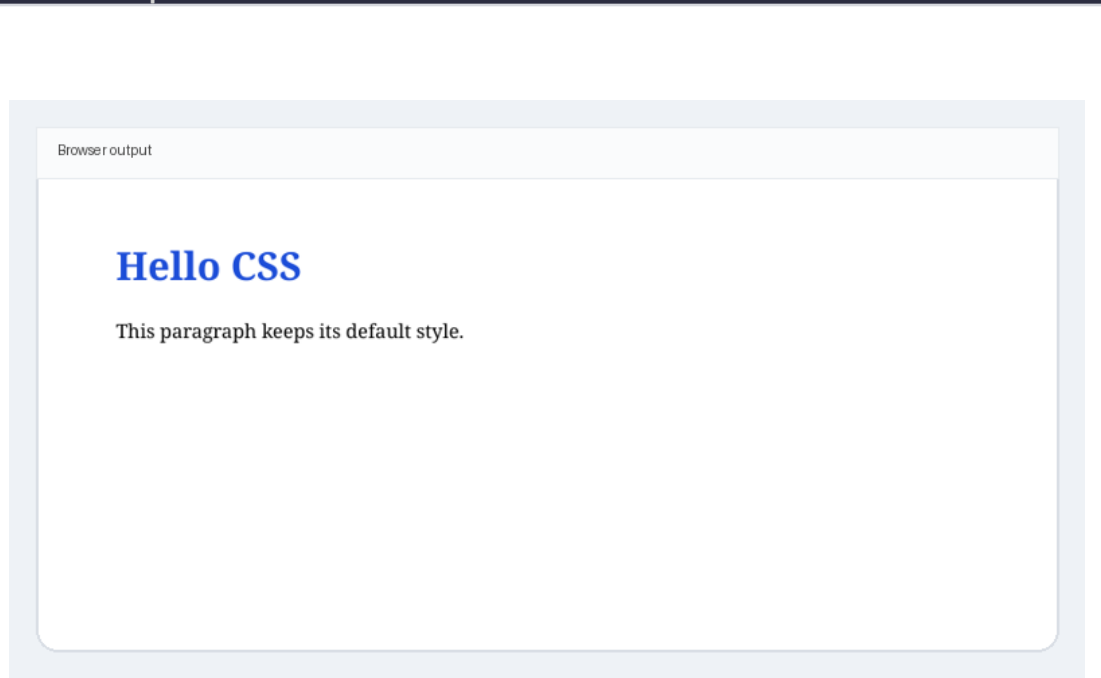
Code

HTML

```
<style>
  h1 {
    color: #1d4ed8;
  }
</style>

<h1>Hello CSS</h1>
<p>This paragraph keeps its default style.</p>
```

Rendered output



Styling a class

CSS Unit 01 • Example

What to notice: A class selector starts with a dot. • Classes can be reused on many elements.

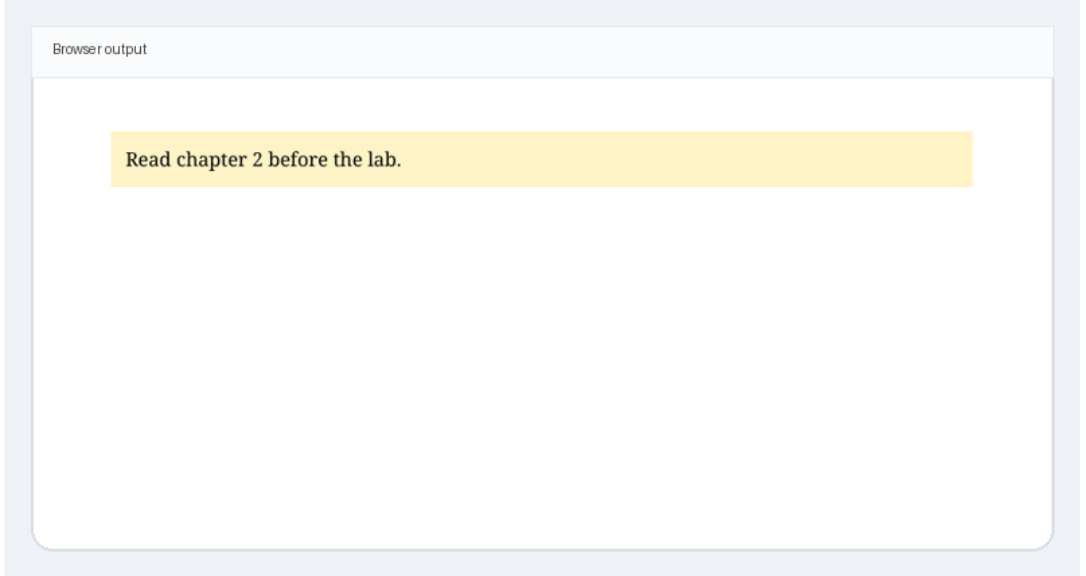
Code

HTML

```
<style>
  .note {
    background: #fef3c7;
    padding: 12px;
  }
</style>

<p class="note">Read chapter 2 before the lab.</p>
```

Rendered output



Multiple declarations in one rule

CSS Unit 01 • Example

What to notice: One selector can define several visual properties at once. • CSS reads each declaration independently inside the block.

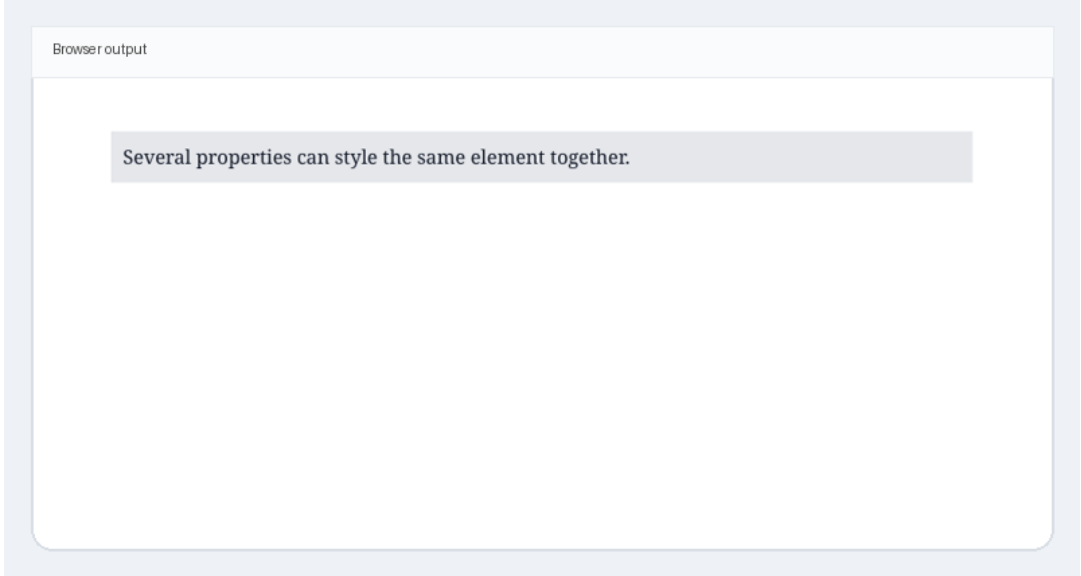
Code

HTML

```
<style>
  p {
    color: #111827;
    background: #e5e7eb;
    padding: 10px;
  }
</style>

<p>Several properties can style the same element together.</p>
```

Rendered output



Grouping selectors

CSS Unit 01 • Example

What to notice: A comma lets multiple selectors share the same declarations. • Grouping avoids repeated code.

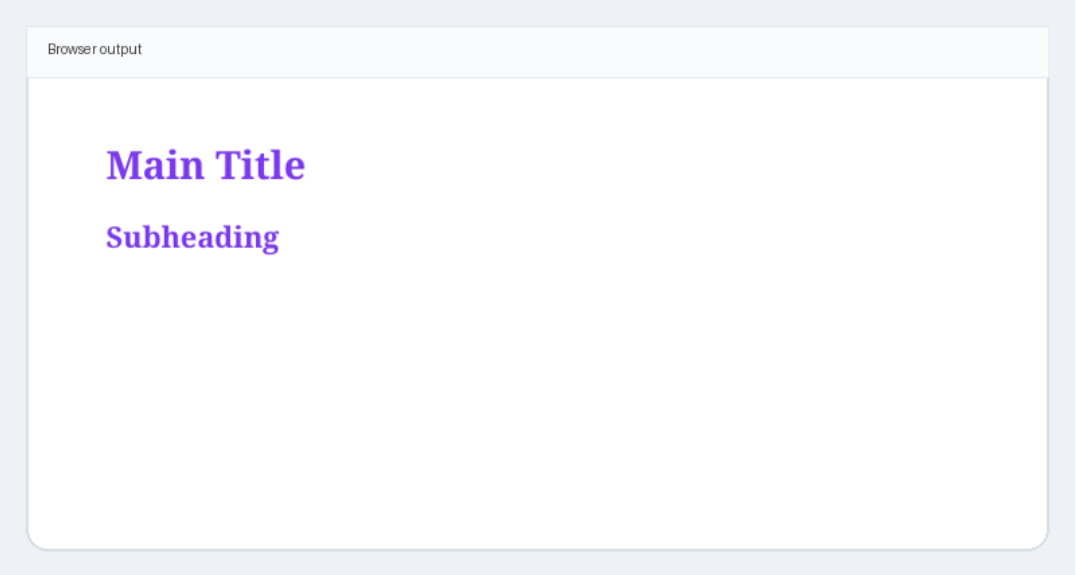
Code

HTML

```
<style>
  h1, h2 {
    color: #7c3aed;
  }
</style>

<h1>Main Title</h1>
<h2>Subheading</h2>
```

Rendered output



Inline style example

CSS Unit 01 • Example

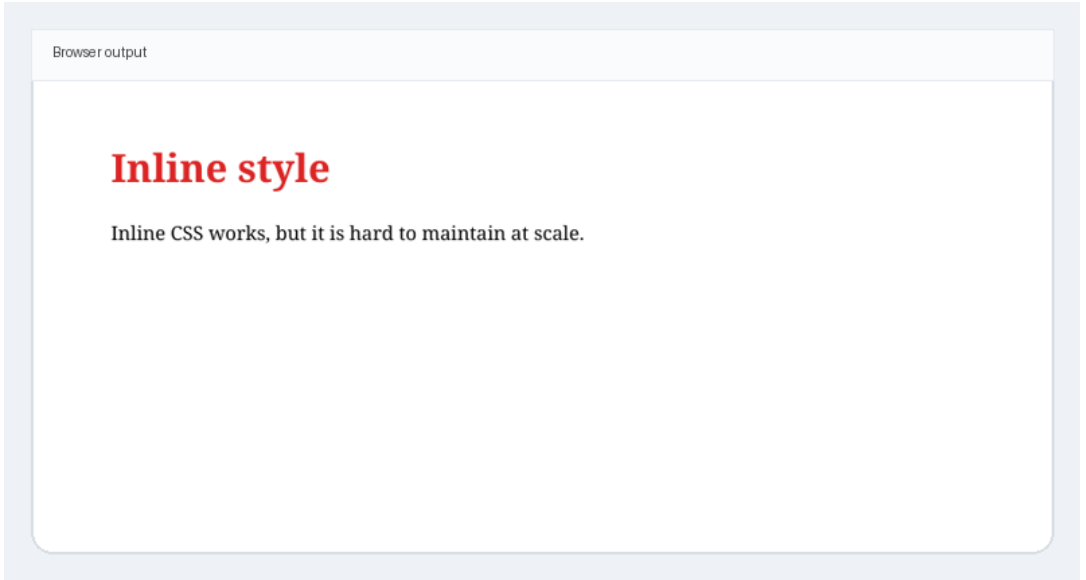
What to notice: Inline style applies directly to one element. • It is useful for tiny demos, but not for larger projects.

Code

HTML

```
 1  <h1 style="color: #dc2626;">Inline style</h1>
 2  <p>Inline CSS works, but it is hard to maintain at
scale.</p>
```

Rendered output



CSS Basics and Rule Syntax: Common mistakes

- Do not mix many inline styles into real projects.
- Do not forget semicolons when writing multiple declarations.
- Do not choose selectors so generic that they style everything accidentally.
- Write CSS with readability and reuse in mind from the beginning.

CSS Basics and Rule Syntax: Recap

- CSS adds visual design to the structure built in HTML.
- A rule combines a selector with one or more declarations.
- Classes and grouped selectors support reuse.
- Clean syntax is the first step toward maintainable stylesheets.

Web Programming

CSS Unit 02

Selectors and Specificity

- Selectors decide which elements a rule targets.
- Common selectors include element, class, id, descendant, child, sibling, and attribute selectors.

Fenerbahce University

Selectors and Specificity: Key ideas

- Selectors decide which elements a rule targets.
- Common selectors include element, class, id, descendant, child, sibling, and attribute selectors.
- Specificity helps resolve conflicts when multiple rules apply.
- Clear selectors reduce bugs and unexpected styling.

Selectors and Specificity: Practical notes

- Element selectors are broad; class selectors are more targeted.
- Descendant selectors match nested elements anywhere inside an ancestor.
- Child selectors match only the direct children.
- Specificity should be managed intentionally, not fought with random !important.

Descendant selector

CSS Unit 02 • Example

What to notice: Only paragraphs inside article match the selector. • Selectors can describe structural relationships, not only single tags.

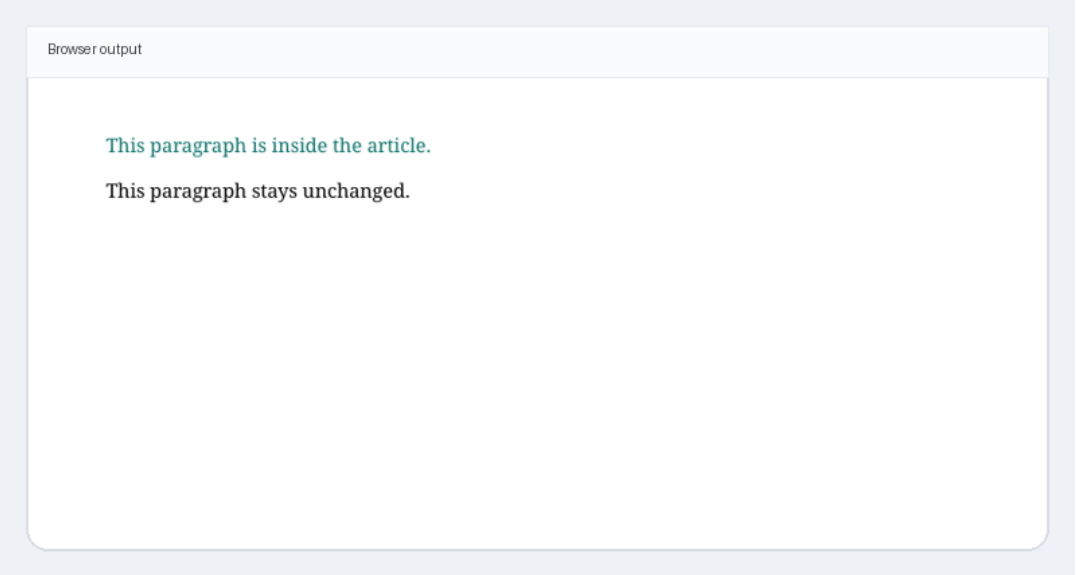
Code

HTML

```
<style>
  article p {
    color: #0f766e;
  }
</style>

<article>
  <p>This paragraph is inside the article.</p>
</article>
<p>This paragraph stays unchanged.</p>
```

Rendered output



Child selector

CSS Unit 02 • Example

What to notice: The > combinator targets only direct children. • Nested descendants do not match the same way.

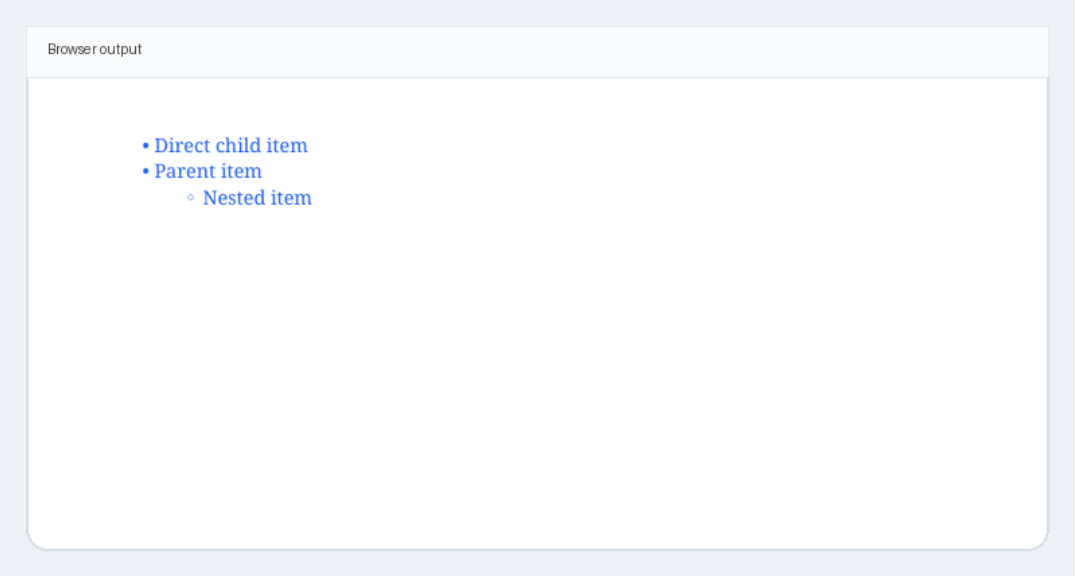
Code

HTML

```
<style>
  ul > li {
    color: #2563eb;
  }
</style>

<ul>
  <li>Direct child item</li>
  <li>
    Parent item
    <ul><li>Nested item</li></ul>
  </li>
</ul>
```

Rendered output



Attribute selector

CSS Unit 02 • Example

What to notice: Attribute selectors target elements based on attribute values. • This is useful for form styling and reusable patterns.

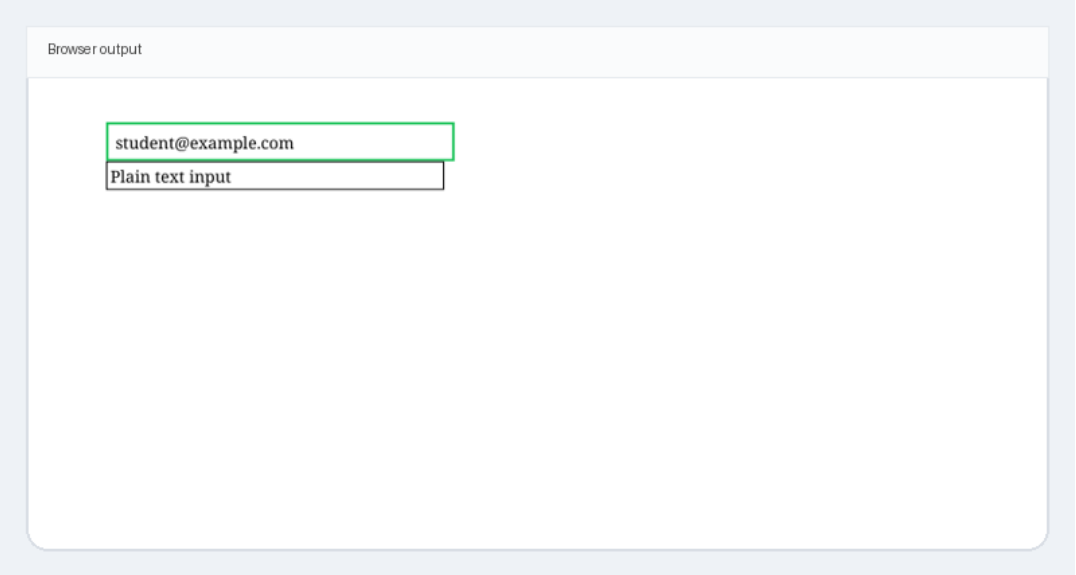
Code

HTML

```
<style>
  input[type="email"] {
    border: 2px solid #22c55e;
    padding: 6px;
  }
</style>

<input type="email" value="student@example.com">
<input type="text" value="Plain text input">
```

Rendered output



Sibling selector

CSS Unit 02 • Example

What to notice: The adjacent sibling selector matches the next sibling only. • This is useful for local patterns such as intro paragraphs.

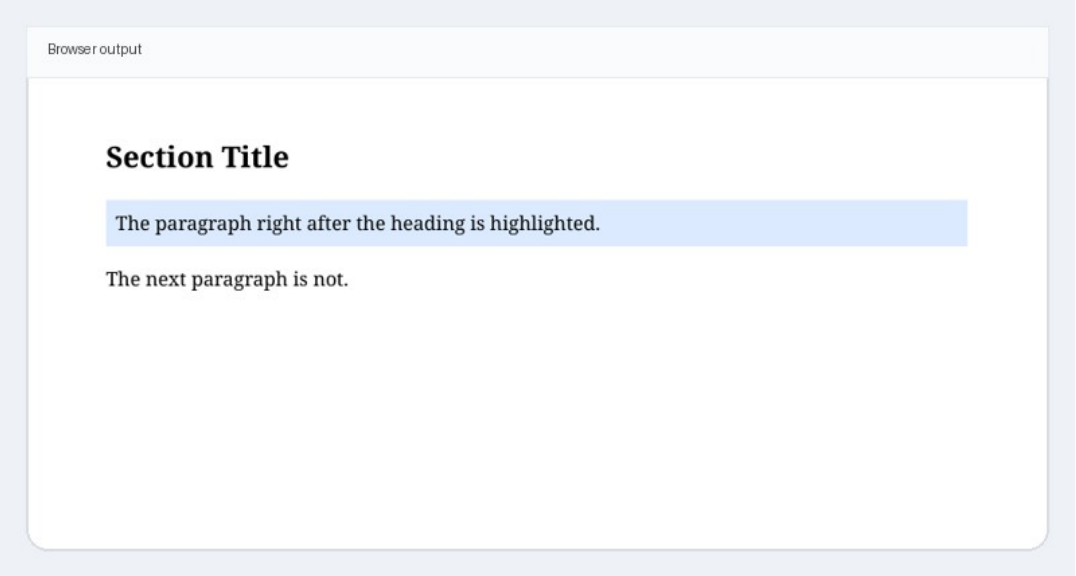
Code

HTML

```
<style>
  h2 + p {
    background: #dbeafe;
    padding: 8px;
  }
</style>

<h2>Section Title</h2>
<p>The paragraph right after the heading is highlighted.</p>
<p>The next paragraph is not.</p>
```

Rendered output



Specificity example

CSS Unit 02 • Example

What to notice: A class selector is more specific than a plain element selector. • Understanding this prevents many styling surprises.

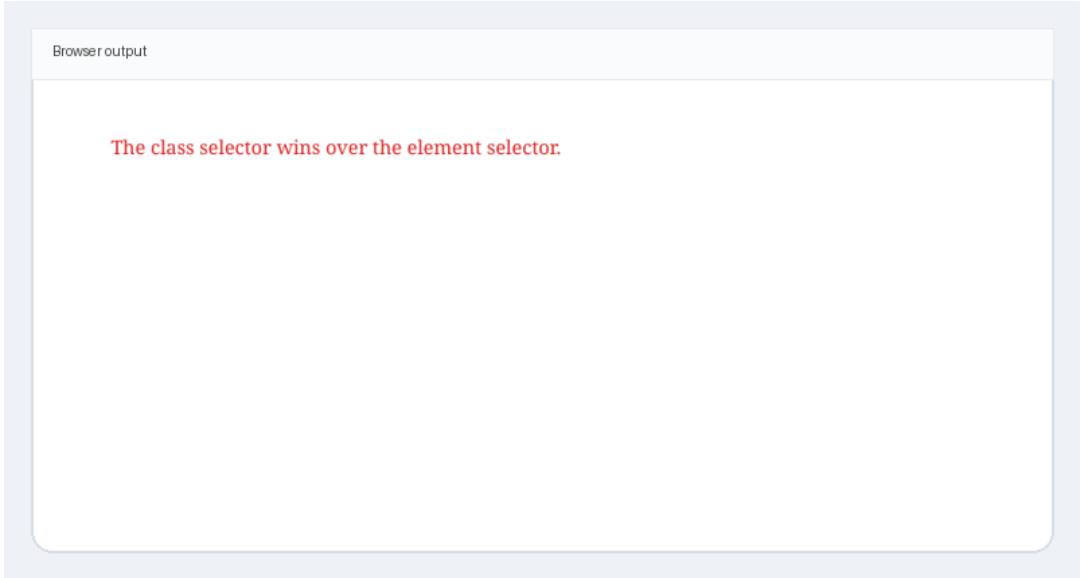
Code

HTML

```
<style>
  p { color: #6b7280; }
  .highlight { color: #dc2626; }
</style>

<p class="highlight">The class selector wins over the element selector.</p>
```

Rendered output



Selectors and Specificity: Common mistakes

- Do not write selectors that depend on a fragile deep nesting chain.
- Do not overuse id selectors when classes would be more reusable.
- Do not jump to !important before checking specificity.
- Prefer selectors that are easy to read and reason about.

Selectors and Specificity: Recap

- Selectors can express identity, attributes, and relationships.
- Specificity explains why one matching rule wins over another.
- Broad selectors are fast to write but risky at scale.
- Smart selector choices keep a stylesheet calm and predictable.

Web Programming

CSS Unit 03

Colors, Backgrounds, and Units

- CSS can express color with named values, hex, rgb(), and hsl().
- Backgrounds can use solid colors, images, and gradients.

Fenerbahce University

Colors, Backgrounds, and Units: Key ideas

- CSS can express color with named values, hex, rgb(), and hsl().
- Backgrounds can use solid colors, images, and gradients.
- Units such as px, %, em, rem, vw, and vh describe size in different ways.
- Good visual systems use a small, consistent set of values.

Colors, Backgrounds, and Units: Practical notes

- Hex and rgb are common for exact color definitions.
- Relative units such as rem and % adapt better than fixed units in many cases.
- Background color affects the element box, not only the text.
- Width and spacing decisions should match the design intent.

Different text colors

CSS Unit 03 • Example

What to notice: CSS offers several equivalent ways to define color. • Choose one system and keep it consistent within the project.

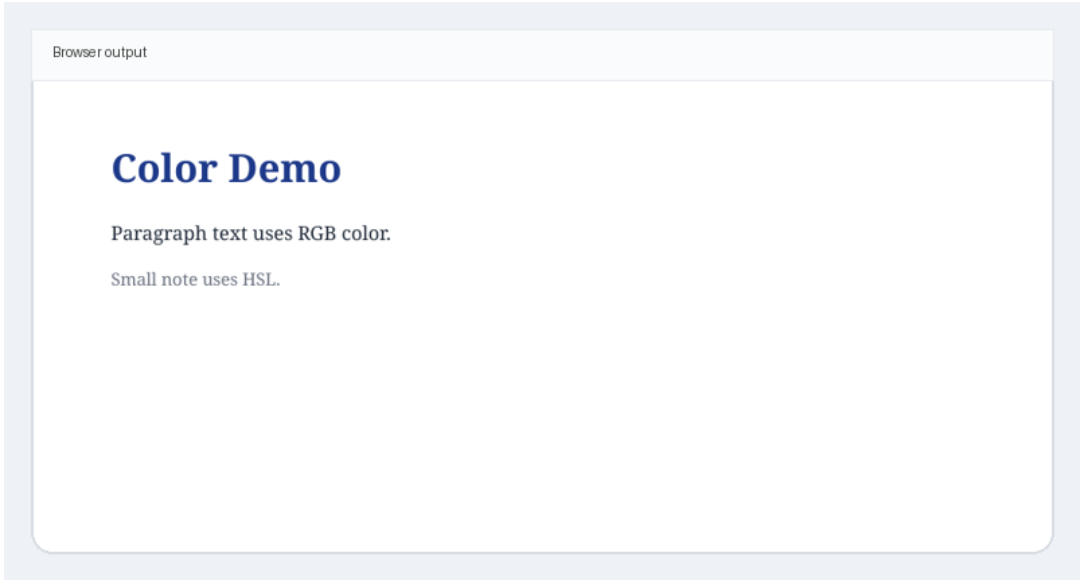
Code

HTML

```
<style>
  h1 { color: #1e3a8a; }
  p { color: rgb(31, 41, 55); }
  small { color: hsl(220, 10%, 45%); }
</style>

<h1>Color Demo</h1>
<p>Paragraph text uses RGB color.</p>
<small>Small note uses HSL.</small>
```

Rendered output



Background color and border

CSS Unit 03 • Example

What to notice: Background color helps create visual grouping. • A border can separate the card from the page.

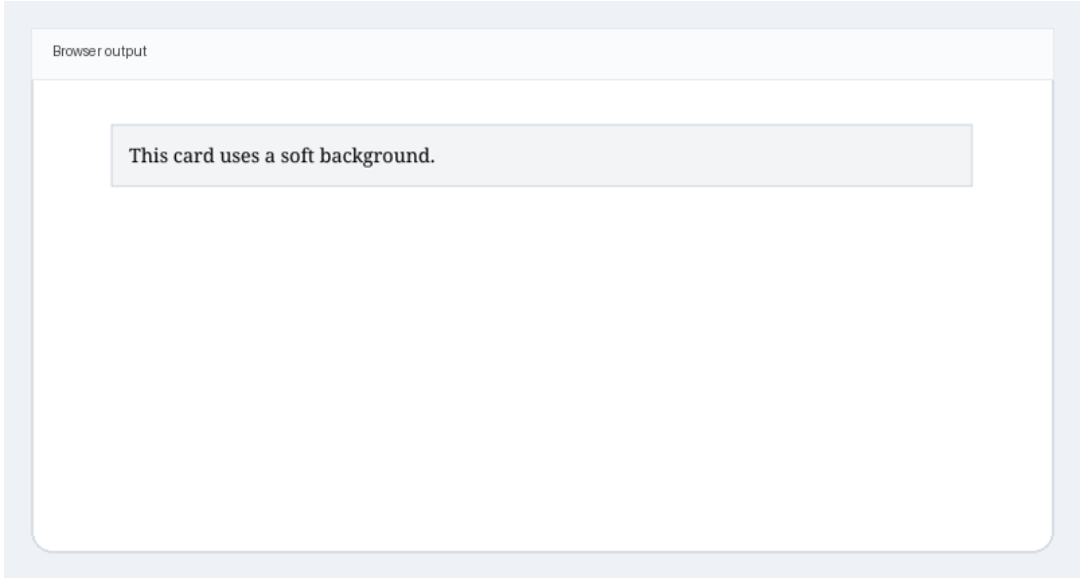
Code

HTML

```
<style>
  .card {
    background: #f3f4f6;
    border: 1px solid #cbd5e1;
    padding: 14px;
  }
</style>

<div class="card">This card uses a soft background.</div>
```

Rendered output



Linear gradient background

CSS Unit 03 • Example

What to notice: Gradients smoothly blend multiple colors. • They are backgrounds, not text effects.

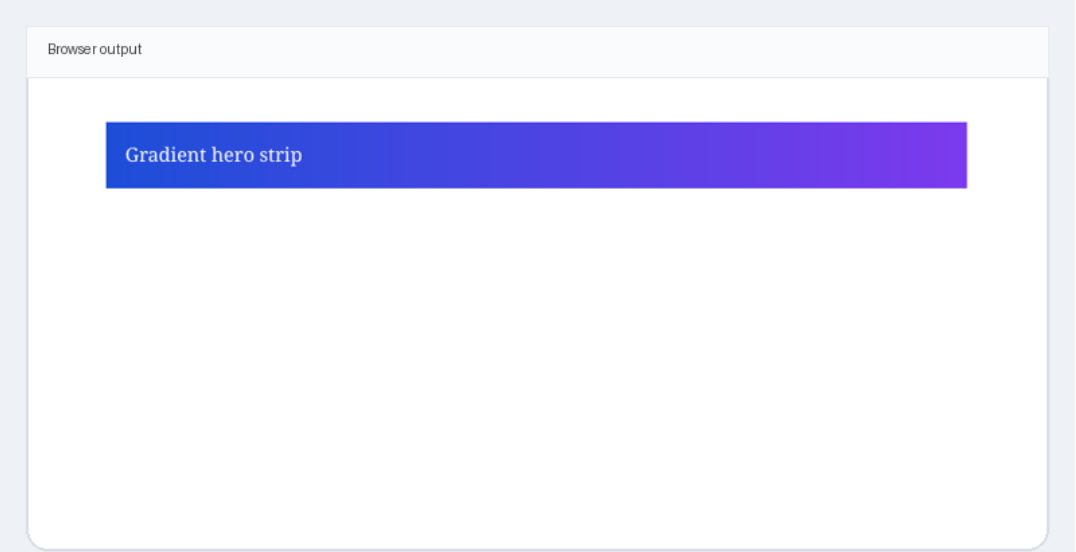
Code

HTML

```
<style>
  .banner {
    background: linear-gradient(90deg, #1d4ed8, #7c3aed);
    color: white;
    padding: 16px;
  }
</style>

<div class="banner">Gradient hero strip</div>
```

Rendered output



Relative width with percentage

CSS Unit 03 • Example

What to notice: Percentage width depends on the width of the parent container. • Relative sizing is useful for adaptable layouts.

Code

HTML

```
<style>
  .box {
    width: 60%;
    background: #dbeafe;
    padding: 12px;
  }
</style>

<div class="box">This box uses a percentage width.</div>
```

Rendered output

rem versus px

CSS Unit 03 • Example

What to notice: px is fixed; rem scales from the root font size. • Relative units can support more flexible design systems.

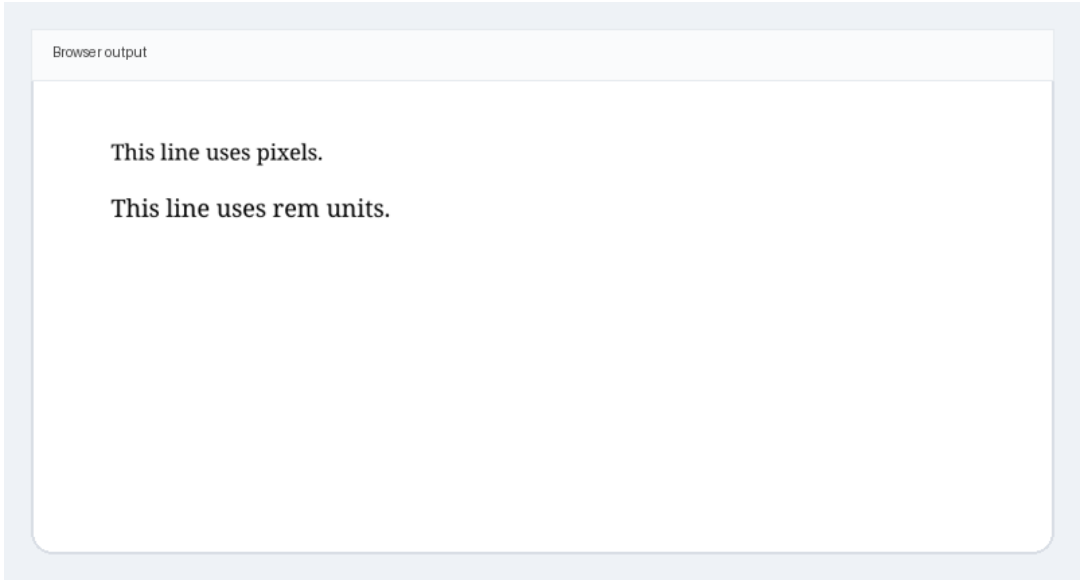
Code

HTML

```
<style>
  .px-size { font-size: 18px; }
  .rem-size { font-size: 1.25rem; }
</style>

<p class="px-size">This line uses pixels.</p>
<p class="rem-size">This line uses rem units.</p>
```

Rendered output



Colors, Backgrounds, and Units: Common mistakes

- Do not use too many unrelated colors without a system.
- Do not mix many units randomly unless you know why.
- Do not rely on background color alone for meaning or state.
- Start simple and document the palette and unit logic.

Colors, Backgrounds, and Units: Recap

- Color and units define much of a site’s visual rhythm.
- Backgrounds shape containers and emphasis.
- Relative units are powerful for scalable design.
- Consistency matters more than flashy variety.

Web Programming

CSS Unit 04

Typography and Text Styling

- Typography controls how text feels, not only how it looks.
- Common properties include font-family, font-size, font-weight, line-height, text-align, and letter-spacing.

Fenerbahce University

Typography and Text Styling: Key ideas

- Typography controls how text feels, not only how it looks.
- Common properties include font-family, font-size, font-weight, line-height, text-align, and letter-spacing.
- Readable text relies on hierarchy, spacing, and contrast.
- Good typography is one of the strongest differences between an amateur and polished page.

Typography and Text Styling: Practical notes

- Use font stacks so the browser can fall back gracefully.
- Line height affects readability as much as font size.
- Text transformations should support the message, not overpower it.
- Long paragraphs benefit from comfortable line spacing and measure.

Font family and weight

CSS Unit 04 • Example

What to notice: Different font families create different tones and roles. • Weight helps establish hierarchy between title and body text.

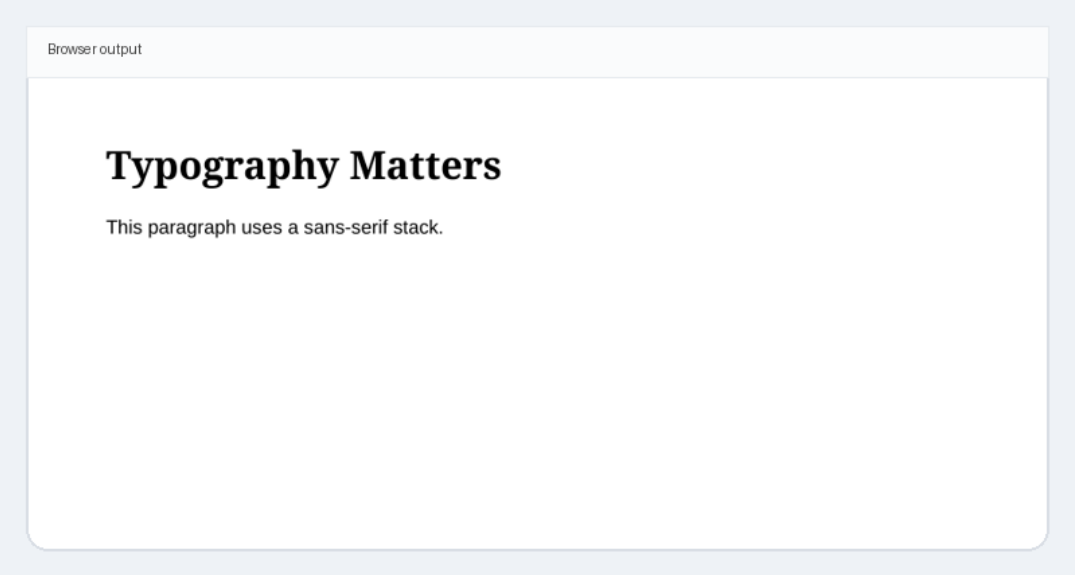
Code

HTML

```
<style>
  h1 {
    font-family: Georgia, serif;
    font-weight: 700;
  }
  p {
    font-family: Arial, sans-serif;
  }
</style>

<h1>Typography Matters</h1>
<p>This paragraph uses a sans-serif stack.</p>
```

Rendered output



Line height for readability

CSS Unit 04 • Example

What to notice: Line height creates breathing room between text lines. • Dense paragraphs feel harder to read even with the same font size.

Code

HTML

```
<style>
  p {
    width: 300px;
    line-height: 1.7;
  }
</style>

<p>Comfortable line spacing makes a paragraph easier to scan and read over several lines.</p>
```

Rendered output

Browser output

Comfortable line spacing makes a paragraph easier to scan and read over several lines.

Alignment and letter spacing

CSS Unit 04 • Example

What to notice: Alignment changes the visual rhythm of a block of text. • Letter spacing is useful in moderation, especially for headings.

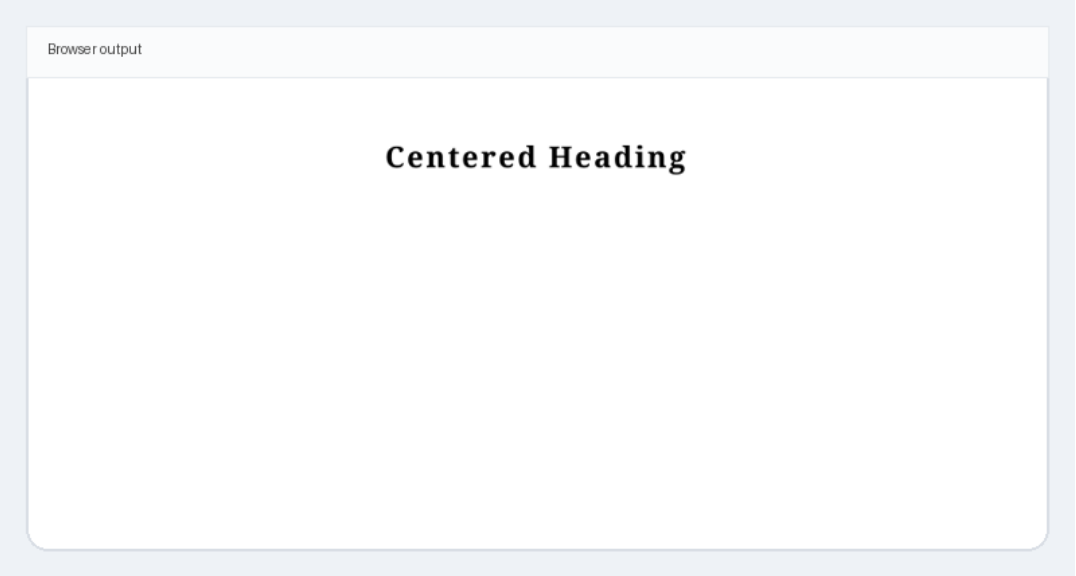
Code

HTML

```
<style>
  h2 {
    text-align: center;
    letter-spacing: 2px;
  }
</style>

<h2>Centered Heading</h2>
```

Rendered output



Text transform and decoration

CSS Unit 04 • Example

What to notice: Transform changes how text is displayed without changing the HTML content. • Decoration can add emphasis, but it should remain readable.

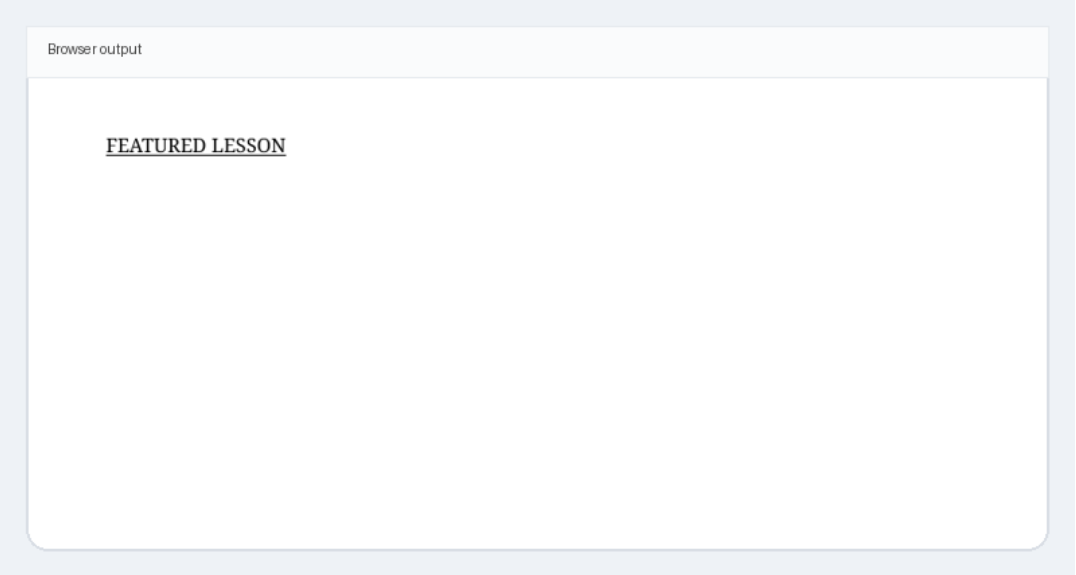
Code

HTML

```
<style>
  .label {
    text-transform: uppercase;
    text-decoration: underline;
  }
</style>

<p class="label">featured lesson</p>
```

Rendered output



Indented quote block

CSS Unit 04 • Example

What to notice: Typography and spacing can give quoted text a distinct tone. • Simple adjustments often create strong visual meaning.

Code

HTML

```
<style>
  blockquote {
    font-style: italic;
    border-left: 4px solid #cbd5e1;
    padding-left: 12px;
  }
</style>

<blockquote>Design is intelligence made visible.</blockquote>
```

Rendered output

Browser output

> *Design is intelligence made visible.*

Typography and Text Styling: Common mistakes

- Do not use too many font families in one page.
- Do not reduce line-height until paragraphs feel cramped.
- Do not force uppercase on long blocks of body text.
- Typography choices should support content hierarchy and readability.

Typography and Text Styling: Recap

- Readable typography depends on several small properties working together.
- Hierarchy comes from size, weight, spacing, and alignment.
- Body text deserves as much care as headings.
- Good type design makes content easier to trust and understand.

Web Programming

CSS Unit 05

The Box Model

- Every element is treated as a rectangular box in CSS layout.
- The box model includes content, padding, border, and margin.

Fenerbahce University

The Box Model: Key ideas

- Every element is treated as a rectangular box in CSS layout.
- The box model includes content, padding, border, and margin.
- Padding adds inner space; margin adds outer space.
- box-sizing controls how width and height are calculated.

The Box Model: Practical notes

- A background fills the content and padding areas by default.
- Margin creates distance from surrounding elements.
- Borders help reveal the actual edges of a box.
- Many layout bugs come from misunderstanding box sizing.

Padding and border

CSS Unit 05 • Example

What to notice: Padding creates breathing room between content and border. • The border marks the visible edge of the element.

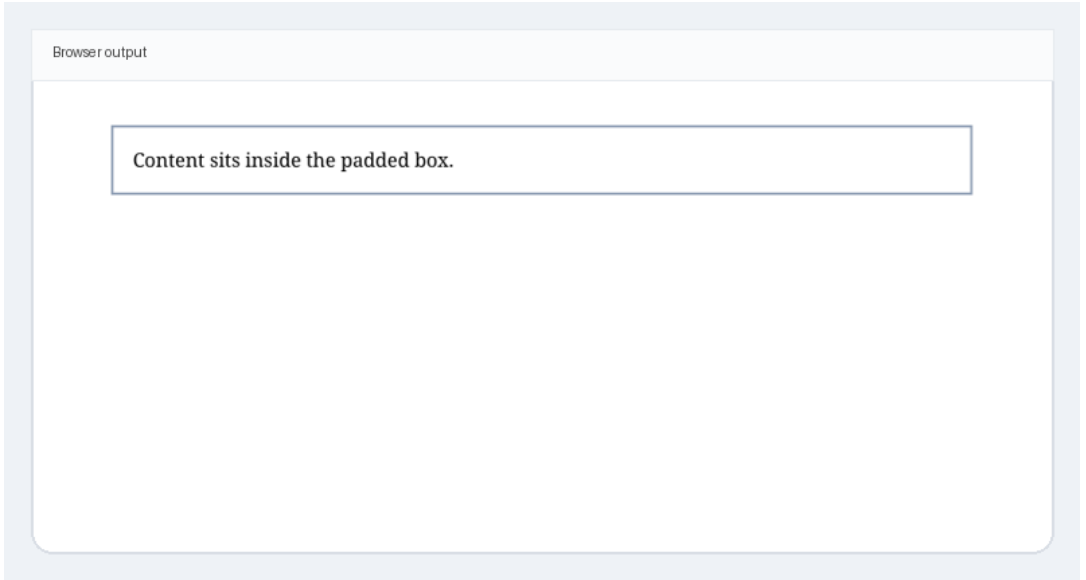
Code

HTML

```
<style>
  .panel {
    border: 2px solid #94a3b8;
    padding: 16px;
  }
</style>

<div class="panel">Content sits inside the padded box.</div>
```

Rendered output



Margin between boxes

CSS Unit 05 • Example

What to notice: Margin separates one box from the next. • This is different from padding, which acts inside the box.

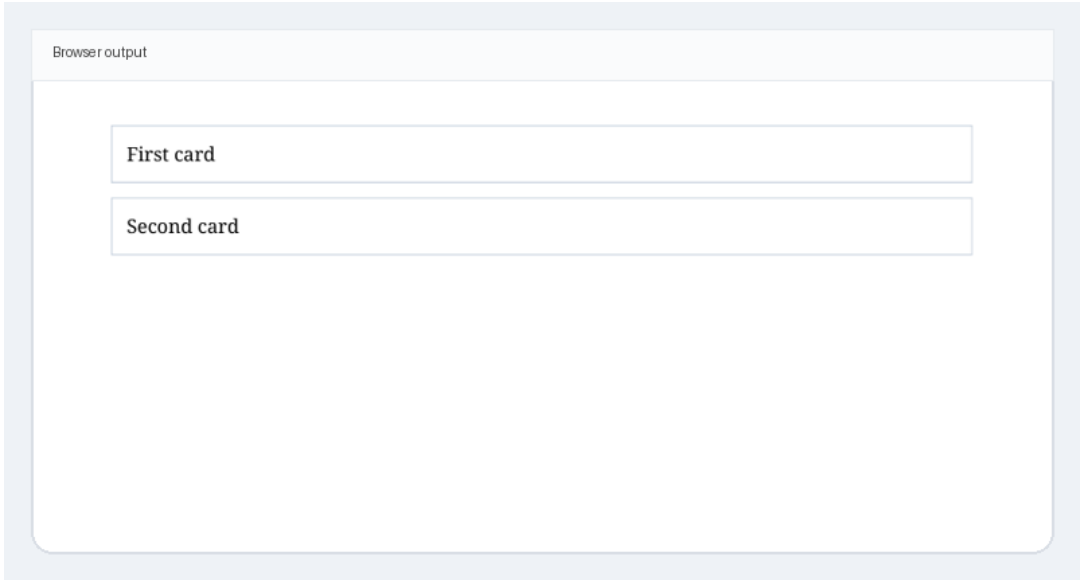
Code

HTML

```html
<style>
  .card {
    border: 1px solid #cbd5e1;
    padding: 12px;
    margin-bottom: 12px;
  }
</style>

<div class="card">First card</div>
<div class="card">Second card</div>
```

Rendered output



box-sizing border-box

CSS Unit 05 • Example

What to notice: border-box makes width include padding and border. • This often makes layout math easier and more predictable.

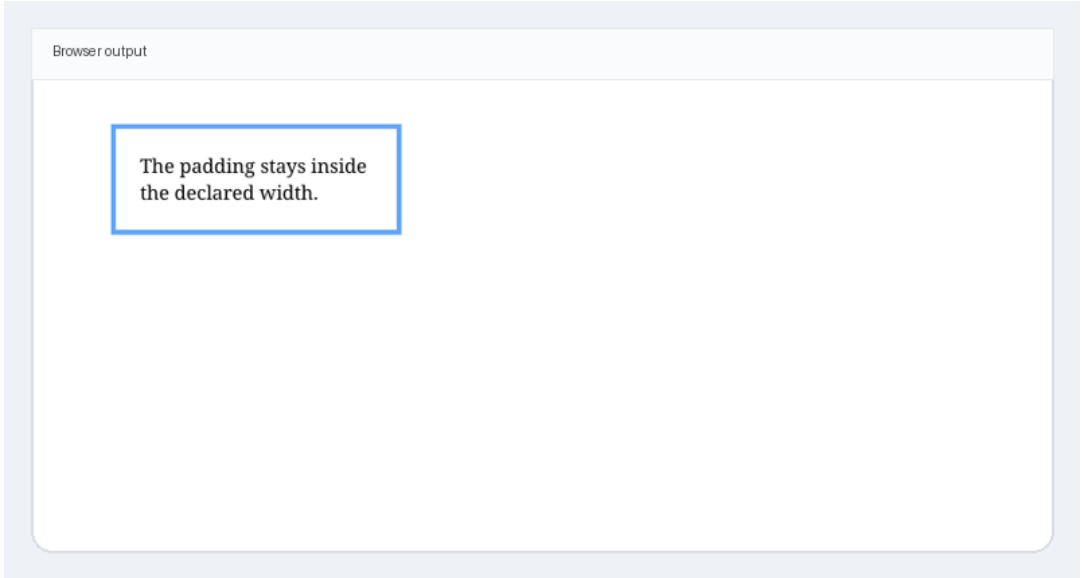
Code

HTML

```
<style>
  .item {
    width: 240px;
    padding: 20px;
    border: 4px solid #60a5fa;
    box-sizing: border-box;
  }
</style>

<div class="item">The padding stays inside the declared width.</div>
```

Rendered output



Rounded corners and outline

CSS Unit 05 • Example

What to notice: Radius changes the corner shape of the box. • Outline sits outside the border and is useful for focus styles.

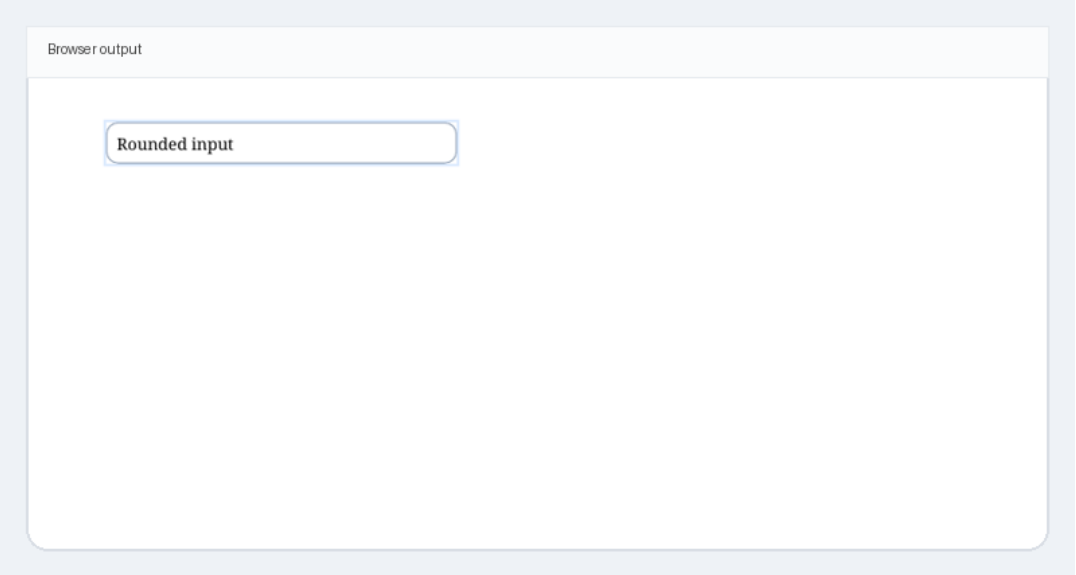
Code

HTML

```html
<style>
  input {
    padding: 8px;
    border: 1px solid #94a3b8;
    border-radius: 10px;
    outline: 2px solid #dbeafe;
  }
</style>

<input value="Rounded input">
```

Rendered output



Overflow handling

CSS Unit 05 • Example

What to notice: Overflow controls what happens when content exceeds the box size. • auto adds scrolling only when needed.

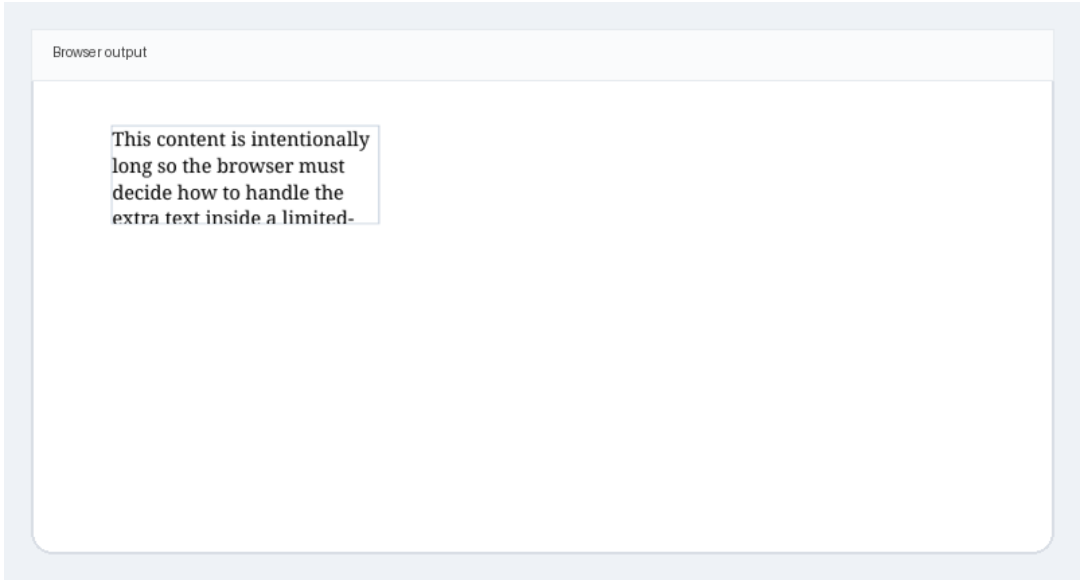
Code

HTML

```
<style>
  .box {
    width: 220px;
    height: 80px;
    overflow: auto;
    border: 1px solid #cbd5e1;
  }
</style>

<div class="box">This content is intentionally long so the browser must decide how to handle the extra text inside a limited-height box.</div>
```

Rendered output



The Box Model: Common mistakes

- Do not confuse margin and padding—they solve different spacing problems.
- Do not forget that borders and padding can change total box size.
- Do not hide overflow unless cutting content is acceptable.
- Use box-sizing intentionally across the project.

The Box Model: Recap

- The box model is the foundation of CSS layout.
- Padding, border, and margin each control different layers of space.
- box-sizing can simplify sizing rules dramatically.
- Many layout systems become easier once the box model is clear.

Web Programming

CSS Unit 06

Display and Positioning

- Display controls how an element participates in layout.
- Common display values include block, inline, inline-block, flex, grid, and none.

Fenerbahce University

Display and Positioning: Key ideas

- Display controls how an element participates in layout.
- Common display values include block, inline, inline-block, flex, grid, and none.
- Positioning changes how elements are placed relative to normal flow.
- Important position values are static, relative, absolute, fixed, and sticky.

Display and Positioning: Practical notes

- Block elements usually start on a new line.
- Inline elements flow inside text and ignore width and height in many cases.
- relative positioning establishes a reference point for absolutely positioned children.
- Fixed elements stay attached to the viewport instead of the document flow.

Block and inline behavior

CSS Unit 06 • Example

What to notice: Block elements take their own line by default. • Inline elements sit next to each other inside the line flow.

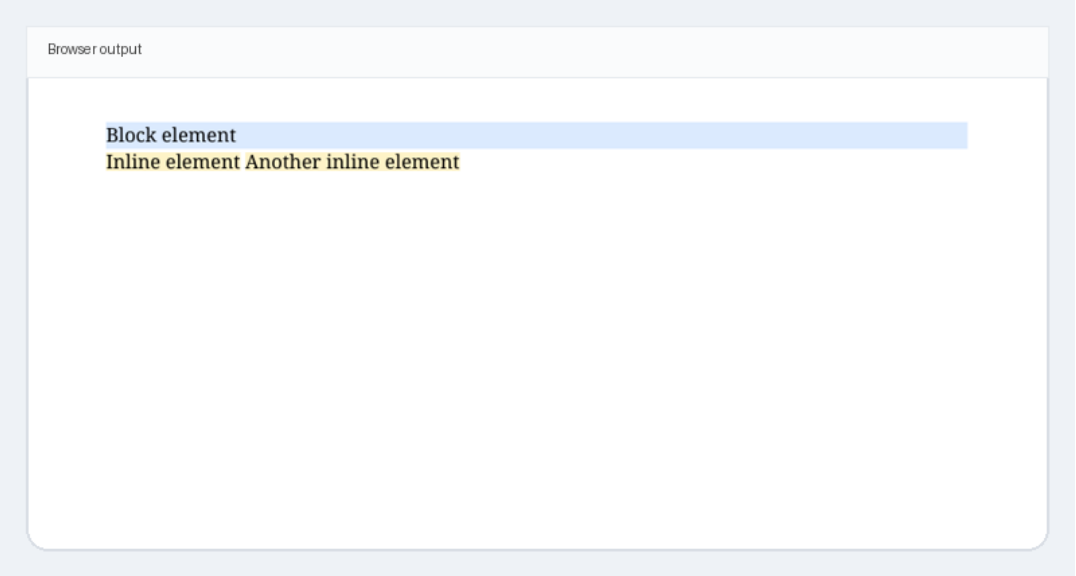
Code

HTML

```
<style>
  span {
    background: #fef3c7;
  }
  div {
    background: #dbeafe;
  }
</style>

<div>Block element</div>
<span>Inline element</span>
<span>Another inline element</span>
```

Rendered output



Inline-block cards

CSS Unit 06 • Example

What to notice: inline-block allows boxes to sit side by side while keeping dimensions. • It was a common layout tool before flexbox and grid.

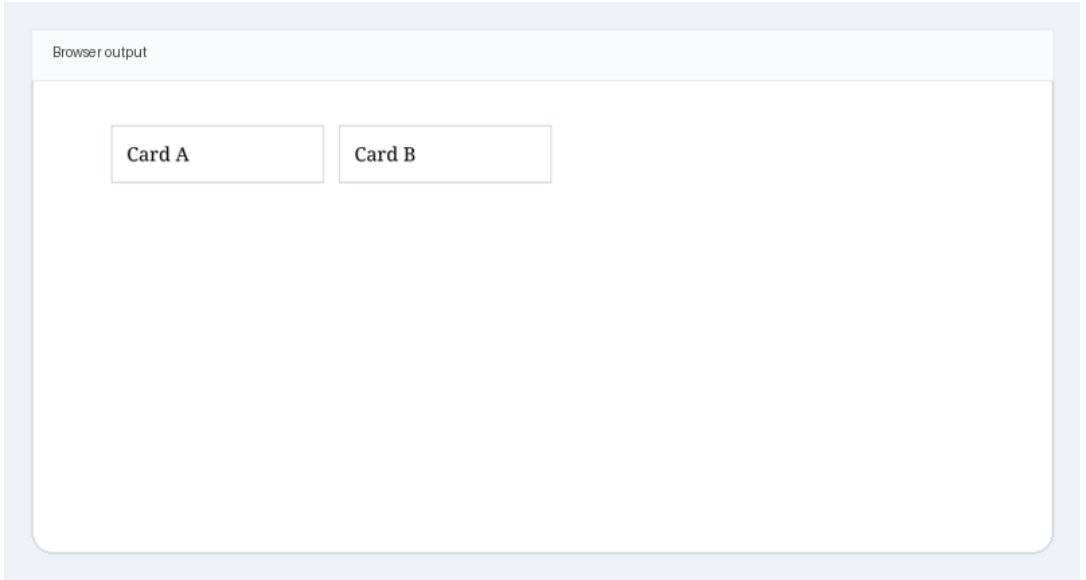
Code

HTML

```
<style>
  .card {
    display: inline-block;
    width: 150px;
    padding: 12px;
    border: 1px solid #cbd5e1;
    margin-right: 8px;
  }
</style>

<div class="card">Card A</div>
<div class="card">Card B</div>
```

Rendered output



Relative and absolute positioning

CSS Unit 06 • Example

What to notice: Absolute positioning uses the nearest positioned ancestor as reference. • relative on the parent creates a stable anchor for the badge.

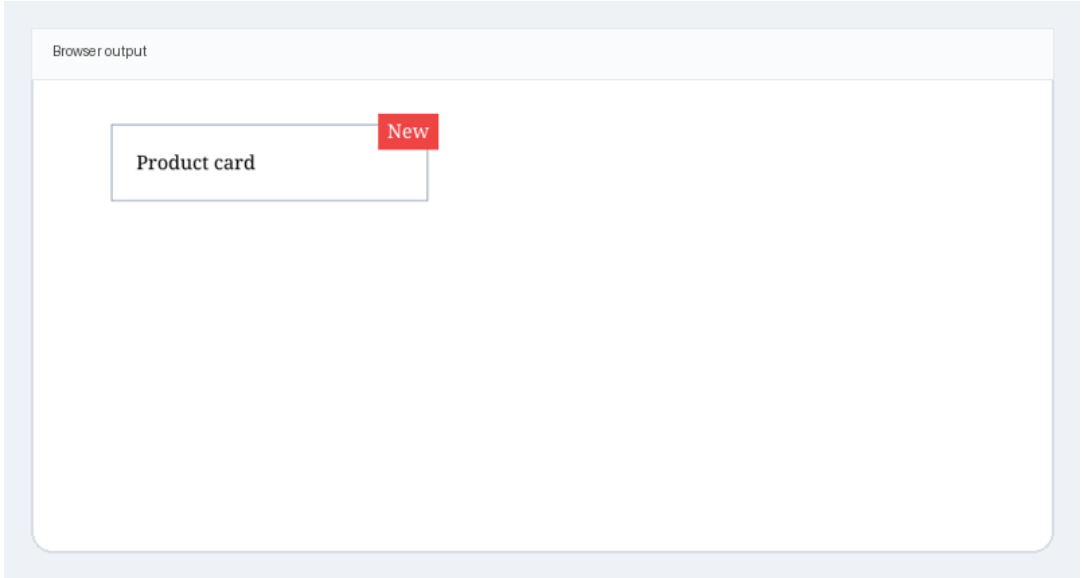
Code

HTML

```
<style>
  .box {
    position: relative;
    width: 220px;
    padding: 20px;
    border: 1px solid #94a3b8;
  }
  .badge {
    position: absolute;
    top: -10px;
    right: -10px;
    background: #ef4444;
    color: white;
    padding: 4px 8px;
  }
</style>

<div class="box">
  Product card
  <span class="badge">New</span>
</div>
```

Rendered output



Fixed notice

CSS Unit 06 • Example

What to notice: Fixed elements stay in the same viewport location while the page scrolls. • They are useful for alerts, chat buttons, and sticky actions.

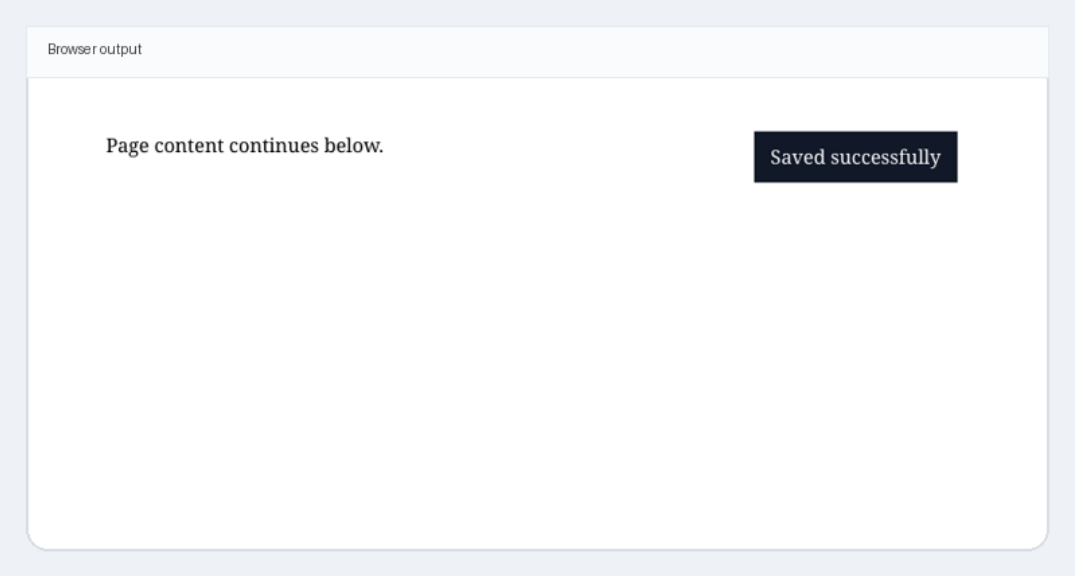
Code

HTML

```
<style>
  .notice {
    position: fixed;
    top: 16px;
    right: 16px;
    background: #111827;
    color: white;
    padding: 10px 14px;
  }
</style>

<div class="notice">Saved successfully</div>
<p>Page content continues below.</p>
```

Rendered output



Z-index overlap

CSS Unit 06 • Example

What to notice: z-index controls stacking order for positioned elements. • This matters whenever elements overlap visually.

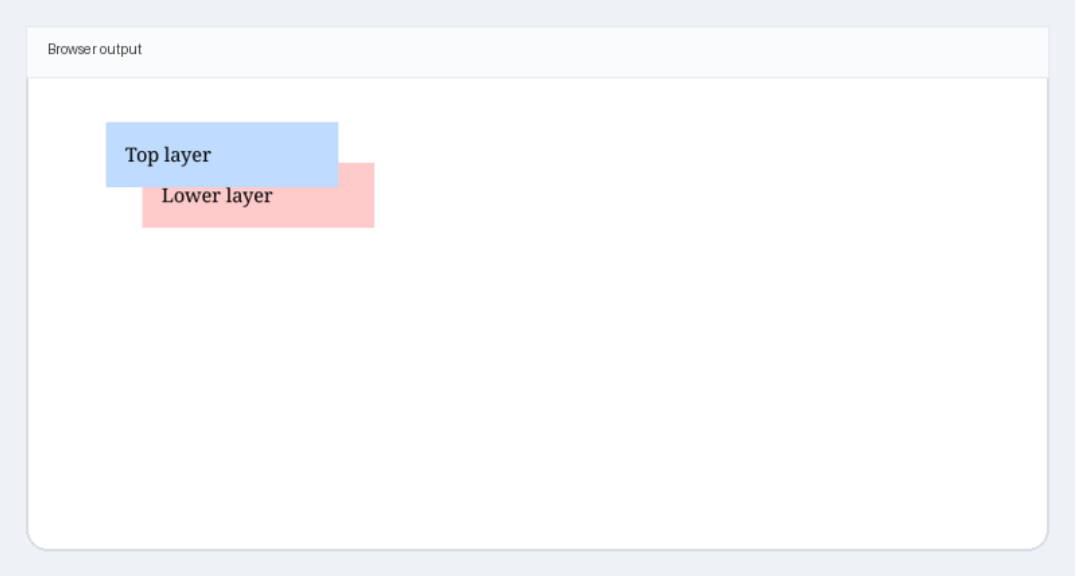
Code

HTML

```
<style>
  .one, .two {
    position: relative;
    width: 160px;
    padding: 16px;
  }
  .one {
    background: #bfdbfe;
    z-index: 2;
  }
  .two {
    background: #fecaca;
    margin-top: -20px;
    margin-left: 30px;
  }
</style>

<div class="one">Top layer</div>
<div class="two">Lower layer</div>
```

Rendered output



Display and Positioning: Common mistakes

- Do not remove elements from normal flow unless the design truly requires it.
- Do not use absolute positioning for every layout problem.
- Do not forget the containing block when placing badges or overlays.
- Choose the simplest display model that solves the layout need.

Display and Positioning: Recap

- Display and positioning shape how boxes behave on the page.
- Normal flow is the default and often the safest starting point.
- Positioned elements are powerful, but easy to misuse.
- Understanding flow prevents many alignment frustrations.

Web Programming

CSS Unit 07

Flexbox

- Flexbox is designed for one-dimensional layout: row or column.
- A flex container controls the arrangement of its direct children.

Fenerbahce University

Flexbox: Key ideas

- Flexbox is designed for one-dimensional layout: row or column.
- A flex container controls the arrangement of its direct children.
- Key properties include display: flex, flex-direction, justify-content, align-items, gap, and flex-wrap.
- Flexbox is excellent for navigation bars, cards, toolbars, and centering tasks.

Flexbox: Practical notes

- The main axis follows flex-direction; the cross axis is perpendicular.
- justify-content distributes items along the main axis.
- align-items controls alignment across the cross axis.
- gap creates spacing without extra margin hacks.

Simple horizontal menu

CSS Unit 07 • Example

What to notice: display: flex places the links in one horizontal row by default. • gap adds equal spacing between the items.

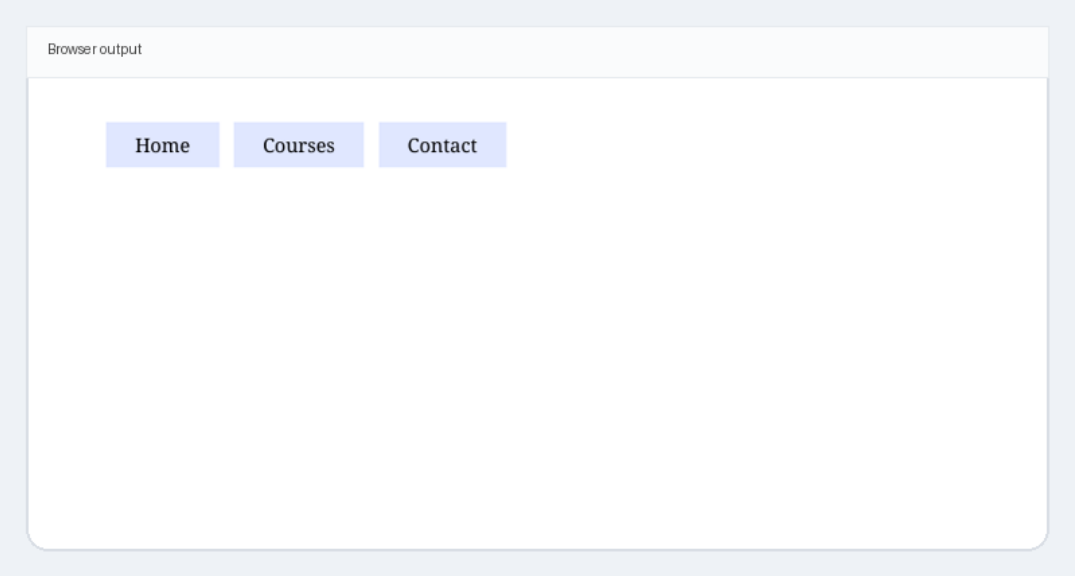
Code

HTML

```
<style>
  nav {
    display: flex;
    gap: 12px;
  }
  a {
    padding: 8px 12px;
    background: #e0e7ff;
  }
</style>

<nav>
  <a>Home</a>
  <a>Courses</a>
  <a>Contact</a>
</nav>
```

Rendered output



Centered card content

CSS Unit 07 • Example

What to notice: Flexbox makes both horizontal and vertical centering straightforward. • This is one of the most common everyday flex patterns.

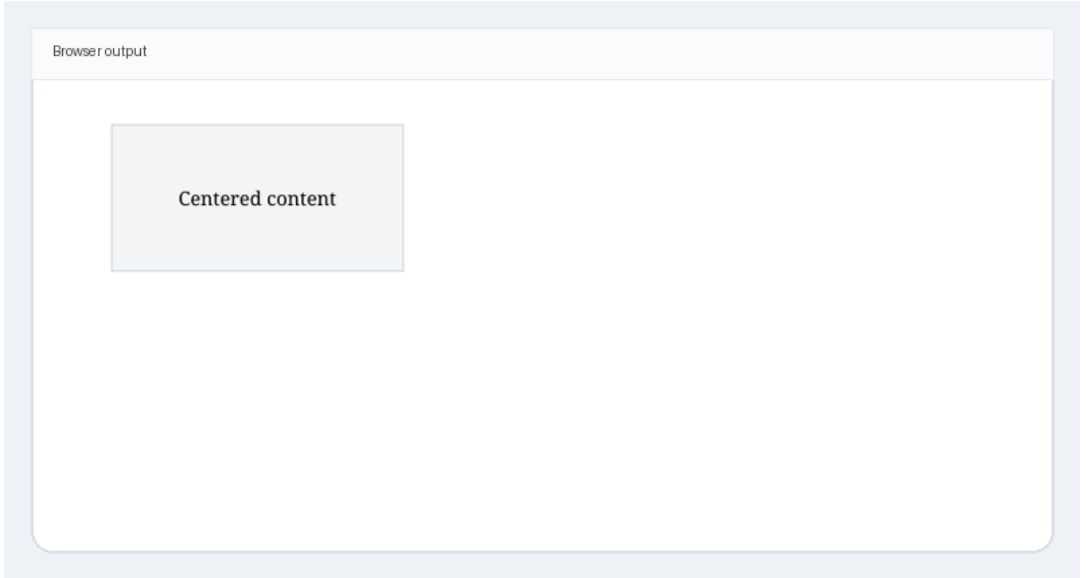
Code

HTML

```html
<style>
  .card {
    display: flex;
    justify-content: center;
    align-items: center;
    width: 240px;
    height: 120px;
    background: #f3f4f6;
    border: 1px solid #cbd5e1;
  }
</style>

<div class="card">Centered content</div>
```

Rendered output



Space-between toolbar

CSS Unit 07 • Example

What to notice: space-between pushes first and last items to opposite sides. • This pattern is common in toolbars and headers.

Code

HTML

```
<style>
  .toolbar {
    display: flex;
    justify-content: space-between;
    align-items: center;
    width: 360px;
    border: 1px solid #cbd5e1;
    padding: 10px;
  }
</style>

<div class="toolbar">
  <span>Logo</span>
  <button>Sign In</button>
</div>
```

Rendered output

Wrapping cards

CSS Unit 07 • Example

What to notice: flex-wrap allows items to move to a new line when space runs out. • This is useful for tags, cards, and responsive rows.

Code

HTML

```html
<style>
  .grid {
    display: flex;
    flex-wrap: wrap;
    gap: 10px;
    width: 340px;
  }
  .item {
    width: 100px;
    padding: 12px;
    background: #dbeafe;
  }
</style>

<div class="grid">
  <div class="item">A</div>
  <div class="item">B</div>
  <div class="item">C</div>
  <div class="item">D</div>
</div>
```

Rendered output

Flexible columns with grow

CSS Unit 07 • Example

What to notice: Flex growth values divide available space proportionally. • The main column receives twice as much free space as the sidebar.

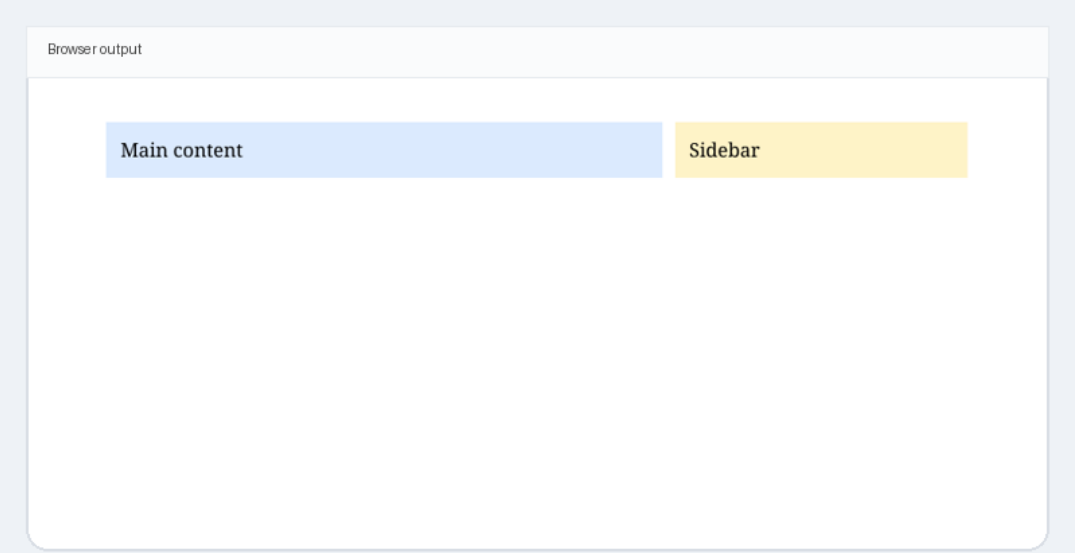
Code

HTML

```
<style>
  .layout {
    display: flex;
    gap: 10px;
  }
  .main {
    flex: 2;
    background: #dbeafe;
    padding: 12px;
  }
  .side {
    flex: 1;
    background: #fef3c7;
    padding: 12px;
  }
</style>

<div class="layout">
  <div class="main">Main content</div>
  <div class="side">Sidebar</div>
</div>
```

Rendered output



Flexbox: Common mistakes

- Do not use flexbox when you actually need a two-dimensional grid.
- Do not fight spacing with margins when gap already solves it cleanly.
- Do not forget that flex properties affect only direct children.
- Start by identifying the main axis before choosing alignment properties.

Flexbox: Recap

- Flexbox is the workhorse of modern one-dimensional layout.
- Alignment becomes much easier once the main and cross axes are clear.
- Gap and wrap reduce many older layout hacks.
- Flexbox is simple in syntax but powerful in practice.

Web Programming

CSS Unit 08

Grid

- CSS Grid is designed for two-dimensional layout: rows and columns together.
- A grid container defines tracks using grid-template-columns and grid-template-rows.

Fenerbahce University

Grid: Key ideas

- CSS Grid is designed for two-dimensional layout: rows and columns together.
- A grid container defines tracks using grid-template-columns and grid-template-rows.
- Grid excels at page shells, galleries, dashboards, and card layouts.
- Items can span multiple tracks when the design requires it.

Grid: Practical notes

- Grid works best when the layout has a visible overall structure.
- gap creates spacing between grid tracks.
- repeat(), minmax(), and auto-fit support responsive patterns.
- Named areas can make layout code more readable.

Two-column grid

CSS Unit 08 • Example

What to notice: Grid tracks create columns explicitly. • Fraction units divide available space proportionally.

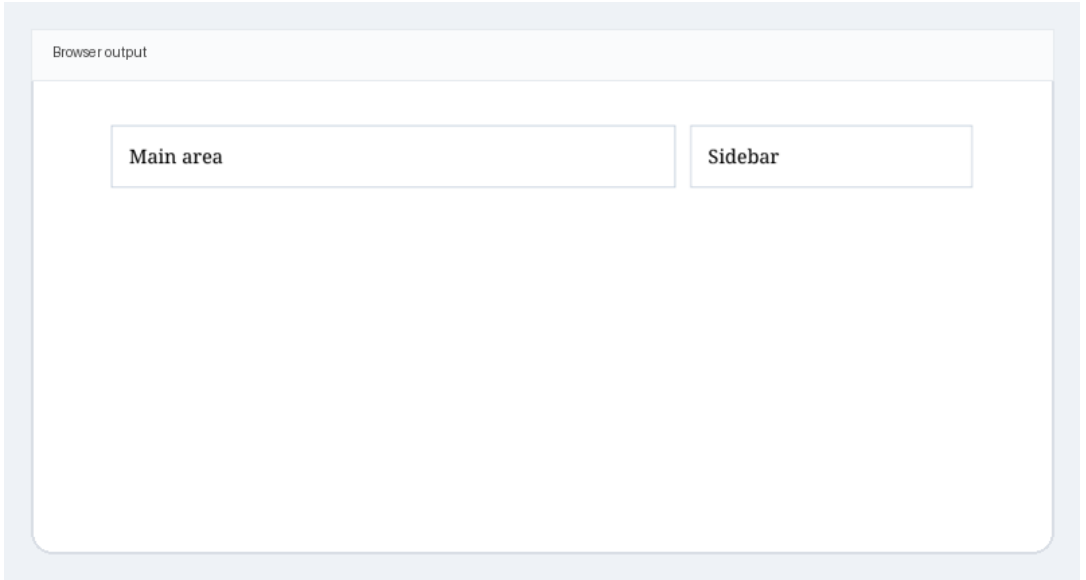
Code

HTML

```
<style>
  .layout {
    display: grid;
    grid-template-columns: 2fr 1fr;
    gap: 12px;
  }
  .main, .side {
    padding: 14px;
    border: 1px solid #cbd5e1;
  }
</style>

<div class="layout">
  <div class="main">Main area</div>
  <div class="side">Sidebar</div>
</div>
```

Rendered output



Simple gallery grid

CSS Unit 08 • Example

What to notice: Grid makes evenly sized galleries easy to build. • Each child becomes a grid item automatically.

Code

HTML

```
<style>
  .gallery {
    display: grid;
    grid-template-columns: repeat(3, 1fr);
    gap: 10px;
  }
  .photo {
    padding: 30px 0;
    text-align: center;
    background: #e5e7eb;
  }
</style>

<div class="gallery">
  <div class="photo">1</div>
  <div class="photo">2</div>
  <div class="photo">3</div>
  <div class="photo">4</div>
</div>
```

Rendered output

Item spanning columns

CSS Unit 08 • Example

What to notice: Grid items can span across multiple tracks. • This is useful for feature cards and dashboard highlights.

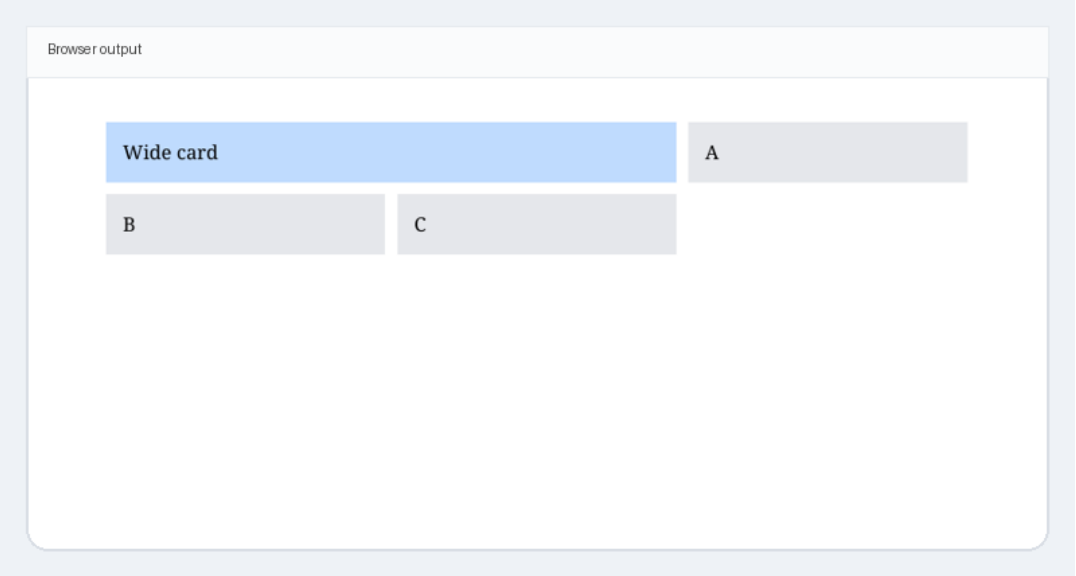
Code

HTML

```html
<style>
  .board {
    display: grid;
    grid-template-columns: repeat(3, 1fr);
    gap: 10px;
  }
  .wide {
    grid-column: span 2;
    background: #bfdbfe;
    padding: 14px;
  }
  .box {
    background: #e5e7eb;
    padding: 14px;
  }
</style>

<div class="board">
  <div class="wide">Wide card</div>
  <div class="box">A</div>
  <div class="box">B</div>
  <div class="box">C</div>
</div>
```

Rendered output



Responsive auto-fit grid

CSS Unit 08 • Example

What to notice: auto-fit and minmax create flexible columns automatically. • This pattern is common in responsive card grids.

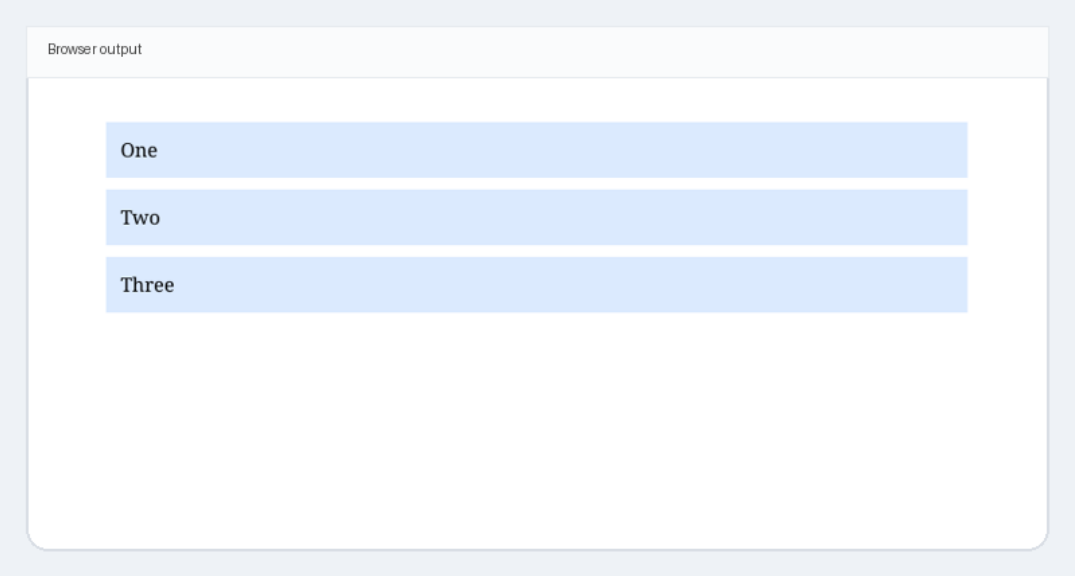
Code

HTML

```
<style>
  .cards {
    display: grid;
    grid-template-columns: repeat(auto-fit, minmax(120px,
1fr));
    gap: 10px;
  }
  .card {
    background: #dbeafe;
    padding: 12px;
  }
</style>

<div class="cards">
  <div class="card">One</div>
  <div class="card">Two</div>
  <div class="card">Three</div>
</div>
```

Rendered output



Named grid areas

CSS Unit 08 • Example

What to notice: Named areas make larger grid layouts easier to read. • The layout can be understood almost like a diagram.

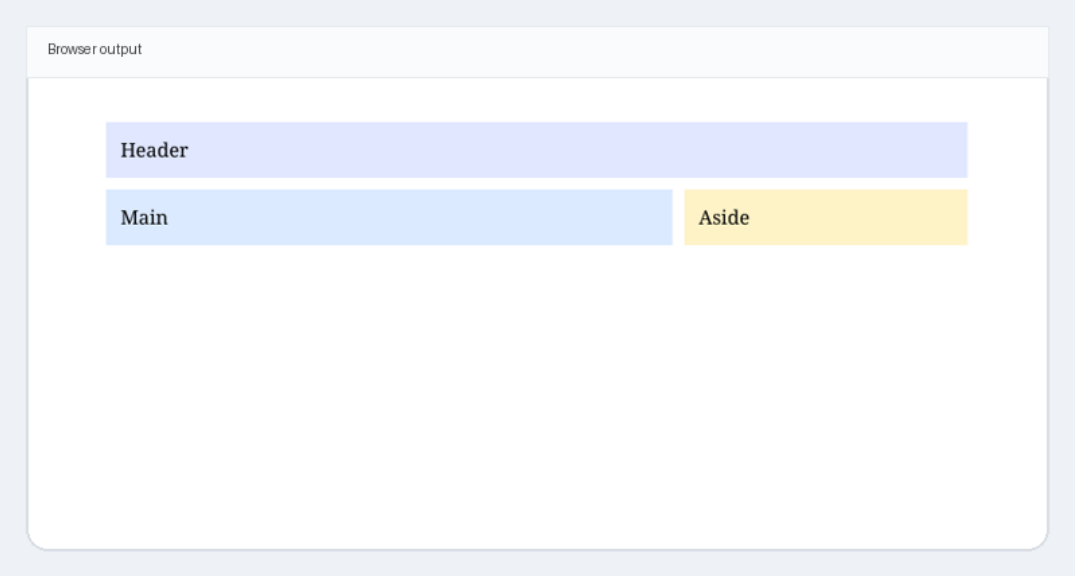
Code

HTML

```
<style>
  .page {
    display: grid;
    grid-template-areas:
      "header header"
      "main aside";
    grid-template-columns: 2fr 1fr;
    gap: 10px;
  }
  header { grid-area: header; background: #e0e7ff; padding: 12px; }
  main { grid-area: main; background: #dbeafe; padding: 12px; }
  aside { grid-area: aside; background: #fef3c7; padding: 12px; }
</style>

<div class="page">
  <header>Header</header>
  <main>Main</main>
  <aside>Aside</aside>
</div>
```

Rendered output



Grid: Common mistakes

- Do not use grid for a simple row when flexbox already fits naturally.
- Do not create overly complex track systems before the content is clear.
- Do not forget that implicit row height can affect the final look.
- Use grid when both rows and columns matter to the design.

Grid: Recap

- Grid is ideal for layouts that need both horizontal and vertical structure.
- Tracks, gaps, spans, and named areas give precise control.
- Responsive grids become elegant with repeat() and minmax().
- When the layout feels architectural, grid is usually a strong candidate.

Web Programming

CSS Unit 09

Responsive Design and CSS Variables

- Responsive design adapts the interface to different screen sizes and contexts.
- Media queries apply styles when certain conditions are true.

Fenerbahce University

Responsive Design and CSS Variables: Key ideas

- Responsive design adapts the interface to different screen sizes and contexts.
- Media queries apply styles when certain conditions are true.
- Relative sizing, max-width, and fluid media are foundational responsive tools.
- CSS variables help keep repeated values centralized and consistent.

Responsive Design and CSS Variables: Practical notes

- A mobile-first workflow starts with a simple small-screen layout.
- max-width can keep content readable on large screens.
- Images should usually scale within their containers.
- Variables make theme colors and spacing easier to update across the project.

Fluid image inside a card

CSS Unit 09 • Example

What to notice: max-width: 100% keeps images from breaking the layout. • height: auto preserves the aspect ratio.

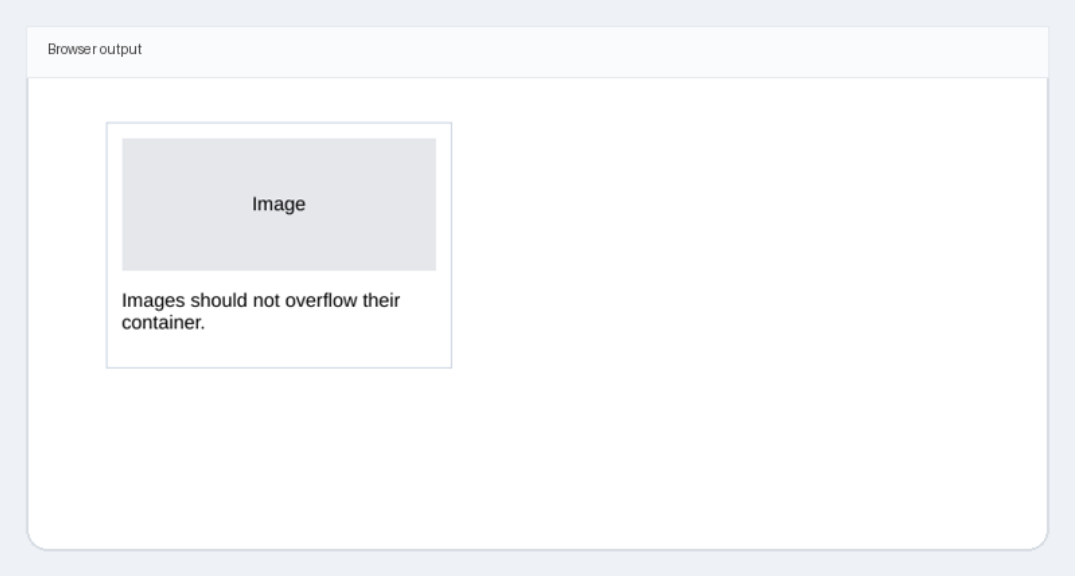
Code

HTML

```
<style>
  .card {
    width: 260px;
    border: 1px solid #cbd5e1;
    padding: 12px;
  }
  img {
    max-width: 100%;
    height: auto;
  }
</style>

<div class="card">
  <img alt="Placeholder image" src="photo.png">
  <p>Images should not overflow their container.</p>
</div>
```

Rendered output



max-width content container

CSS Unit 09 • Example

What to notice: max-width prevents lines from becoming too wide. • margin: 0 auto centers the box when extra space exists.

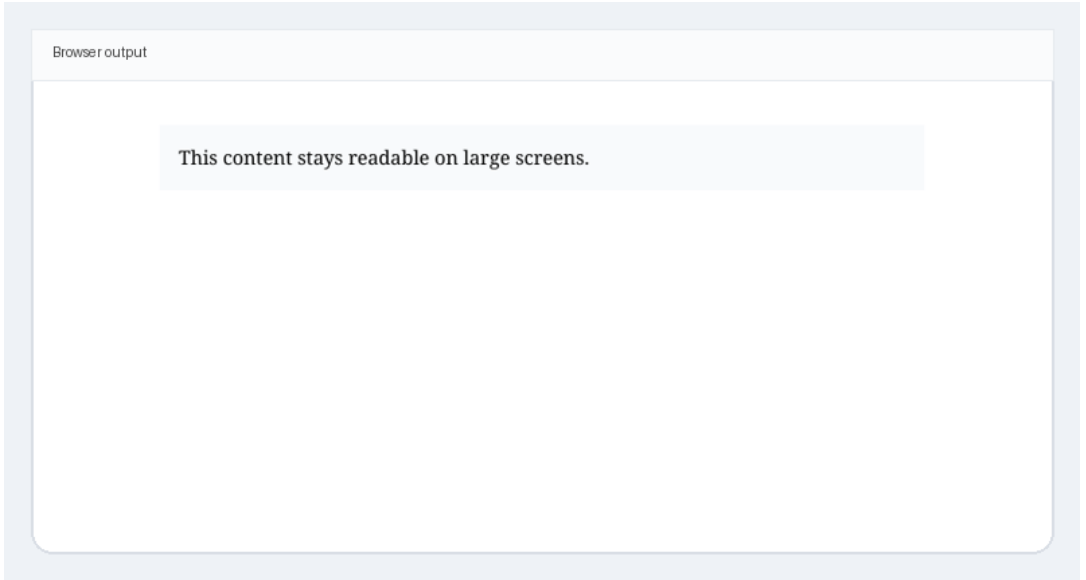
Code

HTML

```
<style>
  .page {
    width: 90%;
    max-width: 600px;
    margin: 0 auto;
    background: #f8fafc;
    padding: 16px;
  }
</style>

<div class="page">This content stays readable on large screens.</div>
```

Rendered output



Media query for a wider layout

CSS Unit 09 • Example

What to notice: Media queries apply extra rules when the viewport meets a condition. • This supports responsive changes without changing the HTML.

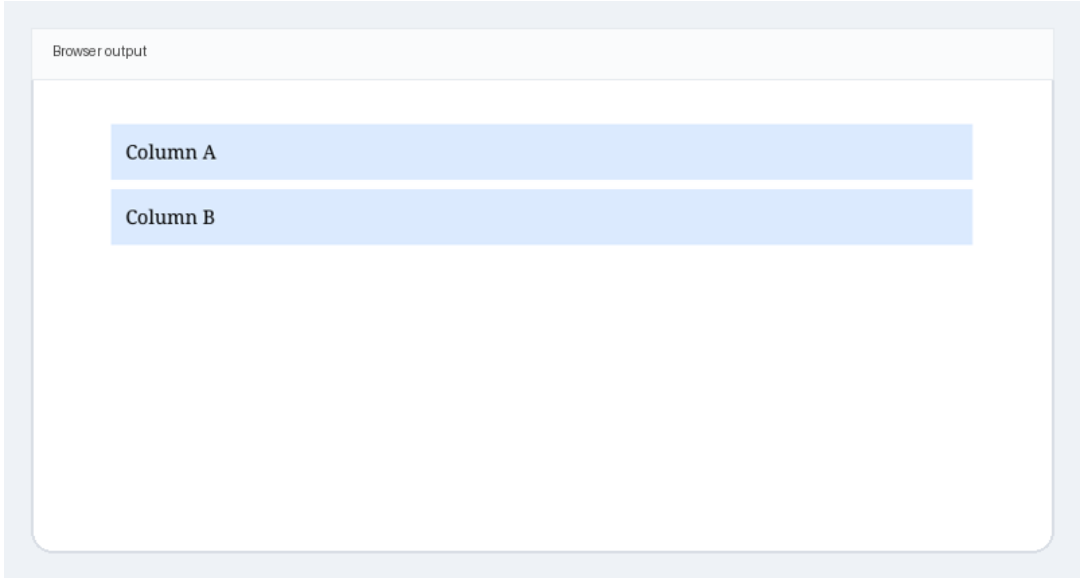
Code

HTML

```
<style>
  .layout {
    display: block;
  }
  @media (min-width: 700px) {
    .layout {
      display: flex;
      gap: 12px;
    }
  }
  .box {
    padding: 12px;
    background: #dbeafe;
    margin-bottom: 8px;
  }
</style>

<div class="layout">
  <div class="box">Column A</div>
  <div class="box">Column B</div>
</div>
```

Rendered output



Clamp for responsive heading size

CSS Unit 09 • Example

What to notice: clamp sets a minimum, preferred, and maximum value. • This is useful for fluid typography across screen sizes.

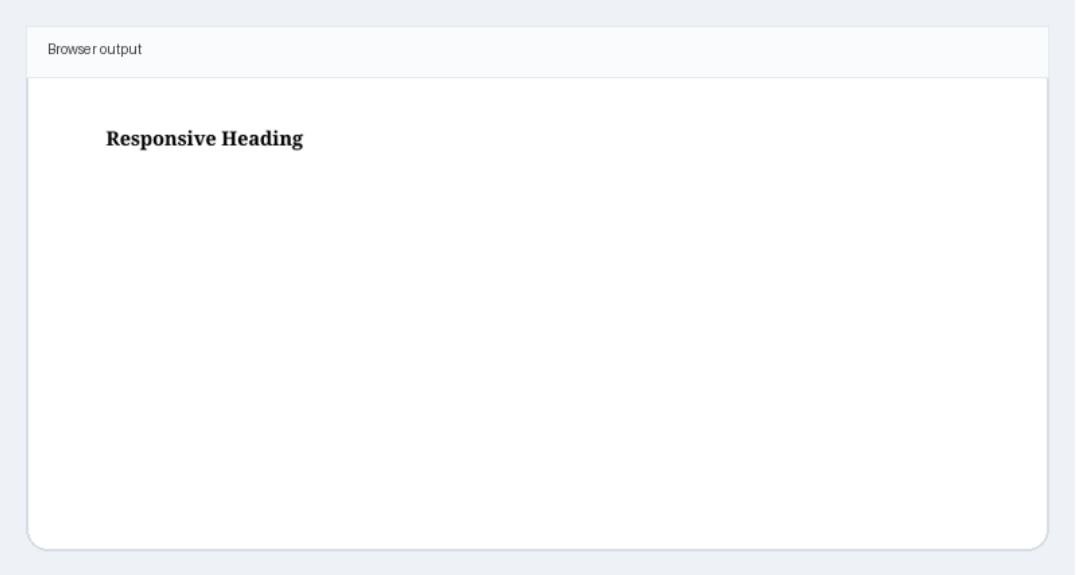
Code

HTML

```
<style>
  h1 {
    font-size: clamp(1.5rem, 4vw, 3rem);
  }
</style>

<h1>Responsive Heading</h1>
```

Rendered output



CSS variables for theme colors

CSS Unit 09 • Example

What to notice: Custom properties are declared with a double hyphen. • var() reads the value wherever it is needed.

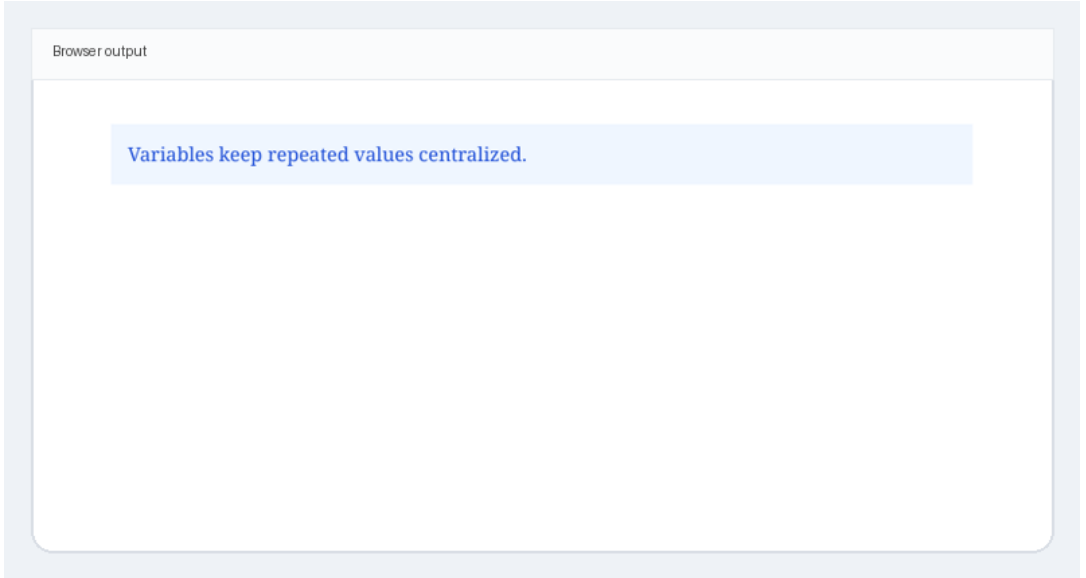
Code

HTML

```
<style>
  :root {
    --brand: #1d4ed8;
    --surface: #eff6ff;
  }
  .card {
    color: var(--brand);
    background: var(--surface);
    padding: 14px;
  }
</style>

<div class="card">Variables keep repeated values centralized.</div>
```

Rendered output



Responsive Design and CSS Variables: Common mistakes

- Do not wait until the end to think about small screens.
- Do not hard-code enormous widths that break narrow devices.
- Do not repeat the same color values everywhere when variables can help.
- Responsive design starts with flexible thinking, not only media queries.

Responsive Design and CSS Variables: Recap

- Responsive design keeps content usable across many screens.
- Fluid sizing and max-width solve many issues before media queries are needed.
- Variables improve consistency and reduce maintenance effort.
- A responsive system is usually simpler than it first appears.

Web Programming

CSS Unit 10

Transforms, Transitions, Animation, and Final Projects

- CSS can add motion and visual state changes without JavaScript.
- Transform changes an element’s shape or position in a composited way.

Fenerbahce University

Transforms, Transitions, Animation, and Final Projects: Key ideas

- CSS can add motion and visual state changes without JavaScript.
- Transform changes an element’s shape or position in a composited way.
- Transition animates a property change over time.
- Keyframes create multi-step animations for loaders, highlights, and effects.

Transforms, Transitions, Animation, and Final Projects: Practical notes

- Motion should support feedback, hierarchy, or delight—not distract users.
- Small transitions often feel more professional than instant jumps.
- Transforms are commonly used for translate, scale, and rotate effects.
- Animation should remain readable, accessible, and restrained.

Transformed card

CSS Unit 10 • Example

What to notice: Transform changes the rendered shape without rewriting the layout code. • Small effects are often enough to create interest.

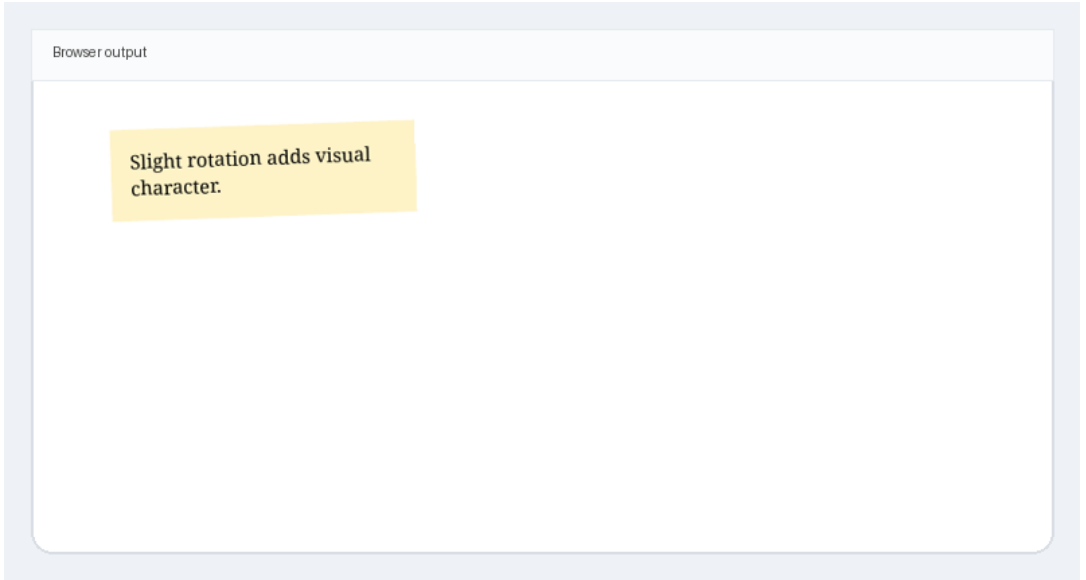
Code

HTML

```
<style>
  .card {
    transform: rotate(-2deg);
    background: #fef3c7;
    padding: 16px;
    width: 220px;
  }
</style>

<div class="card">Slight rotation adds visual character.</div>
```

Rendered output



Transition-ready button

CSS Unit 10 • Example

What to notice: Transition prepares the browser to animate a future state change. • The hover rule defines the visual destination state.

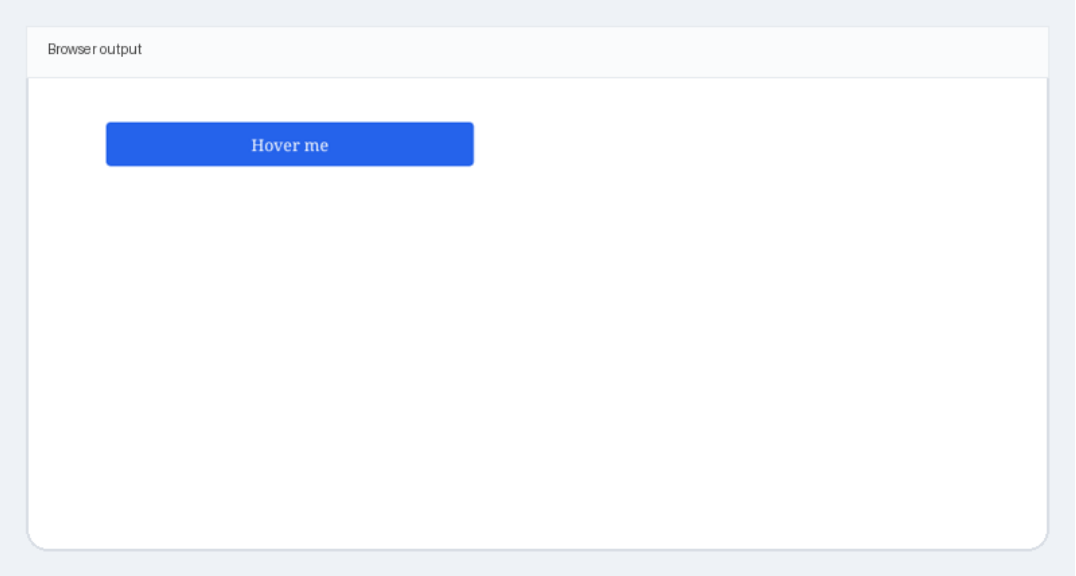
Code

HTML

```
<style>
  button {
    background: #2563eb;
    color: white;
    border: 0;
    padding: 10px 16px;
    transition: background 0.3s ease;
  }
  button:hover {
    background: #1d4ed8;
  }
</style>

<button>Hover me</button>
```

Rendered output



Keyframe pulse animation

CSS Unit 10 • Example

What to notice: Keyframes describe stages of an animation over time. • The animation property applies that sequence to an element.

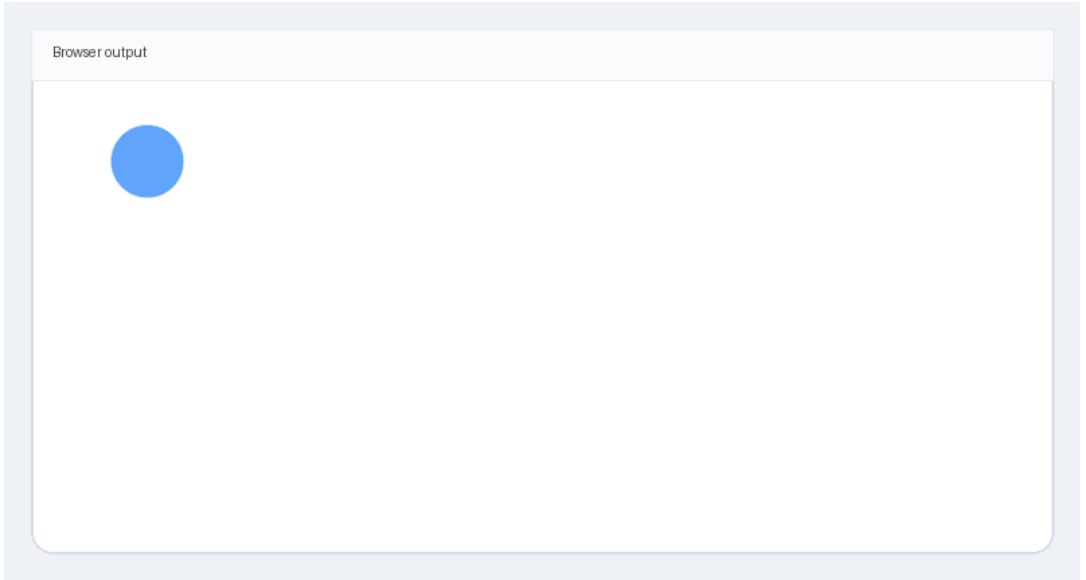
Code

HTML

```
<style>
  @keyframes pulse {
    0% { transform: scale(1); }
    50% { transform: scale(1.08); }
    100% { transform: scale(1); }
  }
  .dot {
    width: 60px;
    height: 60px;
    background: #60a5fa;
    border-radius: 50%;
    animation: pulse 1.5s infinite;
  }
</style>

<div class="dot"></div>
```

Rendered output



Mini project: styled landing hero

CSS Unit 10 • Example

What to notice: This mini project combines color, spacing, typography, and layout. • Small components show how CSS properties work together as a system.

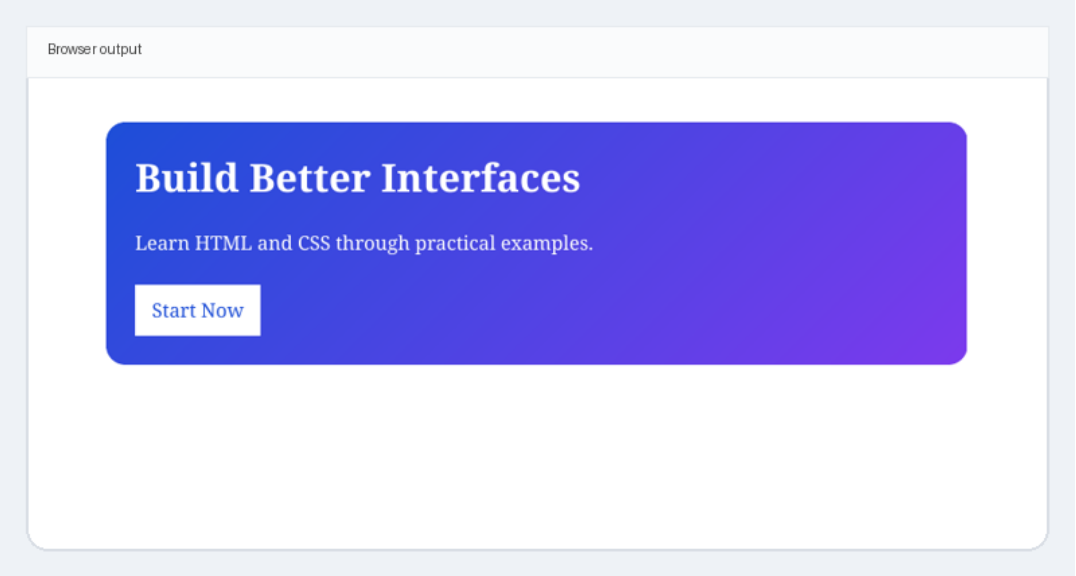
Code

HTML

```
<style>
  .hero {
    background: linear-gradient(135deg, #1d4ed8, #7c3aed);
    color: white;
    padding: 24px;
    border-radius: 16px;
  }
  h1 { margin-top: 0; }
  a {
    display: inline-block;
    margin-top: 8px;
    padding: 10px 14px;
    background: white;
    color: #1d4ed8;
    text-decoration: none;
  }
</style>

<section class="hero">
  <h1>Build Better Interfaces</h1>
  <p>Learn HTML and CSS through practical examples.</p>
  <a href="#">Start Now</a>
</section>
```

Rendered output



Mini project: pricing cards

CSS Unit 10 • Example

What to notice: Real projects combine layout, hierarchy, and visual emphasis. • A featured card can be created with only a small rule difference.

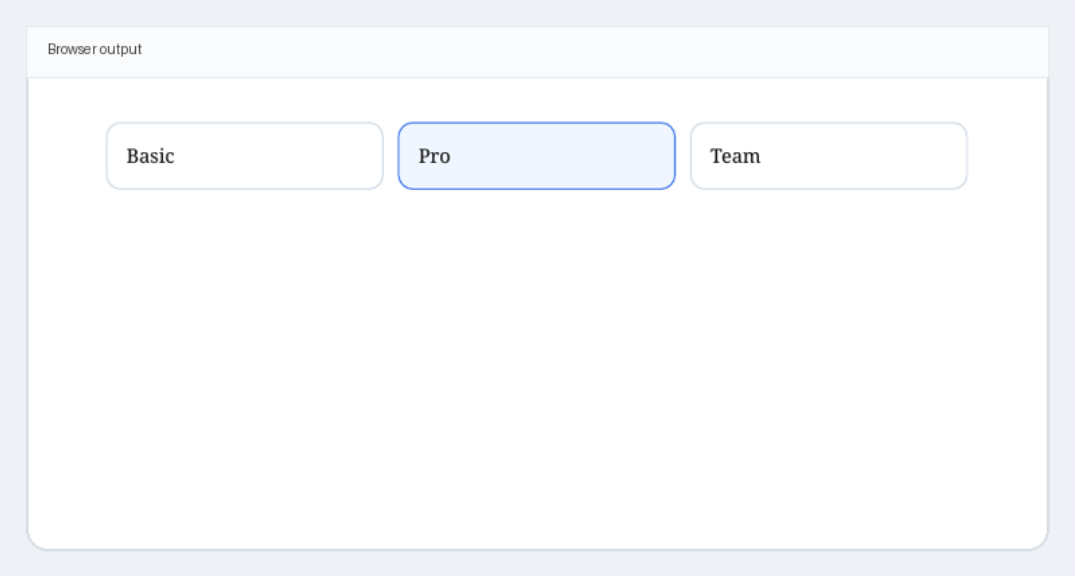
Code

HTML

```
<style>
  .pricing {
    display: flex;
    gap: 12px;
  }
  .plan {
    flex: 1;
    border: 1px solid #cbd5e1;
    padding: 16px;
    border-radius: 12px;
  }
  .featured {
    border-color: #2563eb;
    background: #eff6ff;
  }
</style>

<div class="pricing">
  <div class="plan">Basic</div>
  <div class="plan featured">Pro</div>
  <div class="plan">Team</div>
</div>
```

Rendered output



Transforms, Transitions, Animation, and Final Projects: Common mistakes

- Do not animate every element on the page just because you can.
- Do not use motion that makes reading or interaction harder.
- Do not treat CSS as a list of isolated tricks—think in systems.
- Finish each project by checking responsiveness, readability, and consistency.

Transforms, Transitions, Animation, and Final Projects: Recap

- Transforms, transitions, and animations add polish when used thoughtfully.
- Modern CSS can build complete interface patterns from simple building blocks.
- Final mini projects prove that structure and style now work together.
- The strongest CSS is clear, reusable, and easy to debug.