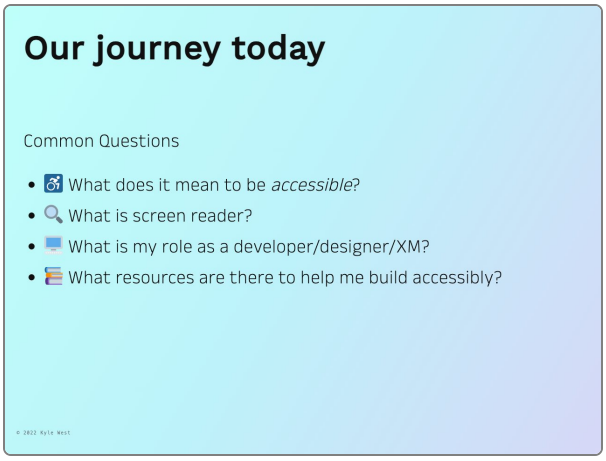
Accessibility and why you should give a DOM

Abstract

Learn about accessibility in design and development. We will cover the history of accessibility in the tech space and the implications this has on building software today. Accessibility is as much of a usability problem as it is a technical feat. We will address best practices, common pitfalls, and resources around building products that are usable for everyone.

Slides and Presentation Notes

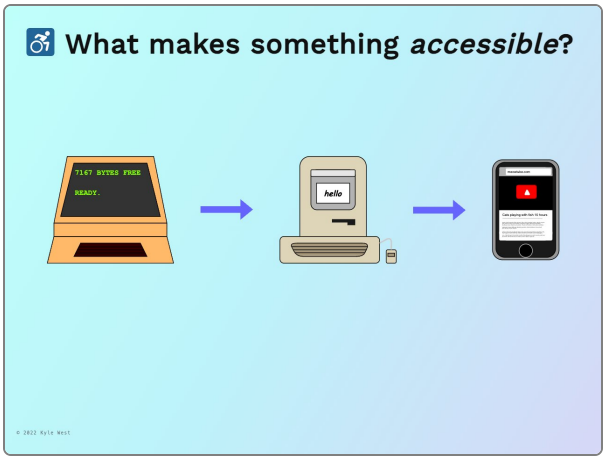
Hello everyone, thank you for coming. My name is Kyle West, and today we will be discussing how to design and build accessible applications. This subject is very dear to me, and I hope you have a chance today to see the tremendous amount of good you can do as you build your products.



There is a lot that goes into this subject, and today we are going to cover a variety of topics. These are a few questions you may have already. We will answer them, of course. But I actually hope by the end you leave with more questions than you came with. This is such a rich topic. I have spent the last 3 years actively researching and I still have plenty of questions all the time.

You have probably seen the abbreviated version of Accessibility from time to time. Some people pronounce it *alley* like an *alley-way*. I like to pronounce it *ally*, since we ought to be a support for people.

Now that that's out of the way, let's get going.

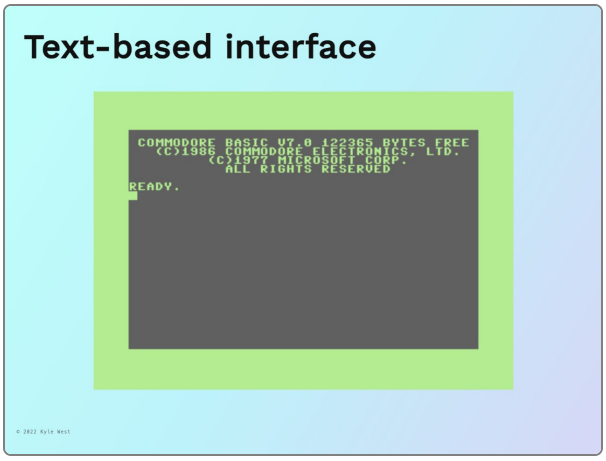


In order to answer the question, "what does it mean to be accessible?" we need to have a conversation about context. The evolution of interfaces has had a drastic effect on how accessibility concerns are addressed.

- Text Based
- Graphical
- Mobile

1977 was a monumental year for computing. By this time, IBM had established that computers were for the modern business folk, but it wasn't until this year that computers really started to be advertized to the average person.

What made this technology accessible in the general sense, is that people could afford to own a machine while having a median salary.



Although most of those machines from 1977 had games developed for them, the primary interface people were introduced to was text only. The PET didn't even have any graphics capabilities (the character set was stored in ROM and could not be modified)!

Text in - and text out. Compared to today's standards, this sounds really primitive, but at that time it was a world of difference from punch cards.

You could tell the computer in *almost* english to do something, and it would do it.

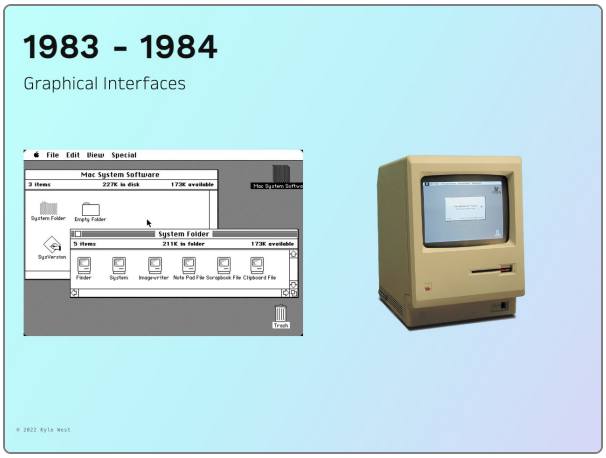
As the markets grew and more people adopted technology into their homes and work places, the need to support those of differing abilities arose.

For the visually impaired, one option was VersaBraille (or EuroBraille) retailing around $3000. In 1986 IBM created the first screen reader called, "Screen Reader". Both of these were easy to use because they could plug into a PC or mainframe just like any other terminal could do.

I should also note that in 1982 SAM (Software Automatic Mouth) was created and sold for $200 (C64, Apple II, Atari 8-bit). This later became MacInTalk, which would eventually influence the first version of VoiceOver for the Mac.

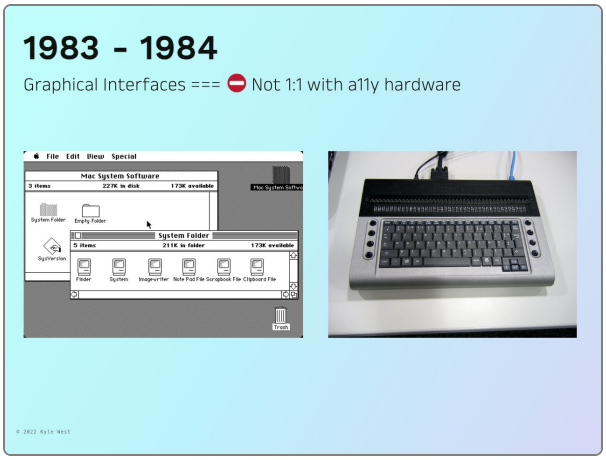
What made these really special in my mind, is that they were almost a 1:1 mapping of what the visual user had. Visually it was "text-in-text-out". For VersaBraille, it was "key-in-braille-out" and for the Screen Reader it was "key-in-sound-out". What one person could do, any other could too (with maybe a little patience).

This was *enabling* because those with differing visual abilities were made equal, with a little help from technology.



Although Douglas Engelbart demonstrated the first concept of a graphical user interface in 1968, GUIs were mostly an academic endeavor until 1983 when Apple released the Lisa and then in the following year the (much more simple) Macintosh.

This graphical interface allowed a new dimension of interaction and expression that was not possible with simple text.



However, we lost that parity we had with text based interfaces. This meant that accessibility could not entirely be handled by hardware.

It would have to be a software solution.

1985-2005

A need for standards and regulation emerged

- 1990: The Americans with Disabilities Act (ADA)
- 1994: Birth of the Internet and mainstream adoption of HyperText
- 1995: Windows 95 was the first OS to come with built in accessibility features
- 1996: pwWebSpeak by Productivity Works **first web browser with built in text to speech**
- 1998: Rehabilitation Act Section 508 passed, requiring all government services to be accessible
- 1999: World Wide Web Consortium (W3C) released **Web Content Accessibility Guidelines**.
- 2000: Microsoft Windows 2000 released, with an on screen keyboard option and the ability to translate text to speech for illiterate or blind users (5).
- 2005: OSX 10.4 (Tiger) released with VoiceOver included, though Windows SR was still much better until around 2008

In these years we started to see a lot of legislative growth in this space, and along with it - companies began to become more responsible with their products. Two of these I would like to call out is that in 1996 we saw the first web-focused screen reader! And in 1999 WC3 released the WCAG, which we will talk about more later.

There are many things I left out, including standards for media captioning, automated Kiosk support for things like ATMs, etc. The main deal is that software started to close the gap that hardware could not solve. Companies and Governments started to take responsibility for their applications.

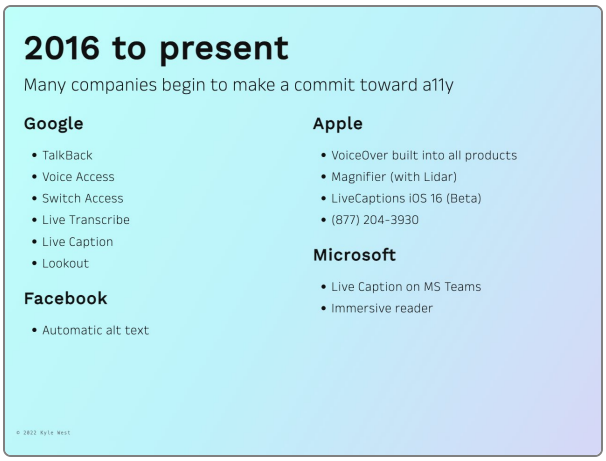
SOURCES:

- https://cs.stanford.edu/people/eroberts/courses/soco/projects/2005-06/accessibility/firstwave.html
- https://www.afb.org/aw/17/6/15322

2007-2011

Smart Phones and gesture based interactions

- 2007: Apple releases iPhone. I don't have to tell you how big of a deal that was for the tech space, but I do want to point out that touch devices were as big of a jump from GUIs as GUIs were from text based interfaces.
- 2009: Apple adds VoiceOver with the iPhone 3GS - the first touch screen native SR, and then later added gesture support for the Mac too
- 2009: Google releases Android, and later that year TalkBack
- 2010: Oratio, a mobile SR for the BlackBerry Curve was released
- 2011: Some creators of Oratio split off and wrote MobileAccessibility suite for Android



Google

- TalkBack - the native screen reader
- VoiceAccess - navigate and control your phone with voice commands
- SwitchAccess - navigate using peripheral hardware
- Uses AI heavily to make their products integral to every day life
    - Live Transcribe: live speech to text
    - Live Caption: live text to speech for content on screen (including phone calls)
    - Lookout: point your phone at an object and it will tell you what it is, or what text is on it

Facebook

- One of the pioneers in using OCR to caption images

Apple

- Since 2011 all apple products ship with VoiceOver (my favorite of the SRs, personally)
- Magnifier (if your phone comes with Lidar) will read out objects recognized in the camera
- Dedicated number for Apple VoiceOver support with experts trained on the a11y experience

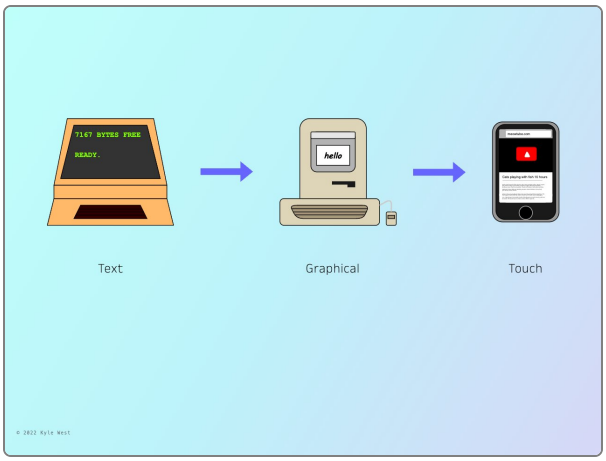
Microsoft

- Like everyone else, you can live caption your video calls with MS Teams
- Immersive Reader is specifically designed to help people with different comprehension and capabilities read text on the page
STORY TIME

In 2016 there was Robles v. Domino’s Pizza, LLC. A man named Guillermo Robles sued Dominos for not being able to use the website or mobile app with a screen reader. In 2019 Domino's petitioned the need for ADA compliance but was denied. It wasn't until June of 2022 that the case concluded - with a settlement.

It is important to note that the Department of Justice does not enforce standards (WCAG), but it does enforce the ADA which prohibits discrimination on the basis of disability. Unless of course, if you are a government website, then you are required to comply to WCAG explicitly under 508.

In 2020 there was Sierra vs General Motors Company. Apparently you can't purchase a car when using a screen reader this is still on going.



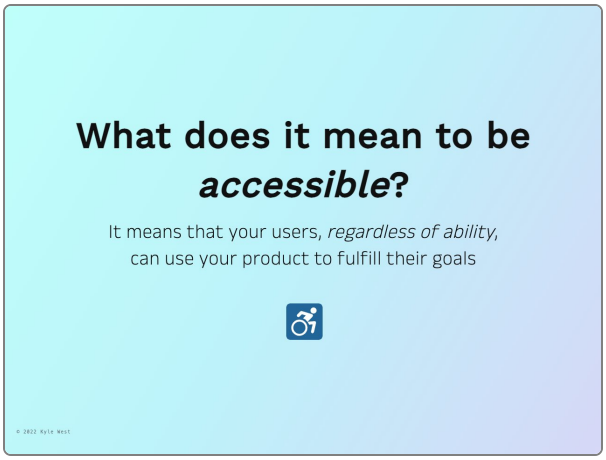
So there you have it: an abridged history of accessible technology. I debated on adding this part because it takes up time I could spend on something else. But in the end I kept it because this knowledge demonstrates one key precept:

Accessibility is less to do with fulfilling standards, and more to do with solving actual user experience problems.

Theses interfaces all had/have standards that guide their development.

1. Text based machines had hardware protocols
2. GUI's have content and semantic standards
3. Mobile devices have both

What makes any of these accessible isn't the standards. It's sum of the whole system's capabilities which enable (or disable) the user to fulfill their own goals.



Before we get into this section, I'd like to pause for a moment and tell you a story of an encounter I had.

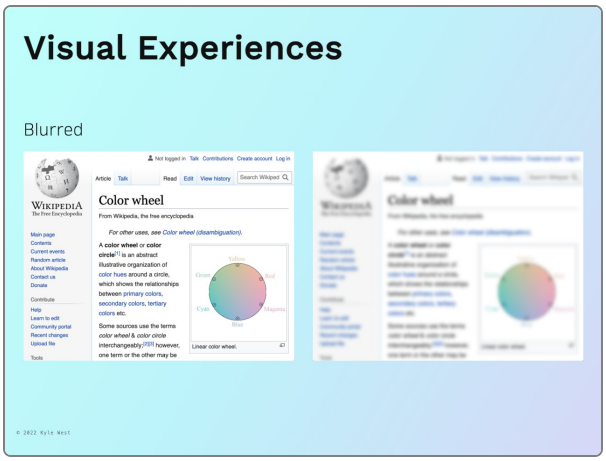
Almost every day I take the bus to work. As such, I get to meet a variety of people. One day I had a perspective changing conversation with a man (whom I will call, Dillon). Dillon has a condition medically described as Trisomy 21 or Down Syndrome. He and I were talking about some of his recent experiences at work and with family, then seemingly out of no where he said something unexpected:

> *I hate it when people call me "Down Syndrome". I hate it when people say I'm "disabled". I'm not, I'm just different... But people tell me I can't do things all the time, and I hate it. I'm not disabled.*

I would have never guessed that he did not consider himself to have a disability. In his mind, he was only "different". In fact, something he said later in the conversation indicated that he did not view Down Syndrome as any more different than hair or eye color.

I mention this story because we have a lot of labels to describe peoples' physical or mental conditions. Though that can be helpful in the medical world when discussing health and treatment, what Dillon taught me is that referring to people by their disability is *unkind* in the general sense.

So out of respect of this concept, for the rest of this presentation I am going to talk with more empathetic language. Instead of mentioning "types of disabilities", we will talk about people's actual experiences and the variety that is found in life.

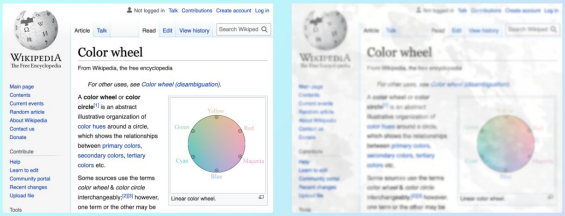


What can be done to improve this experience?

- Glasses! - This tool is often so successful users don't think of it as assistive technology
- Good Contrast (there are tools to help with this)
- Support Screen Magnifiers

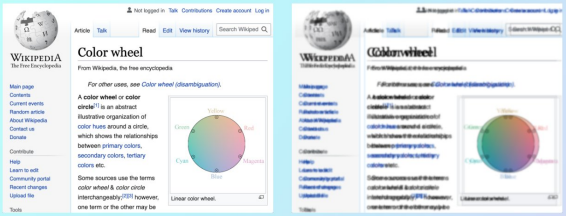
Visual Experiences

Cloudy



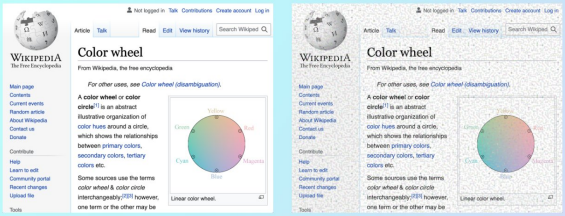
Visual Experiences

Ghosting



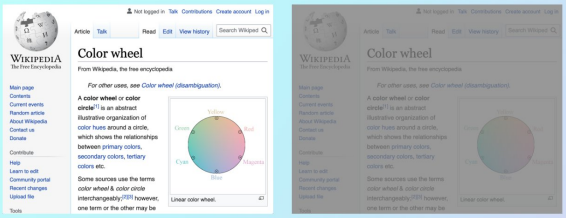
Visual Experiences

Snowy (static)



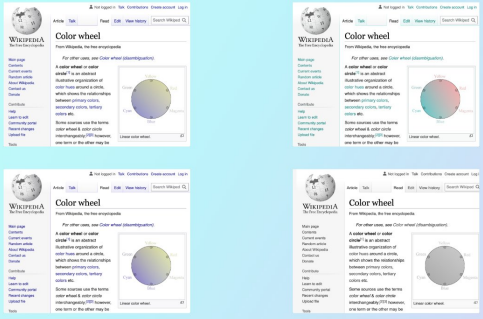
Visual Experiences

Contrast Loss



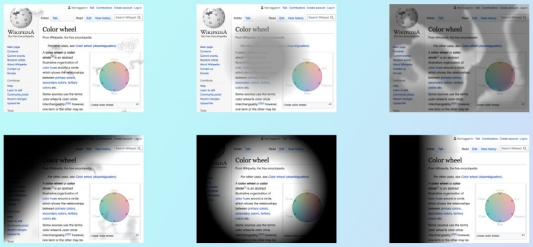
Visual Experiences

Color loss



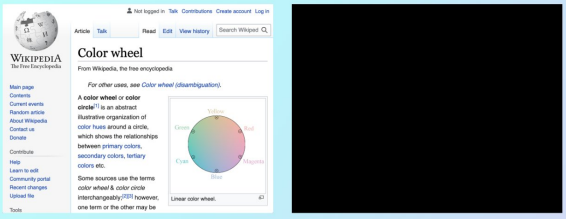
Visual Experiences

Partial Sight Loss



Visual Experiences

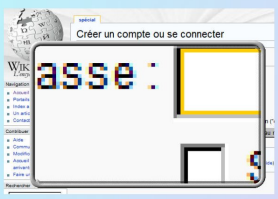
Total Sight Loss



Visual Experiences

What should we support as we design and build our apps?

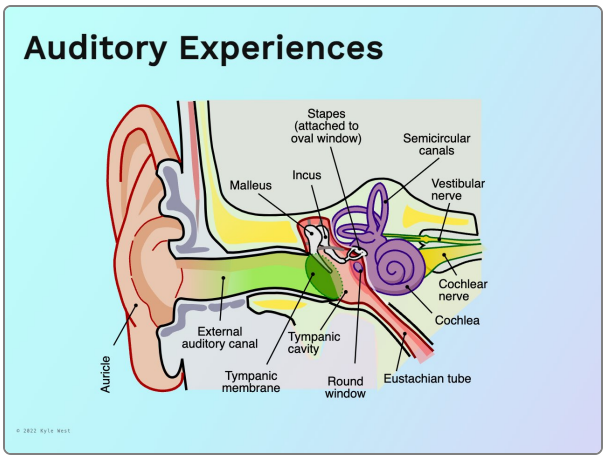
- Contrast and Customizable Colors
- Screen Magnifiers and Zoom
- Screen Reader Compatible



Contrast is one of the easiest things to validate on a website. There are countless tools that can help with the design and testing of this. Contrast itself needs to be both high enough that people can distinguish elements on your site, and colors should be pleasing enough not to cause discomfort to others. Those that benefit from "uncomfortable" levels of contrast will likely have software running that changes the colors presented. To be most compatible with that, please ensure that color is not used as the only way of labeling or providing meaning in your content.

Screen Magnifiers are quite popular. You should make sure that graphics and text do not become pixilated when enlarged. Designing you page to be "responsive" will solve most of these problems for you.

For many experiences, people may turn to using a screen reader to augment their experience on the web. It's not reserved for people with total sight loss, but rather those that may prefer to have addition information given to them as they navigate a site. We will go into great detail on this later for how to make your website compatible with a screen reader.



I spent a lot more time on Visual Experiences than I will other spaces because frontend development has the most control there.

There are other experiences that we need to consider that are non-visual. Personally, I find a lot of the details fascinating, but we will mostly focus on the implications that those details have on your application design.

There are (literally) many moving parts that effect our auditory experiences. Depending on your personal configuration of the outer, middle, and inner ear some sounds may be more easy to perceive than others.

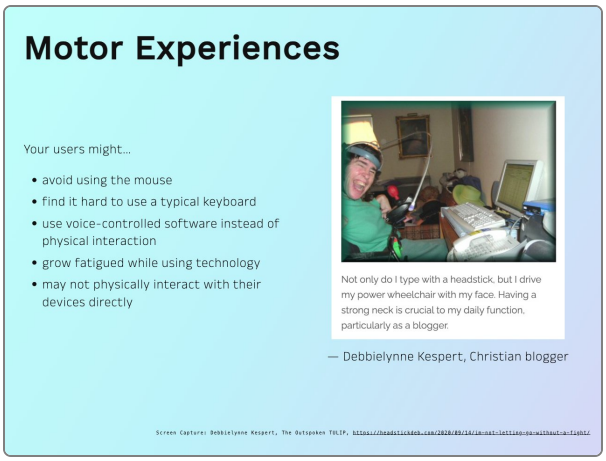
Some people experience actual pain with certain frequencies. Others' ears may physically transduce (or "receive") the sound waves, but they may have a hard time perceiving the sounds.

In all these cases, there are a few things you should be doing:

- All audio should accompanied by either a transcription or captions
- Avoid loud and/or sudden sounds
- If audible indication is needed for your app, include other forms of feedback like visual indicator or haptic feedback where supported by the device

==========

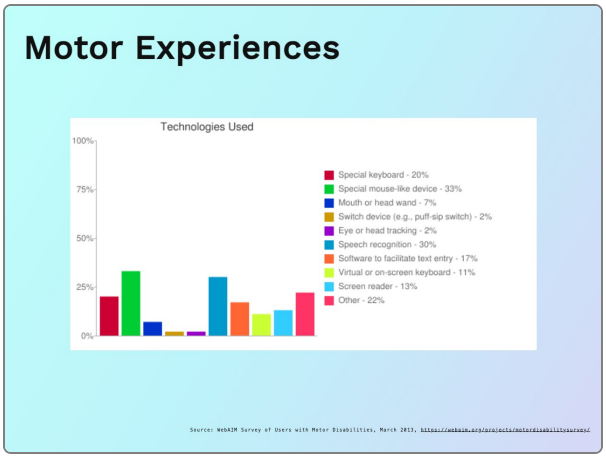
I should also mention that there is one really important exception to my "talk about experiences instead of disabilities" rule of thumb. That is: the Deaf (with a capital "D"). They have such a rich community that includes depth of language, culture, jokes, and values that it would be rude to them to "beat around the bush" about their ability. The Deaf LOVE who they are and are quite proud to celebrate it. So call them what they want to be called.



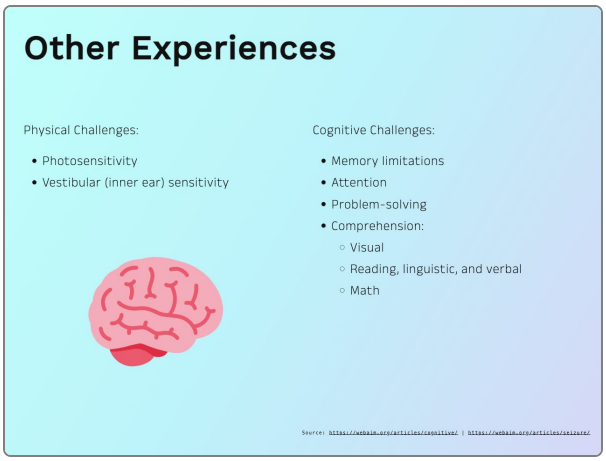
_footer: Screen Capture: Debbielynne Kespert, The Outspoken TULIP, https://headstickdeb.com/2020/09/14/im-not-letting-go-without-a-fight/

1. Ensure all functions are available to both mouse/touchpad and keyboard users
2. Ensure flows are error-tolerant (e.g., ask "are you sure you want to delete this file?"). Create sufficiently large links and buttons that remain in a static position. Allow users to remap or disable single key shortcuts.
3. Voice-activated software can replicate mouse movement, but not as efficiently as keyboard functionality. Ensure all functions are available from the keyboard. Provide descriptive link and button text.
4. Provide a method for skipping over long lists of links or other lengthy content.
5. Ensure content works in both horizontal or vertical orientation. Do not rely on motion actuation (such as shaking or panning the device) or pointer gestures (such as swiping or dragging).

SOURCE: https://webaim.org/articles/motor/motordisabilities "Key Concepts"



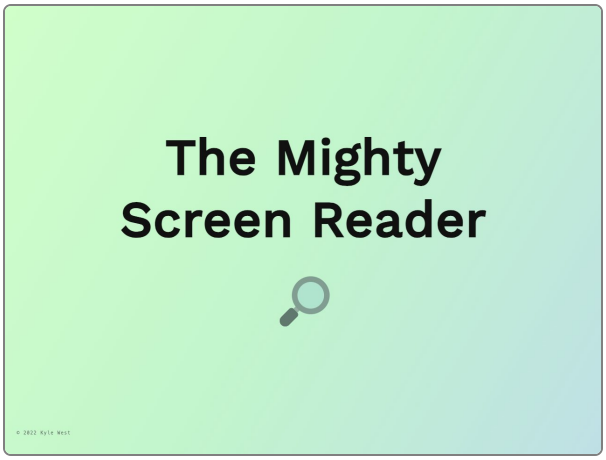
_footer: Source: WebAIM Survey of Users with Motor Disabilities, March 2013, https://webaim.org/projects/motordisabilitysurvey/



_footer: Source: https://webaim.org/articles/cognitive/ | https://webaim.org/articles/seizure/ There are a variety of other experiences your users may face. Some of them (like Photosensitivity) can be very serious while others are more mild.

For the ones I labeled as physical challenges, there are standards and associated tooling that can help ensure you do not cause harm to individuals (like seizures or nausea).

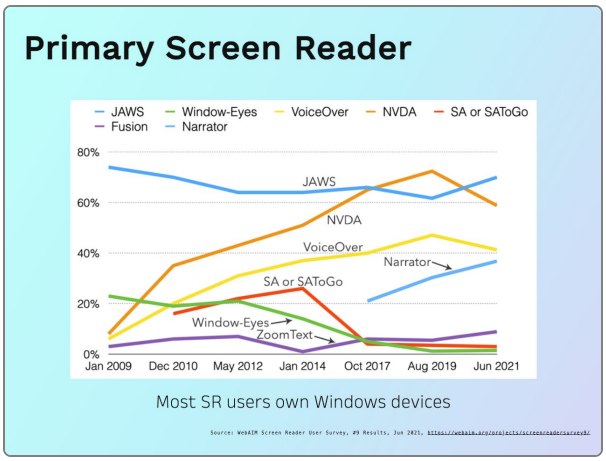
Cognitive challenges are more difficult to account for since the solution to your user's problems will depend heavily on the domain of your applications. So "it depends" on what your overall goal for the user experience is. There are a lot of great resources, however, that we will cover in a later portion that can help you design with these considerations in mind.



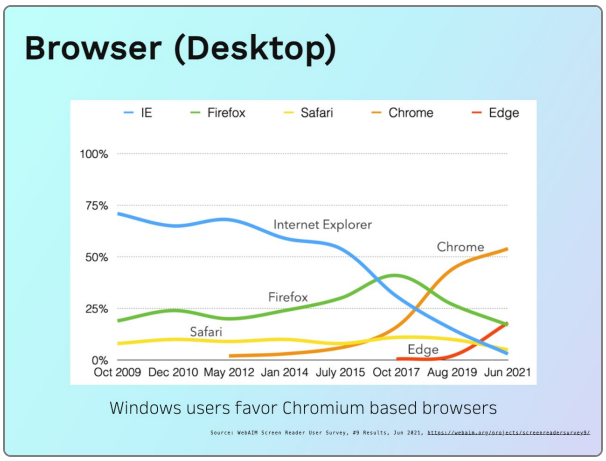
The most important of assistive technologies is the screen reader. Although it was designed for primarily those who experience difficulty seeing, there are plenty of others who use the screen reader because it offers greater control of their device with a keyboard.

Because of this, the majority of the web standards, patterns, and best practices are focused around the screen reader experience.

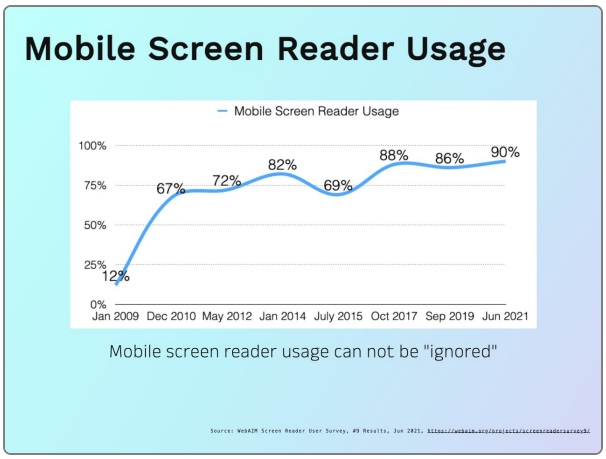
Let's talk a bit about how users interact with the web, from the view of those who prefer a screen reader.



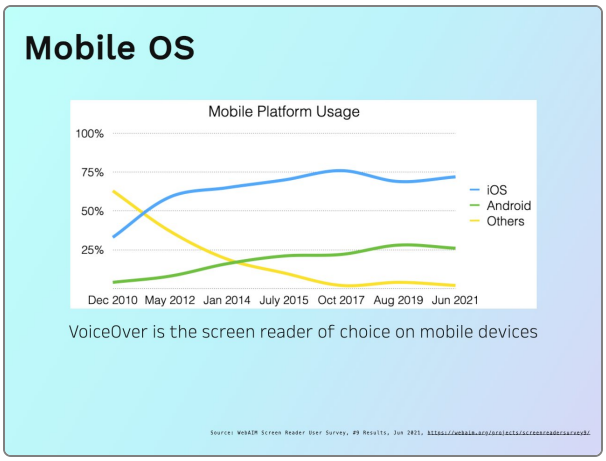
_footer: Source: WebAIM Screen Reader User Survey, #9 Results, Jun 2021, https://webaim.org/projects/screenreadersurvey9/



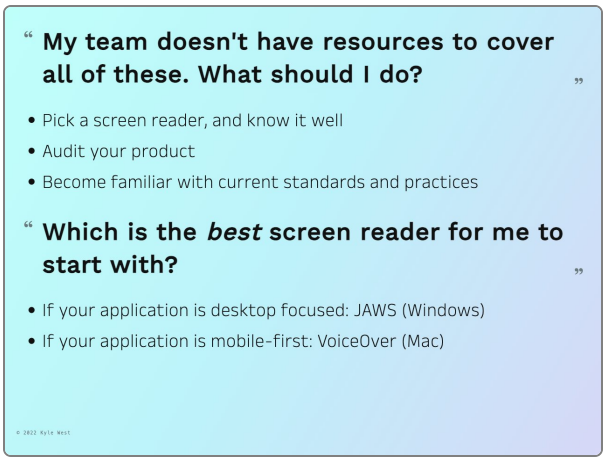
_footer: Source: WebAIM Screen Reader User Survey, #9 Results, Jun 2021, https://webaim.org/projects/screenreadersurvey9/



_footer: Source: WebAIM Screen Reader User Survey, #9 Results, Jun 2021, https://webaim.org/projects/screenreadersurvey9/



_footer: Source: WebAIM Screen Reader User Survey, #9 Results, Jun 2021, https://webaim.org/projects/screenreadersurvey9/



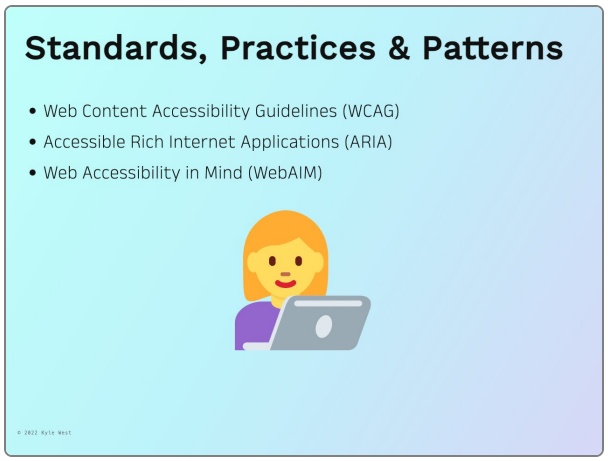
Pick a screen reader, and become proficient with it. If you simply learn how to use the screen reader that is most popular for the device you develop on (or for) then you will cover 80% of the usages of the other device/browser/screen-reader combinations.

Audit what you are building. Understand the flow (or struggle) your users might take when trying to find something or perform common actions. Regularly audit your product's critical work flows to ensure they can be performed with keyboard and screen reader navigation.

There are some pretty major differences between each screen reader, but each screen reader is built to be compatible with the ARIA standards. Know and implement those.

==========

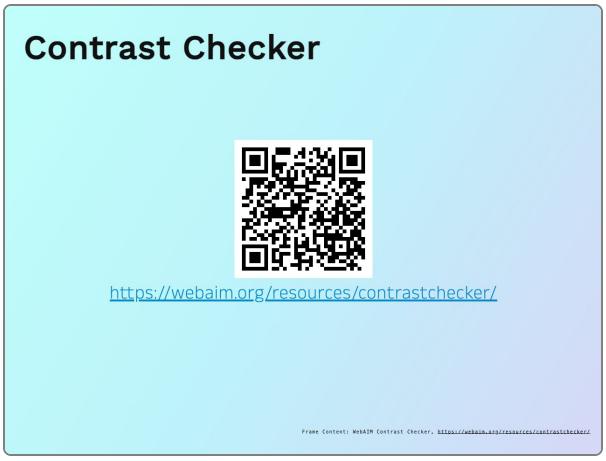
Now I put these suggestions up for which screen reader would be the best start for you to use, but that suggestion is purely based off of statistics. Just pick the one that is most accessible to you. If you develop on a Mac, use VoiceOver. If you don't have the funds for JAWS, use NVDA. Like I said, if you cover one screen reader and follow current standards, most of the other use cases are covered.



Web Content Accessibility Guidelines (WCAG) is a "Standard" that is content based, not technology specific. It tells you what should be on the page and how it should be structured, and what things to avoid. Effectively, it is a design centered focus. WCAG also replaces the original Section 508 compliance requirements from 1998's addition to the ADA.

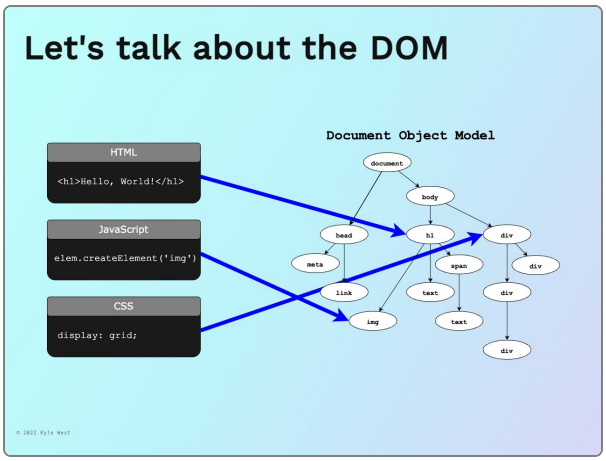
Accessible Rich Internet Applications (ARIA) is another "Standard", but focused on establishing an interface with assistive technologies. This standard is an imperative resource on *how* exactly to create accessible affordances. We will take a look later at this in more detail.

Web Accessibility in Mind (WebAIM) is a great general resource. It is developed by the Institute for Disability Research, Policy, and Practice at Utah State University. It is considered one of the most effective tools for training on accessible design. WebAIM digests the formal concepts of the WCAG and ARIA specifications into plain language. In doing this, they offer explanations as to the *why* behind the standards. They also developed a tool called WAVE which is an effective tool at evaluating how well your applications match the WCAG and ARIA standards. All of my stats in this presentation come from their research efforts.



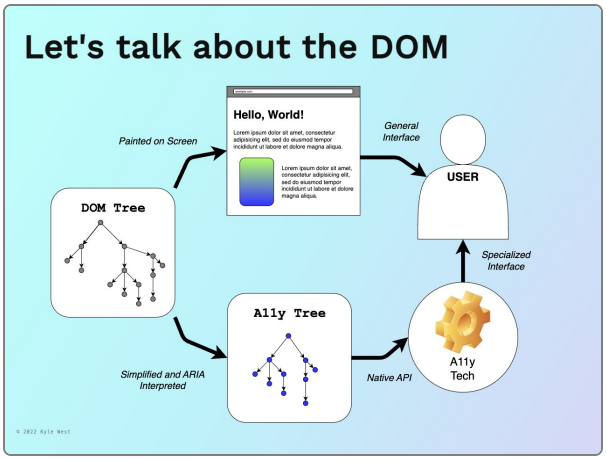
_footer: Frame Content: WebAIM Contrast Checker, https://webaim.org/resources/contrastchecker/ WebAIM also make a really useful contrast checker. Why I love it, is it has the explanation inline to why the contrast matters, and the subtle contexts that change the requirements for contrast.

Contrast issues are far too common. You don't even have to have a physical condition to have a disability. If you use a cheap and/or sun-bleached monitor, then you too will have a hard time with low contrast on some sites. Developers and designers are so spoiled with high quality tech that color contrast can be easily overlooked. Just use a contrast checker. It will make your product more usable for more people.



As you may already know, the code you write is interpreted by the browser to compose a Document Object Model, or DOM for short.

HTML, CSS, and JavaScript all impact the DOM Tree to describe how you want your application to look and behave.



In order to make something useful of the DOM, the browser interprets it a few different ways.

The most common interface for the DOM is the GUI, which is painted onto the user's screen. You are quite familiar with how this works.

The second interface is also quite familiar to you, it's the DevTools. Funny thing, even I find myself sometimes asking someone to "inspect the DOM". This is not technically correct. You aren't inspecting the DOM, you are using DevTools inspect the elements as rendered in the GUI.

The last interface is only familiar to developers who work with accessibility, that is: the Accessibility Tree. This tree is a simplified version of the DOM where:

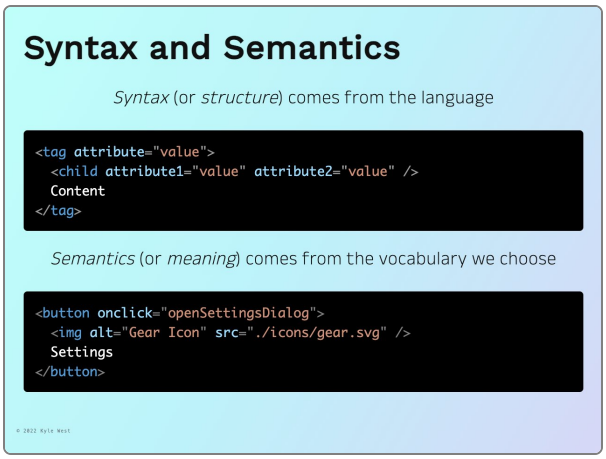
- related text nodes are merged into a single semantic structure
- presentation-only elements are removed entirely
- Headings, form elements, links, and page landmarks are recorded and organized for easy traversal
- implied semantics are interpreted (more on this later)

This Accessibility Tree is accessed via a native API by Screen Readers, Refreshable Braille Displays, and other assistive tech. This API is then wrapped into a variety of user commands that allow your end user to use your site according to their own needs and preferences.

Y'all ready to talk about code now?

Let's start off by talking about something every web developer knows and loves. The humble `div`. What is a div? Well, it stands for "division", but it used in so many context we can also say it means "anything". And we love it.

With a little CSS you can make a dialog, a menu, or anything you want with a div. Let's talk about that.



Since the Accessibility Tree is constructed from the DOM and not directly from our source code, we have to be careful how we structure our code. Specificity this means picking the right tagnames and attributes, as well as establishing a meaningful structure for our code.

Static HTML is inherently accessible because, when used properly, it has meaning. Take this button code for example. The syntax (or structural arrangement) of our code comes from the HTML grammatical definition (shown above). What makes our code useful is not the language itself, but the semantics (or meaning) behind the words we use from the language.

Notice that I don't have to show you a screenshot of the rendered button. You already have a pretty accurate picture in your mind of how this is rendered, don't you? Likewise it sits with the Accessibility Tree. If you use the right vocabulary to provide the intended meaning, then assistive technology will know how to present what you intend for the user experience.

Here is a section of code not too dissimilar to what you can find on a popular shopping website (that I won't name)

Thanks to the good class names, you can kinda figure out that this is a set of search filters probably off to one side of the main page content.

Let's play a game. What if I tried not to use any `div` s? Could we still make this?

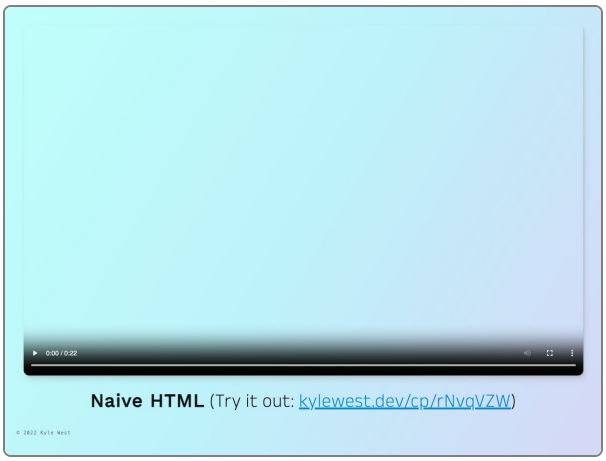
Of course we can! Now there are a few ways to go about it, but I chose this one because it is rich with semantic meaning.

In the end, we could have kept the class names if we wanted, or changed our selectors - that's not what I am wanting to talk about.

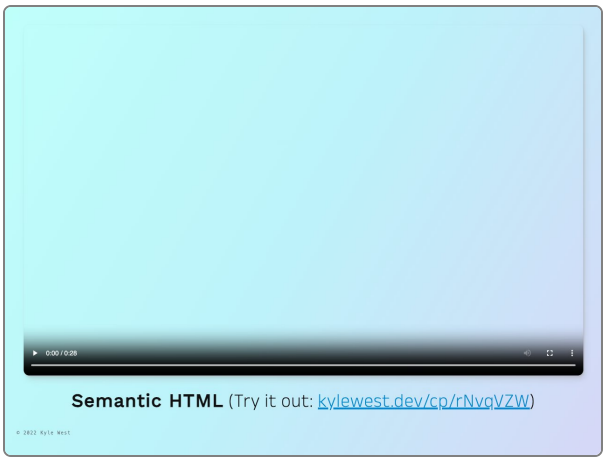
Notice here the value that comes out naturally when just reading through this code.

- The inputs and label text are not only grouped visually, but explicitly through the id/for pattern
- The legend "price" labels all the checkbox options in the fieldset with relevance
- The form element ties all the controls together
- and lastly, we know this piece of code is secondary to the main content because it is under an `aside` element

Even though these two implementations have the same visual result, this structure has an inherit sense of meaning that the other is missing. All of that because we challenged ourselves not to use any `div` s.



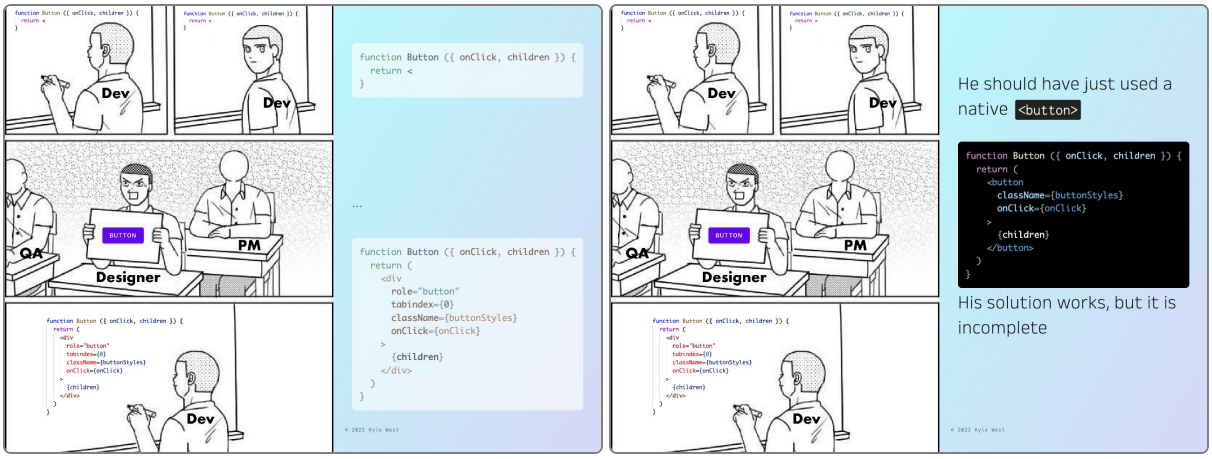
Notice as I navigate here with the screen reader, that none of the options tell me what the value is I am selecting!



Notice as I navigate here with the screen reader, that none of the options tell me what the value is I am selecting! Notice here that it announces that I navigated to a "complementary" (or side content) part of the page. Additionally, the labels are working nicely. And when I go back, it re-announces that I have entered the "Price" selection.

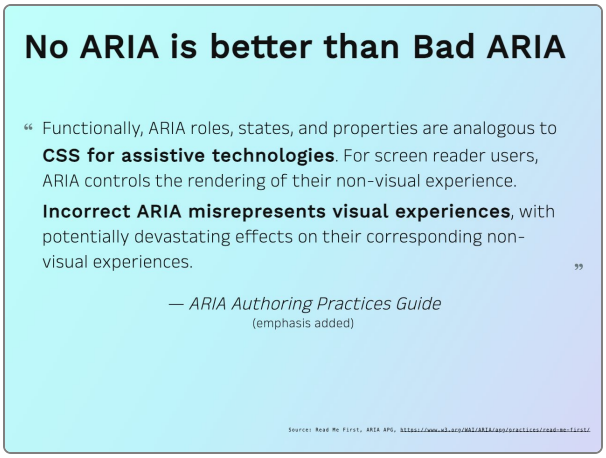
This is a much more prime experience, and it didn't cost anything more than simply being thoughtful with what element I chose.

In fact, the sighted user experience is better too with regards to the label. If we click the naive labels, nothing happens! But if I click the semantic labels, they select the corresponding radios.



When you choose to opt out of using a native element, there is a lot you have to juggle. Notice that he had to add a role and tabindex to his code. This works for the trivial case, but say the button needs to be disabled, well `div` does not support the `disabled` attribute. He will have to remember to use `aria-disabled` instead.

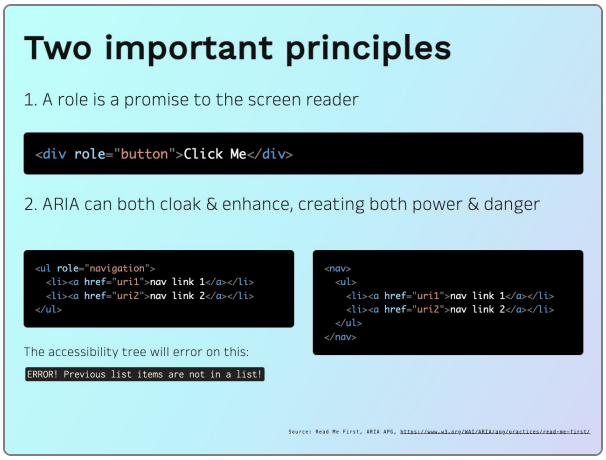
It works, but why? What could he hope to gain?



_footer: Source: Read Me First, ARIA APG, https://www.w3.org/WAI/ARIA/apg/practices/read-me-first/ Now that we have covered a couple examples, I'd like to start taking a closer look at ARIA. Before I do that though, could I please get a volunteer to read this quote?

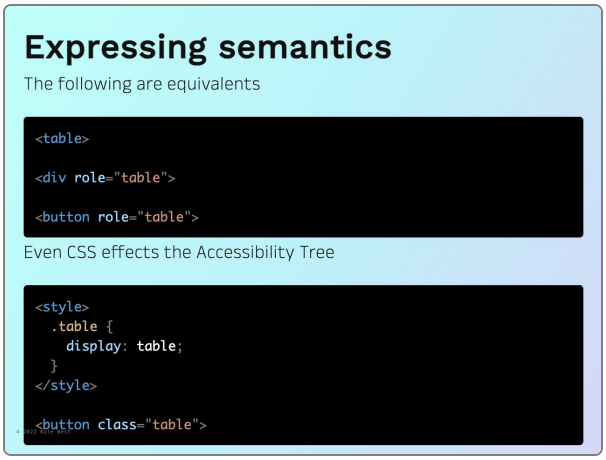
Thank you.

Let's talk about ARIA roles

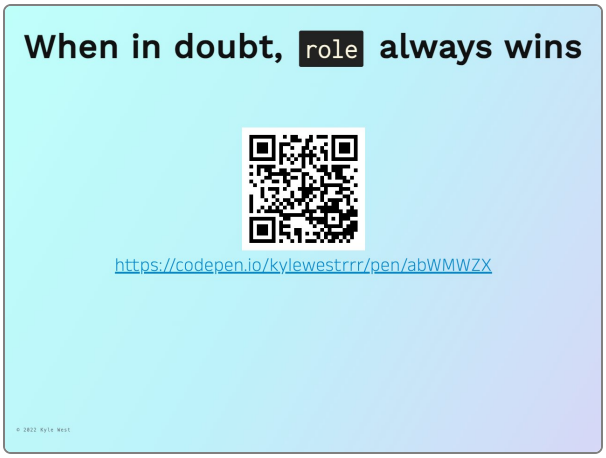


_footer: Source: Read Me First, ARIA APG, https://www.w3.org/WAI/ARIA/apg/practices/read-me-first/ When you set a role on an element, you are promising to the A11y Tree that you also are providing all the JavaScript that handles the keyboard interactions expected for that role. Unlike some native HTML elements, ARIA roles do not cause browsers to suddenly provide keyboard behaviors, nor does the Screen Reader add the expected behavior for you (most of the time).

Roles also override the native ARIA features. For example, You can tell a link to be a menuitem and that is exactly what it will be treated as.



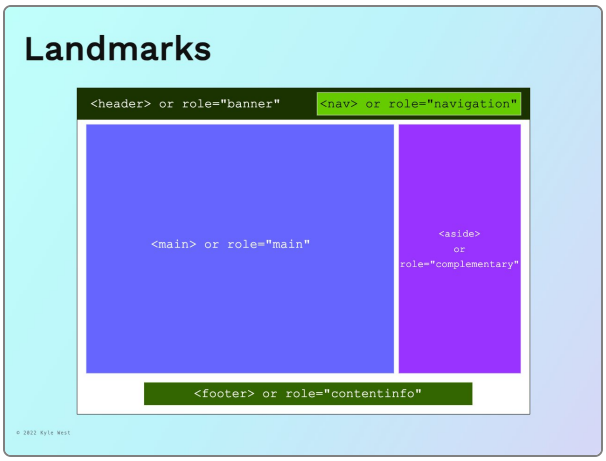
I remember the day I learned that CSS effected semantics. I thought I was being clever by structuring a collection of items using table/tr/td, but then I applied "display: flex" to it so that it had a nice visual layout. This backfired. When I tried it out with VoiceOver, it did not register as a table. The very thing I was try to solve by "not using a div", was caused by a single CSS rule.



Whatever role you specify will get applied

- Then CSS implications
- Then HTML tag semantics

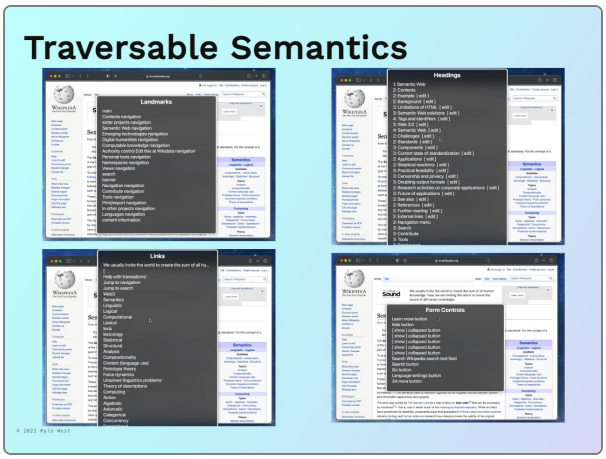
In order to demonstrate this, here is a codepen I made we can play with



Here is a wire frame of a simple website. Notice that you can choose to use the role or the semantic HTML element.

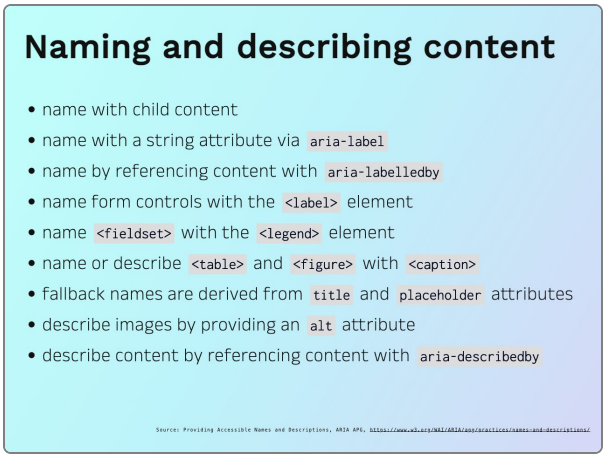
Your main top level navigation should go inside the header or footer. Your "content" of the site should be inside a main, and additional secondary content should sit inside an aside element.

These semantics provide convenience for your users to jump around the page using a screen reader.



If you use landmarks, proper headings, links, and form elements on your page, your users will have a much easier time navigating your product. This is because the screen reader organizes these menus to help the user "skim" the site and more easily search for what they are looking for.

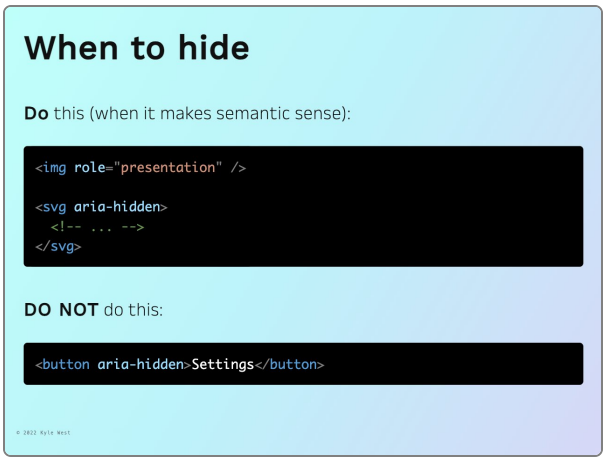
In fact, if you only did one thing to your code, I would ask you to make sure your heading levels are correctly ordered on the page. If you have good headings, your users will be able to navigate effectively to find whatever it is they are looking for.



_footer: Source: Providing Accessible Names and Descriptions, ARIA APG, https://www.w3.org/WAI/ARIA/apg/practices/names-and-descriptions/ There are several rules when it comes to naming and describing content

1. Heed Warnings and Test Thoroughly (WAVE, linters, and other tools can help)
2. Prefer Visible Text (for parity between experiences)
3. Prefer Native Techniques (label-input, fieldset-legend, table-caption)
4. Avoid Browser Fallback (prefer other methods where possible)
5. Compose Brief, Useful Names (avoid noise)

Doing this (especially the aria-label or alt attribute on an icon or image, and the label for form elements) can make a world of difference for your users. I have seen so many unlabeled IconButtons and images. It frustrates me a lot because those are so easy to fix, but cause so much confusion if not done.



Sometimes, it is possible for images or other not interactive elements to provide no value to the end user if they are non-visual. If captions or descriptions are not possible or effective, feel free to mark that DOM node as presentational with the role or the aria attributes.

However, be careful not to mark any interactive content as hidden, because if you do the screen reader will never be able to navigate to that element. It is removed from the Accessibility Tree altogether when that is done.

In case it is unclear, someone has gone to the trouble to write effectively two applications, the "normal" application, and then what turned out to be a semantically correct but rather watered down version of their product.

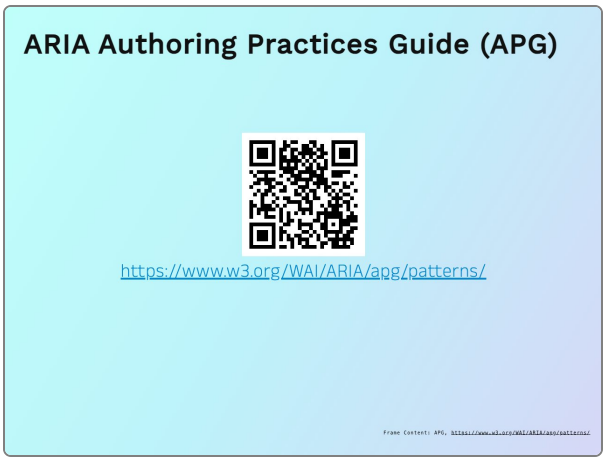
History has taught us that "separate" is NOT equal. Even if the intent behind his was to provide a better experience to blind users, it's not the correct way to go about it. What's wrong with this, is that it is actual discrimination. Ethically speaking, you are better off to fix your application so that it is using best practices. Your visual users will benefit from it too.

There is a lot I have not covered today. I left out:

- Keyboard interactions
- Grid and Table semantics
- All the types of roles
- TONS aria attributes (like form validation, loading content, menus, and dialogs, to name a few)

The reason why, is there is no way I could cover everything in a single presentation. Some people spend years actively studying this subject. Today I covered the very basic principles.

Before I end though, I want to quickly walk you through guides so that you know where to turn to have your questions answered.



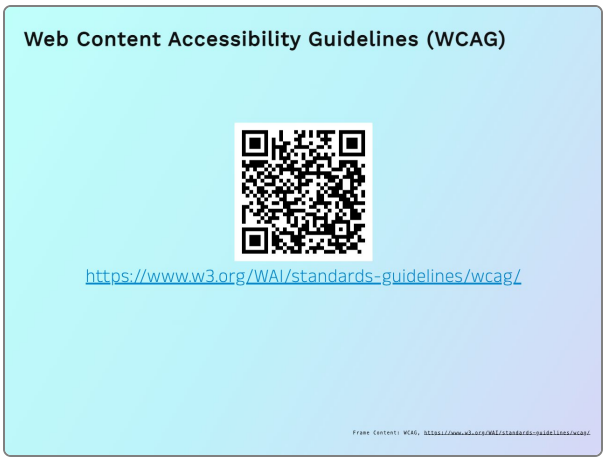
_footer: Frame Content: APG, https://www.w3.org/WAI/ARIA/apg/patterns/ If you know what you need to build, but don't know how to make it accessible, this is the site to turn to. They do a deep dive into semantics, setting up keyboard interfaces, and other details for a large number of widgets.

Let's take a look at Combobox for example

_footer: Frame Content: WebAIM, https://webaim.org WebAIM is an excellent resource for everyone. Where the WCAG and APG cover a lot of the *whats* and *hows* of accessibility, WebAIM cover's the *whys*.

Show:

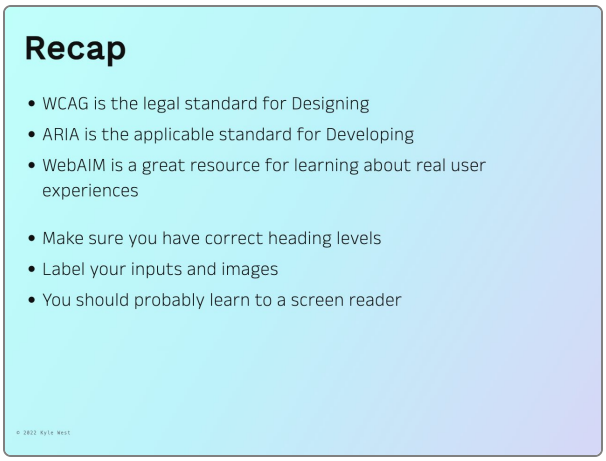
- articles
- projects
  - Surveys
  - WAVE



_footer: Frame Content: WCAG, https://www.w3.org/WAI/standards-guidelines/wcag/ They themselves say:

> *WCAG is not an introduction to accessibility.*

Actually, you could probably never read this yourself and be fine. It's the legal requirements for what defines something as accessible. WebAIM is a much more useful reference for designing accessible applications. But here you go.

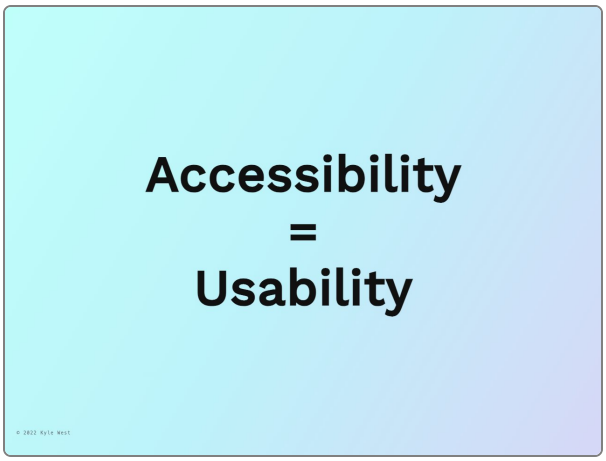


FamilySearch has made a tremendous effort in the last couple years to adopt the standards and practices mentioned. We still aren't perfect, but we have greatly improved the user experience across the board. Our users have given us very positive feedback.

Three major key points have contributed into our recent successes:

- WebDev, QA, and UX coordination and involvement
- Regular auditing and training
- The adoption of a unified Design System and Component Library

But success doesn't have to start as a large organizational effort. Grass roots efforts can go a long way. We had a lot of successes before any formal committee was formed. In fact, we formed the committee to coordinate the efforts already happening by individuals in the org.



Lastly, if you remember only one things from this presentation, it's that Accessibility is less about standards and compliances than it is about building truly usable products.

Life is so hard for so many people in the world. Please consider the capacity you have to reduce their burdens - at least in your interactions with them, or their interactions with your products.

Do good. Thank you. ? Pathos:

- watch https://www.youtube.com/watch?v=Oc_QMQ4QHcw