Accessibility

How HTML can be used to ensure maximum accessibility

[**What is Accessibility?**](#_vgfbrgyrn4r7) **1**

[**Good Semantics**](#_s98qwa6a458y) **2**

[Text content](#_w6p6jaquvkq7) 2

[Using clear language](#_7nlqbzo95fo5) 3

[Page layouts](#_x8hgwmfd6xxz) 3

[UI controls](#_sktz43rgziv9) 5

[Meaningful text labels](#_6hqfjhbqq6z8) 7

[Forms](#_xd9dzt5eq52r) 7

[**Accessible Data Tables**](#_cw53i9jyy3ec) **8**

[Text Alternatives](#_r7fqhczg58k6) 10

[Empty Alt Attributes](#_1il4z4dunfse) 11

# What is Accessibility?

Accessibility is the practice of making your websites usable by as many people as possible. We traditionally think of this as being about people with disabilities, but the practice of making sites accessible also benefits other groups such as those using mobile devices, or those with slow network connections.

You might also think of accessibility as treating everyone the same, and giving them equal opportunities, no matter what their ability or circumstances. Just as it is wrong to exclude someone from a physical building because they are in a wheelchair (modern public buildings generally have wheelchair ramps or elevators), it is also not right to exclude someone from a website because they have a visual impairment. We are all different, but we are all human, and therefore have the same human rights.

Accessibility is the right thing to do. Providing accessible sites is part of the law in some countries, which can open up some significant markets that otherwise would not be able to use your services or buy your products.

Building accessible sites benefit everyone:

* Semantic HTML, which improves accessibility, also improves SEO, making your site more findable.
* Caring about accessibility demonstrates good ethics and morals, which improves your public image.
* Other good practices that improve accessibility also make your site more usable by other groups, such as mobile phone users or those on low network speed. In fact, everyone can benefit from many such improvements.
* Did we mention it is also the law in some places?

As you learn more about HTML — read more resources, look at more examples, etc. — you'll keep seeing a common theme: the importance of using semantic HTML (sometimes called POSH, or Plain Old Semantic HTML). This means using the correct HTML elements for their intended purpose as much as possible.

You might wonder why this is so important. After all, you can use a combination of CSS and JavaScript to make just about any HTML element behave in whatever way you want. For example, a control button to play a video on your site could be marked up like this:

| **<div>Play video</div>** |
| --- |

But as you'll see in greater detail later on, it makes sense to use the correct element for the job:

| **<button>Play video</button>** |
| --- |

# Good Semantics

We've already talked about the importance of proper semantics, and why we should use the right HTML element for the job. This cannot be ignored, as it is one of the main places that accessibility is badly broken if not handled properly.

Out there on the web, the truth is that people do some very strange things with HTML markup. Some abuses of HTML are due to legacy practices that have not been completely forgotten, and some are just plain ignorance. Whatever the case, you should replace such bad code.

Sometimes you are not in the position to get rid of lousy markup — your pages might be generated by some kind of server-side framework over which you don't have full control, or you might have third party content on your page (such as ad banners) over which you have no control.

The goal isn't "all or nothing"; every improvement you can make will help the cause of accessibility.

## Text content

One of the best accessibility aids a screen reader user can have is an excellent content structure with headings, paragraphs, lists, etc. An excellent semantic example might look something like the following:

| <h1>My heading</h1>  <p>This is the first section of my document.</p>  <p>I'll add another paragraph here too.</p>  <ol>  <li>Here is</li>  <li>a list for</li>  <li>you to read</li> </ol>  <h2>My subheading</h2>  <p>This is the first subsection of my document. I'd love people to be able to find this content!</p>  <h2>My 2nd subheading</h2>  <p>This is the second subsection of my content. I think it is more interesting than the last one.</p> |
| --- |

## Using clear language

The language you use can also affect accessibility. In general, you should use clear language that is not overly complex and doesn't use unnecessary jargon or slang terms. This not only benefits people with cognitive or other disabilities; it benefits readers for whom the text is not written in their first language, younger people ... everyone, in fact! Apart from this, you should try to avoid using language and characters that don't get read out clearly by the screen reader. For example:

* Don't use dashes if you can avoid it. Instead of writing 5–7, write 5 to 7.
* Expand abbreviations — instead of writing Jan, write January.
* Expand acronyms, at least once or twice, then use the tag to describe them.

## Page layouts

In the bad old days, people used to create page layouts using HTML tables — using different table cells to contain the header, footer, sidebar, main content column, etc. This is not a good idea because a screen reader will likely give out confusing readouts, especially if the layout is complex and has many nested tables. **[🤮](https://emojipedia.org/face-vomiting/)**

| <table width="1200">  <!-- main heading row -->  <tr id="heading">  <td colspan="6">   <h1 align="center">Header</h1>   </td>  </tr>  <!-- nav menu row -->  <tr id="nav" bgcolor="#ffffff">  <td width="200">  <a href="#" align="center">Home</a>  </td>  <td width="200">  <a href="#" align="center">Our team</a>  </td>  <td width="200">  <a href="#" align="center">Projects</a>  </td>  <td width="200">  <a href="#" align="center">Contact</a>  </td>  <td width="300">  <form width="300">  <input type="search" name="q" placeholder="Search query" width="300">  </form>  </td>  <td width="100">  <button width="100">Go!</button>  </td>  </tr>  <!-- spacer row -->  <tr id="spacer" height="10">  <td>   </td>  </tr>  <!-- main content and aside row -->  <tr id="main">  <td id="content" colspan="4" bgcolor="#ffffff">   <!-- main content goes here -->  </td>  <td id="aside" colspan="2" bgcolor="#ff80ff" valign="top">  <h2>Related</h2>   <!-- aside content goes here -->   </td>  </tr>  <!-- spacer row -->  <tr id="spacer" height="10">  <td>   </td>  </tr>  <!-- footer row -->  <tr id="footer" bgcolor="#ffffff">  <td colspan="6">  <p>(c)Copyright 2050 by nobody. All rights reversed.</p>  </td>  </tr>  </table> |
| --- |

If you try to navigate this using a screen reader, it will probably tell you that there's a table to be looked at (although some screen readers can guess the difference between table layouts and data tables). You'll then likely (depending on which screen reader you're using) have to go down into the table as an object and look at its features separately, then get out of the table again to carry on navigating the content.

Table layouts are a relic of the past — they made sense back when CSS support was not widespread in browsers, but now they just create confusion for screen reader users. Additionally, their source code requires more markup, which makes them less flexible and more difficult to maintain. You can verify these claims by comparing your previous experience with a more modern website structure example, which could look something like this:

| <header>  <h1>Header</h1> </header>  <nav>  <!-- main navigation in here --> </nav>  <!-- Here is our page's main content --> <main>   <!-- It contains an article -->  <article>  <h2>Article heading</h2>   <!-- article content in here -->  </article>   <aside>  <h2>Related</h2>   <!-- aside content in here -->  </aside>  </main>  <!-- And here is our main footer that is used across all the pages of our website -->  <footer>  <!-- footer content in here --> </footer> |
| --- |

If you try the more modern structure example with a screen reader, you'll see that the layout markup no longer gets in the way and confuses the content readout. It is also much leaner and smaller in terms of code size, which means easier to maintain code, and less bandwidth for the user to download (particularly prevalent for those on slow connections).

## UI controls

By UI controls, we mean the main parts of web documents that users interact with — most commonly buttons, links, and form controls. In this section, Let’s look at the basic accessibility concerns to be aware of when creating such controls.

| <!-- create a page to test here - > |
| --- |

You can then press Enter/Return to follow a focused link or press a button (we've included some JavaScript to make the buttons alert a message), or start typing to enter text in a text input. Other form elements have different controls; for example, the [<select>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/select) element can have its options displayed and cycled between using the up and down arrow keys.

You essentially get this behaviour for free, just by using the appropriate elements, e.g.

| <h1>Links</h1>  <p>This is a link to <a href="https://www.mozilla.org">Mozilla</a>.</p>  <p>Another link, to the <a href="https://developer.mozilla.org">Mozilla Developer Network</a>.</p>  <h2>Buttons</h2>  <p>  <button data-message="This is from the first button">Click me!</button>  <button data-message="This is from the second button">Click me too!</button>  <button data-message="This is from the third button">And me!</button> </p>  <h2>Form</h2>  <form>  <div>  <label for="name">Fill in your name:</label>  <input type="text" id="name" name="name">  </div>  <div>  <label for="age">Enter your age:</label>  <input type="text" id="age" name="age">  </div>  <div>  <label for="mood">Choose your mood:</label>  <select id="mood" name="mood">  <option>Happy</option>  <option>Sad</option>  <option>Angry</option>  <option>Worried</option>  </select>  </div> </form> |
| --- |

This means using links, buttons, form elements, and labels appropriately (including the [<label>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/label) element for form controls).

## Meaningful text labels

UI control text labels are very useful to all users, but getting them right is particularly important to users with disabilities.

You should make sure that your button and link text labels are understandable and distinctive. Don't just use "Click here" for your labels, as screen reader users sometimes get up a list of buttons and form controls.

Make sure your labels make sense out of context, read on their own, as well as in the context of the paragraph they are in. For example, the following shows an example of good link text:

| <p>Orcas are really awesome creatures. <a href="orcas.html">Find out more about orcas</a>.</p> |
| --- |

but this is bad link text **💩**:

| <p>Orcas are really awesome creatures. To find more out about orcas, <a href="orcas.html">click here</a>.</p> |
| --- |

## Forms

Form labels are also important for giving you a clue about what you need to enter into each form input. The following seems like a reasonable enough example:

| Fill in your name: <input type="text" id="name" name="name"> |
| --- |

However, this is not so useful for disabled users. There is nothing in the above example to associate the label unambiguously with the form input and make it clear how to fill it in if you cannot see it. If you access this with some screen readers, you may only be given a description along the lines of "edit text."

The following is a much better example:

| <div>  <label for="name">Fill in your name:</label>  <input type="text" id="name" name="name"> </div> |
| --- |

With code like this, the label will be clearly associated with the input; the description will be more like "Fill in your name: edit text." In the Forms module, we’ll discuss this further.

## 

# Accessible Data Tables

The good news is when we did tables in the last module, we already used accessibility principles when designing our table. However, it’s a good idea to understand what we did, and why we did it.

A basic data table can be written with very simple markup, for example:

| <table>  <tr>  <td>Name</td>  <td>Age</td>  <td>Gender</td>  </tr>  <tr>  <td>Gabriel</td>  <td>13</td>  <td>Male</td>  </tr>  <tr>  <td>Elva</td>  <td>8</td>  <td>Female</td>  </tr>  <tr>  <td>Freida</td>  <td>5</td>  <td>Female</td>  </tr> </table> |
| --- |

But this has problems — there is no way for a screen reader user to associate rows or columns together as groupings of data. To do this, you need to know what the header rows are and if they are heading up rows, columns, etc. This can only be done visually for the above table.

Now let’s look at another example. Notice the table headers are defined using the <th> tags. By using the scope attribute you can specify if they’re rows or columns. The caption attribute acts as alt text for a table, which would give a screen reader user a quick summary of the table’s contents.

| <table>  <caption>A summary of the UK's most famous punk bands</caption>  <thead>  <tr>  <th scope="col">Band</th>  <th scope="col">Year formed</th>  <th scope="col">No. of Albums</th>  <th scope="col">Most famous song</th>  </tr>  </thead>  <tbody>  <tr>  <th scope="row">Buzzcocks</th>  <td>1976</td>  <td>9</td>  <td>Ever fallen in love (with someone you shouldn't've)</td>  </tr>  <tr>  <th scope="row">The Clash</th>  <td>1976</td>  <td>6</td>  <td>London Calling</td>  </tr>  <tr>  <th scope="row">The Damned</th>  <td>1976</td>  <td>10</td>  <td>Smash it up</td>  </tr>  <tr>  <th scope="row">Sex Pistols</th>  <td>1975</td>  <td>1</td>  <td>Anarchy in the UK</td>  </tr>  <tr>  <th scope="row">Sham 69</th>  <td>1976</td>  <td>13</td>  <td>If The Kids Are United</td>  </tr>  <tr>  <th scope="row">Siouxsie and the Banshees</th>  <td>1976</td>  <td>11</td>  <td>Hong Kong Garden</td>  </tr>  <tr>  <th scope="row">Stiff Little Fingers</th>  <td>1977</td>  <td>10</td>  <td>Suspect Device</td>  </tr>  <tr>  <th scope="row">The Stranglers</th>  <td>1974</td>  <td>17</td>  <td>No More Heroes</td>  </tr>  </tbody>  <tfoot>  <tr>  <th scope="row" colspan="2">Total albums</th>  <td colspan="2">77</td>  </tr>  </tfoot> </table> |
| --- |

## Text Alternatives

Whereas textual content is inherently accessible, the same cannot necessarily be said for multimedia content — image and video content cannot be seen by visually-impaired people, and audio content cannot be heard by hearing-impaired people. We cover video and audio content in detail in the [Accessible multimedia](https://developer.mozilla.org/en-US/docs/Learn/Accessibility/Multimedia), but for this article we'll look at accessibility for the humble [<img>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/img) element.

We have a simple example written up which features four copies of the same image:

| <img src="dinosaur.png">  <img src="dinosaur.png"  alt="A red Tyrannosaurus Rex: A two legged dinosaur standing upright like a human, with small arms, and a large head with lots of sharp teeth.">  <img src="dinosaur.png"  alt="A red Tyrannosaurus Rex: A two legged dinosaur standing upright like a human, with small arms, and a large head with lots of sharp teeth."  title="The Mozilla red dinosaur">  <img src="dinosaur.png" aria-labelledby="dino-label">  <p id="dino-label">The Mozilla red Tyrannosaurus Rex: A two legged dinosaur standing upright like a human, with small arms, and a large head with lots of sharp teeth.</p> |
| --- |

The first image, when viewed by a screen reader, doesn't really offer the user much help — VoiceOver for example reads out "/dinosaur.png, image". It reads out the filename to try to provide some help. In this example the user will at least know it is a dinosaur of some kind, but often files may be uploaded with machine-generated file names (e.g. from a digital camera) and these file names would likely provide no context to the image's content.

When a screen reader encounters the second image, it reads out the full alt attribute — "A red Tyrannosaurus Rex: A two legged dinosaur standing upright like a human, with small arms, and a large head with lots of sharp teeth.".

This highlights the importance of not only using meaningful file names in case so-called **alt text** is not available, but also making sure that alt text is provided in alt attributes wherever possible. Note that the contents of the alt attribute should always provide a direct representation of the image and what it conveys visually. Any personal knowledge or extra description shouldn't be included here, as it is not useful for people who have not come across the image before.

One thing to consider is whether your images have meaning inside your content, or whether they are purely for visual decoration, and thus have no meaning. If they are decorative, it is better to write an empty text as a value for alt attribute or to just include them in the page as CSS background images.



If you do want to provide extra contextual information, you should put it in the text surrounding the image, or inside a title attribute, as shown above. In this case, most screen readers will read out the alt text, the title attribute, and the filename. In addition, browsers display title text as tooltips when moused over.

### Empty Alt Attributes

| <h3>  <img src="article-icon.png" alt="">  Tyrannosaurus Rex: the king of the dinosaurs </h3> |
| --- |

There may be times where an image is included in a page's design, but its primary purpose is for visual decoration. You'll notice in the code example above that the image's alt attribute is empty — this is to make screen readers recognize the image, but not attempt to describe the image (instead they'd just say "image", or similar).

The reason to use an empty alt instead of not including it is because many screen readers announce the whole image URL if no alt is provided. In the above example, the image is acting as a visual decoration to the heading it's associated with. In cases like this, and in cases where an image is only decoration and has no content value, you should include an empty alt in your img elements. Another alternative is to use the aria [role](https://developer.mozilla.org/en-US/docs/Web/Accessibility/ARIA/Roles) attribute [role="presentation"](https://developer.mozilla.org/en-US/docs/Web/Accessibility/ARIA/Roles/presentation_role) as this also stops screen readers from reading out alternative text.