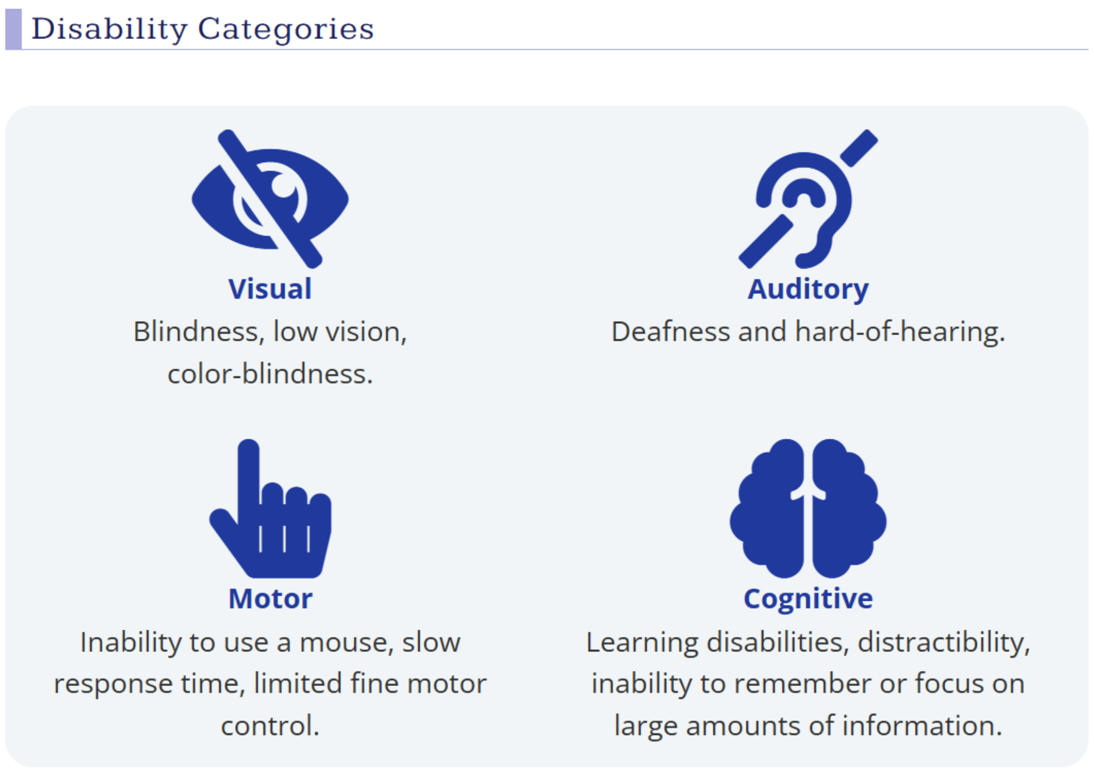
Table of Contents

1. Introduction

1.1 What is web accessibility?

Web accessibility refers to the inclusive practice of ensuring there are no barriers that prevent interaction with, or access to, websites on the World Wide Web by people with disabilities. It benefits everyone, including users on slow connections, users with temporary disabilities, and users in challenging environments.



Disability Categories

Visual
Blindness, low vision, color-blindness.

Auditory
Deafness and hard-of-hearing.

Motor
Inability to use a mouse, slow response time, limited fine motor control.

Cognitive
Learning disabilities, distractibility, inability to remember or focus on large amounts of information.

1.2 Why Accessibility Matters?

- Ensures equal access and opportunity for all users
- Improves SEO and overall usability
- Often a legal requirement (compliance with laws like ADA, Section 508)
- Enhances brand reputation

1.3 Web Accessibility Standards and Laws

1.3.1 WCAG (Web Content Accessibility Guidelines)

Organized by principles: Perceivable, Operable, Understandable, and Robust (POUR). Levels of compliance:

- A: Minimum
- AA: Mid-range (legal standard in many places)
- AAA: Highest

1.3.2 Legal Frameworks

- **ADA (Americans with Disabilities Act)**: Requires public websites to be accessible.
- **Section 508 (US)**: Federal agencies must make electronic content accessible.
- **EN 301 549 (EU)**: Covers ICT accessibility in Europe.
- **RPwD Act (India)**: Mandates digital accessibility for all organizations.

1.4 The Four Principles of WCAG (POUR)

1.4.1 Perceivable

Information must be presented in ways users can perceive:

- Content must be available to the senses (sight, hearing, touch)
- Use text alternatives for non-text content
- Provide captions and transcripts
- Use semantic HTML for meaningful content structure

1.4.2 Operable

Interface components must be operable (e.g., via keyboard).

- All functionality must be available via keyboard
- Avoid content that causes seizures (no more than 3 flashes per second)
- Provide clear navigation and allow users to skip repetitive content

1.4.3 Understandable

Information and operation must be understandable (predictable behavior,readable content)

- Content must be readable and predictable
- Use clear instructions and consistent navigation
- Identify input errors and offer suggestions

1.4.4 Robust

Content must be compatible with current and future assistive technologies.

- Content must be compatible with assistive technologies
- Use valid HTML/CSS/JS
- Follow ARIA specifications accurately

Web Content Accessibility Guidelines (WCAG) 2.0

WebAIM: WebAIM's WCAG 2 Checklist

2. Keyboard Accessibility

Keyboard accessibility is a core principle of web accessibility, ensuring that all interactive elements on a webpage are fully operable using only a keyboard, without requiring a mouse or touch device. This is critical for users who rely on assistive technologies (like screen readers), have motor disabilities, or simply prefer navigating via the keyboard.

2.1 What Makes a Web Page Keyboard Accessible?

2.1.1 Focus

Focus determines where keyboard events go on a webpage. When a user clicks on an input field, that field receives focus and can accept keyboard input.

2.1.2 Keyboard Navigation

Navigate through all interactive elements using the **Tab key (forward)** and **Shift + Tab (backward)**. Arrow keys can navigate within components.

2.1.3 Focus Management

Semantic HTML elements like input fields and buttons are automatically focusable, while non-interactive elements like headers and images are not. Ensuring a logical tab order is crucial for accessibility.

2.1.4 Focus Indicators

The currently focused item is often indicated by a focus ring, which varies based on browser and additional styling.

2.1.5 Skip Links

Provide a "**Skip to main content"** link at the top of your page to let keyboard users bypass repetitive navigation.

2.1.6 No Keyboard trap

Ensure users can **enter and exit** components like modals or menus **without getting stuck**.

2.2 Keyboard Focus

A large part of keyboard accessibility is centred around focus. Focus refers to the element on the screen actively receiving input from the keyboard.

2.2.1 Focus Order

Focus Order also called tab or navigation order, is the order in which elements receive focus. The default focus order must be logical, intuitive, and match the visual order of a page, follows the structure of the DOM, can be disrupted using tabindex improperly or DOM manipulations.

Note: Your tab key moves the keyboard focus up the DOM, shift+tab moves the focus down the DOM.

2.2.2 TabIndex

The tabindex attribute allows developers to modify the default tab order of elements, which is useful for custom components that lack a native focusable equivalent, like dropdowns or modals.

Value	Behaviour
0	Element is focusable in the natural tab order.
-1	Element is programmatically focusable (e.g., with .focus()), but skipped in tabbing.
>0	Element is placed in a custom (ascending) tab order (not recommended as it breaks natural flow).

2.2.3 Focus Indicator

A focus indicator is a visual outline or effect that shows which element is currently focused, especially when navigating using a keyboard (e.g., Tab key).

Default Browser Focus (e.g. Chrome)

```
<a href="#">Link</a>
```

This gets a blue outline on focus by default.

Bad Practice: outline: none

```
button:focus {
  outline: none;
}
```

Good Practice: Custom Focus Style

```
button:focus {
  outline: 2px solid #005fcc;
  outline-offset: 2px;
}
```

2.3 Skip Links

Skip links are anchor links that jump to a different section of the same page, using that section's ID, instead of sending the user to another page on the website or an external resource.

Skip links are typically added as the first focusable element a user will encounter when arriving at a website and can be visible or visually hidden until a user tabs to it, depending on what the design calls for.

```
<a href="#main" class="skip-link">Skip to main content</a>
<main id="main">...</main>
```

Make sure the skip link is visibly styled on focus.

2.4 Keyboard Testing

Testing with a keyboard is an essential part of any accessibility evaluation.
The following table includes many of the most common online interactions, the standard keystrokes for the interaction, and additional information on things to consider during testing.

Interaction	**Keystrokes**	**Notes**
Navigate to interactive elements	• **Tab** - navigate forward • **Shift** + **Tab** - navigate backward	• Keyboard focus indicators must be present. • Navigation order should be logical and intuitive.
Link	• **Enter** - activate the link	•
Button	• **Enter** or **Spacebar** - activate the button	Ensure elements with ARIA role="button" can be activated with both key commands.
Checkbox	• **Spacebar** - check/uncheck a checkbox	Users can typically select zero, one, or multiple options from group of checkboxes.
Radio buttons	• **Spacebar** - select the focused option (if not selected) • ↑/↓ or ←/→ - navigate between options • **Tab** - leave the group of radio buttons	Users can select only one option from a group of radio buttons.
Select (dropdown) menu	• ↑/↓ - navigate between options • **Spacebar** - expand • **Enter**/**Esc** - select option and collapse	You can also filter or jump to options in the menu as you type letters.
Autocomplete	• Type to begin filtering • ↑/↓ - navigate to an option	•

	• **Enter** - select an option	
Dialog	• **Esc** - close	• Modal dialogs should maintain keyboard focus. • Non-modal dialogs should close automatically when they lose focus. • When a dialog closes, focus should usually return to the element that opened the dialog.
Slider	• ↑/↓ or ←/→ - increase or decrease slider value • **Home**/**End** - beginning or end	• For double-headed sliders (to set a range), **Tab**/**Shift** + **Tab** should toggle between each end. • In some sliders **PageUp**/**PageDown** can move by a larger increment (e.g., by 10%).
Menu bar	• ↑/↓ - previous/next menu option • **Enter** - expand the menu (optional) and select an option. • ←/→ - expand/collapse submenu	• A menu bar dynamically changes content within an application. Links that utilize **Tab**/**Enter** are NOT menu bars.
Tab panel	• **Tab** - once to navigate into the group of tabs and once to navigate out of the group of tabs • ↑/↓ or ←/→ - choose and activate previous/next tab.	• This is for 'application' tabs that dynamically change content within the tab panel. If a menu looks like a group of tabs, but is actually a group of links to different pages, **Tab** and **Enter** are more appropriate.
'Tree' menu	• ↑/↓ - navigate previous/next menu option • ←/→ - expand/collapse submenu, move up/down one level.	•
Scroll	• ↑/↓ - scroll vertically • ←/→ - scroll horizontally • **Spacebar**/**Shift** + **Spacebar** - scroll by page	The space bar will, by default, scroll the page, but only if an interactive control that allows space bar input is not focused. Horizontal scrolling within the page should be minimized.

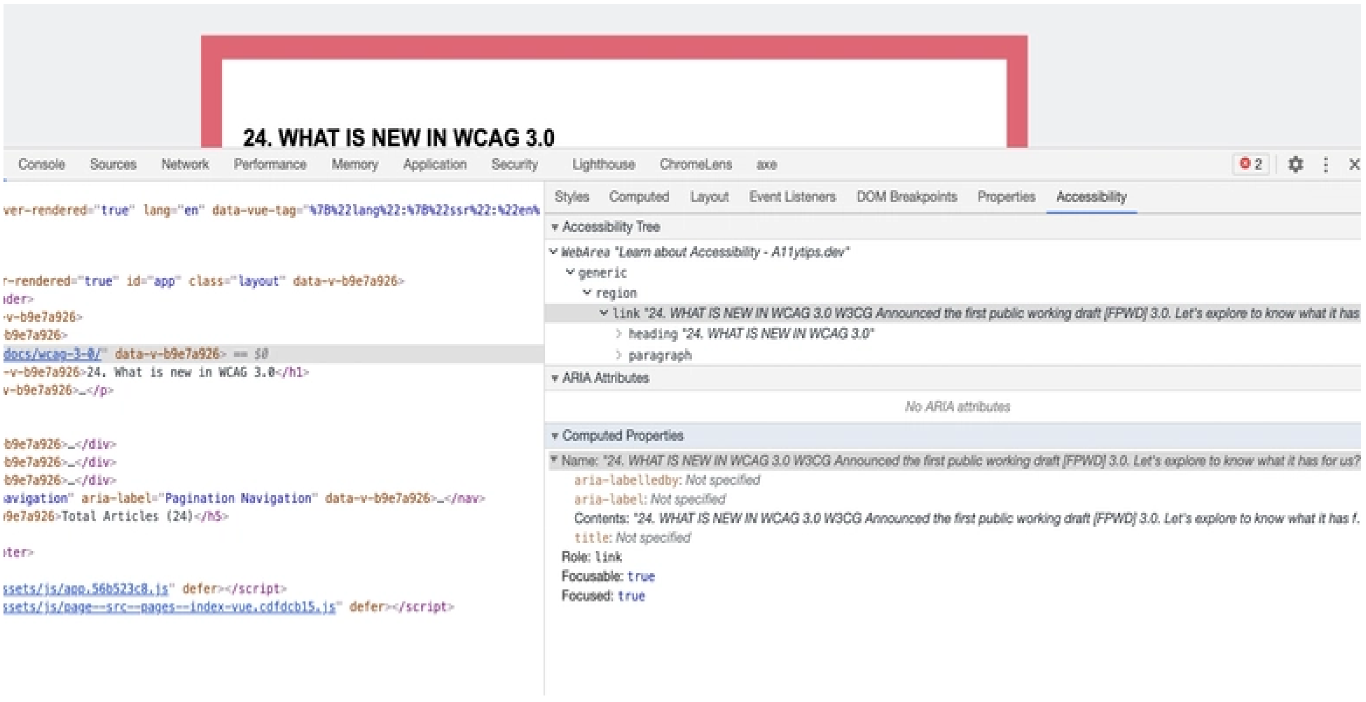
3. Semantics

Semantic HTML elements convey meaning and structure. Assistive technologies use these to interpret the page.

Example: <button> is focusable, announces itself as a button, and handles key events —
<div role="button"> needs additional work

3.1 Accessibility Tree

The browser modifies the DOM tree to create the accessibility tree, which is a simplified version that assists assistive technologies like screen readers.
The accessibility tree is created by the browser and based on the standard Document Object Model (DOM) tree. Like the DOM tree, the accessibility tree contains objects representing all the markup elements, attributes, and text nodes. The accessibility tree is also used by platform-specific accessibility APIs to provide a representation that assistive technologies can understand.
We can see this tree in the firefox and Chrome debugger too. This helps the developer to see how the accessibility elements are getting added such as role, focusable, alt tag, aria-labelledby, etc.



```
ver-rendered="true" lang="en" data-vue-tag="%7B%22lang%22:%7B%22ssr%22:%22en%

r-rendered="true" id="app" class="layout" data-v-b9e7a926>
ider>
-v-b9e7a926>
b9e7a926>
docs/wcag-3-0/" data-v-b9e7a926> == $0
-v-b9e7a926>24. What is new in WCAG 3.0</h1>
v-b9e7a926>…</p>

b9e7a926>…</div>
b9e7a926>…</div>
b9e7a926>…</div>
avigation" aria-label="Pagination Navigation" data-v-b9e7a926>…</nav>
9e7a926>Total Articles (24)</h5>

ter>

ssets/js/app.56b523c8.js" defer></script>
ssets/js/page--src--pages--index-vue.cdfdcb15.js" defer></script>
```

```
▾ Accessibility Tree
⌄ WebArea "Learn about Accessibility - A11ytips.dev"
  ⌄ generic
    ⌄ region
      ⌄ link "24. WHAT IS NEW IN WCAG 3.0 W3CG Announced the first public working draft [FPWD] 3.0. Let's explore to know what it has
        › heading "24. WHAT IS NEW IN WCAG 3.0"
        › paragraph
▾ ARIA Attributes
                No ARIA attributes
▾ Computed Properties
▾ Name: "24. WHAT IS NEW IN WCAG 3.0 W3CG Announced the first public working draft [FPWD] 3.0. Let's explore to know what it has for us?
    aria-labelledby: Not specified
    aria-label: Not specified
    Contents: "24. WHAT IS NEW IN WCAG 3.0 W3CG Announced the first public working draft [FPWD] 3.0. Let's explore to know what it has f.
    title: Not specified
  Role: link
  Focusable: true
  Focused: true
```

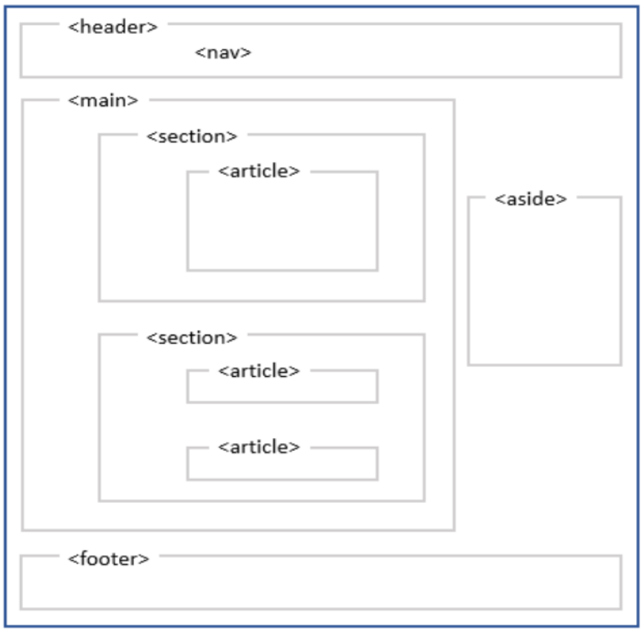
3.2 Semantic Elements

When you use semantic HTML elements, the inherent meaning of each element is passed on to the accessibility tree and used by the Assistive Technologies8, giving more meaning to the content than non-semantic elements.

There may be cases where you need to add additional ARIA attributes to build relationships or to enhance the overall user experience, but in most situations, one of the 100+ HTML elements available should work fairly well on its own.

3.2.1 Landmarks

Landmarks are the way to mark the areas of a webpage for the keyboard and assistive technology users (screen readers). By using landmarks, we identify the organization and structure of the webpage. Landmarks should provide the same structure programmatically of the webpage as it provides visually.



- **Header**
  The <header> element is used to include header content like web page logo, login link, website settings link etc. Ideally, every web page has one header. However, multiple headers may also be included as per need. Can serve as a page banner or introductory content for sections.

- **Nav**
  The <nav> element is used for navigational content like navigation menu for the website. There is no limit to the number of times <nav> tag can be used on a web page. As long as there are navigation links, links can be wrapped inside <nav>. Contains links to other pages or sections within the page.

- **Section**
  The <section> HTML element represents a generic standalone section of a document. Sections should always have a heading, with very few exceptions.

- **Main**
  The <main> element is used for demarking the main content of the web page. Only one main tag per webpage is allowed.

- **Article**
  The <article> HTML element represents a self-contained composition in a document, page, application, or site, which is intended to be independently distributable or reusable (e.g., in syndication).Represents self-contained content like blog posts or news articles.

- **Aside**
  The <aside> HTML element represents a portion of a document whose content is only indirectly related to the document's main content. Asides are frequently presented as sidebars or call-out boxes.

- **Footer**
  The <footer> element is used to include footer content like copyright, about us, terms and conditions link,etc. One footer is included per page.

3.2.2 Headings

Headings communicate the organization of the content on the page. Web browsers, plug-ins, and assistive technologies can use them to provide in-page navigation.
The most important heading has the rank 1 (<h1>), the least important heading rank 6 (<h6>). Headings with an equal or higher rank start a new section, headings with a lower rank start new subsections that are part of the higher ranked section.

Skipping heading ranks can be confusing and should be avoided where possible: Make sure that a <h2> is **not** followed directly by an <h4>.
For example. It is ok to skip ranks when closing subsections, for instance, a <h2> beginning a new section, can follow an <h4> as it closes the previous section.

```
<header>
  <h1>Stamp collecting</h1>
</header>
<main>
   <section aria-label="Introduction to stamp collecting">
      <h2>What is stamp collecting?</h2>
      <p>Stamp collecting, also known as philately, is the study of
   postage stamps, stamped envelopes, postmarks, postcards, and
   other materials relating to postal delivery.</p>
   </section>

   <section aria-label="Start a stamp collection">
      <h2>How do I start a stamp collection?</h2>
   <h3>Required equiment</h3>
   <p>...</p>

   <h3>How to acquire stamps</h3>
   <p>...</p>

   <h3>Organizations you can join</h3>
      <p>...</p>
   </section>
</main>
```

3.2.3 Lists

HTML lists are a way to semantically group items similar to one other giving them inherent meaning, much like your grocery store list or that never-ending to-do list you keep ignoring.

There are three types of HTML lists:

- ordered <ol>
- unordered <ul>
- description <dl>

The list item <li> element is used to represent an item in an ordered or unordered list, while the description term <dt> and definition <dd> elements can be found in the description list.

When programmed correctly, these elements can inform non-sighted assistive technology users about the visible structure of the list. When an assistive technology encounters a semantic list, it can tell the user the list name and how many items are in it. As the user navigates within the list, the assistive technology will read each list item out loud and tell which number it's in the list—item one of five, item two of five, and so on.

Grouping items into lists also helps sighted people who have cognitive and attention disorders and those with reading disabilities, as list content is typically styled to have more visual whitespace and the content is to the point.

3.2.4 Images

Images must have text alternatives that describe the information or function represented by them. This ensures that images can be used by people with various disabilities. This tutorial demonstrates how to provide appropriate text alternatives based on the purpose of the image:

- Informative images: Images that graphically represent concepts and information, typically pictures, photos, and illustrations. The text alternative should be at least a short description conveying the essential information presented by the image.

- Decorative images: Provide a null text alternative (alt="") when the only purpose of an image is to add visual decoration to the page, rather than to convey information that is important to understanding the page.

- Functional images: The text alternative of an image used as a link or as a button should describe the functionality of the link or button rather than the visual image. Examples of such images are a printer icon to represent the print function or a button to submit a form.

- Images of text: Readable text is sometimes presented within an image. If the image is not a logo, avoid text in images. However, if images of text are used, the text alternative should contain the same words as in the image.

- Complex images such as graphs and diagrams: To convey data or detailed information, provide a complete text equivalent of the data or information provided in the image as the text alternative.

- Groups of images: If multiple images convey a single piece of information, the text alternative for one image should convey the information for the entire group.

- Image maps: The text alternative for an image that contains multiple clickable areas should provide an overall context for the set of links. Also, each individually clickable area should have alternative text that describes the purpose or destination of the link.

For a quick overview on deciding which category a particular image fits into, see the alt Decision Tree. The text alternative needs to be determined by the author, depending on the usage, context, and content of an image. For example, the exact type and look of a bird in an image might be less relevant and described only briefly on a website about parks, but may be appropriate on a website specifically about birds.

3.2.5 Forms

Forms are commonly used to provide user interaction on websites and in web applications. For example, login, registering, commenting, and purchasing. This tutorial shows you how to create accessible forms. The same concepts apply to all forms, whether they are processed client or server-side.

Aside from technical considerations, users usually prefer simple and short forms. Only ask users to enter what is required to complete the transaction or process; if irrelevant or excessive data is requested, users are more likely to abandon the form.

- Labelling Controls: Use the <label> element, and, in specific cases, other mechanisms (e.g. WAI-ARIA, title attribute etc.), to identify each form control.
- Grouping Controls: Use the <fieldset> and <legend> elements to group and associate related form controls.
- Form Instructions: Provide instructions to help users understand how to complete the form and individual form controls.
- Validating Input: Validate input provided by the user and provide options to undo changes and confirm data entry.
- User Notifications: Notify users about successful task completion, any errors, and provide instructions to help them correct mistakes.
- Multi-Page Forms: Divide long forms into multiple smaller forms that constitute a series of logical steps or stages and inform users about their progress.
- Custom Controls: Use stylized form elements and other progressive enhancement techniques to provide custom controls.

Note:

If possible, forms should not be subject to a time limit to allow users to complete the form at their pace. If a time limit needs to be in place, for example, for security reasons, the user should have the option to turn it off or extend it. This restriction does not apply if the time limit is due to a live event, such as an auction or a game, or if the time to complete the form is essential for a valid submission.

3.2.6 Tables

Data tables are used to organize data with a logical relationship in grids. Accessible tables need HTML markup that indicates header cells and data cells and defines their relationship. Assistive technologies use this information to provide context to users.

Header cells must be marked up with <th>, and data cells with <td> to make tables accessible. For more complex tables, explicit associations may be needed using scope, id, and headers attributes.

This tutorial shows you how to apply appropriate structural markup to tables. It includes the following pages:

- Tables with one header for rows or columns: For tables with content that is easy to distinguish, mark up header cells with <th> and data cells with <td> elements.
- Tables with two headers have a simple row header and a simple column header: For tables with unclear header directions, define the direction of each header by setting the scope attribute to col or row.
- Tables with irregular headers have header cells that span multiple columns and/or rows: For these tables, define column and row groups and set the range of the header cells using the colgroup and rowgroup values of the scope attribute.
- Tables with multi-level headers have multiple header cells associated per data cell: For tables that are so complex that header cells can't be associated in a strictly horizontal or vertical way, use id and headers attributes to associate header and data cells explicitly.
- Caption & Summary: A caption identifies the overall topic of a table and is useful in most situations. A summary provides orientation or navigation hints in complex tables.

Some document formats other than HTML, such as PDF, provide similar mechanisms to markup table structures. Word processing applications may also provide mechanisms to markup tables. Tables markup is often lost when converting from one format to another, though some programs may provide functionality to assist converting table markup.

Many web authoring tools and content management systems (CMS) provide functions to define header cells during table creation without having to edit the code manually.

3.2.7 Audio & Video

It's important to know the basics of accessibility requirements for media. This knowledge will help you design and build the appropriate layouts and features to accommodate users with different environmental and sensory needs, such as the millions of people with hearing loss or visual impairment worldwide.

Alternative media types were developed to support the media needs of people with disabilities. This gives people additional formats to choose from when accessing audio and video content.

The alternative media types you must include with your media files depend on:

- The type of media you are supporting—audio-only, video-only, or video with audio (multimedia) formats
- Whether the media is live or prerecorded
- The version and level of WCAG compliance you are targeting
- Any additional media-related user needs

To create accessible audio and video content for websites and apps, there are three main types of alternative:

- **Captions**

One of the most widely used alternative media types are captions. Captions are written text synchronized with multimedia content for people who can't hear or understand spoken words. They are presented in the same language as the main audio track and include important non-speech information, such as sound effects, background noises, and essential music.

Captions benefit people who are deaf, hard of hearing, or have cognitive disabilities, but are useful to many other people as well.

Captions come in two forms—open or closed.

- **Closed captions (CC)** are text on top of a video that can be turned on or off by the viewer and, depending on the media player, styled in a way that fits the user's needs.
- **Open captions (OC)** are text burned into the video and cannot be turned off or styled differently.

One method might be preferable, depending on the situation or how the multimedia will be consumed.

- **Transcripts**

Close cousins to captions, transcripts are detailed, text-based documents that capture all essential words, sounds, and important visual information in your media. Transcripts primarily help people who are hard of hearing or deaf, and descriptive transcripts help people who are deafblind.

Transcripts are also useful for people with cognitive disabilities or for people who want to review the content at their own speed.

Search bots can't access your captions but can crawl your text transcripts. When you include transcripts with your media files, your search engine optimization gets a boost. It's one of those rare exceptions when duplicate content isn't confusing to users or penalized by search engine algorithms.

You can ensure you've made your transcript available to all users by:

- Including the transcript text directly in-context, on the page with the embedded video.
- Adding a link to an accessible PDF containing the transcript.
- Linking out to the copy on another page.
- Including a link to the transcript, wherever it lives, within the video description on whatever media player platform you've used (such as YouTube or Vimeo).

- **Sign language interpretation**

Another major alternative media type you may encounter is sign language interpretation, where an interpreter narrates the auditory portion of the audio-only or multimedia content using sign language. This is very important for many people who are deaf, as sign language is their first and most fluent language.

Sign language interpretation is often more expressive and detailed than written documents, providing a much richer experience than captions or transcripts alone.

That said, sign language interpretation can be time-consuming and cost-prohibitive to many organizations. And even if you have the time and budget to add sign language interpretation to your media, there are over 300 different sign languages worldwide. Adding one sign language interpretation to your media wouldn't be enough to support a global audience.

4. ARIA (Accessible Rich Internet Applications)

WAI-ARIA (Accessible Rich Internet Applications or ARIA) is a W3C specification for enhancing accessibility in ways that plain HTML cannot. ARIA is not a true programming language but a set of attributes you can add to HTML elements to increase their accessibility.

These attributes communicate role, state, and property to assistive technologies using accessibility APIs found in modern browsers. This communication happens through the accessibility tree. ARIA modifies incorrect or incomplete code to create a better experience for those using assistive technologies by changing, exposing, and augmenting parts of the accessibility tree.

When used properly, ARIA can...

- enhance accessibility of interactive controls, such as tree menus, sliders, pop-ups, etc.
- define helpful landmarks for page structure
- define dynamically-updated "live regions"
- improve keyboard accessibility and interactivity
- and much more

4.1 Rules for ARIA

4.1.1 Rule 1: Do not use ARIA

ARIA is required when an HTML element does not have accessibility support. This could be because the design does not allow for a specific HTML element or the wanted feature or behaviour isn't available in HTML. However, these situations should be scarce.

```
Don't: Assign a role.
 <a role="button">Submit</a>

Do: Use the semantic element.
<button>Submit</button>
```

4.1.2 Rule 2: Do not add (unnecessary) ARIA to HTML

In most circumstances, HTML elements work well as-is and do not need additional ARIA added to them. In fact, developers using ARIA often must add additional code to make the elements functional in the case of interactive elements.

```
Don't: Assign a misleading role.
<h2 role="tab">Heading tab</h2>

Do: Assign roles correctly.
<div role="tab"><h2>Heading tab</h2></div>
```

4.1.3 Rule 3: Always support keyboard navigation

All interactive (not disabled) ARIA controls must be keyboard accessible. You can add tabindex= "0" to any element that needs a focus that doesn't normally receive keyboard focus. Avoid using tab indexes with positive integers whenever possible to prevent potential keyboard focus order issues.

```
Don't: Add a tabindex.
<span role="button" tabindex="1">Submit</span>

Do: Set the tabindex to `0`.
<span role="button" tabindex="0">Submit</span>
```

4.1.4 Rule 4: Don't hide focusable elements

Don't add role= "presentation" or aria-hidden= "true" to elements that need to have focus—including elements with a tabindex= "0". When you add these roles and states to elements, it sends a message to the AT that these elements are not important and to skip over them. This can lead to confusion or disrupt users attempting to interact with an element.

```
Don't: Hide focusable elements

<div aria-hidden="true">
  <button>Submit</button>
</div>

Do: Expose focusable elements
<div>
  <button>Submit</button>
</div>
```

4.1.5 Rule 5: Use accessible names for interactive elements

The purpose of an interactive element needs to be conveyed to a user before they know how to interact with it. Ensure that all elements have an accessible name for people using Assistive technology devices. Accessible names can be the content surrounded by an element (in the case of an <a>), alternative text, or a label. For each of the following code samples, the accessible name is "Red leather boots."

```
<!-- A plain link with text between the link tags. -->
<a href="shoes.html">Red leather boots</a>

<!-- A linked image, where the image has alt text. -->
<a href="shoes.html"><img src="shoes.png" alt="Red leather boots"></a>

<!-- A checkbox input with a label. -->
<input type="checkbox" id="shoes">
<label for="shoes">Red leather boots</label>
```

4.2 Core Components (Roles, Properties, State)

4.2.1 Roles

Roles define what an element is or does. Most HTML elements have a default role that is presented to assistive technology. For example, <button> has a default role of "button" and <form> has a default role of "form". ARIA can define roles that are not available in HTML, and can also override the default roles of HTML elements (see Rule #2 above).

4.2.1.1 Landmark Roles

ARIA can define roles for significant page areas or regions. These are identified by screen readers and help with orientation and navigation in the page. The available document landmark roles are:

- **banner**

Site-orientated content that typically contains the name of the web site, logo, search, and/or main navigation. Semantically equivalent to <header>

- **navigation**

The area that contains the navigation links for the document or web site. Semantically equivalent to <nav>

- **main**

The main or central content of the document. Semantically equivalent to <main>

- **complementary**

Supporting content for the main content, often presented in a side bar. Semantically equivalent to <aside>

- **contentinfo**

Informational child content, such as footnotes, copyrights, links to privacy statement, links to preferences, and so on. Semantically equivalent to <footer>

- **search**

A section that contains the search functionality for the site. There is no equivalent element in HTML.

- **form**

The form role can be used to identify a group of elements on a page that provide equivalent functionality to an HTML form. The form is not exposed as a landmark region unless it has an **accessible name**.

```
<div role="form" id="contact-info" aria-label="Contact information">
  <!-- form content -->
</div>
```

- **region**

The region role is used to identify document areas the author deems significant. It is a generic landmark available to aid in navigation when none of the other landmark roles are appropriate.

```
<div role="region" aria-label="Example">
  <!-- region content -->
</div>
```

On a typical web page, the logo and header content might be within an element with role of banner.
The navigation links across the top would be identified within navigation, typically contained within the banner. The site search form would be given <form role="search">.
The main body of an article would be main. The related links in a side bar might be identified as complementary.
The footer content and links at the bottom of a page would have a role of contentinfo.

4.2.1.2 Live Region Roles

Live region roles are used when content updates dynamically without moving focus.

- **alert**

The alert role is used to communicate an important and usually time-sensitive message to the user. When this role is added to an element, the browser will send out an accessible alert event to assistive technology products which can then notify the user.

```
<div id="expirationWarning" role="alert">
  <span class="hidden">Your log in session will expire in 2 minutes</span>
</div>
```

- **status**

A status is a type of live region providing advisory information that is not important enough to justify an alert, which would immediately interrupt the announcement of a user's current activity.

- **log**

The log role is used to identify an element that creates a live region where new information is added in a meaningful order and old information may disappear.

- **timer**

The **timer** role indicates to assistive technologies that an element is a numerical counter listing the amount of elapsed time from a starting point or the remaining time until an end point. Assistive technologies will not announce updates to a timer as it has an implicit aria-live value of off.

```
<div role="timer" id="eggtimer">0</div>
```

- **marquee**

The marquee role defines an area as a type of live region that presents non-essential information that changes frequently. Examples of marquees include stock tickers and ad banners; information that is not necessarily sought out by the user that may be presented in any order.

4.2.1.3 Window Roles

Window roles are used for dialog boxes and modals.

- **dialog**

The dialog role is used to mark up an HTML based application dialog or window that separates content or UI from the rest of the web application or page.

Dialogs are generally placed on top of the rest of the page content using an overlay. Dialogs can be either non-modal (it is still possible to interact with content outside of the dialog) or modal (only the content in the dialog can be interacted with).

```
<div
  role="dialog"
  aria-labelledby="dialog1Title"
  aria-describedby="dialog1Desc">
  <h2 id="dialog1Title">Your personal details were successfully updated</h2>
  <p id="dialog1Desc">
    You can change your details at any time in the user account section.
  </p>
  <button>Close</button>
</div>
```

- **alertdialog**

The **alertdialog** role is to be used on modal alert dialogs that interrupt a user's workflow to communicate an important message and require a response.

```
<div
  id="alert_dialog"
  role="alertdialog"
  aria-modal="true"
  aria-labelledby="dialog_label"
  aria-describedby="dialog_desc">
  <h2 id="dialog_label">Confirmation</h2>
  <div id="dialog_desc">
    <p>Are you sure you want to delete this image?</p>
    <p>This change can't be undone.</p>
  </div>
  <ul>
    <li>
      <button id="close-btn" type="button">No</button>
    </li>
    <li>
      <button id="confirm-btn" type="button" aria-controls="form">Yes</button>
    </li>
  </ul>
</div>
```

4.2.1.4 Abstract Roles

Abstract roles are only intended for use by browsers to help organize and streamline a document. They should not be used by developers writing HTML markup. Doing so will not result in any meaningful information being conveyed to assistive technologies or to users.

Avoid using command, composite, input, landmark, range, roletype, section, sectionhead, select, structure, widget and window.

4.2.1.5 Document Structure Roles

Document Structure roles are used to provide a structural description for a section of content. Most of these roles should no longer be used as browsers now support semantic HTML elements with the same meaning. The roles without HTML equivalents, such as presentation, toolbar and tooltip roles, provide information on the document structure to assistive technologies such as screen readers as equivalent native HTML tags are not available.

- **toolbar**

A toolbar is a collection of commonly used controls, such as buttons or checkboxes, grouped together in a compact visual form. The toolbar role can be used to communicate the presence and purpose of such a grouping to screen reader users and can help reduce the number of tab stops for keyboard users. Only use the toolbar role to group 3 or more controls.
https://www.w3.org/WAI/ARIA/apg/patterns/toolbar/examples/toolbar/

- **tooltip**

A tooltip is a contextual text bubble that displays a description for an element that appears on pointer hover or keyboard focus.

```
<label for="password">Password:</label>
<input aria-describedby="passwordrules" id="password" type="password" />
<div role="tooltip" id="passwordrules">
  <p>Password Rules:</p>
  <ul>
    <li>Minimum of 8 characters</li>
    <li>
      Include at least one lowercase letter, one uppercase letter, one number
      and one special character
    </li>
    <li>Unique to this website</li>
  </ul>
</div>
```

- **feed**

A feed is a dynamic scrollable list of articles in which articles are added to or removed from either end of the list as the user scrolls. A feed enables screen readers to use the browse mode reading cursor to both read and scroll through a stream of rich content that may continue scrolling infinitely by loading more content as the user reads.
https://www.w3.org/WAI/ARIA/apg/patterns/feed/examples/feed/

- **math**

The math role indicates that the content represents a mathematical expression.

The above pythagorean theorem is written accessibly as:

```
<div role="math" aria-label="a^{2} + b^{2} = c^{2}">
  a<sup>2</sup> + b<sup>2</sup> = c<sup>2</sup>
</div>
```

Had an image been used, the alt attribute would be used along with the math role:

```
    <img src="pythagorean_theorem.gif" alt="a^{2} + b^{2} = c^{2}" role="math" />
```

- **note**

The note role can be added to parenthetic or ancillary content if no other native element or other role fits the purpose.

```
<h1>Madam C. J. Walker</h1>
<p>
  Madam C.J. Walker was an African American entrepreneur, philanthropist, and
  political and social activist.
</p>
<h2>Early Life</h2>
…
<h2>Career</h2>
…
<p role="note" class="highlight-box">
  At the height of the depression, Madam C. J. Walker trained 20,000 women to
  sell hair pomade door-to-door
</p>
<h2>Activism and Philanthropy</h2>
```

- **presentation/none**

The presentation role and its synonym none remove an element's implicit ARIA semantics from being exposed to the accessibility tree.
The content of the element will still be available to assistive technologies; it is only the semantics of the container — and in some instance, required associated descendants — which will no longer expose their mappings to the accessibility API.

If presentation or none is applied to a <table> element, the descendant <caption>, <thead>, <tbody>, <tfoot>, <tr>, <th>, and <td> elements inherit the role and are thus not exposed to assistive technologies. But, elements inside of the <th> and <td> elements, including nested tables, are exposed to assistive technologies.

For most document structure roles, semantic HTML equivalent elements are available and supported. Avoid using:

- article (use <article>)
- cell (use <td>)
- columnheader (use <th scope="col">)
- definition (use <dfn>)
- directory
- document
- figure (use <figure> instead)
- group
- heading (use h1 through h6)
- img (use <img> or <picture> instead)
- list (use either <ul> or <ol> instead)
- listitem (use <li> instead)
- meter (use <meter> instead)
- row (use the <tr> with <table>)
- rowgroup (use <thead>, <tfoot> and <tbody>)
- rowheader (use <th scope="row">)
- separator (use <hr> if it doesn't have focus)
- table (use <table>)
- term (use <dfn>)

4.2.1.6 Widget Roles

Widget roles are used to define common interactive patterns. Like document structure roles, some widget roles have the same semantics as well-supported native HTML elements, and therefore should be avoided. The key difference is that widget roles typically require JavaScript for interaction, while document structure roles often do not.

- **scrollbar**

A scrollbar is a graphical object that controls the scrolling of content within a viewing area.

```
<div id="pi-label">Pi</div>
<div id="pi">
3.1415926535897932384626433832795028841971693993751058209749445923078164062862089
986280348253421170679
</div>
<div
  role="scrollbar"
  aria-labelledby="pi-label"
  aria-controls="pi"
  aria-orientation="horizontal"
  aria-valuenow="0"
  aria-valuemin="0"
  aria-valuemax="100">
  <div id="thumb"></div>
</div>
```

- **searchbox**

The searchbox role indicates an element is a type of textbox intended for specifying search criteria.

```
<div tabindex="0" aria-label="search" role="searchbox" contenteditable></div>
```

```
<form role="search">
  <input
    type="search"
    role="searchbox"
    aria-description="search results will appear below"
    id="search"
    value="" />
  <label for="search">Search this site</label>
  <button>Submit search</button>
</form>
<div aria-live="polite" role="region" aria-atomic="true">
  <div class="sr-only"></div>
</div>
<div id="search-results"></div>
```

- **separator**

A separator is a divider that separates and distinguishes sections of content or groups of menuitems. There are two types of separators: a static structure that provides a visible boundary, identical to the HTML <hr> element, and a focusable, moveable widget.

```
<div role="separator"><h3>Title of my separator</h3></div>
```

- **slider**

The slider role defines an input where the user selects a value from within a given range.

```
<div>
  <div id="temperatureLabel">Temperature</div>
  <div id="temperatureValue">20°C</div>
  <div id="temperatureSlider">
    <div
      id="temperatureSliderThumb"
      role="slider"
      aria-labelledby="temperatureLabel"
      aria-orientation="vertical"
      tabindex="0"
      aria-valuemin="15.0"
      aria-valuemax="25.0"
      aria-valuenow="20.0"
      aria-valuetext="20 degrees Celsius"
      style="top: calc((25 - 20)*2rem - 0.5rem)"></div>
  </div></div>
```

- **spinbutton**

The spinbutton role defines a type of range that expects the user to select a value from among discrete choices.

```
<p id="day">Enter the day of the month</p>
<button type="button" tabindex="-1" aria-label="previous day">‹</button>
<div
  role="spinbutton"
  tabindex="0"
  aria-valuenow="1"
  aria-valuetext="first"
  aria-valuemin="1"
  aria-valuemax="31"
  aria-labelledby="day">
  1
</div>
<button type="button" tabindex="-1" aria-label="next day">›</button>
```

- **switch**

The ARIA **switch** role is functionally identical to the checkbox role, except that instead of representing "checked" and "unchecked" states, which are fairly generic in meaning, the switch role represents the states "on" and "off."

```
<button role="switch" aria-checked="true" id="speakerPower" class="switch">
  <span>off</span>
  <span>on</span>
</button>
<label for="speakerPower" class="switch">Speaker power</label>
```

- **tab**

The ARIA tab role indicates an interactive element inside a tablist that, when activated, displays its associated tabpanel.

```
<button role="tab" aria-selected="true" aria-controls="tabpanel-id" id="tab-id">
    Tab label
</button>
```

- **tabpanel**

The ARIA tabpanel is a container for the resources of layered content associated with a tab.

```
<div class="tabs">
  <div role="tablist" aria-label="Sample Tabs">
    <button
      role="tab"
      aria-selected="true"
      aria-controls="panel-1"
      id="tab-1"
      tabindex="0">
      First Tab
    </button>
    <button
      role="tab"
      aria-selected="false"
      aria-controls="panel-2"
      id="tab-2"
      tabindex="-1">
      Second Tab
    </button>
    <button
      role="tab"
      aria-selected="false"
      aria-controls="panel-3"
      id="tab-3"
      tabindex="-1">
      Third Tab
    </button>
  </div>
  <div id="panel-1" role="tabpanel" tabindex="0" aria-labelledby="tab-1">
    <p>Content for the first panel</p>
  </div>
  <div id="panel-2" role="tabpanel" tabindex="0" aria-labelledby="tab-2" hidden>
    <p>Content for the second panel</p>
  </div>
  <div id="panel-3" role="tabpanel" tabindex="0" aria-labelledby="tab-3" hidden>
    <p>Content for the third panel</p>
  </div>
</div>
```

- **treeitem**

A tree is a hierarchical list with parent and child nodes that can expand and collapse.
A treeitem is a node in a tree. The root of the tree is tree, but all tree nodes are treeitem elements, even if they themselves have nested treeitem nodes.

```
<div>
  <h3 id="treeLabel">Developer Learning Path</h3>
  <ul role="tree" aria-labelledby="treeLabel">
    <li role="treeitem" aria-expanded="true">
      <span>Web</span>
      <ul role="group">
        <li role="treeitem" aria-expanded="false">
          <span>Languages</span>
          <ul role="group">
            <li role="treeitem" aria-expanded="false">
              <span>HTML</span>
              <ul role="group">
                <li role="treeitem">Document structure</li>
                <li role="treeitem">Head elements</li>
                <li role="treeitem">Semantic elements</li>
                <li role="treeitem">Attributes</li>
                <li role="treeitem">Web forms</li>
              </ul>
            </li>
            <li role="treeitem">CSS</li>
            <li role="treeitem">JavaScript</li>
          </ul>
        </li>
        <li role="treeitem" aria-expanded="false">
          <span>Accessibility</span>
          <ul role="group">
            <li role="treeitem" aria-label="accessibility object model">AOM</li>
            <li role="treeitem">WCAG</li>
            <li role="treeitem">ARIA</li>
          </ul>
        </li>
        <li role="treeitem" aria-expanded="false">
          <span>Web Performance</span>
          <ul role="group">
            <li role="treeitem">Load time</li>
          </ul>
        </li>
        <li role="treeitem">APIs</li>
      </ul>
    </li>
  </ul>
</div>
```

Avoid using button, checkbox, gridcell, link, menuitem, menuitemcheckbox, menuitemradio, option, progressbar, radio, and textbox, which we've included for completeness. For most, semantic equivalents with accessible interactivity are available and supported. See the individual role documentation for more information.

Composite widget roles

- **combobox**

The combobox role identifies an element as either an input or a button that controls another element, such as a listbox or grid, that can dynamically pop up to help the user set the value. A combobox can be either editable (allowing text input) or select-only (only allowing selection from the popup).

```
<label for="jokes">Pick what type of jokes you like</label>
<div class="combo-wrap">
  <input
    type="text"
    id="jokes"
    role="combobox"
    aria-controls="joketypes"
    aria-autocomplete="list"
    aria-expanded="false"
    data-active-option="item1"
    aria-activedescendant="" />
  <span aria-hidden="true" data-trigger="multiselect"></span>
  <ul id="joketypes" role="listbox" aria-label="Jokes">
    <li class="active" role="option" id="item1">Puns</li>
    <li class="option" role="option" id="item2">Riddles</li>
    <li class="option" role="option" id="item3">Observations</li>
    <li class="option" role="option" id="item4">Knock-knock</li>
    <li class="option" role="option" id="item5">One liners</li>
  </ul>
</div>
```

- **menu**

The menu role is a type of composite widget that offers a list of choices to the user.

```
<div>
  <button
    type="button"
    aria-haspopup="menu"
    aria-controls="colormenu"
    tabindex="0"
    aria-label="Text Color: purple">
    Purple
  </button>
  <ul role="menu" id="colormenu" aria-label="Color Options" tabindex="-1">
    <li
      role="menuitemradio"
      aria-checked="true"
      style="color: purple"
      tabindex="-1">
      Purple
    </li>
    <li
      role="menuitemradio"
      aria-checked="false"
      style="color: magenta"
      tabindex="-1">
      Magenta
    </li>
    <li
      role="menuitemradio"
      aria-checked="false"
      style="color: black;"
      tabindex="-1">
      Black
    </li>
  </ul>
</div>
```

- **menubar**

A menubar is a presentation of menu that usually remains visible and is usually presented horizontally.

https://www.w3.org/WAI/ARIA/apg/patterns/menubar/examples/menubar-navigation/

- **tablist**

The tablist role identifies the element that serves as the container for a set of tabs. The tab content are referred to as tabpanel elements.

See the tabpanel, tab, and tablist example in the tabpanel role definition.

- **tree**

A tree is a widget that allows the user to select one or more items from a hierarchically organized collection.

Avoid using grid, listbox, and radiogroup, which we've included for completeness. See the individual role documentation for more information.

https://www.w3.org/TR/wai-aria-1.3/#role_definitions
https://developer.mozilla.org/en-US/docs/Web/Accessibility/ARIA/Reference/Roles
https://www.w3.org/WAI/ARIA/apg/patterns/
https://inclusive-components.design/

4.2.2 States & Properties

ARIA properties define additional semantics not supported in standard HTML.

An example is <button aria-haspopup="true">.
This property extends the standard button to cause a screen reader to announce that the button, if activated, will trigger a pop-up.

ARIA states are attributes that define the current condition of an element. They generally change based on user interaction or some dynamic variable.

An example is <input aria-invalid="true">.
This property will cause a screen reader to read this input as currently being invalid (meaning it needs to be corrected), but this state value could easily be changed to false dynamically based on user input.

There are 4 categories of ARIA states and properties:

4.2.2.1 Widget attributes

- **aria-autocomplete**

The aria-autocomplete attribute indicates whether inputting text could trigger display of one or more predictions of the user's intended value for a combobox, searchbox, or textbox and specifies how predictions will be presented if they are made.

values:
✓ "inline" – Suggestion appears inline (e.g., grayed-out text).
✓ "list" – Suggestions appear in a list (e.g., dropdown).
✓ "both" – Both inline and list suggestions are provided.
✓ "none" – No autocomplete is provided.

- **aria-checked**

Indicates the current checked state of checkboxes, radio buttons, or tree items.

values:
✓ "true" – Checked
✓ "false" – Not checked
✓ "mixed" – Partially checked (used in tri-state checkboxes)

- **aria-disabled (deprecated)**

Indicates whether an element is disabled (non-interactive), even if it doesn't use the HTML disabled attribute. It is used on buttons, form controls, custom widgets.

values:
"true" or "false"

Note: Unlike disabled, an aria-disabled element can still receive focus.

- **aria-errormessage**

Identifies the ID of an element that describes an error message related to the input. It is used on: inputs, form controls.

value:
ID reference of the error message element.

```
<p>
  <label for="email">Email address:</label>
  <input
    type="email"
    name="email"
    id="email"
    aria-invalid="true"
    aria-errormessage="err1" />
  <span id="err1" class="errormessage">Error: Enter a valid email address</span>
</p>
```

Note: Use with aria-invalid="true" to trigger screen reader announcement.

- **aria-expanded**

Indicates whether a UI element (like a menu, accordion, or disclosure widget) is expanded or collapsed. Used on: button, link, or toggleable container elements.

values:
"true", "false", or "undefined"

```
<label for="username">Username</label>
<input id="username" name="username" aria-describedby="username-desc" />
<button
  aria-expanded="false"
  aria-controls="username-desc"
  aria-label="Help about username"
  type="button">
  <span aria-hidden="true">?</span>
</button>
<p id="username-desc" hidden>
  Your username is the name that you use to log in to this service.
</p>
```

- **aria-haspopup**

The aria-haspopup attribute indicates the availability and type of interactive popup element that can be triggered by the element on which the attribute is set. Used on: Buttons or elements that open a menu, dialog, etc.

values:
- ✓ "true" – Generic popup
- ✓ "menu", "listbox", "tree", "grid", "dialog" – Specific types of popups

- **aria-hidden**

Hides elements from assistive tech (like screen readers) without hiding it visually.

values:
"true" or "false"

Use with caution — don't hide interactive content that should be accessible.

- **aria-invalid**

Marks a form control as having failed validation. Used on: Form fields, custom inputs.

values:
✓ "true" – Invalid
✓ "false" – Valid
✓ "grammar" – Grammar error
✓ "spelling" – Spelling error

- **aria-label**

Defines a text label for an element (overrides visible text). Used on: Any element, especially custom controls.

value:
A string label.

Alternative: use aria-labelledby to reference another element as a label.

- **aria-level**

Indicates the hierarchical level of an element in a structure (like headings, trees). Used on: Elements with roles like heading, treeitem, etc.

A positive integer (e.g., 1, 2, 3)

- **aria-modal**

Indicates whether a dialog prevents interaction with the rest of the page. Used on: Dialogs, modals.

Values:
"true" or "false"

When true, focus is expected to stay within the modal.

- **aria-multiline**

Indicates if a text box can accept multiple lines. Used on: Text inputs, textbox role elements.

values:
✓ "true" (multi-line)
✓ "false" (single-line)

- **aria-multiselectable**

Indicates whether multiple options can be selected in a list/grid. Used on: listbox, grid, tree, etc.

values:
"true" or "false"

- **aria-orientation**

Specifies the orientation of the element. Used on: slider, separator, tablist, toolbar, etc.

values:
✓ "horizontal" (default for most)
✓ "vertical"

- **aria-placeholder**

Provides a placeholder text description of the expected input. Used on: Elements with role textbox, searchbox, combobox, etc.Complements or replaces the native placeholder attribute.

value:
A string

- **aria-pressed**

Indicates the pressed state of a toggle button. Used on: button elements acting as toggle buttons.

values:
✓ "true" – Pressed
✓ "false" – Not pressed
✓ "mixed" – Indeterminate
✓ undefined – Normal button

- **aria-readonly**

Indicates the element is read-only (cannot be modified by the user). Used on: textbox, combobox, gridcell, slider, etc.

values:
"true" or "false"

- **aria-required**

Indicates that user input is required before submission. Used on: Inputs, form fields.

values:
"true" or "false"

- **aria-selected**

Indicates the current selection state of an item in a group (like tabs, options). Used on: Items within listbox, grid, tablist, etc.

values:
"true", "false", or "undefined"

- **aria-sort**

Indicates sort direction of column or grid data. Used on: Column headers (role="columnheader").

values:

✓ "none" – Not sorted
✓ "ascending" – Sorted low → high
✓ "descending" – Sorted high → low
✓ "other" – Custom sort

- **aria-valuemax, aria-valuemin, aria-valuenow, aria-valuetext**

These are used together for range widgets like sliders, spinbuttons, and progress bars.

Attribute	Description
aria-valuemax	Maximum allowed value (number)
aria-valuemin	Minimum allowed value (number)
aria-valuenow	Current value (number)
aria-valuetext	Human-readable version of value (e.g., "High")

```
<div role="slider"
    aria-valuemin="0"
    aria-valuemax="100"
    aria-valuenow="75"
    aria-valuetext="75% Volume">
</div>
```

4.2.2.2 Live region attributes

- **aria-live**

Indicates that an element is a live region, and defines how assertively updates to the region should be announced by screen readers.
values:
✓ "off" (default) – Screen readers ignore updates.
✓ "polite" – Announces changes when the user is idle. for less critical updates (e.g., new chat messages).
✓ "assertive" – Announces changes immediately, interrupting the user if necessary, for urgent alerts

```
<div aria-live="polite">New notification will appear here</div>
```

- **aria-busy**

Tells assistive technologies that the element is loading or updating, so screen readers should wait before announcing content changes.

values:
✓ "true" – The region is busy.
✓ "false" – The region is ready.

```
<div aria-busy="true">Loading messages...</div>
```

When set to true, screen readers will delay announcing any aria-live updates until aria-busy="false".

- **aria-relevant**

Specifies what kinds of changes in the live region should be announced (additions, removals, text updates, etc.).

values:
✓ "additions" – Announce inserted nodes (default).
✓ "removals" – Announce removed nodes.
✓ "text" – Announce changes to text.
✓ "all" – Announce all changes (combines the above).

Multiple values can be space-separated.

```
<div aria-live="polite" aria-relevant="additions removals">
  <!-- content dynamically updated -->
</div>
```

- **aria-atomic**

Controls **how much** of the live region is announced when a change occurs.

values:
✓ "true" – Announces the **entire region**, even if only a part changed.
✓ "false" (default) – Announces **only the changed part**.

```
<div aria-live="polite" aria-atomic="true">
  Message from <span>John</span>: <span>Hello</span>
</div>
```

If the message text changes from “Hello” to “How are you?”, screen readers will read:
“Message from John: How are you?”
Instead of just “How are you?”

4.2.2.3 Drag-and-Drop attributes (Depracted)

- aria-dropeffect
- aria-grabbed

4.2.2.4 Relationship attributes

- **aria-activedescendant**

Identifies the currently active child element of a composite widget. Used on: The container, not the active element. Common in combobox, listbox, grid, etc.

value:
ID of the child element.

```
<div role="listbox" aria-activedescendant="option-2">
  <div id="option-1" role="option">Item 1</div>
  <div id="option-2" role="option">Item 2</div>
</div>
```

The user’s focus remains on the container, but screen readers treat the child as active.

- **aria-controls**

Indicates that the element controls the contents of another element. Used on: Buttons, tabs, accordions.

value:
ID reference(s) of controlled elements.

```
<button aria-controls="panel1">Toggle Panel</button>
<div id="panel1">This is the panel content</div>
```

- **aria-describedby**

Provides additional descriptive text for assistive tech. Used on: Inputs, form controls, widgets.

value:
One or more ID references.

```
<input id="email" aria-describedby="email-help" />
<div id="email-help">We'll never share your email.</div>
```

Works like aria-label, but is meant for supplementary info.

- **aria-description**

Purpose: Provides a programmatic description of the element that may not be visually present. Used on: Any interactive element.

value:
String description.

```
<button aria-description="Submits the form to our servers.">Submit</button>
```

Supported only in some screen readers (JAWS, NVDA).

- **aria-details**

Links to detailed content about the element (like additional instructions or help text). Used on: Form fields, graphs, etc. More detailed than aria-describedby.

value:
ID reference.

```
<input aria-details="info" />
<div id="info">Explanation of how your data will be used.</div>
```

- **aria-flowto**

Specifies the next logical reading order (when it's not DOM order). Rarely used — behavior is screen reader-dependent.

value:
ID(s) of the next element(s) in reading order.

```
<p aria-flowto="summary">See the summary next</p>
<p id="summary">Here's a summary of the above content.</p>
```

- **aria-labelledby**

Identifies element(s) that label this one. Used on: Almost any element. Preferred over aria-label when there is visible text already on the page.

value:
ID reference(s).

```
<label id="lbl">Email Address</label>
<input aria-labelledby="lbl" />
```

- **aria-owns**

Declares element ownership when DOM structure doesn't match visual hierarchy.

value:
One or more IDs of child elements.

```
<div id="container" aria-owns="item2">
  <div id="item1">A</div>
</div>
<div id="item2">B</div>
```

Treats item2 as a logical child of container, even though it’s outside in DOM.

⚠ Use with caution. Complex and can confuse screen readers if used improperly.

The following attributes are primarily used for accessible tables, grids, and treegrids.

- **aria-colcount**

Specifies the total number of columns in a grid or table. Used on: grid, table, rowgroup.
value:
Integer (including off-screen columns).

```
<div role="grid" aria-colcount="5">...</div>
```

- **aria-colindex**

Gives the column index (1-based) of a cell. Used on: cell, gridcell, columnheader.

value:
Integer

```
<div role="gridcell" aria-colindex="2">B</div>
```

- **aria-colspan**

Indicates how many columns a cell spans across. Used on: cell, gridcell, columnheader.

value:
Integer

```
<div role="gridcell" aria-colspan="2">Merged Cell</div>
```

- **aria-rowcount**

Specifies the total number of rows in a grid or table. Used on: grid, table.

value:
Integer

```
<div role="grid" aria-rowcount="10">...</div>
```

- **aria-rowindex**

Indicates the row position (1-based index) of a cell or row. Used on: row, cell, gridcell, etc.

value:
Integer

```
    <div role="row" aria-rowindex="3">...</div>
```

- **aria-rowspan**

Indicates how many rows a cell spans vertically. Used on: gridcell, cell, rowheader.

value:
Integer

```
    <div role="gridcell" aria-rowspan="2">Merged Vertically</div>
```

The following attributes are used in sets of items (like tree views, menus, lists, etc.) to define position and structure.

- **aria-posinset**

Purpose: Describes the position of an item in a set (e.g., 3rd item in a list). Used on: treeitem, option, menuitem, listitem, etc.

value: Integer

```
  <li role="treeitem" aria-posinset="2">Node B</li>
```

- **aria-setsize**

Indicates the total number of items in the current set. Used on: Same roles as aria-posinset. Often paired with aria-level, aria-posinset, and aria-expanded.

value:
Integer

```
  <li role="treeitem" aria-setsize="5">Node A</li>
```

Global ARIA attributes

Some states and properties apply to all HTML elements regardless of whether an ARIA role is applied. These are defined as "Global" attributes. Global states and properties are supported by all roles and base markup elements.
Many of the above attributes are global, meaning they can be included on any element unless specifically disallowed:

- aria-atomic
- aria-busy
- aria-controls
- aria-current
- aria-describedby
- aria-description
- aria-details
- aria-disabled
- aria-dropeffect
- aria-errormessage
- aria-flowto
- aria-grabbed
- aria-haspopup
- aria-hidden
- aria-invalid
- aria-keyshortcuts
- aria-label
- aria-labelledby
- aria-live
- aria-owns
- aria-relevant
- aria-roledescription

https://developer.mozilla.org/en-US/docs/Web/Accessibility/ARIA/Reference/Attributes
https://www.w3.org/TR/wai-aria-1.3/#state_prop_def

5 Color and Contrast

Contrast and colour use are vital to accessibility. Users, including users with visual disabilities, must be able to perceive content on the page

5.1 Defining colours

Colours can be defined in a few ways. For example, this shade of blue may commonly be defined in three different ways in webpage styles:

- **rgb (97, 97, 255):** The amount of red, green, and blue that form a color are each presented as a number between 0 and 255.
- **#6161FF:** This is a "hexadecimal" format where the red/green/blue values are presented as a combination of six letters or numbers. Typically called "Hex," this is a very common format in webpages.
- **hsl (240, 100%, 69%):** Hue, saturation, and lightness map more closely to the way people perceive colours. Changing the "lightness" of a color will change its contrast ratio to another color.

Alpha, which is the opacity or transparency of a color, will also impact contrast. Alpha is presented as a number between 0 (completely transparent) and 1 (completely opaque). Reducing the alpha for an element will reduce its contrast because you are allowing an underlying color to bleed through.

5.2 Contrast Ratio

Colour contrast ratio is a numerical value that expresses the difference in luminance (brightness) between two colours — usually between text (foreground) and its background. It plays a critical role in accessibility, especially for users with low vision or colour blindness.

The visual presentation of text and images of text has a contrast ratio of at least 4.5:1, except for the following:

WCAG Guidelines (from WCAG 2.1/2.2)-Level AA

Text Type	**Minimum Contrast Ratio Required**
Normal text	4.5:1
Large text	3:1
Icons/UI components	3:1

5.3 Tools for testing

- WebAIM Contrast Checker
- axe DevTools (browser extension)
- Color Contrast Analyzer