Introduction to semantics

- Assistive technology
- Affordances
- Screen readers

You've seen how to make a site accessible to users who can't use a mouse or pointing device — whether due to physical impairment, a technology issue, or personal preference — by addressing keyboard-only use. While it requires some care and thought, it's not a huge amount of work if you plan it from the beginning. Once that basic work is done, you're a long way down the path to a fully accessible and more polished site.

In this lesson, we'll build on that work and get you thinking about other accessibility factors, such as how to build websites to support users like Victor Tsaran, who can't see the screen.

First, we'll get some background on *assistive technology*, the general term for tools like screen readers that help users with impairments that can keep them from accessing information.

Next, we'll look at some general user experience concepts, and build on those to take a deeper dive into the experience of users of assistive technology.

Finally, we'll see how to use HTML effectively to create a good experience for these users, and how it overlaps quite a bit with the way we addressed focus earlier.

Assistive technology

Assistive technology is an umbrella term for devices, software, and tools that help any person with a disability complete a task. In the broadest sense this could be something low-tech like a crutch for walking or a magnifying glass for reading, or something high-tech like a robotic arm or image recognition software on a smartphone.

Assistive technology examples including crutch magnifying glass and robotic prosthesis.

Assistive technology can include something as general as browser zoom, or as specific as a custom-designed game controller. It can be a separate physical device like a braille display, or be implemented completely in software like voice control. It can be built-in to the operating system

like some screen readers, or it can be an add-on like a Chrome extension.

> *More assistive technology examples including browser zoom braille display and voice control.*

The line between assistive technology and technology in general is blurry; after all, all technology is meant to assist people with some task or another. And technologies can often move into and out of the "assistive" category.

For example, one of the earliest commercial speech synthesis products was a talking calculator for the blind. Now speech synthesis is all over the place, from driving directions to virtual assistants. Conversely, technology that was originally general-purpose often finds an assistive use. For example, people with low vision may use their smartphone's camera zoom to get a better look at something small in the real world.

In the context of web development, we must consider a diverse range of technologies. People may interact with your website using a screen reader or braille display, with a screen magnifier, via voice control, using a switch device, or with some other form of assistive technology that adapts the page's default interface to create a more specific interface that they can use.

Many of these assistive technologies rely on *programmatically expressed semantics* to create an accessible user experience, and that's what most of this lesson is about. But before we can explain programmatically expressed semantics, we need to talk a bit about *affordances*.

Affordances

When we use a man-made tool or device, we typically look to its form and design to give us an idea of what it does and how it works. An *affordance* is any object that offers, or affords, its user the opportunity to perform an action. The better the affordance is designed, the more obvious or intuitive its use.

A classic example is a kettle or teapot. You can easily recognize that you should pick it up by the handle, not the spout, even if you've never seen a teapot before.

> *A teapot with handle and spout.*

That's because the affordance is similar to those you have seen on many other objects -- watering pots, beverage pitchers, coffee mugs, and so on. You probably *could* pick up the pot by the spout, but your experience with similar affordances tells you the handle is the better option.

In graphical user interfaces, affordances represent actions we can take, but they can be ambiguous because there is no physical object to interact with. GUI affordances are thus specifically designed to be unambiguous: buttons, check boxes, and scroll bars are meant to convey their usage with as little training as possible.

For example, you might paraphrase the use of some common form elements (affordances) like this:

- Radio buttons — "I can choose one of these options."
- Check box — "I can choose 'yes' or 'no' to this option."
- Text field — "I can type something into this area."
- Dropdown — "I can open this element to display my options."

You are able to draw conclusions about these elements *only because you can see them*. Naturally, someone who can't see the visual clues provided by an element can't comprehend its meaning or intuitively grasp the value of the affordance. So we must make sure that the information is expressed flexibly enough to be accessed by assistive technology that can construct an alternative interface to suit its user's needs.

This non-visual exposure of an affordance's use is called its *semantics*.

Screen readers

One popular type of assistive technology is the *screen reader*, a program that enables visually impaired people to use computers by reading screen text aloud in a generated voice. The user can control what is read by moving the cursor to a relevant area with the keyboard.

We asked Victor Tsaran to explain how, as a blind person, he accesses the web using a the built-in screen reader on OS X, called VoiceOver. See this video of Victor using VoiceOver.

Now, it's your turn to try using a screen reader. Here is a page with *ChromeVox Lite*, a minimal but functioning screen reader written in JavaScript. The screen is purposefully blurred to simulate a low-vision experience and force the user to complete the task with a screen reader. Of course, you'll need to use the Chrome browser for this exercise.

ChromeVox lite demo page

You can use the control panel at the bottom of the screen to control the screen reader. This screen reader has very minimal functionality, but you can explore the content using the `Previous` and `Next` buttons, and you can click things using the `Click` button.

Try using this page with ChromeVox lite enabled to get a feel for screen reader use. Think about the fact that a screen reader (or other assistive technology) actually creates a complete alternate user experience for the user based on the programmatically expressed semantics. Instead of a visual interface, the screen reader provides an audible interface.

Notice how the screen reader tells you some information about each interface element. You should expect a well-designed reader to tell you all, or at least most, of the following information about the elements it encounters.

- The element's *role* or type, if it is specified (it should be).
- The element's *name*, if it has one (it should).
- The element's *value*, if it has one (it may or may not).
- The element's *state*, e.g., whether it is enabled or disabled (if applicable).

The screen reader is able to construct this alternate UI because the native elements contain built-in accessibility metadata. Just as the rendering engine uses the native code to construct a visual interface, the screen reader uses the metadata in the DOM nodes to construct an accessible version, something like this.

> *A screen reader uses the DOM to create accessible nodes.*

□□□ #3
□ □□□ □□□ 2026-10-31 22:15:25 UTC
□ cocobook-sync □□□ 2026-11-04 17:38:00 UTC