WebAIM Campus Training-Session 3

I want to do a real quick introduction to ARIA. ARIA is Accessible Rich Internet Applications. This is a specification by the W3C, like WCAG and HTML and CSS. ARIA is a specification for enhancing accessibility of web content primarily, focused on applications. But there are some things that can be relevant to all web content. ARIA is just additional markup that you can add to existing web pages-- new HTML attributes and values that we can add to our existing web pages to enhance accessibility above and beyond what we can do with standard HTML.

With this introduction to ARIA, I just-- we'll see where we get to, but this afternoon we're probably not going to get into heavy duty ARIA stuff, but it's just a little bit of a primer. There are some foundational rules of ARIA. The first one, I'm not going to read it, but basically it can be boiled down to, if you can use HTML, then do that. If you can make it accessible using what we have available in our toolbox with HTML, then you must. If there are places where HTML is not sufficient, then ARIA can help fill those gaps and can enhance accessibility. And I'll talk about some of the areas where that can be helpful, especially in regards to form accessibility.

But what I want to really cover and why I introduced ARIA now is in the context of regions and landmarks. So with HTML, we can define significant page areas using what are called regions-- header, nav, main, aside, and footer. So header the head stuff-- logo, site name, navigation, maybe search at the top of our page. Nav for navigation elements. Main for our main content area. Aside for sidebar elements, typically with the main element, typically related to the main content. So like See Also or Related Links. And then footer for our footer stuff.

Really helpful for accessibility. These elements will be identified by screen readers. A screen reader user can also navigate by these elements to jump between these two the significant areas within the page. You can also define significant page areas using ARIA, using landmarks. So we can give an element, typically a div element, role of banner, navigation-- banner would be the same as our header, navigation, main complementary, content, info, and search.

So there's a bit of a mapping between these HTML regions and ARIA landmarks. So header aligns to role of banner, nav to navigation, main is the same, aside maps to role of complimentary, footer to content info, whis is a little bit weird. Now, ARIA does have role equal search, but there is no equivalent in a HTML to that. There's no search tag or search element.

So these can be really, really helpful for accessibility. In accessibility, we tend to use the term regions and landmarks a little-- kind of synonymously. Some screen reader will call these regions, some will call them landmarks, regardless of how they're identified, whether they use the HTML or the ARIA role attribute. Don't worry too much about that, but add them. These are just really, really great for accessibility. And yeah, like I said, don't worry what the screen reader might call them, or the terminology.

Now remember rule number one of ARIA that I introduced is, if you can do it with HTML, then you should. So we generally recommend the HTML elements, but the ARIA can also be used and would provide the same functionality. Generally, you would not do both. For instance, you

wouldn't do something like header role equals banner or nav role equals navigation. It's really redundant. It's markup that's not necessary. Using one or the other is sufficient, and nav is what we recommend.

And that's pretty much it for regions or landmarks. Generally, if you look at your-- if you think of your page content, everything on your page should be contained within a region. In other words, if I navigate from region to region and listen to the content inside of those, I should get all of the content of the page, which kind of makes sense. You usually have a header, main content, maybe a sidebar, footer, usually everything should fit in one of those.

Also if a few of these is really good for accessibility, it doesn't mean that 87 is better. You can really overuse these. So an example-- sometimes we'll see, like, every group of links in a web page marked up in an nav element. Every heading marked up in a header element. Header and headings are different than-- header is like your site, header, headings are what we talked about for headings.

It's expected that you're going to have groups of links on the page, like a sidebar. If it's in the side element, generally we have links. Those don't need to have a nav element also. Links in your footer-- footers usually have links. They don't also have to be a nav element. If you have 7 or 20 nav elements on one page, it's then more difficult to navigate to the actual main navigation. Some pages will have a primary navigation and maybe a secondary navigation. Sure, if you have two, maybe three, if there are significant navigation elements, sure, use nav. But otherwise, usually for significant page elements.

Questions about regions or landmarks?

[INAUDIBLE]

Yes. You can put nav inside of a header, you can put nav inside of an aside element, things like that. Role of search would usually go inside of header, or maybe in aside element. Typically only one main element per page, usually doesn't make sense to have two main contents, because main suggests most important, and you can't have two mosts, right? So usually only one main element. Usually only one header, one footer. HTML, the rules are pretty loose. You can have as many as you want to, but it kind of makes sense to only have one or a couple of those. So yeah. Good, any other questions on regions or landmarks? OK.

I mentioned earlier the color requirements. We talked about color reliance, we talked about sufficient contrast. I mentioned briefly that there's an area where they tend to come together, and that's for non-underlying links. If you look at a web page with content, if the links within that content are not underlined, then you're using color as the only means to identify those links. That's color reliance. If I can't perceive those colors, then your links are not going to be perceivable. [INAUDIBLE] allows that, if you do two other things. One is a three to one contrast ratio between the link text and the non-link text.

So if you have a really dark blue line text and your surrounding body text is black, there might not be as much of a contrast or brightness difference between those. WCAG requires a 3 to 1

contrast ratio, so maybe a lighter color of blue with the black. Also, the link must present a non color indicator-- usually that means the reintroduction of the underline-- on both mouse hover and keyboard focus. So this can be helpful for users to override page colors.

So if you use light blue for your links, they're not underlined by default. But I override your page colors to something else, your links just disappeared. At the minimum I could use my mouse and hover over until I get the finger and know that it's a link, or I can have the Tab key to navigate to find one so it's not very friendly, but at least it provides a mechanism if the link becomes underlined on hover and on keyboard focus. You do have to do both of these. Then it's no longer color reliant. You're relying on colored, but you have alternative mechanisms to differentiate that-- contrast difference and the non-color designator.

So here's an example of what this might look like. We have this text with these blue links inside of it-- kind of difficult to differentiate what those links are. If I have low vision or a form of color blindness or have overridden page colors, it can be difficult to know what those links actually are. This presentation is a little interesting, because of the background colors not a pure white, and the text colors is not pure black. It's a very dark gray.

It turns out that there is no color of blue that can be used that will meet all of the contrast requirements. In order to get a 3 to 1 contrast ratio between the dark gray and the blue, it needs to become a lighter blue. If you get it light enough to where do you mean the 3 to 1 contrast requirement, it's no too light to meet the 4.5 to 1 contrast ratio difference with the background. So if you don't underline your links, you have three contrast requirements-- text to background, link text to background, and text to link-- is at the 3 to 1 ratio. You can have a window that closes entirely, as is the case with these color choices. There's no color blue that could be used to meet all of those requirements.

In this case, in working with PayPal, they really didn't want to underline their links. That would be our primary recommendation-- underline them. It's the most universal indicator of links. Especially, I think on mobile-- underlined links have come back into style, just because it's so easy to tell, even with small text, lower contrast, you can always tell that those are links. They chose not to. If you turn off the colors on this page, you can see how the links pretty much disappear.

If you just make this grayscale, the links have pretty much disappeared. I mean, not disappeared. They're there. You just can't differentiate them from the rest of the text. That's a good way to test for contrast and color reliances-- just desaturate the page. What they chose to do instead is to actually bold their text-- they made it actually a slightly darker blue than they had before. Actually, it decreases the contrast difference between the gray text and the blue links. It decreased it, but by making it bold, it caused them to stand out some more.

So the three-- they were no longer relying on color alone. They have bold in addition to the color to differentiate the links. These do become underlined on both mouse hover and keyboard focus. They didn't have to, technically, for WCAG, but they knew-- we recommended that would create a better end user experience.

So if you desaturate this page, you can still differentiate the links. Is this as good as underlining? No. But it works. There are some difficulties with this. You'll notice that, for instance, user agreement and privacy policy-- is that one link or two links? You can't really tell. You could hover over and get the underline to tell that it's two links, but it's a little difficult to tell. Also in this presentation, what if they were using bold to indicate something else? Like for emphasis, not links-- now you actually have difficulty in differentiating between bold text and bold link text.

Like, you must agree, and must is bold. Is must a link or is it not? So this is not a perfect solution, but it tended to work and got around the WCAG thing, and certainly increased accessibility, though not as good as probably underlining the links. This really only is relevant to links where there can be visual ambiguity as to whether it's a link or not. This is a purely visual thing. This is not a screen reader thing. Screen reader user can hear the links. It can navigate to them, and so forth.

So, like a list of links in a footer-- if you can visually tell, it's clear that those things are links, they don't have to be underlined, or you don't have to do these additional things. Sometimes in footers, though, we have some things that are links, some text that are not links. If you can't always tell the difference, then this would become applicable. You would need to do the 3 to 1 contrast ratio and the underline, unfocus, and hover. So use color. Use color to differentiate your links. But if you're only using color, know that there are some additional requirements.

Any questions about this? OK.

All right, next topic is visually hiding content. Sometimes, especially more in applications or in page templates, things like that, you may have content that you would want to hide either from everyone, or content and you may want to hide just from-- visually, but still have accessible to screen reader users. So we can do CSS to do that. And if you're not a developer, don't worry too much about this. Just focus on the concept.

But we can use CSS of display none or visibility hidden or the HTML hidden attribute to hide content from everyone. This can be really, really helpful. Consider, for instance, like dropdown menus. You wouldn't want a screen reader reading through all of the menus if they don't show it visibly. Or error messages in a form. You have dynamic error messages, if you make a mistake it'll automatically show you this error message. Those are hidden by default, you wouldn't want a screen reader reading the error messages when they don't show up visibly. They're going to think there was an error when there actually isn't one. In that case, hide it from everyone using it primarily display none.

Screen readers will ignore that content. They want to avoid things like zero pixels or same color as the background, things like that. Just-- they aren't very universal. Some screen readers may read them, others may not. There might be searched engine implications. So display none really is the best technique there.

Occasionally you may have content that you want to have read by a screen reader that does not show up visibly. You can do this by positioning the content off screen. I'll show you how to do that in a moment. But just be very cautious with this. Sometimes we see this technique really

overused for like very verbose, lengthy descriptions and explanations, instructions for screen reader users. Just not-- just be really careful with that.

We evaluated a major American bank web site. You might even call it the Bank of America. It was years ago. And they had all of these form fields, and almost every single form field on their site-- and you imagine a banking site, there are a lot of forms-- almost every single form field had hidden screen reader specific content that would be read. So visibly you would see user name, [INAUDIBLE] box-- the screen reader user would hear something like, this is the text box where you enter your username. Probably not necessary.

You'd see password, and it'd be like password, and then the password field, and then below would be like password must be 8 to 13 characters, screen reader user would navigate to it and they'd hear something like, your account integrity is very important to us. Therefore we have certain requirements for your password. Your password must be at least 8 characters. It must be no more than 13 character. And went on and on and on. And to be honest, it was insulting.

It really was insulting to screen reader users. [INAUDIBLE] the suggestion that they'd never used the web before, and this-- the Bank of America is the first web site they've ever come to. We better tell them how to use a text box. I mean, it was-- it just wasn't cool. And it was so much. So much stuff. Sometimes-- very often when we consider accessibility and start to implement it, we very-- we usually consider the user's first experience when they first come to the page.

And that's important. We want it to be accessible. But sometimes thinking that way lends itself to being more verbose-- lengthier alternative text, off screen text that provides additional instructions and information for screen reader users. Which can maybe be helpful the first time. Every other time after that, it's just extra noise. Consider the users thousandth time on that web page. That little extra word-- I'm not saying you just have to trim it down to nothing, but be mindful of that. It can become noise after that. It's OK to expect your users to orient when they come to your site, to know where searches, where form fields are. So just be very cautious with this type of screen made or specific content.

If you do need to add it, and there are some use cases, we would recommend some CSS like this, and again, you don't really understand CSS, don't worry about it. But what this does is that positions the content absolutely-- so it essentially removes it from the flow of the page. Everything will collapse so it doesn't take any space. Positions it off the left hand side of the screen-- 10,000 pixels, [INAUDIBLE] says don't move it up or down, just move it straight to the left.

And then the rest is maybe a little bit overkill. But it sizes it down to one pixel wide and high, and hides everything that doesn't fit within that one pixel block. So we essentially have a 1 pixel by 1 pixel block, 10,000 pixels off the left hand side of the page. Because it's still part of the page, it's read by a screen reader, but it doesn't show up visibly. A nice little technique to have this content be read by a screen reader. Yes.

[INAUDIBLE]

Correct, yeah, thank you. That's a great question and comment. Yes, it will be read based on its code order, source code order. So you can, with CSS, you can position things, you can float them. Now with CSS Grid, you can do this whole layout and position elements wherever you want to. You want the visual order and the underlying code order to both generally be the same. At least be logical and intuitive. So yeah, it's based on the underlying source code order.

Yeah. Remember, that also impacts keyboard users. So as I start hitting the Tab key on a web page, it's going to navigate based on the source of the code. So usually I want that to be left to right, top to bottom, through the page. Screen reader will read it in that order. Visual orders-- you want them all to a align to create the most consistent experience. Yeah, good. Any other questions about this technique?

[INAUDIBLE]

Yeah, height-- width or height of zero and sometimes-- some browsers don't pass that information on the screen reader users-- well, onto the screen reader technology. Screen readers work on [INAUDIBLE]-- I'm going to do this thing with my hands probably several times before the day is over, this kind of layer idea that you have your content, it's interpreted by the browser, then it communicates information to the screen reader that lives on top of it. And so sometimes browsers with zero-- things are sized to zero pixels-- won't convey that to the screen reader. So just depends on the browser.

There is this alternative technique that uses clip and or clip path. That's in CSS. So the clip basically clips the dimensions down to zero pixels. Clip path does the same thing. Clip is old in CSS and is no longer supported in CSS. Clip path is brand new, not yet supported by all browsers. So if you're going to use clip or clip path, we recommend using both and hope that there's not a gap where clip is not supported and clip path is not yet supported. Becayse if there's a gap there in browsers, then your content will show up. Offscreen left works pretty universally.

OK, continuing on with keyboard accessibility-- there are standard keystrokes for elements within your web browser. All browsers support these standard keyboard interactions for standard things. For links and form controls, we're talked about the Tab key. Shift plus Tab willl navigate backwards through those things. If you hit Tab and go to a link, you activate that link, hitting the Enter key. That's the same as clicking on it with the mouse. If you navigate to a button, you can activate the button using the Enter key or the Spacebar.

Either of those will trigger it. In most screen readers, it will tell the user to press the Spacebar to activate the button. It becomes a little bit-- no, not a little bit. This becomes very important if you're dealing with dynamic scripted applications, maybe where you have elements of function like a button but aren't a true button. You need to program them to work with both the enter and the spacebar. For checkboxes, you'd hit the Tab key to navigate to a checkbox. The Spacebar to check or uncheck that checkbox.

Radio buttons-- you would navigate to the set of radio buttons, usually the first or selected radio button, then you use the arrow keys-- up or down or left or right arrow keys-- to cycle through the radio buttons within a set-- say, multiple choice answers. Arrow keys to select the one that

you want, tap and then take you to the next interactive thing within the page. Now, it is up or down or left or right for radio buttons. A lot of that's because a screen reader user can't see the interface. Very often the radio buttons are oriented vertically, like multiple choice answers, consider like the Likert scale that's left or right. A visual the user might look to that and try to hit the Left or Right arrow keys. The browser will support both-- up or down or left or right. So you can use either of those.

For select menus like a drop down with a list of states or countries, there's a lot of keyboard interaction. You hit Tab to get to it, you hit the Spacebar to expand that, you hit Up or Down arrow keys to navigate through the menu till you get to the item that you want, you can type letters on the keyboard to quickly jump to things, which is nice if you live in Utah. There's only one state that starts with U. You hit Tab, U, tab, and here you're done.

So the browser does all of this. You don't have to do any of this. What you want to try to do is don't break it-- implement something in your page that would interfere with these standard keyboard interactions. If you're building other types of custom widgets-- sliders, tree menus, dropdown menus, things like that-- there are standard keyboard interactions for those types of things. Now, you may need to program those to align with the standard keyboard interactions, but those things are defined. You just need to make sure.

If I am a keyboard to use and I come to something on your page, I need to know how to interact with it and it needs to be standardized as much as it can be. If you are a Mac user, there is a setting you'll need to set within your preferences. Go to System Preferences, Keyboard Shortcuts, at the bottom there's this option for a full keyboard access. Set that to all controls.

So the default on a Mac when you hit the Tab key, it won't navigate everything that's interactive. You're going to do keyboard testing, so I suggest in that setting that all controls and navigating. Just leave it on. If you're going to do any keyboard testing, you want that set. Every couple months I get a hateful email from someone saying, I can't believe your website's not keyboard accessible, I hit the Tab key and I can't get to these things. I say, well, that's your Mac settings. That's not our website. So make sure that's set to all controls.

Questions about some of those standard keyboard things? Yes.

[INAUDIBLE]

Yeah, there are a few approaches there. That's a very common one, where you have-- maybe have something in your interface that looks like-- doesn't look like a checkbox but it functions like a checkbox. Like maybe a list of options that you can turn on or turn off. Sometimes that will work you have a set of checkboxes that's hidden, offscreen, so that a keyword user are actually Tabs or a screen reader user interacts with these hidden checkboxes that are then kind of tied to the visual interface.

So if I click on it, it's actually checking the hidden checkbox. If I interact with the hidden checkbox, and check it, it visually becomes selected within the interface. Yeah, that can work

very well. You need to do some testing, especially look for things like focus indicators, that I'll talk about next. So that's one approach.

Another approach is to maybe use ARIA. And for a lot of custom widgets, they're supported by ARIA, and there's documentation that can tell you how to build those, and the proper keyboard interactions for them. Any other questions on this?

So I mentions just setting your keyboard focus indicators or keyboard outlines. When you're navigating with the keyboard, hitting the Tab key, it's important that you can visually tell what thing has focus within the page. As I hit Tab, I need to tell what I'm on so I can then interact with or activate that thing. By default, the browsers will show a focus indicator, or a focus outline. Most browsers it's like a blue kind of Halo that surrounds the currently focused element. In Firefox, it's like a dotted line that surrounds the focused element.

You can, with just one little piece of CSS, you can turn off or inhibit that default focus or outline. With outline 0 or outline none applied to interactive elements, that focus indicator will be turned off. This is hugely detrimental for sighted keyboard users. If I start hitting the Tab key, and I have no idea what I'm on within the page, it's pretty much totally unusable. I can't interact with anything on that page, because I don't know what I'm on to interact with.

So don't do this. Don't turn off the focus indicators. This is very prevalent on the web. Maybe a third of the web has no focus indicators to find. It's very, very common. Don't do it, just because it's a huge, huge thing. This is a WCAG 2.0 AA failure. It should be single A. Treat it as single A, because it's hugely significant for keyboard users.

It works by default. You have to break it. And it's stupid simple to fix. Just get rid of outline zero or outline none from your CSS. It'll impact nothing else, but will ensure that there's a default focus indicator.

Why are these things turned off? Why is it so common that they're disabled? One is sometimes it's just implemented with CSS that you might be using, and you're not familiar with it, and you haven't tested it. So maybe you're using some sort of design template or something like that. Plug it into your site. And it turns these off, and you haven't really considered the keyboard accessibility implications.

Or sometimes they're turned off we just people don't like the way that they look. They see that blue halo and they're like, oh, what's that blue thing? I don't like that, that doesn't match our visual design, let's turn it off. If you don't like the way that it looks, you can customize it. You can use CSS to style it to taste. Make it look the way you want it to, you can change the color, size, there's quite a few things you can change to ensure that it's still visually distinctive. You don't want it to be super subtle. If I'm hitting the Tab key, I need to really be able to tell where it's jumping out within the page. But still have it match your visual design, if you so choose.

Test this. On your pages, just go to your page, hit the Tab key, make sure you can tell where you're at within the page. It's almost always going to be this that breaks that.

Now, I mentioned earlier in WCAG 2.1 there is the new requirement for 3 to 1 contrast ratio for interface elements. This includes focus indicators. But only if you customize them. If you stick with the browser default-- in other words, you don't do anything in your CSS will change the outline-- there's nothing you have to do.

There is, however, there can be some issues with that. And that is if you use blue. Because the default focus indicator in most browsers is blue, what happens if your page background is blue? When they come to a link, that focus indicator may be the same color or very similar to the background, and it can disappear. WCAG does not require that you fix that. I think it's a good idea, though, because sighted keyboard users may not know what they're doing, or where they're at on the page.

If you do customize your focus indicators, WCAG does require that 3 to 1 contrast. So if you have a blue background and you just make it a light blue, that's not going to be distinctive enough, unless there's a 3 to 1 contrast ratio. So you'd want to style this to taste, but make sure they're visually distinctive. Questions about this?

If you have hover interactions in your web page, things are defined with like CSS hover-- like menu items, when you roll over them, maybe they get a little drop shadow or change color or something like that-- typically you want to provide those same types of visual indications on keyboard focus. So if I hover over something and it changes slightly, usually that's a reinforcement that this thing is interactive. Provide the same thing for keyboard users-- as they're tabbing, through provide that same visual change, in addition to the outline, to provide an indication, hey, this thing is truly interactive, and you can activate it.

Pretty easy in CSS. If you have, like, a color hovered and you defined hover styles, just add comma space a colon focus, it will duplicate those styles on the focus state when you hit Tab to that particular element.

Another important thing for keyboard users are skip links, or skip to main content links. So I mentioned before, our colleague uses the stick in his mouth to navigate. And you came up with a number of how many items he'd have to navigate through to get through to the main content area on your page. You can add a link to the beginning of the page that he could activate to skip over those links to jump directly to the main content area.

This can be really helpful, especially for sighted keyboard users. Screen readers, we talked about before, provide a lot of different ways to navigate. I can navigate by headings, I can navigate by regions. I can also navigate links, buttons, tables, forms, lists. All sorts of things in the page I can navigate by using the keyboard.

Standard browsers-- IE, Chrome, Firefox, pretty much limited to Tab and shift Tab. That's pretty much it. So if I'm not using a screen reader, but I need to use the keyboard, if you have a whole bunch of links at the top, I don't have an easy mechanism to say jump to your first level heading, or jump to your main element, main region within the page.

A skip link can help facilitate that. So the idea is it's one of the first things on the page. Hit the Tab key once or twice, come to the skip to main content link, hit Enter, it then jumps focus down to the main content area of the page generally where are your main element, main region, would begin.

Really, that's really beneficial for accessibility. This is not required by WCAG. WCAG has a requirement that says that your page provides a mechanism to skip repetitive links. So if your page starts with an H1, your main content starts with an H1, is that a mechanism? Yes, you pass. Do you use the main element? Is out a mechanism. Yes, you pass.

But main and H1 aren't navigable foresighted keyboard users. We recommend the skip links. Now, skip links are a hack. No other way to put it. They're kind of obtrusive, they're not very pretty, but they're still useful for sighted keyboard users. We're working with browser vendors to try to get them to support some other keyboard interactivity. I would love the ability to hit the M key to jump to the main content, S to jump to search, N to cycle through the navigation elements, H to navigate through headings, 1 to jump to the first level heading.

I'd use the heck out of that, I think a lot of other people would too. Were trying to convince browsers to do that. If they will implement that functionality, skip links can go away. And good riddance to them, because they're a bit of a hack. But today is not yet that day. We're not quite there yet with browsers.

So we still do recommend these skip links. Skip links, however, are an area where there can be some tension between accessibility recommendation and visual design, or maybe even usability for others that don't use a keyboard. Again, to be useful, it needs to be one of the first things on the page. It also needs to be visible within the page. If I can't see it, it doesn't do me any good if I'm a sighted keyboard user.

That's pretty intrusive to visual design. I can even argue that it can at least provide overhead or could cause confusion for other users, users that don't need to use the keyboard. So there's an area here of a little bit of tension. This can be resolved relatively easily, that's just by hiding that skip link so it only becomes visible to the users that need it.

So here's that same CSS as we talked about before to take that skip link, we hide it off the side of the page. But then when the link receives focus, we position it back within the page. It becomes visible. So sighted mouse users never see it. It's off the side of the page. Sighted keyboard users, when they hit the Tab key, it becomes visible in the page, they can then activate it. And screen reader users don't generally care if it shows up visibly or not. They just want to be able to hit Tab hear it and activate it.

So this provides the accessibility benefits without intrusion to visual design or usability for others. So just a little bit of creativity. There is the note there to not use display none on the link. I mentioned display none hides content from everyone. So if you do have that skip link display none, and then style it to become visible when it receives focus, the problem is the link cannot receive focus, because it's hidden from everyone. You can't tab to something that's hidden with display none, or within something hidden with display none. So don't do that. Test your skip

links, if you have these, make sure that they work appropriately. Very often they're hidden in a way where they're not actually accessible.

Questions about skip links? Sometimes people ask about multiple skip links. I don't know, usually not. The primary benefit of a skip link is getting in from the beginning of the page to the main content area. Sometimes we see, like, skip to main navigation, skip to secondary navigations, skip to footer. I don't know, at what point do you need a skip this skip links link? Because you're now adding enough overhead that they might lose some of their utility. And they're the first links on every single page. So just-- I don't know, one or two, usually, maybe three if you have a complex layout.

OK, so [INAUDIBLE] little bit of code technical stuff. We'ge going to come back through some more things that are really applicable to anybody, even if you aren't really doing code, if you're just doing content authoring. One of those is the use of lists. I mentioned before, the screen reader when it comes to a list, it'll tell you the list type. It will tell you the number of items within the list.

So use these. Use them for groups of items. Use them appropriately-- OL for ordered or hierarchical lists. For numbered lists. UL for unordered or bulleted list. Things where if you change the order it doesn't impact the meaning. In HTML there's also DL for definition or description list, where you can have things are paired together, like a question and answer, question answer. The browser will provide this visual styling of indenting the value below that item.

So a screen reader will identify these-- use them. They're awesome, great for accessibility. Even if you aren't presenting it visually as a list, this can still be helpful. For instance, the WebAIM site, our navigation across the top of the site-- Services, Articles, Resources, Community-- that's really a group of parallel items. We've marked that up structurally in the code as a list. It's an unordered list with four items, and we use CSS to get the visual presentation across the top of our page.

And you don't have to do that. But it's really useful when the screen reader comes to this thing and they hear that there are four items coming, they know what to expect if it's-- four is way different than 27. And the screen reader user can navigate by list item or they can skip over the list to go to the next item. Questions about lists? Uh-huh?

[INAUDIBLE]

Yeah. Yeah, like right here, under community, you might consider this a list of items. It doesn't have bullets, but yeah. So you have styles where you can do that in the sidebar. If People don't want bullets, they can style it that way. But maintain the semantics of a list. Excellent, very good. And yeah, these things down here are also lists in our page. And they don't have bullets, but semantically they're parallel items or hierarchical items so we mark them up as a list appropriately. Good.

Any other questions or comments there? OK.

Another important one is page title. That's the text that appears in the tab at the top of the browser. This is the title element in HTML. Typically the first thing that's read by a screen reader when it comes to the page. So it's important to have it be descriptive of your page content. Generally pretty succinct. Succinct not only because a screen reader will read it when it comes to the page, but also because the number of characters that are visible within the tab is limited, and more so if you have more tabs open.

So you usually want to keep that pretty succinct. Usually put the more descriptive, distinct stuff first in the page title, as opposed to at the end. if all of your page titles started Michigan Technological University colon and then the discriptive stuff, I have more than handful of tabs open, I can't differentiate between those. So maybe MTU colon might work, or just put that at the end of the distinctive stuff if you think you need it.

Because the page title should be descriptive of the page content, and your H1 should describe the main content area of the page, they're usually going to be the same or similar. And there are some automated tools out there that will flag it as an error if they're not identical. That's quite extreme. But usually they kind of serve similar purposes.

Now, they are distinct. They are different. They do something different. Sometimes what will happen is you might listen to the page in the screen reader and you hear the page title, and then you get into the page and here the H1 and if they're the same or similar, you think, well there's duplication there. That's bad.

No, they serve different functions. The H1 describes the content within the page, the page title. It differentiates that page from other pages. So if I switch tabs in my browser and I want to come back to the page I'm looking for, it would read that page title when I come back to it to help differentiate. So it's important that they be descriptive, even if the H1 and the page title are the same or similar. Don't worry about that repetition if you hear it.

Questions about page titles? OK. I think in your CMS. I'm assuming the user can define a page title. And I think you said that that also becomes the H1. perfect. That's a way to just ensure that consistency. And that's great. You use OU campus? Is that right? OK, good.

All right, real quickly, you can also use the title attribute. Now, don't confuse this with the title element. [INAUDIBLE], element is the tag, attribute in quotes, attribute values in quotes. So if you're using frames or I frames, you can give them the title attributes. For frames, if you're still using frames, heaven help you, but frames we don't seem as often anymore.

They're not part of HTML 5. But if you were conceivably you were asking for a friend about accessibility of how do you make frames accessible, you would use the title attribute. To that's what would describe the content or functionality of that frame to differentiate it.

For I frames, I frames are in line frames. They essentially allow you to embed a web page within another web page. For an I frame, it's just treated as content of the parent page. So let's say you embed a YouTube video, most often embedded with I frames. Screen reader would read to the

page content, when it gets to the I-frame, it'll just read the front end of the I frame from YouTube and then go back to the main page content.

You can give I frames titles to just describe them, to differentiate them from the rest of the page content. So like advertisements or a YouTube video-- visually, when you look at it, this is a distinct piece of content. A YouTube video usually has like a still frame or even text that displays the title of the video inside of it. Put that in the I frame title so it's clear to a screen reader user that this is something distinct.

Now, if the I frame is transparent, like you're embedding, I don't know, content from a third party, but it just looks like tonton in the main page, if it's not important that the user know that it's a distinct block of content, and you don't need to give it a title. Most common with like YouTube videos, Facebook feeds, Twitter feeds, things like that, most cases a good idea give it a title to describe what that block of content is.

A few things about fonts and text. So you can, in web content, you can define text sizes using absolute values or relative values. Pixels, points for absolute, percents, [INAUDIBLE], things like that for relative values.

We recommend relative values. It's not absolutely critical for accessibility. Some people say you have to use relative sizing. There are some browsers, especially older browsers, that will not resize text if it's defined in pixel values. Which is maybe correct. Because if you say something is 12 pixels, 12 pixels is 12 pixels.

But users can change their text sizing. Some older browsers, some browsers won't redo that, but if you define the text in relative sizes, it will respond or change to the user preference. So we recommend relative sizing. But it's not-- I don't know. We talking about fighting the right battle, especially if you've like the legacy system, if things are defined using pixels, it's probably not the best use of your energy to totally redo all of the CSS to support relative sizes.

For something new going forward, build it in relative sizes. It's going to make your life easier, too, once you get it set up. Just so much more flexible. But it's not a huge impact for accessibility. When it comes to the font faces that you might just, use good fonts. If you're using non-standard fonts, like [INAUDIBLE], things that are downloaded to allow custom font faces in your page. That's fine. It can be fully accessible, as long as your code is just standard text, the string reader will read that.

Just made sure that the fonts you choose are legible. If they're super fancy, it's just been impact readability for everyone. So don't go don't go crazy with those. Do be careful with use of all caps. So when a screen reader is reading your page, it's going to do its best to read the content in a way that's hopefully legible. But it can't always differentiate between certain things. A lot of words in English have the same spelling but multiple pronunciations. Attribute versus attribute-- the screen reader may read it the wrong way.

We can't really control that don't worry about that kind of pronunciation. Some things are just going to be read weird. We're at Utah State University-- USU-- screen reader very often we'll

read it as USU. WebAIM-- sometimes it reads it as WebAIM. We can't control that. Don't worry about it. There's no hack that you wouldn't want implement to force the screen reader to read things a certain way. They just do it their standard way.

And the screen that users are typically used to that. They can navigate letter by letter if they choose to figure out the spelling of something if they want to. Things like numbers and things like that can also be read in a way that seems atypical-- my phone number is read 435. 797, and 8,284 It's not how I would read a telephone number, but that's the way the screen reader just does that in a standard way. Don't worry about those things.

[INAUDIBLE]

Yeah, so what happens is when you do all caps, this-- when a screen reader comes to something that it can't form as a word, it will very often read it letter by letter-- acronyms, abbreviations, things like that. Very often it'll read it like-- MTU, it's probably not going to say mtu. Although I was in a town in Africa that had -mtu-- M-T-U-- in the name. Sorry, I was there a couple weeks ago. [INAUDIBLE] was the name of the town. So it'd probably read it M-T-U, because it doesn't naturally form a word unless you're in Tanzania.

So-- but if you use all caps, it can kind of trigger the screen reader to more likely treat it as an acronym. An example might be a contact us link that's all uppercase. The screen reader may read contact U-S, because it thinks uppercase US is the United States as opposed to us. Or add to cart, all uppercase, it might read A-D-D to car. So we recommend CSS text [INAUDIBLE] uppercase.

If the all caps is purely stylistic, use CSS to style it as being uppercase. In your code just have it be standard case. Unfortunately, some screen readers will treat text transform uppercase the same way, and still may read it-- again, we can't really control that. But if it is stylistic, use CSS. All right, any questions about any of those items?

Another consideration with text is to be careful with full justification, where text is aligned on both the left and the right margins. When we read, we don't read characters. We don't look at letter by letter reading through the page. We don't even always look at words. When look at shapes and patterns and outlines of text and the ways in which words kind of flow together. And it's incredible what our mind can do to read through content very, very quickly without actually seeing letters and words.

When you're reading text, though, like a block of text, your mind very, very quickly adapts to that text. What is the font? How do the letters go together? That's why a certain font faces can cause things to be a little more difficult, like fonts where like I-- uppercase I, lowercase l, and the number 1 all look really similar.

When you get to that, your brain has to do this real quick, what is that thing? Is it an I or an L? Or when, like, characters kind of intersect a little bit, your brain just-- we do it incredibly quick, most of us, to decipher what those texts are based on their contexts-- the surrounding characters. And character spacing and word spacing has a big impact on that.

If you fully justify your text, what the browser does is it changes the word and letter spacing for every line to enforce the justification on both margins. What that causes is, every time, when you're reading the text, every new line, there's a subtle difference in those, and it impacts your readability. Your speed and-- every time you come to a new line, your brain has to reset just a little bit to that slight change in letter and word spacing to adapt to that text. So studies have shown it has a pretty notable impact on readability-- speed. At least for large blocks of text.

Another thing you can do with full justification is what are called reverse of white, because of that variable word spacing, you can see here, there's kind of this line kind of goes vertically up through that-- it's called rivers of white. It can be really distracting. As you're reading, your mind's kind of like, what's this other, this kind of white space that forms some sort of shape in here. It can also aggravate some users with dyslexia, just that little bit of spacing, something in there kind of that tends to cause the mind to jumble up characters a little bit more. So you'd want to generally avoid full justification.

A couple other things about fonts and text. So responsive design-- if you're not familiar with responsive design, it's the idea that you can have a web page, you can program that page so it responds, or adapts, based on the user environment. So you can have a page that looks one way on desktop, a different way on tablet, different way on a phone.

It's not three different web pages. It's one web page that adapts. So our home page has some basic responsiveness built into it. So this is our-- zoom that up just a little bit-- this is our default presentation of our web page. As I narrow the viewport width here, notice as I narrow it down, notice it adapts-- first of all, we have this-- we talked about line length before.

Let me just come to an article real quick. So our line lengths are adapted so there's kind of this happy medium. If I go too wide, I don't get really, really long lines of text. It just adds a whitespace around the edges. If I narrow this down, I can go narrow, narrow, narrow, to a certain point, and then it stops. I Can't get any more narrow than that to create really super short line lengths.

But you might have seen that there, some of that responsiveness. So as I narrow this window down, our default presentation has three columns. When I get to here, notice that that third column has adapted and kind of linearized down below. If I continue to narrow this down some more, it adapts again. Now I get a single column layout.

So this is more like the tablet presentation. This is more of the phone presentation. Also notice the navigation has oriented vertically in the top right. Again, this would be kind of more of the phone presentation. If I go narrow enough, the menu changes yet again and now appears below the menu so it doesn't overlap the menu. This would be a low resolution type of phone presentation, or a very small screen.

So there's one web page. We're not serving different content for phones or tablets versus desktop. It's one page that responds or adapts. It turns out that this is really awesome for users with low vision. Because, as you enlarge large web content, you're essentially making the relative viewport width smaller.

So decreasing your browser size to 50% is essentially the same as taking the content of the page and making it twice as big, right? The relatively-- you're dealing with the same thing. So if I start to zoom, or magnify, this page, watch what happens-- as I continue to zoom it, it gets bigger, bigger, bigger, right here-- actually right there-- I triggered the two column layout, essentially our tablet display. Here I triggered the phone display.

For somebody with low vision, this is awesome. You guys have probably all experienced on a phone where you have a web page has really long lines and you have to like pan back and forth, back and forth, back and forth, to read lines of text, just how much of a pain that is to have to horizontally scroll.

That's the reality for many users that use screen magnification if pages aren't responsive. They use their mouse and they kind of have to pan back and forth, back and forth to read the text within the page. Responsive design supports really good adaptation of that content for users that enlarge the page content.

This is is great. We did a training a few months ago. There was a lady on the front row she sat right up front, and she had like these huge really thick glasses. She was using screen magnification software, and we talked about this, and at the break she came up to us and she said, do you mean to tell me that in my browser, I can just hit Control plus, and pages, if they're designed that way, will respond so I don't have to horizontally scroll with my screen magnification, that generally just makes everything bigger.

Like, yeah. and It was like Ah, the angels singing, and it was like-- seriously, it was life changing. She was emotional about it. She's like, this changes everything. So my experiences on the web are going to change significantly now that I know that pages can respond. But only if they're built that way. You have to build these things into your content in order for it to work.

What you need to consider what this, however, is that your phone presentation might be seen by somebody on a high resolution desktop. You need to consider that. If you're significantly changing that experience, just know that your phone responsive design doesn't necessarily mean phone. It can mean something else. So their experiences may vary. Which is what you want. You just enable your page to support different experiences and environments, but just know that what they're experiencing might be different than what you experience or what you would even expect that they would experience, which is they've adapted that content to their needs.

So yeah, that zoom content will trigger those responsive break points, again, which is awesome. WCAG 2.0 has a 200% tech sizing requirement. But it does say that if you support zoom at 200%, you're good. Congratulations, you pass. It's pretty much impossible to have something that's readable at 100% when you make it twice as big isn't readable. WCAG 2.0 allows horizontal scrolling, so you pass. You don't really need to worry about that.

However, we know that there are some users that increase just their text size. Bump up their text size a little bit, like my grandmother, her vision's not as good as it used to be. She probably isn't using a screen magnifier. May not be using browser zoom, but she might increase her text size a little bit.

You want to make sure that your pages don't break when that happens. Typically what happens is when things are defined using pixel heights, so if you have a box of text you define as to be 100 pixels high, if the user increases their text size and it no longer fits within that block, it may bleed out of that element and slide under or overlap other elements or be presented in low contrast.

An easy solution to that that is to use min height instead. CSS min height will define a minimum height, so [INAUDIBLE] be the default height, but if the text inside of it expands and it'll no longer fit in the element, it'll expand or grow to accommodate its contents.

So I mentioned the requirement-- and that's not required in WCAG 2.0. WCAG 2.0 simply says if you make everything twice as big, it has to work. It works. It's almost impossible to break for standard web content. Consider users that increase their text size. In WCAG 2.1, however, there is a new requirement that's pretty significant.

This requires-- a short version of all of this is it requires 400% zoom without loss of content or functionality, and without horizontal scrolling. So you can measure this by sizing your browser window to 1,280 pixels. So I'm going to do this-- [INAUDIBLE] little technical here. I'm actually going to open up the developer tools.

With that open, when I change the browser window size, it will show me the dimensions up in the top right. So I keep pointing this way. You guys are all looking this way, [INAUDIBLE] up here. So you see the top right, it shows the browser window size. I'm going to size this to 1,280 pixels right there. Maybe get it exact. Doesn't matter. There we go, 1,280 pixels. And then I'm going to zoom this to 400%.

So I'm hitting Command +, Control + on Windows, so there's 110%, 125%, 150%, 175%, 200%, 250%, 300%, 400%. This is the new WCAG 2.1 requirement that you support this without horizontal scrolling or loss of content or functionality. So this page, we have designed to support this. All of the content is there. There's no horizontal scrolling required.

Short version is if you support this, you probably support the WCAG 2.1 requirement. If you have a good responsive design, you probably will meet this requirement. This does, however, essentially require responsiveness. Unless you have a very, very pretty much just plain text base-- if you have any kind of layout, you're going to want to consider this if you want to meet WCAG 2.1.

There are exceptions to this, for things that require a certain size. Things like data tables-- if I had a large data table, it's not going to fit in this very small view. Images where the sizing is important, like a map, a campus map. If I make it super tiny, it's going to be useless. So those things are omitted from these requirements. Questions about that?

Be thinking about this. This is a WCAG 2.1 requirement. I mentioned before, just being ahead of the curve with 2.1. So you want to start thinking about responsiveness, certainly for things moving forward, making sure those are built-- that's built into it.

Another WCAG 2.1 requirement deals with text spacing. There's a whole bunch of stuff here, basically what this means is, if the user of adapts their text and makes their texts bigger, increases the line height, the character spacing, things like that, that your page doesn't just fall apart.

So consider a user that has dyslexia. And they have a particular font face that works best for them. So they may override your font preferences. So I always view it in a particular thing. With that font face, they may also increase the word spacing. They may increase the letter spacing to decrease the likelihood that words are really close to each other. It just is easier. I mean, it's going to decrease the likelihood that I'll get letters mixed up if they're further apart, and words are further apart still.

And maybe the line height as well-- so the lines aren't close to each other. I have the ability as an end user to override and customize those things. WCAG 2.1 says if the user does that, if they adapt their text to these settings, that you have no loss of content or functionality. If you support what I mentioned just a few minutes ago, the user increasing their text size a little bit, you probably will meet this requirement.

As before, where this typically fails is height, pixel heights. Any time you see the word height anywhere, in HTML or CSS, if it's defined with pixels, if it contains text, it very likely will fail this. Min height is a good solution to that. Questions about any of these? You're being very quiet. Are you all just overwhelmed?

All right, couple other things that are important for accessibility. One is language of the page. This is very easy to use, very easy to define, very important for accessibility, especially for screen reader users that are multilingual, that understand multiple languages. In your code on the HTML tag, at the very beginning of your page, simply added lang equals, then the two character code for the language of that page.

So if I speak French and English, if my default language is French, and I come to your English page, if you have not identified the document language, my screen reader may start to read it with French intonation and pronunciation. It probably isn't going to make a lot of sense. I have an example of this. So this is English text you're going to hear it read by a screen reader that thinks it is a different language.

[INAUDIBLE]

I probably don't need to say anything more about the impact that this can have if it's not defined, and doesn't align, or if there's a mismatch. This actually Czech-- the Czech languages is what this is. So we had somebody from Czechoslovakia in a training a few months ago, and they're like, I recognize that, I know what that is. I know the letters 1, 4, and 0 and Czech pretty well now. So make sure that you've identified this correctly.

This is a WCAG 2.0 and 2.1-- [INAUDIBLE] a little clarification I need to make. WCAG 2.1 only adds to 2.0. It doesn't change anything in 2.0. It only adds additional success criteria-- it extends it. So everything that's in 2.0 is still there with 2.1, it just adds new stuff.

Now, with that said, there are a few areas where new things and 2.1 kind of negate what's in 2.0. We just talked about one of those-- 2.0 says you have to support 200% zoom. You do. 2.1 says you have to support 400% zoom without horizontal scrolling or loss of content functionality, and you have to support text adaptation. That kind of makes the 200% zoom useless. I mean, there's-- it's such a broader requirement than the 200% zoom, 2.0 is kind of doesn't do anything, right, unless you're only looking at 2.0.

Yeah, so if it was really-- that would be an exception to that. There are some ways though that you can set character hyphenation. So it'll break great words. Browsers do a pretty good job of that. Overflow word rate-- there are some things you can do that may not be appropriate. And if it's not appropriate in those cases, that would be attempted. Does that make sense?

So this is a single A WCAG requirement-- to define the language of the page. Easy. Lang equals whatever the two character code is. There's also language of parts. This is a WCAG 2.0 AA requirement. This is for content within the page that may be in a different language from the rest of the page. So have an English page, within the page you have a quote in French, just give it div lang equals FR. So the screen reader, if it supports English and French, would read the page in English. When it gets to the code, it would switch to le sexy voice, and read in French, then back to English when it's done.

Also really important for multilingual users-- only do this for words that are truly, or content that's truly in the other language. Things like piñata, Los Angeles-- I mean, the root might be in Spanish, but they're English words. They're part of English now, they'd be understood generally by English speakers.

Where we see this most commonly impactful are things like language choosers. Let's say you have a footer, and there's a list of links for different translations of your content in the footer. If the text of that link is, for instance, Japanese-- it says Japanese-- that's English. Don't give it lang equals JA. Because it's-- the word Japanese is English.

If the characters are in Japanese, then give it lane equals JA. Then when the screen reader gets to it, if it understands Japanese, it will read the Japanese-- whatever that content is in Japanese. Now, if the screen reader doesn't understand Japanese, and it comes to that link, it may not read anything, because the Japanese characters can't be read in English. So it may read nothing.

Voiceover on Mac will actually say Japanese-- it'll give you an indication hey, there's something here, it's in a different language that I'm not set to speak. So you're aware of that. And you can't control that. Make sure you mark these things up appropriately. Questions about language? All right, OK.

Next topic is tables. We can use tables on the web really for two different things. One is to use tables to control layout or positioning of content within the page. Where we have this layout here, where visually we've associated text to values below it, just to lay out and create this nice kind of visual columns of information.

Screen readers will read content, as I mentioned a few minutes ago, in the order of their source code. So if you define this in HTML using a table, a screen reader would read that content left to right, top to bottom, row by row. So for this layout table, it would read average score, high score, low score, standard deviation, average time, 39%, 67%, and so on. It's not going to be very useful to a screen reader user. How could you fix this? How could you address this with this table? What would you maybe do differently?

[INAUDIBLE]

That's one way, is you could do a list and lay this out somehow. Good. Other ideas? What's that?

[INAUDIBLE]

Yeah, so like this. Is that what you mean? No?

[INAUDIBLE]

Oh, OK. Yeah, yeah, so that would be another way to do it, or this-- merge them together. There are a few different approaches, just so that the text and the values are associated together. List vertical table, merging the cells, in this horizontal table, all of those cause it to be read appropriately. A few different approaches there.

Layout tables are not the end of the world. You're not supposed to use layout tables in HTML 5. Of course, we know the rules of HTML are meant to be broken. We're not supposed to use layout tables. We have so much flexibility now with CSS that we can control layout. It's not usually a good idea, but they're not the end of the world when it comes to accessibility if the reading order makes sense.

Now, with the new WCAG 2.1 responsive requirement, layout tables are probably going to be a little difficult to meet that, and also make it responsive. But again, when it comes to accessibility, you want to fight the right battles. If you have some system that uses layout tables, totally restructuring an old systems to support CSS as opposed to layout tables probably isn't the best use of your time. Put it into other areas moving forward.

What tables are typically used for on the web, however, are for tabular data, which is what we will cover after the break. So let's take 15 minutes, still lots to cover this afternoon. Take 15 minutes, we'll get into data tables and then some on forms.