HTML, CSS and JavaScript Basics

Standalone references for building structured, responsive and accessible web interfaces with HTML, CSS and supporting JavaScript.

- HTML Reference
  - HTML Text and Headings
  - HTML Links and Images
  - HTML Lists and Tables
  - HTML Forms
  - HTML Embeds and Comments
- CSS Reference
  - Connecting a CSS Stylesheet
  - CSS Selectors, Classes and Reusable Rules
  - CSS Box Model and Spacing
  - Responsive Layouts with Flexbox and Grid
  - Accessible Colour, Typography and Focus
  - Styling Forms and Validation Messages
  - Styling Tables, Navigation and Data Displays
- JavaScript Reference
  - Connecting JavaScript and Selecting Elements
  - Handling Events and User Input
  - Filtering and Sorting Displayed Data
  - Updating Interfaces Accessibly
  - Presenting PHP Data with JavaScript Charts

HTML Reference

Standalone HTML references for structuring text, links, images, lists, tables, forms and embedded content.

HTML Text and Headings

HTML provides elements for displaying text, headings and page structure.

Headings Headings are used to organise content on a webpage.

HTML provides six heading levels.

Heading 1

Heading 2

Heading 3

Heading 4

Heading 5

Heading 6

Result:
Heading 1 Heading 2 Heading 3 Heading 4 Heading 5 Heading 6 Paragraphs Paragraphs are used for normal text content.

This is a paragraph.

This is another paragraph.

Result:
This is a paragraph.

This is another paragraph.

Bold Text Use the strong element to make text bold.

Important information Result:

Important information

Italic Text Use the em element to emphasise text.

Important information Result:

Important information

Line Breaks Use the br element to move text onto a new line.

Line One
Line Two Result:

Line One Line Two

Horizontal Lines Use the hr element to create a horizontal divider.

Result:
Subscript Subscript is commonly used in science and mathematics.

H_2O Result:

H2O

Superscript Superscript is commonly used for powers and exponents.

x^2 Result:

x2

Combining Elements Multiple text elements can be combined.

Science Report

Water is written as H_2O.

The formula for area is x^2.

Result:
Complete Example

```
<h1>Heading 1</h1>
```

```
<h2>Heading 2</h2>

<p>This is a paragraph.</p>

<strong>Bold text</strong>

<br><br>

<em>Italic text</em>

<hr>

H<sub>2</sub>O

<br>

x<sup>2</sup>
```

Quick Reference Headings

Heading 1

Heading 6

Paragraph
Paragraph text

Bold **Bold text** Italic *Italic text* Line Break
Horizontal Rule

Subscript Superscript You now know how to create headings and display text content in HTML.

HTML Links and Images

Links support navigation and images communicate visual information. Both need meaningful text alternatives.

Links

```
<a href="results.php">View tournament results</a>
```

Link text should describe the destination without relying on “click here”.

For an external link opened in a new tab:

```
<a href="https://example.com"
   target="_blank"
   rel="noopener">Visit the example website</a>
```

Only open a new tab when useful, and make the behaviour clear in surrounding text.

Email and page sections

```
<a href="mailto:help@example.com">Email fictional support</a>
<a href="#testing">Jump to testing evidence</a>

<h2 id="testing">Testing evidence</h2>
```

Use `example.com` addresses in demonstrations rather than real school or personal details.

Images

```
<img src="images/fictional-team.jpg"
     alt="Fictional Falcons team logo">
```

Alternative text communicates the image’s purpose. A decorative image uses an empty alternative:

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

Do not put essential words only inside an image. Avoid using width and height attributes to distort the aspect ratio; use responsive CSS.

Figures

```
<figure>
  <img src="images/leaderboard-example.png"
       alt="Leaderboard with the Falcons ranked first on 12 points">
  <figcaption>Example leaderboard produced from fictional data.</figcaption>
</figure>
```

Check

- ☐ Link text is meaningful.
- ☐ External new tabs use `rel="noopener"`.
- ☐ Demonstrations use fictional addresses and data.
- ☐ Informative images have purposeful alternative text.
- ☐ Decorative images use `alt=""`.
- ☐ Images resize without distortion.

HTML Lists and Tables

Lists organise related items. Tables represent information with meaningful row and column relationships.

Unordered and ordered lists

```
<ul>
  <li>Team registration</li>
  <li>Match results</li>
  <li>Leaderboard</li>
</ul>

<ol>
  <li>Choose a fictional dataset.</li>
  <li>Validate its headings.</li>
  <li>Import valid rows.</li>
</ol>
```

Use an ordered list when sequence matters. Do not create visual bullets with hyphens and line breaks.

Description list

```
<dl>
  <dt>Administrator</dt>
  <dd>Imports and manages fictional tournament data.</dd>
  <dt>User</dt>
  <dd>Views processed results and statistics.</dd>
</dl>
```

Accessible table

```
<div class="table-wrap">
<table>
  <caption>Fictional team standings</caption>
  <thead>
    <tr>
      <th scope="col">Team</th>
      <th scope="col">Wins</th>
      <th scope="col">Points</th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <th scope="row">Falcons</th>
      <td>4</td>
      <td>12</td>
    </tr>
  </tbody>
</table>
</div>
```

The caption identifies the table. Header cells and `scope` communicate relationships to assistive technology.

Do not use tables for page layout. For narrow screens, keep the semantic table and place it inside a horizontally scrollable wrapper.

Check

- [ ] List type matches the content.
- [ ] Table has a caption.
- [ ] Header cells use suitable scope.
- [ ] Values remain understandable without colour.
- [ ] Layout tables are avoided.
- [ ] Screenshots have meaningful alternative text or are marked decorative.

HTML Forms

Forms collect user input. Accessible forms connect every control to a visible label and provide instructions and errors that do not rely on placeholders or colour.

Basic form

```
<form action="register.php" method="post">
  <div class="form-group">
    <label for="username">Username</label>
    <p id="username-help">Use 3–50 letters, numbers or underscores.</p>
    <input id="username" name="username"
           autocomplete="username"
           aria-describedby="username-help"
           maxlength="50" required>
  </div>

  <div class="form-group">
    <label for="password">Password</label>
    <input id="password" name="password"
           type="password"
           autocomplete="new-password"
           required>
  </div>

  <button type="submit">Create account</button>
</form>
```

The label’s `for` value matches the input’s `id`. The `name` identifies the value submitted to PHP.

Useful input types

```
<input type="email" name="email">
<input type="number" name="capacity" min="1" max="100">
<input type="date" name="event_date">
<input type="file" name="dataset" accept=".csv,text/csv">
```

Input types and HTML constraints help users, but PHP must validate every submitted value again.

Related choices

```
<fieldset>
  <legend>Preferred output</legend>
  <label><input type="radio" name="output" value="table"> Table</label>
  <label><input type="radio" name="output" value="chart"> Chart</label>
</fieldset>
```

Use `fieldset` and `legend` for a related group.

Error summary

```
<div role="alert">
  <p>Please correct the following:</p>
  <ul>
    <li>Capacity must be from 1 to 100.</li>
  </ul>
</div>
```

Preserve valid values after an error when safe, and never return passwords to the form.

GET or POST

Use GET for non-sensitive searches and filters that may be bookmarked. Use POST for registration, login, file upload and operations that change stored data.

Check

- [ ] Every control has an associated label.
- [ ] Instructions remain visible.
- [ ] Appropriate input types and autocomplete values are used.
- [ ] PHP repeats validation.
- [ ] Errors are specific and accessible.
- [ ] Forms contain only fictional data.

HTML Embeds and Comments

HTML can be used to embed videos, maps and other content into a webpage.

Comments can also be added to help explain code without displaying anything on the page.

HTML Comments

Comments are ignored by the browser.

They are useful for explaining code and leaving notes for yourself or other developers.

```
<!-- This is a comment -->
```

Example:

```
<!-- Navigation Menu -->

<nav>

</nav>
```

Comments are not visible on the webpage.

Embedding a YouTube Video

YouTube provides embed code for videos.

Example:

```
<iframe
    width="560"
    height="315"
    src="https://www.youtube.com/embed/W-Q7RMpINVo"
    allowfullscreen>

</iframe>
```

Result:

A YouTube video displayed on the page.

https://www.youtube.com/embed/dQw4w9WgXcQ

Responsive YouTube Videos

To help videos work on phones and tablets:

```
<iframe
    width="100%"
    height="315"
    src="https://www.youtube.com/embed/W-Q7RMpINVo"
    allowfullscreen>

</iframe>
```

The video will automatically adjust to the available width.

Embedding a Google Map

Google Maps can also be embedded.

Example:

```
<iframe
    src="https://www.google.com/maps/embed..."
```

```
    width="600"
    height="450">

</iframe>
```

Result:

An interactive map displayed on the page.

https://www.google.com/maps/embed?pb=!1m18!1m12!1m3!1d3520.366280272018!2d153.37404667577476!3d-28.07436817597404!2m3!1f0!2f0!3f0!3m2!1i1024!2i768!4f13.1!3m3!1m2!1s0x6b911cb972c15b2f%3A0xc34ef94511e30812!2sRobina%20State%20High%20School!5e0!3m2!1sen!2sau!4v1781436815960!5m2!1sen!2sau

---

Displaying a Video File

Videos stored in your project can be displayed using the `video` element.

```
<video
    width="600"
    controls>

    <source
        src="videos/demo.mp4"
        type="video/mp4">

</video>
```

Result:

A video player with play and pause controls.

Displaying Audio Files

Audio can be embedded using the `audio` element.

```
<audio controls>

    <source
        src="audio/music.mp3"
        type="audio/mpeg">

</audio>
```

Result:

An audio player with playback controls.

Displaying PDF Files

PDF documents can be embedded directly into a webpage.

```
<iframe
    src="files/document.pdf"
    width="100%"
    height="600">

</iframe>
```

Result:

The PDF document is displayed inside the webpage.

Common Uses for Embeds

Embeds are commonly used for:

- YouTube videos
- Google Maps
- PDF documents
- Audio players

- Demonstration videos
- Interactive content

Complete Example

```
<!DOCTYPE html>

<html>

<head>
    <title>Embeds Example</title>
</head>

<body>

    <h1>Media Examples</h1>

    <!-- YouTube Video -->

    <iframe
        width="560"
        height="315"
        src="https://www.youtube.com/embed/W-Q7RMpINVo"
        allowfullscreen>

    </iframe>

    <br><br>

    <!-- Audio Player -->

    <audio controls>

        <source
            src="audio/music.mp3"
            type="audio/mpeg">
```

```
    </audio>

</body>

</html>
```

Quick Reference

Comment

```
<!-- Comment -->
```

YouTube Video

```
<iframe>
```

Google Map

```
<iframe>
```

Video File

```
<video>
```

Audio File

```
<audio>
```

PDF

```
<iframe src="document.pdf">
```

You now know how to embed media and add comments in HTML.

Next chapter: **CSS Reference**

CSS Reference

Style responsive and accessible interfaces using reusable CSS rules, layout systems and clear visual states.

Connecting a CSS Stylesheet

CSS controls presentation while HTML provides structure.

Connect the file

Create `css/style.css`, then add this inside the page’s `<head>`:

```
<link rel="stylesheet" href="css/style.css">
```

A relative path is interpreted from the current page. A PHP page in a subfolder may need a different path.

Test the connection

```
body {
  font-family: Arial, sans-serif;
  color: #1f2933;
  background: #f7f9fc;
}
```

Refresh without cache if an old version remains.

Good practice

Keep reusable rules in external stylesheets. Avoid large blocks of inline CSS and do not use colour as the only way to communicate meaning.

Check

- [ ] Link is inside `head`.
- [ ] Path matches folder structure.
- [ ] Browser developer tools show no 404.
- [ ] Page remains understandable if CSS does not load.

CSS Selectors, Classes and Reusable Rules

Selectors identify the HTML elements a rule styles.

```
body { margin: 0; }
h1 { color: #123f73; }
.card { border: 1px solid #cbd5e1; }
#main-search { max-width: 40rem; }
nav a:hover,
nav a:focus-visible { text-decoration: underline; }
```

Use element selectors for broad defaults, classes for reusable components and IDs mainly for unique relationships or scripting. Avoid selectors tied to one accidental nesting structure.

```
<article class="card">
  <h2>Team summary</h2>
</article>
```

Cascade and specificity

When rules conflict, source order and specificity decide which applies. Prefer simple classes over repeated `!important`.

Check

- [ ] Class names describe purpose.
- [ ] Repeated designs reuse the same class.
- [ ] Focus styles are not removed.
- [ ] Selectors remain understandable.

CSS Box Model and Spacing

Every element is a box made from content, padding, border and margin.

```
*,
*::before,
*::after {
  box-sizing: border-box;
}

.card {
  max-width: 42rem;
  padding: 1rem;
  border: 1px solid #cbd5e1;
  margin-block: 1rem;
}
```

With `border-box`, declared width includes padding and border.

Use padding inside a component and margin between components. Prefer relative units such as `rem` for scalable spacing.

Avoid fixed layouts

Large fixed widths can overflow small screens. Combine `width: 100%` with a suitable `max-width`.

Check

- [ ] Global border-box rule is present.
- [ ] Spacing has a consistent scale.
- [ ] Content does not touch borders.
- [ ] Page has no horizontal scrolling at narrow widths.

CSS Reference

Responsive Layouts with Flexbox and Grid

Responsive layouts adapt to available space rather than one device size.

Flexbox navigation

```
.navigation {
  display: flex;
  flex-wrap: wrap;
  gap: 0.75rem;
  align-items: center;
}
```

Grid cards

```
.card-grid {
  display: grid;
  grid-template-columns: repeat(auto-fit, minmax(min(16rem, 100%), 1fr));
  gap: 1rem;
}
```

Media query

```
@media (max-width: 40rem) {
  .page-header {
    align-items: stretch;
  }
```

```
}
```

Allow text to wrap and avoid fixed heights. Test zoom, long labels and real data.

Check

- [ ] Layout works at narrow and wide widths.
- [ ] Reading order remains logical.
- [ ] Keyboard order matches visual order.
- [ ] Tables have a deliberate small-screen treatment.
- [ ] Controls remain large enough to use.

Accessible Colour, Typography and Focus

Accessible styling supports perception, reading and keyboard use.

```
body {
  color: #1f2933;
  background: #ffffff;
  font-family: system-ui, sans-serif;
  line-height: 1.6;
}

a { color: #075985; }

:focus-visible {
  outline: 3px solid #f59e0b;
  outline-offset: 3px;
}
```

Maintain sufficient contrast and never communicate status through colour alone. Pair colour with text, icons or patterns.

Do not disable zoom, remove focus outlines or use tiny fixed text. Keep line lengths comfortable and headings visually distinct.

Reduced motion

```
@media (prefers-reduced-motion: reduce) {
  *, *::before, *::after {
    scroll-behavior: auto;
    animation-duration: 0.01ms !important;
  }
```

```
}
```

Check

- [ ] Text/background contrast is sufficient.
- [ ] Focus indicator is obvious.
- [ ] Status includes words, not colour alone.
- [ ] Page works at 200% zoom.
- [ ] Motion is not required to understand content.

CSS Reference

Styling Forms and Validation Messages

Form styling should make labels, controls, instructions and errors easy to identify.

```
.form-group {
  display: grid;
  gap: 0.35rem;
  margin-block: 1rem;
}

input,
select,
button {
  font: inherit;
  min-height: 2.75rem;
}

input {
  border: 1px solid #64748b;
  border-radius: 0.25rem;
  padding: 0.6rem;
}

input:focus-visible {
  outline: 3px solid #f59e0b;
  outline-offset: 2px;
}

.error {
  color: #8a1c1c;
  border-left: 4px solid #b42318;
  padding: 0.75rem;
}
```

Do not rely on placeholder text as the label. Keep validation messages near the relevant control and provide a summary for multiple errors.

Check

- [ ] Every control retains a visible label.
- [ ] Focus and error states are distinct.
- [ ] Buttons look actionable.
- [ ] Messages remain understandable without colour.
- [ ] Keyboard navigation follows a logical order.

CSS Reference

Styling Tables, Navigation and Data Displays

Data interfaces should prioritise interpretation over decoration.

```
.table-wrap { overflow-x: auto; }

table {
  width: 100%;
  border-collapse: collapse;
}

th, td {
  padding: 0.65rem;
  border: 1px solid #cbd5e1;
  text-align: left;
}

th { background: #123f73; color: #fff; }

tbody tr:nth-child(even) { background: #f8fafc; }

nav ul {
  display: flex;
  flex-wrap: wrap;
  gap: 0.75rem;
  list-style: none;
  padding: 0;
}
```

Use real table markup for tabular data. A responsive wrapper permits horizontal scrolling without changing table meaning.

Highlight the current navigation link with text/shape as well as colour, and retain a visible focus state.

Check

- [ ] Table has a caption and header cells in HTML.
- [ ] Narrow screens can reach every column.
- [ ] Navigation wraps safely.
- [ ] Current location and focus are identifiable.
- [ ] Empty datasets have a visible message.

JavaScript Reference

Use supporting JavaScript for interaction, filtering, accessible updates and data presentation.

Connecting JavaScript and Selecting Elements

JavaScript can add interaction to an HTML interface. Keep the page usable when supporting scripts fail.

Connect a script

Place this before the closing `body` tag:

```
<script src="js/app.js"></script>
```

Or use `defer` in the page head:

```
<script src="js/app.js" defer></script>
```

Select an element

```
<p id="status">Waiting for an action.</p>
```

```
const statusMessage = document.querySelector("#status");

if (statusMessage) {
  statusMessage.textContent = "JavaScript loaded.";
}
```

Use meaningful identifiers and check that an element exists before using it.

Check

- [ ] Script path is correct.
- [ ] Browser console has no error.
- [ ] Text is inserted with `textContent` when HTML is unnecessary.
- [ ] Core information remains available without the script.

Handling Events and User Input

Events allow a script to respond to user actions.

```
<label for="team-filter">Filter teams</label>
<input id="team-filter" type="search">
<p id="filter-status" role="status"></p>
```

```
const filter = document.querySelector("#team-filter");
const status = document.querySelector("#filter-status");

filter.addEventListener("input", () => {
  const value = filter.value.trim();
  status.textContent = value
    ? `Filtering by: ${value}`
    : "Showing all teams";
});
```

Do not use JavaScript as the only validation for server-side operations. PHP must independently validate submitted values.

Accessible interactions

Use native buttons for actions, retain keyboard behaviour, give controls labels and announce important updates through an appropriate status region.

Check

- [ ] Event is attached to the intended control.
- [ ] Empty input is handled.

- [ ] Keyboard users can perform the action.
- [ ] Server-side validation still exists.
- [ ] Status changes are understandable.

Filtering and Sorting Displayed Data

Client-side filtering can help users explore data already present in the browser. Database filtering remains preferable for large or permission-sensitive datasets.

```
const rows = [...document.querySelectorAll("[data-team-row]")];
const search = document.querySelector("#team-search");

search.addEventListener("input", () => {
  const query = search.value.trim().toLowerCase();
  let visibleCount = 0;

  for (const row of rows) {
    const team = row.dataset.team.toLowerCase();
    const visible = team.includes(query);
    row.hidden = !visible;
    if (visible) visibleCount++;
  }

  document.querySelector("#result-count").textContent =
    `${visibleCount} results shown`;
});
```

This demonstrates iteration and selection, but JavaScript itself is supporting technology. Explain where authoritative data processing occurs.

Check

- [ ] Original data is safely rendered.
- [ ] Case and empty searches are handled.
- [ ] Result count updates.

- [ ] Hidden content is not treated as access control.
- [ ] Large datasets are processed server-side.

JavaScript Reference

Updating Interfaces Accessibly

Dynamic interfaces must communicate changes without removing keyboard access or context.

Status message

```
<p id="save-status" role="status" aria-live="polite"></p>
```

```
const status = document.querySelector("#save-status");
status.textContent = "Changes saved.";
```

Use `role="alert"` for urgent errors, not routine success messages.

Toggle a region

```
<button id="details-button" aria-expanded="false"
        aria-controls="details-panel">Show details</button>
<section id="details-panel" hidden>...</section>
```

```
const button = document.querySelector("#details-button");
const panel = document.querySelector("#details-panel");

button.addEventListener("click", () => {
  const opening = panel.hidden;
  panel.hidden = !opening;
  button.setAttribute("aria-expanded", String(opening));
  button.textContent = opening ? "Hide details" : "Show details";
});
```

Check

- [ ] Native controls are used.
- [ ] State is conveyed programmatically and visibly.
- [ ] Focus is not unexpectedly moved.
- [ ] Interface works with keyboard input.
- [ ] Essential content is not available only through animation or colour.

Presenting PHP Data with JavaScript Charts

PHP can query and process database data, then pass a small prepared result to JavaScript for presentation.

```
<?php
$labels = array_column($rows, "team_name");
$points = array_map("intval", array_column($rows, "points"));
?>
<script>
const labels = <?= json_encode($labels) ?>;
const points = <?= json_encode($points) ?>;
</script>
```

Use a chart library or your own display code with these arrays. Do not construct JavaScript by concatenating unescaped database text.

Preserve meaning

Include:

- descriptive chart title
- labels and units
- adequate contrast
- predictable category order
- a table or textual alternative
- an empty-data message

The chart should visualise a result that responds to a user need or success criterion. JavaScript is one presentation option, not the evidence by itself.

Check

- [ ] PHP output uses `json_encode()`.
- [ ] Data is already filtered or aggregated appropriately.
- [ ] Chart has a non-visual alternative.
- [ ] Empty and extreme values are tested.
- [ ] Script errors do not hide all useful output.