



Web Design Accessibility

Course Handbook

Welcome

Welcome to the *Web Design Accessibility* Course Handbook.

Building inclusive websites is a wonderful topic. Not only do we make the world a better, more accessible place, but it also improves the experience for everyone, opens our businesses up to a whole new source of customers and adds a valuable skill to our arsenal as web developers.



This handbook serves as a reference guide to support the video content. It will reinforce what you learn in the lessons but do not forget that you can revisit the lessons in full at any time, too.

Best of luck in your accessibility journey.

Chris

Tools

Using tools and automated reports will help you catch and fix accessibility issues faster. Here are the top tools that I recommend.

Lighthouse

Lighthouse is an audit tool built into the Chrome web browser. You can access it by going to the Developer Tools (F12), audits and then start a new audit. As well as an accessibility report, it will also provide information on performance and SEO.

VoiceOver

VoiceOver is a screen reader built into Mac and iOS devices. If you have a Mac, you already have a screen reader that you can use to test your website. Use Cmd + F5 to activate it.

VoiceOver will offer you a tutorial when you first start it and provide hints as to how to use it as you go along.

Windows screen readers

There are multiple options for screen readers on Windows devices.

- *JAWS* has historically been the most popular screen reader but comes with a high price tag. You can download an evaluation version from the Freedom Scientific website.
- *NVDA* is rising in popularity because it is available for free. You can download it from the NV Access website.
- Newer versions of Windows include a built-in screen reader from Microsoft known as *Narrator*.

Other screen readers

Android comes with a built-in screen reader named *TalkBack*.

There are multiple screen readers available for Linux including *Orca* and *Linux Screen Reader*.

WAVE

The *WAVE Web Accessibility Evaluation Tool* is a browser-based tool that allows you to enter a website URL, and the tool will then load the page with a report on accessibility issues and an annotated version of the website.

WAVE can be found at <https://wave.webaim.org/>.

W3C Validator

The *W3C Markup Validation Service* will check whether your HTML is valid and flag any issues. Writing valid HTML is important because it allows screen readers and browsers to interpret your website correctly.

You can access the tool at <https://validator.w3.org/>.

Disabilities

In this section, we will look at some of the conditions that users of our websites may have so that we can keep these in mind while designing.

Sight

Level of sight varies greatly from 20/20 vision to complete blindness. While some people cannot see at all, others may have blind spots or blurry vision that requires them to zoom the screen and make everything larger.


Colour blindness also affects people differently with various combinations missing. The most common of which is red/green colour blindness.

Hearing

If you are using multimedia content, not everyone will be able to hear. A user may be deaf, experience limited hearing, or be living with a condition such as tinnitus.

Motor

Some users have limited motor functions. For example, someone with Parkinson's may struggle to accurately control a mouse and therefore will be struggling on clicking on a small link or tap target and may end up hitting a nearby element.



Other users may have no ability to control a mouse at all and may navigate via the keyboard or using a specially adapted device.

For this reason, it is vital to check that our websites can be navigated using a keyboard as well as a mouse.

Cognitive

Not all disabilities are physical. Some users may have learning difficulties, memory problems, autism or a range of other conditions.

For this reason, we want to make sure that our designs are easy to understand, use simple language where possible and use clear and consistent call-to-actions (CTAs).

Inclusive design

In this section, we will look at some best practice for technical implementations.

Semantic HTML

HTML4 relied on non-descriptive `div` and `span` tags. HTML5 addressed this issue by adding a range of meaningful tags that made it easier for the browser to understand the page. Using these tags provides more context for browsers and screen readers.

Examples include:

- `header` and `footer`
- `nav`
- `main`
- `article`
- `aside`
- `figure` and `figcaption`

Text

Pages should have a single `h1` tag and a correct heading hierarchy. That is to say that headings should be used in order with no “missing” numbers. It should not jump from `h2` to `h4`, for example.

Text should have a strong contrast with the background and sufficient line spacing to make it easy to read. 1.5 is a good starting point.

Use `p` tags instead of `br` tags.

Links

Links should take the user to a different page. Buttons should perform an action on the current page. Ensure you use the correct element: you can always use CSS to style it as you wish.

Always ensure you indicate something is a link using more than just colour. Traditionally, we use an underline.

Text should have a descriptive anchor and not just “click here”.

Colour


Text should have sufficient colour contrast with the background to be easily reasonable. You can use a colour contrast checker to validate this.

For AA compliance:

- 4.5:1 for standard text
- 3:1 for large text

For AAA compliance:

- 7:1 for standard text
- 4.5:1 for large text



Colour should not be used as an indicator alone. As some users may be colour blind, we must ensure there are multiple pieces of context. For example, links should also be underlined. And red error messages should also include an error icon.

Images

Images must always include an `alt` attribute. If the image is relevant, this should contain a description of the image. If the image is purely for display purposes, the attribute should be blank.

Avoid using text in images where possible as accessibility tools are unable to read the embedded text. If you must use it, ensure it is described in the alt attribute and uses a sufficient colour contrast.

Audio & Video

Multimedia content should not autoplay and, where possible, avoid flashing imagery.

Player controls should be tested to ensure they work with the keyboard.

Videos should provide closed captions and audio should come with a transcript.

Forms

Every input should have a label.

The `placeholder` attribute is not a replacement for a label. You can either skip this attribute or use it as an additional hint. For example, in a date field, it could indicate the required format (DD/MM/YYYY).

Avoid using the `disabled` attribute on buttons as doing so removes it from the tab index. The result is that screen readers may not be able to find the button, and thus the user will become confused.

Ensure your error handling is tested with a screen reader. It should be clear what fields are in error and what the user should do to fix them.

ARIA

Accessible Rich Internet Applications (ARIA) is a standard set of practices and attributes created by the W3C to allow us to make Web 2.0 applications more accessible.

ARIA is not designed to replace semantic HTML: this is always your first port of call. But, when you are unable to describe the website using semantic tags alone, ARIA allows you to add additional context.

Use the `role` attribute to indicate what an element represents. For example:

- `role="navigation"`
- `role="tablist"`

Avoid redundancy: if you are using the `nav` tag, you do not need to add the `role` attribute.

ARIA-prefixed attributes include:

- `aria-label` adds text to an element, for example, a hamburger menu icon
- `aria-hidden` removes anything from the DOM that is not relevant to a screen reader
- `aria-checked` allows you to simulate a checkbox input
- `aria-disabled` allows you to indicate a button or control is disabled without actually disabling it
- `aria-labelledby` and `aria-describedby` add additional context to form elements

If you are updating content interactively after the page has loaded, you should place it inside an `aria-live` region so that a screen reader knows to announce these changes. You can also use `role="alert"` to achieve the same thing.

