Accessibility

Web accessibility ensures that your content is usable by all people, including those with disabilities. It focuses on usability and adherence to standards like **WCAG** (Web Content Accessibility Guidelines).

WCAG is a globally recognized set of guidelines developed by the **World Wide Web Consortium (W3C)** to make web content more accessible.

These guidelines are designed to improve accessibility for individuals with:

- Visual impairments (e.g., blindness, low vision, color blindness)
- Hearing impairments
- Mobility or motor difficulties
- Cognitive or learning disabilities

HTML and Built-In Accessibility

HTML offers built-in accessibility tools such as semantic elements, labels, and ARIA attributes. Below are best practices to help ensure your site is accessible.

Use Semantic HTML

Semantic elements describe their function in the document structure. They help assistive technologies understand and communicate the content's purpose.

Use: `<header>, <nav>, <button>`
Avoid: `<div id="header">, <span onclick="">`

Label All Form Elements

Every input field should have a clear label, and the for attribute in the <label> should match the id of the corresponding input.

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

Avoid using only placeholder text as a label—it disappears when users type and may not be read by screen readers.

Use Alternative Text for Images

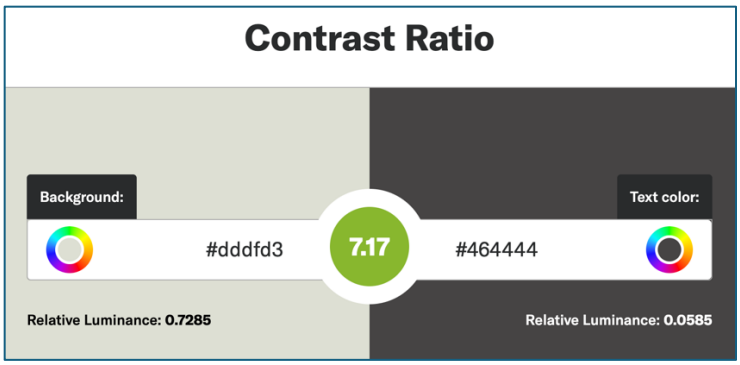
Provide descriptive alt text for meaningful images:

```
<img src="logo.png" alt="Company logo">
```

Use an empty `alt=""` for decorative images so screen readers can skip them.

Ensure Sufficient Color Contrast

Text should have enough contrast against the background to be readable by users with low vision or color blindness.



Make Content Keyboard Accessible

All functionality must be operable using only the keyboard, without needing a mouse or touch input.

This includes:

- Using native HTML elements such as (`<button>`, `<a>`, `<input>`)
    - These elements are inherently accessible using tab, enter, or space
    - They work naturally with screen readers
- Avoiding mouse-only events (`onclick`, `onmouseover`, etc.)
    - Don't use: Submit (not keyboard accessible)
    - Use: <button type="submit">Submit</button>

Use Headings to Structure Content

Use headings (`<h1>` through `<h6>`) to organize content logically and hierarchically.

```
<h1>Main Title</h1>
<h2>Section Title</h2>
<h3>Subsection</h3>
```

This helps screen reader users understand the structure of the page and navigate more easily.

Provide Meaningful Link Text

Make sure link text describes the link's purpose.

Good: `<a href="report.pdf">Download the full report</a>`

Avoid: `<a href="report.pdf">Click here</a>`

Include Captions and Transcripts

Multimedia content should be accessible to users with hearing impairments.

- **Videos** should include captions
- **Audio-only** content should include transcripts

Test with Assistive Technologies

To ensure your content is accessible:

- Use keyboard-only navigation

- Test with screen readers (e.g., NVDA, VoiceOver)
- Check contrast ratios
- Use accessibility tools like Axe, Lighthouse, or WAVE

Use ARIA: Only When Necessary

The **ARIA (Accessible Rich Internet Applications)** specification provides attributes to enhance accessibility, but it should be used sparingly and only when native HTML cannot achieve the desired effect.

As modern websites and applications become increasingly dynamic and interactive, they often use custom components that are not inherently accessible to users relying on assistive technologies. **ARIA** is a set of attributes defined that help bridge the gap between custom elements and accessibility by providing **additional semantic meaning** to HTML elements.

Without ARIA, many custom-built components—like modals, sliders, tab panels, and dropdowns—are invisible or unusable to assistive technology. ARIA provides the **missing information** to ensure full accessibility and compliance with standards like **WCAG**.

ARIA attributes provide information to assistive technologies (like screen readers) about:

- **Roles**: What an element is or does
- **States**: The current condition of a component
- **Properties**: Additional characteristics or relationships

Examples of Common ARIA Attributes

Attribute	Description
`aria-label`	Provides a custom label for screen readers
`aria-hidden`	Hides an element from assistive technologies

Attribute	Description
`aria-expanded`	Indicates whether a collapsible element is open
`aria-checked`	Shows the state of checkboxes or radio buttons
`aria-live`	Declares a region of the page as dynamic/live
`aria-describedby`	Points to text that describes an element

ARIA Roles Example

Define what type of user interface element an object is.

```
<button aria-describedby="tip1">Help</button>

<div role="tooltip" id="tip1">

  This button opens a help guide.

</div>
```

HTML doesn't include a semantic tooltip element. ARIA can fill the gap.The button has aria-describedby="tip1", linking it to the tooltip content.

The button links to the tooltip with aria-describedby, and the role="tooltip" makes it understandable to screen readers.

ARIA States Example

Communicate the current condition of an element. Used when creating collapsible content like dropdowns or accordions:

HTML:

```
<button aria-expanded="false" aria-controls="section1"
id="toggleBtn">Show Details</button>

<div id="section1" hidden>
  <p>Here is some extra content that gets toggled.</p>
</div>
```

Javascript:

```
const button = document.getElementById("toggleBtn");
const section = document.getElementById("section1");

button.addEventListener("click", () => {
  const isExpanded = button.getAttribute("aria-expanded")
=== "true";
  button.setAttribute("aria-expanded",String(!isExpanded));
  section.hidden = isExpanded;
});
```

This allows assistive tech to recognize if the section is visible and updates the interface dynamically.

Remember use semantic HTML whenever possible (e.g., <button>, <nav>, <form>). Use ARIA only when necessary, such as when building custom components. ARIA should enhance, not replace, native HTML accessibility

By following accessibility best practices:

- You make content more usable for all
- You comply with WCAG standards
- You improve the experience for users of all abilities

Use semantic HTML first. Use ARIA only when necessary. Always test your work.