WebAIM Campus Training-Session 4

We're going to continue on with data tables. Tables are intended for tabular data, laying out information in a grid. That's what tables are for. Because layout tables have fallen out of favor-- you're not supposed to use those in HTML5-- sometimes we hear people say things like, tables are evil, you really shouldn't use tables. No. Use tables when it's appropriate. If it's tabular information, use tables. That's what they're intended for.

If we looked at this very basic table-- when we look at it, we can make some associations. First of all, we know this is the class schedule table. We know that, because we've made an association, a visual association between the words class schedule and the table that comes below. We do that based on the proximity and positioning of that text in relation to the table. Someone that's blind can't make that visual association, so we need to make an explicit or programmatic association.

We can also look at, say, BIS 5650. We know that that is the course number for advanced website development. We know that, because we visually scan up or down, left or right to create the association between that cell and its respective column or row headers. Someone that's blind can't make those visual associations, so again, we need to in our market create an explicit association.

So we make this accessible doing three things. First, we use the caption element. Caption would be the first thing inside the table. It will describe what that table is. It will associate that visible text above the table to the table itself. Now, not all tables have to have a caption. But if you have a piece of text that describes the table associated to the table using the caption element, that's step number one.

Step number two is identify all of our table headers using the th attribute, th for table header, as opposed to td for data cells, table data cells. Step number three is identify all of our table headers as being either column headers or row headers using the scope attribute. So th scope equals col for column headers. Th scope equals row for row headers.

If we do those three things, we will have created all of the associations necessary for this table to be accessible. We associated the caption text to the table. When the screen reader comes to this, it would read, class schedule table. So it tells them what that caption of the table is.

The screen reader will generally then say something like three columns and three rows. For a data table, the screen reader will trigger special functionality to begin to navigate that table using the arrow keys. Most screen readers, it's control, option, and then the arrow keys to navigate through that data table.

When we visually consume data table content, we typically scan through columns of information, or we scan through rows of information. Somebody using the keyboard to navigate this table will probably do the same thing. Because we have created the associations between the row and header row-- the column and row headers to the data cells, as the screen reader user navigates the table, there are some efficiencies built in. And it will read the appropriate headers as you navigate.

So let's say that the user started on the BIS 5650 cell. If they navigate to the right, the screen reader would read location B 105. It reads the column header, because we've changed columns. We moved into a new column, so it tells us the header for the column that we moved into. If the user then navigates down a cell, it would read database management B 220. It reads the row header, because we changed rows.

So if you're navigating down a column, it will read the row headers as they change. If you're navigating across a column through a row and the column headers are changing, it will read the column headers as those column headers change. So it doesn't repeat every header on every single cell, only the ones that you need, because they've changed. So there are some really nice efficiencies built in if you create these proper associations.

So if the screen reader user gets confused-- because I'm getting confused processing that-- can they ask the screen reader to re-read those? I need [INAUDIBLE] my row and column headers.

Most screen readers have functionality to do that. So let's say we're in that bottom right hand cell. You could hit a key command that said, give me the context I need, and it would read something like class schedule table, three columns, three rows, column three, row three, location, database management B 220. So it tells you the size of the table, where you're at in the table, and the row and column headers, and then the data table for it. And that's going to vary a little bit based on the screen reader.

If you navigate to the end of a table, or any border, edge of the table, most screen readers will say something like bottom, top, or end, or blank, something like that, just to indicate that there's nothing there. You can't navigate past it. It's pretty cool. Not very complex markup to create those necessary associations. Yes.

I was just going to mention if CMS [INAUDIBLE] documentation [INAUDIBLE] using WYSIWYG in our CMS on our webmaster's blog. And if you're a programmer at Michigan Tech, we have some documentation just like we talked about on our accessibility website.

Awesome. So your CMS supports table headers. Does it also support scope?

Yes.

Sweet. Cool. So those are settings that's right in your CMS. If you're creating a data table, you can enable those options. Now, you've got to make sure you set those up correctly. If you have a table-- maybe you're using a table for layout, don't mark it up as a th, table header. Don't give it a scope. Don't give it a caption. That can really mess things up. Make sure your table headers are marked up appropriately. That's cool if your CMS supports that, then you can define all of this correctly. Good.

Sometimes, you may have more complex tables. Now, using scope-- in this case, scope of col, column headers for both of these, this would work. This would create the proper association. In the fall semester, column header would be associated as a header for all three of the columns that come below it. But it's going to add some overhead. There's multiple layers of column headers. What would you maybe do instead on this if you could simplify it?

[INTERPOSING VOICES]

Yeah, make it a caption. As much as you can, flatten your tables. Try to avoid spanned cells, especially spanned header cells. You can generally make them accessible. It just adds a lot of complexity, a lot of code.

And for a screen reader user, it can be kind of complex to have multiple levels of headers being read. So simply flattening your table-- almost always, if the first row of your table spans the entire table width, it almost always should just be a caption instead. And then it would be associated as, essentially, the descriptor or caption for that table.

Jared, just backing up a minute on tables. So the caption element was-- sounds to me like the equivalent of alt text for a table.

Kind of.

Like in Word, for example, there's actually a table property field for alt text, it's called alt text. Are they treated the same when the screen reader [INAUDIBLE]?

So one thing with caption is the caption does show up visibly, so it will be text visible. It just associates that text as the descriptor or caption for the table that comes below it. In Word, there is an alt text option for tables. We don't usually recommend adding that, typically because if the table is structured correctly, it will be identified as a table. The user can navigate the table and get the content. The alt text is probably going to be extraneous, extra stuff, screen reader only stuff that may not be necessary.

If your table needs a piece of text that describes-- tells you what it is, usually it's good for everybody to see that-- put it in the page itself. Now, there isn't a way in Word, for instance, to associate this text as a caption for the table below it. You can maybe make it a heading, and it serves as the heading for that. By the way, you can put headings inside captions. So if you have a piece of text that not only describes the table, but maybe describes a section of content that includes the table, you can put a heading inside that caption. That's perfectly fine in HTML5.

Usually, alt text on tables in Word doesn't work very well. Also, if you convert it to PDF, I don't believe that alt text does anything in PDF-- alt text for table. Other questions about this? Yes.

Do you have a skip link for tables?

Good question. Skip link for tables is usually not necessary. If it's a very large table with a lot of content, a lot of cells, the user navigating-- they need to get through it-- screen readers do generally provide mechanisms like skip to the next thing if you have a good heading structure, things like that. Probably not.

If it is a really large table, there may be a link to skip past that table. It might be helpful. It solves one potential problem but introduces overhead. And it's now an extra link the user has to figure out and consume and hear. So maybe.

Obviously, if the table has headings [INAUDIBLE], I would assume that means you wouldn't want that whole section and skip to the next page.

If you have good other structures, the user can navigate by those. Usually not. There might be some cases where it could be helpful. It does bring up one thing real quick I want to mention.

Sometimes, people will say something like that. Well, in this case, skipping the table really is only helpful for screen reader users. If I'm a sighted keyboard user, I can consume the complex table, or I just skip that and just look past it real easily.

So sometimes, there is this idea that, well, let's add this skip link, but let's position it off-screen. That way, it's only available to the screen reader users, except it's not. If it's a link, sighted keyboard users can navigate to it.

If I hit Tab and I come to a link that's off the side of the screen, I can't see it. I don't know where I'm at. I just hit Tab. There's no focus indicator anywhere on the page. I can be confused by that. So it's one of the issues-- I wouldn't call it an issue-- a feature.

One of the realities of links is if you can navigate to a link-- well, if you can navigate to anything that's functional, it should be visible in the page. So I know you weren't asking that, but sometimes, we see that technique. And in the end, it solves one potential problem but introduces a pretty notable issue for sighted keyboard users. Good. What do you do if you have a much more complex table, something like this? What would you maybe do instead here?

[INAUDIBLE]

What's that?

Two tables.

Yes, two tables. Separate this out. Again, you're flatting the table, use caption, works much better. For some tables, you might do something like this. So here, I've added a semester column that repeats the information that was duplicated here. And it's a little bit of duplication in the content, but it still works. It's a nice, flat table. I can use th with scope to create all of the associations necessary. It's a pretty basic markup.

If you tried to use th with scope on this complex table, it's not going to work. Here, I have column headers that are associated to cells that are not correct. So for that bottom cell, it would read, fall semester course number, winter semester course number, BIS 3330. Well, is it fall semester or winter semester? Scope does not work in this case.

Sometimes for very complex tables, it just is not going to work to properly create those column or row scopes for your headers. There is an alternative approach that uses headers and ID. It's a lot of code. Basically, every header cell is given a unique ID. Then every single data cell, every single td is given headers attribute, the references, the IDs of the header cells.

So the relationship is kind of opposite. Instead of saying, I'm a table header for this column or I'm a table header for this row, headers and ID says, I'm a data cell, and this is one of my headers, and this is one of my headers, and this is one of my headers. So this works. It creates the semantic association.

But it's a lot of work. It's a lot of code, a lot of stuff you have to put in your page. You have to get it right. Usually, it can be avoided just by simplifying, or flattening your tables, or splitting it into multiple tables. Very often implemented incorrectly, which is you have to get all of these references, all identified correctly. Screen reader support isn't as good for headers and ID. It's not supported on VoiceOver on Mac at all.

Also, that efficiency that we got before where it only reads the headers when they change. With headers and ID, it doesn't do that. If you navigate into the cell, it will read all of the headers. It doesn't really know which ones you'd think it should not read. So it reads all of them, so you lose that efficiency.

This is certainly a case of technical versus functional accessibility. I don't think I've ever seen a table that's so complex that it requires headers and ID to make it accessible that was actually usable. If you're a screen reader user and you start hearing three or four or five or six levels of headers for every single cell, it's just not going to work very well. So try to avoid it just by simplifying your tables.

A couple of other last words on tables. I kind of mentioned this before, but only use data table markup for data tables. Screen readers treat data tables very different from layout tables. A layout table doesn't even tell you there's a table. It just reads the content based on the source order, because it's for layout. It shouldn't even know that it's a data table. It shouldn't know that it's a table at all.

But if you use something like th or scope or caption on a layout table, maybe for layout purposes, it may-- the screen reader will generally think it's a data table. That may cause some content to be repeated for other pieces of content, because it thinks it's a header for that content. It just turns into a big, icky mess. So don't do that.

Screen readers also when they come to a data table-- they can sometimes guess. For instance, if you have a layout table that only contains little bits of information-- like that first table I showed you with just the single-- the two rows of information-- there's a high likelihood that when a screen reader comes to that and says, wow, this is a table. It looks like there's little pieces of information. I'm going to assume it's a data table, even if they haven't marked it up as one and may start to read it as a data table.

That can be a mess if it's not actually a data table. It was intended for layout. You can address that by giving any layout tables role equals presentation. This is some ARIA, but it basically says, hey, this is a presentational table. Don't read it as a data table. Just treat it as a layout table. In other words, don't read it as a table at all. Just read the contents of the table.

Really, if you're doing layout tables at all, I recommend role and presentation. You want to avoid empty th. Kind of like headings, headings always describe a section of content. Table headers always describe a column or row of data, so they should always contain text or an image with alternative text. It should never be empty.

So that top left hand cell, sometimes it's a column header. Sometimes, it's a row header. Sometimes, it's neither. Sometimes, it's empty. If it's empty, make it a td, not a th.

Screen reader and browser support for data tables is still not perfect, which is frustrating. It's not like tables are new. But screen reader supports are sometimes still quirky. But sometimes, if you have a th that's empty, you can shift the header cells. And so it will actually read the wrong header for the wrong column or row. It's really weird. Just avoid that.

There is something called th abbr, abbreviation. It just allows you to have a shorter piece of text that's read when you're navigating inside the table, as opposed to reading all of the cell contents when you get to it. So say you have a first name header with a Sort button, when you get to that table header, you want it to read the Sort button. But when you navigate into that column later on in the table, you don't want it to read, first name, button sort ascending every single time. You only want it to read first name.

So abbr allows you to have an abbreviated header text when you're exploring the table. There is in HTML something called table summary. This is kind of like alt text for a table. It allows you to add content that would be read only by a screen reader. It was intended to allow you to add a summary of the structure and layout of the table, like this table has 27 rows and 14 columns and is structured this way, something like that.

We don't recommend it. First of all, summary has been dropped from HTML5, so no longer part of HTML. It isn't supported in some screen readers at all. And it was kind of like a bad joke. If you have to explain it, it wasn't a very good joke to begin with.

If you have a table that's so complex that you have to explain its layout and structure to a screen reader user, it's probably not very usable to begin with. And you've addressed that by now giving them more stuff. It just doesn't work very well. We don't recommend table summary.

As I mentioned before, use table where it's appropriate for data tables. Where tables can sometimes be problematic is for responsiveness. I mentioned before, with WCAG 2.1, the 400% zoom data tables are exempted from that. If you have a data table that doesn't fit in those, essentially, 400 pixels or 320 pixels, then that's OK. But sometimes, just for responsiveness, it can be a little bit tricky.

Sometimes, for instance, you might have a data table in your desktop presentation but on, say, a tablet or phone, it becomes a list-- like heading with list below it, heading with list below it, or nested list, or something like that. It's more of a linearized presentation. The difficulty with that is it's really difficult to adapt a data table into a list and vise versa using one piece of code. Unless you add a whole bunch of scripting to it, it's just kind of ugly. Tables don't respond that well.

So sometimes, what we see is people try to recreate a data table using all of these divs. And then on phone, the divs kind of restructure, so they look like a list. The problem is if you're using divs, floated or a grid, without actual [INAUDIBLE] market, there's no semantics. It's not going to be accessible. Screen reader will just read the content. It won't read it actually as a data table. Scope, things like that aren't going to do any good on a div, only on th elements.

So one approach there is to actually have two versions of your content in your page, one in tabular format, one in a list format. In the desktop presentation, show your table, hide your list. And in, say, mobile, hide your table, show your list. And that, again, is kind of contrary to the concept of responsiveness, because you have two versions of it. That's probably going to be a much easier-- no, it is going to be a much easier approach and probably more accessible than trying to actually adapt these things into two totally different things.

Questions about tables? We're going to move on and talk about forms. Quite a bit in forms. Beyond links, forms are probably the most significant thing on the web as far as functionality in what the web actually can do. So there are a lot of considerations for accessibility.

One thing of note is there is in HTML5 a lot of new input types that can be used. There's actually a whole bunch of different types of date format kind of inputs. There's week, year, things like that. There's time, search for our search field, URL for web page addresses. Number will limit it to number inputs.

Range will create a basic slider, numeric slider, so you can increase or decrease the value. Color will generate a color picker. The WebAIM contrast checker, that color picker that popped up, that's done-- all that is is input type equals color in our code. And then the browser and operating system will present the default OS color picker, email for email addresses, tel for telephone numbers.

You probably have seen these in use, especially on mobile-- have really good support on mobile. Maybe you've clicked on the telephone number field, and you get the dial pad input. Or an email address field where the keyboard adapts, and you get the @ sign, and/or URL where you get the slashes and the dot.com little button, things like that. Those are simply by using these input types. So start to use these. They're great for accessibility.

Standard browser support is getting better for them all the time. So if you're collecting this type of information, use these input types. It will enhance accessibility. There's basic browser validation and input controls that are built into these. That's the idea. We're just putting more of the burden on the browser to create the interface components as opposed to us having to build them.

How many different date pickers have you experienced on the web? Probably thousands. At least I have, probably thousands of different types-- different presentation, different functionality, buttons. So many of those.

Which is going to be more likely, that we can convince every developer on the planet to build an accessible date picker, or we can get a few browser vendors to support default input type equals date, so it's accessible? Certainly the latter, and that's the idea. We don't have to build these things or plug them in anymore. We let the browser do that work for us and allow it, hopefully, in an accessible way.

So start to use these things. HTML form labeling is really the foundational aspect of form accessibility. Visually when we look at this text box, we know it's for first name. We know that, because we've made a visual association between the text of first name and the text box that's next to it. Someone that's blind can't make that visual association.

As with tables, we need to explicitly define that association in our code. We do that using the label element. So label will surround the text that is the label for our control, and then we use the for and ID attributes to create the associations. So label for equals F name says, I'm the label for the thing on the page that has this ID. In this case, it's the input has ID of F name. That creates the association, the I'm a label for this text box.

With that association now in place, when a screen reader comes to this text box, it would read first name, edit. The screen reader would say the word edit for text boxes. That's what the word edit means is text box. So we've created that association. This will be now accessible to screen reader users.

So we must provide associated labels for text boxes, text areas, multi-line text boxes, select menus, checkboxes, and radio buttons. And that's all, nothing else. The label element should not be used with anything else. It probably won't do anything, and at worst, will break it. So this makes it accessible.

There is an added benefit of using the label element, and that's for mouse users. If you have an associated label to a form control and you click with your mouse or a touch screen on the associated label, the associated form control will become focused or activated. So I could click on the words first name, and the cursor will be placed in the first name text box. And I can start to type.

This is especially helpful for checkboxes and radio buttons, because they're small. They're really tiny. My friend that uses the trackball on the floor-- for him to click on a checkbox, especially in a series of checkboxes, it can be a little difficult. But if you associate a proper label, he can now click anywhere on the adjacent text. It's makes a much bigger clickable area where he can click to activate that or check or uncheck that checkbox. Really helpful, again, also on mobile.

Now, you might be saying, why do we need to create this association? If a screen reader is reading through this page, won't it read the words first name and then read the text box? Yes, it will. We need to keep in mind that there are two primary ways that the screen reader users can consume information. They can read through the page, or they can navigate through the page, usually hitting the Tab key to navigate from form control or some other thing within the screen reader.

So if I'm simply reading through this page, it would read first name, edit. I know that's where I'd put in my text box. That's the text box where I'd put my first name. But what's going to be more likely is I'm hitting the Tab key. If I hit the Tab key and I jump into this text box, it just skipped over the text. It just skipped over first name.

If I haven't made that association, the screen reader says, edit. Edit for what? I know it's a text box, but I don't know what it's for. I hit Tab again and go last name. If I haven't created the association, it says edit. So I can hit Tab. I hear edit, edit, edit. I then have to explore around that.

Usually, the labels come before the fields, except for checkboxes and radio buttons that come after. It's just really difficult. We want to create these associations, especially for forms. Because once I'm in a form, I generally am going to fill out a form control and hit Tab, Tab, Tab to navigate from form control to form control, skipping over the text. I have to have these associations, so they're read in conjunction with the fields themselves.

There is an alternative approach to this. It uses implicit form labeling. What that allows us to do is put the text and the input both within the label itself. This will also create the association. It does the same thing as using For and ID, which we call explicit form labeling.

So implicit works. It's just a little more limiting in that the text has to be immediately adjacent to the control in our code, so to me, limiting as far as layout, and positioning, and things like that. Some other little quirks with this-- we usually recommend For and ID, because it is explicit. The label can be put anywhere in the same page. It definitively says, I am the label for this control. But this can also work.

What you don't ever want to do is try to combine them. What that means is to use For and ID but put the input inside the label element. This will cause issues. What this says is that the input is part of the label for itself, and it can cause some duplication and just some oddities. So don't do that. Either use For and ID with the input outside the label, or don't use For and ID and put the input inside the label, never both.

There's also fieldset and legend. First of all, before I go on, are there questions about form labeling? Fieldset and legend can be useful for groups of checkboxes and radio buttons. Here, we have a set of radio buttons. We know that these radio buttons are for choosing the shipping method, but we've made a visual association between that choose a shipping method text and the group of radio buttons that come after. Someone that's blind can't do that visual association.

So we can associate this as a group using fieldset. It's a set of form fields, a set of fields, a fieldset. So fieldset will surround the grouping of controls. Every fieldset should have a legend that describes that fieldset. In this case, it's choose a shipping method.

With this now in place, when the user hits the Tab key and comes to that first radio button, it would read something like, choose a shipping method, radio button overnight. So it reads the higher level legend for the fieldset. Don't go crazy with these. Usually, it's just groups of checkboxes or radio buttons where there is a higher level piece of text that describes them. That's really where it's going to be most useful for accessibility. Yes.

Is it useful, for example, on a checkout form where there's the shipping address and billing address?

It can.

[INAUDIBLE]

One of the difficulties with that for large groups of form controls is billing address, shipping address, for instance, we put those in fieldset with legend. Some screen readers will read the legend for every control inside the fieldset. So you'd hear like billing address first name, billing address last name, billing address address, billing address city. And it just can get a little repetitive.

Depending on the layout presentation, it might be helpful. Sometimes, in those cases, we absolutely recommend headings instead like an h2-- h2 for billing address, shipping address. As long as, hopefully, there are some other-- it's kind of clear. There's going to be a visual indication that it's billing address, shipping address. So it's not wrong to use fieldset. It just comes with a little overhead that might be better addressed with headings instead.

Remind us of [INAUDIBLE] a tab goes to headings and links?

No. And that's the downside of not using fieldset. When you hit the Tab key, it will only tab to interactive controls, links and form controls, not headings, not tables, those things. So if you don't use fieldset and the user just hits Tab and they come to the first control, they may skip the billing, the address text. They may not know that it's there. So that's the downside of that.

You would just maybe want to consider your presentation. That might. I don't know. If billing address and shipping address were part of a larger form, especially if there were form controls up here, I would probably use fieldset. That ensures when I'm tabbing up here and I jump into billing address, it's going to read billing address, repeated every time.

If the first field is the first form field in the page, then maybe headings might be better. So it's less likely. It's more likely that I would read the heading before I get into the form. There's not a perfect answer there. It's a great question. Any other questions on fieldset and legend?

Sometimes, you may have form controls that do not need or do not have a visible label, something like a search field where based on the visual design, maybe the button design or the positioning within the page. Visually, it's apparent that it's for search. There's no text next to the field to associate as the label for the text box.

So there are a couple approaches here. One is to use the technique we talked about before, the off-screen text. So we'd give it a label. We associate it. The text of search terms is an associated label, but we've positioned it off the side of the screen. So visually, it makes sense that it's search, but a screen reader user when they come to this, it would read search, terms, edit.

Now, make sure that label is not hidden with Display None. If it's hidden with Display None, it hides it from everyone, so it would not be read. So that's one approach.

Another approach is to use the title attribute. So the title attribute can be used pretty much on anything in HTML. It will generate a mouse tooltip if you hover your mouse over that element, so it creates that little tooltip lock next to the mouse cursor.

This will also be read by the screen reader for form controls that do not have a label. So functionally, this will work. The only difference between the off-screen labels is this will generate the mouse tooltip, which you may or may not want.

Title is intended for advisory information. Keep those terms in mind, advisory information. In other words, if it's helpful and advises, that's great. But if the user doesn't get it, no big deal. You can't really rely on it being read, except for, generally, a few instances.

Form controls that are missing the label, it will be read. That's pretty universal in screen readers. Frame titles-- I talked about before-- will be read, frame and iframe titles. Acronym and abbreviation, again, it depends on the screen reader and the screen reader settings. It generally will be read.

We usually recommend usually avoiding or being very careful with abbreviations and acronyms, meaning abbr element or acronym element. If the screen reader supports it, it will always read the expansion. We just recommend, if you have an acronym or abbreviation, make sure it's commonly understood or expanded in text. Or if we have an article about WCAG, we'll put web content accessibility deadlines, parentheses WCAG, and then we just use WCAG. No need to do abbr or acronym after that.

So beyond those use cases, title may or may not be read. It depends on the browser. It depends on the screen reader. It depends on the screen reader settings. Every screen reader has a verbosity setting. How verbose does the user want their screen reader to be?

If you crank it all the way up, it'll read special characters. It'll read punctuation. It'll read title attributes. It'll read everything that it can find that it thinks might be helpful to you.

Crank it all the way to the bottom, it will only read text. It'll ignore special characters, punctuation, things like that. The default's somewhere in the middle. For some of them, it'll read title attributes. For some of them, it won't. For some, it depends on the type of element. On Mac, it tends to read title attributes more often than on Windows.

Essentially, if you have information in the title you want to be read because it will be helpful for accessibility, assume it will be ignored. If you have stuff in title that's extraneous or repetitive of visual content, assume it will be read and cause overhead. So it's kind of the worst of both worlds but be careful with it.

It's also not accessible to keyboard users. You can never visually see that title attribute tooltip using the keyboard, also inaccessible to touchscreen users. It doesn't matter how long you hover your finger over that thing, you're not going to get the tooltip.

So hover interactions are really dead on mobile. And if you say something's dead on mobile, you should probably just say it's dead. So you wouldn't want to rely on that title. Now, there might be some cases where it could be useful as truly advisory information, maybe like a gear icon.

You can maybe presume that your users know that your icon is for settings. But giving it title equals settings could be helpful. If the user hovers over their mouse, they see the tooltip. It reinforces that it's for settings. If they don't ever see it, don't ever hear it, it still works. It'll have alt text of settings for screen reader users. It's going to be fine.

If you do use title and you're duplicating visual content or alt text, make sure that it's identical. So if you have an image button, it has alt of settings, if you give it a title attribute, make sure it's settings. Same case everything. That will decrease the likelihood that the screen reader will read it twice.

If you have alt of settings and title equals preferences, the screen reader will probably say something like settings, button, preferences. It's going to read both of those, so make sure they're the same if that's appropriate. Questions about title attribute?

Another real big one that we see a lot on the web is placeholder. Placeholder puts the text inside the field temporarily until the user enters the field, and then it usually will disappear when you either click or start to enter information within that. It's usually read by a screen reader if there is not a label or a title. And if there is placeholder, it'll generally read it, even though it's not supposed to. The HTML spec specifically says that label-- the placeholder is not to be used as a label. And so don't treat it as the way of identifying the functionality of the field.

Now, if the screen reader comes to a field and there's no label, there's no title, but there is a placeholder, usually the screen reader will read it. Since I've got something here that might be useful, I'm going to read it. I'm going to present it to the user, even though you should never put the screen reader in that scenario. You should always have something that you know will be read.

As before with the title attribute, if you use placeholder, make sure it's the same as your label text. If the intent is to have them be the same, make sure they're case sensitive, exactly the same.

So if you have placeholder of search terms and a label of search, it may read edit-- search edit, search terms, which is a little redundant. Have them be the same.

Interestingly, most of the issues with placeholder are not screen reader issues. There are other general usability issues, primarily contrast. By default, the placeholder text is usually presented in lower contrast by design. If it's presented in high contrast, the user may confuse it as being user entered text.

In the very most recent version-- within the last month or so-- Chrome, at least, has started-- has increased the default contrast of placeholder text. They put it right at 4.5 to 1. So it passes WCAG. But that's also causing users now to more frequently confuse that placeholder text as being user entered text. So they addressed the contrast issue and introduced a usability-- fixed one usability issue and introduced another.

Other browsers, it is low contrast. So if you're relying on the placeholder text to convey the functionality of these fields, as is the case here, this is clearly a WCAG 2.0 contrast failure. This is a screenshot right from the WebAIM site. In this case, we have an off-screen label for screen reader users. The visual design and button indicate that this is a search field.

Is this a WCAG failure if this is below the WCAG contrast threshold? In other words, if the user can't perceive that text, is it inaccessible? I would say no. If you got rid of the text altogether, it would still be accessible. So in this case, the placeholder is kind of an enhancement.

Is it technically a WCAG failure though, because it's low contrast? I don't know. I honestly don't know. WCAG doesn't really talk about that. It doesn't really provide that scenario. This is what we've chosen to do, and I think it works. Now, if you're relying on the placeholder, clearly a contrast failure if it's below 4.5 to 1.

Other issues with placeholder is that text is in place until the user starts to enter something within the field, then that text disappears. So if you're relying on placeholder to identify the functionality of the fields, as soon as the user put something within those fields, there's no reinforcement. There's no way to go back to know for sure what it is.

So you could get scenarios, for instance, here where you have last name, first name. Looking at this, there is no way to know that that is incorrect. You'd have to actually remove the text and see the placeholder to know that it should be first name, last name. This is general usability theme.

Another thing that's interesting here, notice the expiration field. The users entered 1/20, and it says, enter a valid expiration date. Is 1/20 not a valid expiration date?

Well, if you looked at the placeholder text, it indicates that the format is mm/yy. Once users put something in there, there's no indication that that's the proper formatting, which is they relied on placeholder text. How could you address that, in this case, how could you fix that problem?

Put it in the error message.

Yeah, you could put the formatting in the error message, enter a valid expiration date, mm/yy, or put it in the label itself, expires parenthesis, mm/yy. Interestingly, in this case-- this is from the PayPal site some time ago-- they actually had an off-screen label that included the formatting. So it's accessible for screen reader users all the time, but for visual users, that formatting was not indicated, except for in the placeholder text.

Another thing I want to quickly highlight here is the red borders around the invalid fields. If you're relying only on that red border color to indicate error in fields, that would be reliance on color. Gray versus red, that's only a color change to indicate they aren't in fields.

Now, in this case, they use the triangle error icon in addition to that. It's no longer reliant on color. It uses the red border plus the icon to indicate it, except for the CSC field. They didn't have that icon. They're only using the red border. That is a color reliance failure in this case.

One solution-- I guess you might call it a solution. One thing that we've seen implemented more and more are floating labels where you actually have what looks like placeholder text. It actually is the label element associated to the field. But that text, instead of being before or adjacent to the field, it's positioned over the text box or field. When the user clicks into it, that label transitions away, and the user can enter into that field directly.

So for a screen reader user, this is perfectly fine. As long as that text that's associated is labeled, they don't care where it's at or what it looks like. They just want to hear it.

For sighted users, there can be some potential issues with this. One-- at least in this presentation-- is that the text is really small. Once that transitions out, that text is super small. Remember, WCAG doesn't have a text sizing requirement. Nothing wrong with this as far as WCAG goes, but for usability, it's pretty small text.

Notice in the default presentation, the placeholder text has low contrast. Is it a WCAG failure if by default it has low contrast, but when the user clicks in the field, it then has high contrast? I don't know. I would think so. If I'm just looking at the page and I can't read it, it kind of smells like a contrast failure to me, but WCAG doesn't specifically define that.

And then there's maybe the potential for confusion. This is just a new kind of interaction. Really nice for mobile, just because of the limited real estate. But that little animation moving of text could be potentially confusing for some users.

There's been some usability studies on this, and they've ranged from, wow, these things are awesome for mobile, totally the most usable things ever to, they're literally Hitler. They're like horrible for usability. I don't know what that means.

If you have a design constraint to add placeholder text, this is a much better approach than relying on placeholder for accessibility. But I probably wouldn't make this your default just because of the potential issues.

Jared, what are these using? [INAUDIBLE]

Usually it just scripting. The text that you see is actually a label element associated, so label for. And then it just uses CSS and positioning. So when you click in the field, basically on focus, change the styles of this to move it above the field. So it's pretty basic HTML, and even the scripting isn't very complex to do that basic transition.

I've actually seen this done entirely with CSS, no scripting at all, using focus within, which is a cool new attribute in CSS, not supported at all in older browsers but focused within. If you focus inside, this thing changes the style properties of this. Questions about forms stuff?

Now, I didn't get into some of the ARIA stuff. There's a lot of stuff you can do with ARIA to enhance form accessibility. You can do ARIA labels, ARIA labeled by, ARIA descriptions. Those can help overcome some of the inherent limitations of HTML labeling.

With HTML labeling, you can only have one label for one form control. It's a one-to-one relationship. Some forms might have multiple labels, or you might have data entry where you have multiple fields that are visually labeled by one piece of text

ARIA allows you to overcome that by creating any types of associations between form fields and pieces of text on the page. You can add secondary descriptions to elements like a password with password requirements below it. That's important information that you want the user to get, but it's not really a label. If you were to plug all of that in the label, it becomes a lot of stuff. Then you have multiple labels. You can't do that with-- you'd have password and the requirements below it.

With ARIA, you can have a label, a password and then have this text become a description, so it's read in association with that form field when the user gets to it in a screen reader. You can identify required fields, invalid fields, disabled fields, things like that. There's just a lot of really cool things you can do with form accessibility with ARIA, keeping in mind the first rule of ARIA. If you can do it with HTML, then you must. If you're just standard labeling fieldsets, make sure you do it that way.

Any questions on forms? I'm going to jump to-- let me find it here-- a little bit more about evaluation and testing. I hope I've pretty well hammered this into all of your minds that accessibility is about people. It's about the human experience, and that that's so dynamic, so varied.

That's what makes some of this a little complex. It isn't because it's technically difficult, but because it's varied based on the human experience, alt text. It's so difficult because of the subjectivity, not because the alt attribute is hard to add to your code.

But because accessibility is about people, really, only people, only humans can truly evaluate web accessibility. Automated tools have their place. They can be very helpful as part of an accessibility process, but it's important that you understand what automated tools can, and more importantly, what they cannot do in telling you about accessibility. It's always going to require a human.

In WCAG, automated tools can detect maybe 20%, 25% of the possible failures. So that means there's another 75% to 80% of things that could possibly fail in WCAG that have to be done by a human. So that's an inherent limitation.

An automated tool can tell you if an image doesn't have alt text. It can tell you if it's obviously bad alt text, like logo. I got an email from someone that said, why is WAVE throwing an alert for alt equals logo? It's because it would probably never be appropriate, unless your company's name is logo. That is probably not going to be descriptive enough.

So a tool can tell you that, but it can't tell you if the alt text you have defined is the equivalent, if it's appropriate or not. They're just inherently limited because of that. So just keep that in mind, what tools can do, what they can't do. And just know that a human process is always going to be necessary in evaluation and testing.

Word tools tend to be most valuable is to give a quick overview of where you're at. If a tool can identify errors, it probably means there's errors. It means that there's something probably significant there. Interestingly, most of the errors that automated tools can identify tend to be some of the most impactful things, with the exception of keyboard accessibility.

Automated tools are really limited in what they can do to evaluate interaction in the page, simulating the keyboard. They just can't do that. But if something's not keyboard accessible, it's broken. It's inaccessible for a lot of users. Tools can't usually do that. But missing form labels, missing alt text, really, really crappy headings, things like that that are really impactful, tools can usually identify those for you.

So to get a sense as to where you're at, if you're remediating accessibility tools, it can be helpful to see if you've fixed the apparent things. And then it can be helpful later in a process. We've implemented accessibility. We're working on it. Tell us what's outstanding, or if somebody new has broken something.

Sometimes, we just see people that spend a lot of money, like six, seven figures, on accessibility tools, just to figure out that they have millions and millions of issues when six or seven figures could have fixed a lot of those issues. So sometimes, education is a better place to put that. And you've done that, because you're all here.

Now that I've totally poo-pooed all over evaluation tools, we are going to talk about an evaluation tool. We're going to talk about WAVE briefly here. How many of you have used WAVE before? A handful of you.

This is our tool. It was built by WebAIM. It's a freely available tool. It's available at wave.webaim.org. We just built it to work the way that we work. I'm going to demonstrate it. We think it might be helpful to you in the things that you are doing.

So you can go to wave.webaim.org. You can put in your website address, maybe. Here we go. I'm going to zoom this in a little bit here. Is it OK if I check your home page? I don't want to pick on anybody. Is that all right? Sametoyou.edu, right?

So I can just type in the web page address, hit the button. What WAVE does is it's going to go out, it's going to get your page. It's going to present your page and inject into that page icons and indicators to give feedback about accessibility. So it does it through this visual presentation.

Our philosophy and approach with WAVE-- there's a couple things there. One, because we know human evaluation is a critical part of accessibility, we help facilitate that human evaluation. We make it easier for you as a human to get in and have the underlying accessibility information revealed, so you can determine whether it's an issue or not. And if it is an issue, what the impact might be for the user.

The other thing that we tried to do is we try to ensure that we only indicate things that are errors that we know impact users, and that you can do something about. Some other tools have a very, very broad interpretation of error. Anything that might be some semblance of issue based on some broad and creative interpretation of the guidelines, it flags it as an error.

And what we found is that sometimes that can result in people spending a lot of time making tool errors go away, as opposed to addressing accessibility issues. We want you to focus on the things that matter most. That's our philosophy with WAVE.

So we show you the original page. The sidebar here will give you an overview of what WAVE found, in this case, zero errors. Yay, that's a good sign. Does that mean it's accessible? Does it mean it's compliant? Not necessarily.

Again, WAVE can only check 20%, 25% of compliance things in WCAG 2.0. It's going to be more to that. But that's a good sign. If you see an error, it almost always indicates a use and accessibility and compliance issue. So zero errors, that's a really good sign.

So if there were any errors, they would be indicated in the page using red icons. So red is bad. Red indicates an error. Yellow icons indicate alerts. Alerts are things that we don't know are compliance or accessibility issues, but that you probably ought to look at. They tend to be things that you need to look at to determine for yourself. I'll see if I can find some alerts.

Here's one right here. I can click on it, and notice it does a little tooltip. This indicates that there's an access key on this. We didn't cover access key. It's just a way to define a shortcut key for elements within the page, so the user using the shortcut can quickly get to it.

It can be useful in some cases. It adds a little bit of overhead. There's a potential for conflicts between the access key that you've defined and, say, screen reader or browser shortcuts. That's why it's an alert. It's not an error. We know it's not wrong, but we're just alerting you to the fact.

Access key is something that does not show up visibly at all within the page. WAVE, however, has revealed that. It showed an icon, so now you can see it, and you can determine for yourself whether it's going to be potentially an issue or not. Here's another access key here. Notice it actually shows, in this case, the access key, which is the number nine.

There are actually some others here in the menus. This one here indicates it's a redundant link. It says, hey, there are two links right next to each other that go to the same place. Is that bad? No, not always. It might be-- add a little overhead if it's not appropriate to have these redundant links. So WAVE has indicated that with an alert.

Something weird happened here. I'm going to refresh. There we go. I'm logging a bug in my mind. I don't know if you saw that. It showed question marks.

Green icons indicate accessibility features. These are things that probably are good for accessibility but that you need to verify on your own. So for instance, this image-- it shows alt text when I click on it. It's an image. Alternative text is present, and it shows in text the alternate text, "Michigan Technological University square logo." So that's the alt text. You can see it in the context of the image itself to make sure if that's appropriate alternative text for that image or not.

Blue icons indicate structural and semantic elements, things like headings, heading level two. Let's see if we can find some other ones here. Lists, tables, regions, and landmarks, things like that are defined with blue elements. Purple are HTML5 and ARIA, so there's quite a bit of ARIA in here. And again, WAVE is revealing that ARIA, so you can make sure it's been implemented correctly.

And then there's also this final category of contrast errors. It has zero contrast errors. That's good. Does that mean there's no WCAG contrast failures on this page? Not necessarily. WAVE can only check text in backgrounds.

There might be images or other things that can introduce stuff. You can click on the Contrast tab. And if there were any contrast issues, you could click on those and actually see the contrast ratio, the colors that are used. Essentially, the tools that we showed before, you can see those right in the context of the page.

You can go to the details panel here. It will show you all of the icons within the page. You can click on any of those to jump directly to them within the page. There's this No Styles option at the top. If you enable it, it will turn off the styles of the page. This can be helpful to allow you to see the underlying content, as well as the reading and navigation order of the page.

So if you turn off styles, you can see the order in which the keyboard user will navigate it, and the screen reader will read it. That can be helpful to ensure that it's logical and intuitive. Also, other hidden elements, for instance, our skip links that are present in this page are now visible now that I've turned off styles.

Again, I can click on any of these elements. It will jump me directly to it. And again, here's our regions, our header, navigation elements, search, main content. You can jump right to these things from the sidebar.

For any element within the page, you can click More Information or on the little i icon in the sidebar. It will give you documentation, so you understand what that icon is, what it means, what you need to do to fix it, if necessary, what WAVE used to detect in any relevant WCAG 2.0 or section 5 weighed guidelines.

We have an outline panel. This will show you the HTML-- basically, the headings for the document. You can look at these to make sure that they're appropriate, things that are identified as headings are truly headings and see if there's a proper structure to those. There's a code panel here at the bottom. That will show you the underlying code for any individual element. You can click on it to see that code within the context of the page.

So that's a quick overview of WAVE. We really built WAVE to be educational, to help inform, teach about accessibility along the way. So it might be helpful to you.

We also have WAVE Chrome and Firefox extensions. So with the Firefox and Chrome extensions, you install it directly within your browser. This is Chrome. I have WAVE installed. It has this little WAVE icon at the top right. I can click on the icon. It will perform that analysis directly within the page and shows you, basically, the same thing.

The advantage of using the Chrome extension is it's faster. It also does the analysis directly on the rendered page within your browser as opposed to on our server, so it can be a little more accurate. Also, it can be useful for evaluating dynamic pages, things that use scripting to manipulate content, things like that.

You can manipulate the page into the state that you want. Hit the icon, and it will evaluate it at that state. Also, password protected pages, intranet pages, local pages on your computer can be evaluated using the extensions.

One thing of note here-- I'm going to zoom this in just a little bit. Notice that in the extension, this actually found four contrast errors. That's probably a result, maybe, of scripting. Some of you would want to go in and probably explore these a little bit more to figure out what it is that's causing these. So it looks like these blue links over here-- I'm thinking these were maybe pulled in using an I-frame.

RSS.

RSS? So yes. So probably some scripting that pulled that content in. So here, you can see that it's flagged that blue text on the white background as being just a little bit below the 4.5 to 1, again, another advantage of using the extensions.

If this is the worst of your accessibility issues, you guys are in a great spot. A slight contrast failure there, that's good. Thank you. This is actually a really good site to demonstrate WAVE. There's a lot of really good stuff here so good work.

One other thing. You mentioned how you can use a tool, but then you also really need to use a human to test. So we did keyboard testing on our home page, and really realized that the border of our code is geared towards the mobile device. And the mobile device, the order of the header shows a little bit differently than on desktop.

As a result, the tabbing on desktop is a little bit weird. And so we just actually had an outside consultant re-code our whole header for us, so that visually it looks the same but the tabbing is more user friendly for a user. And we'll be releasing that in the next, I think, week or two.

Awesome. Those are the types of things we're-- that type of user testing is so important to identify that. Very good.

We also learned from the consultant that we overuse some of the ARIA tagging. I think you noticed it was 50 to 70 [INAUDIBLE] quite a bit too much. So rather than programming for the tool, they educated us more on when the user [INAUDIBLE], so we're going to be making some changes to that in the next week or two as well.

Cool. Awesome. Very good. I always love to hear people using less ARIA, because typically, ARIA is overused, abused, and misused in a way that it actually decreases accessibility, sometimes more often than it increases accessibility. So that's good. That's always a flag for me.

When I see hundreds of ARIA things on a page, that's usually an indication of poor accessibility. It means that they have not followed rule number one of ARIA of using good semantic HTML first. They've used ARIA as the solution, and usually, it doesn't work very well.

So that's great. I commend you for your efforts. This looks really, really good. Now, it's a very cursory view using WAVE. User testing can get in and identify some of those additional types of issues.

We do also have a new enterprise version, essentially, of WAVE. It's called Dinolytics. Some people want lots of data. This is one page at a time. If you want to collect, basically, WAVE data across an entire website, we have some options available for that.

Dinolytics is one of those. It's a subscription service where you can basically spider and analyze your entire website and collect data, assign users to particular things. So I'm not trying to sell you anything. It is a subscription service that might be useful to you. So that's a little bit about WAVE. Any questions or comments on that?

Another helpful thing that's part of your evaluation process is to use a checklist. It just helps ensure that you don't miss anything. That you don't overlook things that might-- maybe can't be identified via an automated tool that maybe might be overlooked, even via some of your user testing.

So we have on the WebAIM site-- I mentioned the WCAG 2.0 Checklist, soon to be updated to WCAG 2.1. It's just, again, six pages, checklist of the big, important things to look at in WCAG 2.0. A common one that might be overlooked is document language.

You're testing an English page and an English screen reader. If the document language is misidentified, you probably wouldn't know that, because you don't have your screen reader set to read that language. Using the checklist, you can say, check the document language, view source, the document language isn't identified or is identified incorrectly. That can be really helpful.

Keyboard accessibility-- a real big one. You just highlighted that, Joel. Just the idea that you get in with a user and start testing with a keyboard, that sometimes, you can find these areas of inefficiency or mismatch between the visual presentation and the navigation order. Really easy to do, just take your mouse, put it away, use the keyboard to interact with the page, look for focus indicators, look for logical navigation order, look for efficiencies, or other places where structural elements, headings, regions, maybe a skip link could facilitate that. And there's all of the functionality available to keyboard users that are available to mouse users.

You can also do basic screen reader testing. Screen readers can be a little bit daunting. We do have on the WebAIM site basic tutorials for getting started with JAWS, NVDA, and VoiceOver. JAWS is the most common Windows screen reader. It's rather expensive. [INAUDIBLE], I don't know. Do you have access to JAWS here?

We have a license. It's over $1,000.

It's very expensive for JAWS. Sometimes, at higher institutions, they have bulk licensing that you can maybe tie into. It is rather expensive. NVDA is free in Open Source for Windows, generally, what we would recommend for testing for most people on Windows. VoiceOver comes on the Mac. So if you have Mac or an IOS device, it has VoiceOver built into it. It can be used for basic testing.

Really, what we generally do for basic testing with a screen reader-- I'm just going to come to your website, and I'm going to fire up VoiceOver. Usually, what I do is I come to the page in the screen reader, and I just let it read. I just listen to it. I listen for things like alternative text for reading order, things that just don't make sense. And so I'm going to go ahead and do that and let you listen for just a minute.

[COMPUTER READING TEXT] [INAUDIBLE] You have one new message. Page six or seven. You are now connected to Hangouts. You are now connected to Hangouts. Michigan Technological University web content.

Here we go. It's reading my notifications from Gmail, I think.

Michigan Technological University, net contents, skip, navigation, link, skip to page contents, link, skip to site, navigation, link, skip to footer, search using tech website. Search Michigan Tech website. Edit text Michigan Tech, submit button. Search the directory, search the directory, edit text, submit button, visit link, Michigan Technological University square logo. Michigan Technological University text logo. Link, navigation. List six items. Link, undergraduate admissions. Link, graduate admissions. Link, [INAUDIBLE]. Link.

You get a sense there as to-- just letting it read through. Notice, it was reading a few things that I couldn't really see. I'm not quite sure what was going on there. It might be something to look into. But just let it read through the page. Try to identify if there are issues. Then you would want to go back and start to navigate within the page, hitting the Tab key--

[COMPUTER READING TEXT]

I'm hitting Tab.

[COMPUTER READING TEXT]

So you can see as you're navigating through. In this case, they're linked images. It's reading the alternative text. Does that align? Is it working? The Control key, that's the most important key in screen readers. It's the be quiet key. So just navigate the page. And then what you can do is you want to go back and start to explore structural elements within the page, other ways that the user might be able to navigate.

On Mac, you do that using what's called the rotor. So I'm going to open up the rotor. It allows you to navigate various types of elements. Drag this over. I'm dealing with two monitors here. Here we go. So here's all of the links on the page. You can see those links in isolation. If you have an ambiguous link test, it would become pretty obvious there if you're not quite sure what those links do.

And then hit the arrow keys. Here are all of the headings on the page. Notice that the heading levels are also identified, so I can filter down to, say, second level headings just by hitting the 2 key. Now, only the second level headings. Let's say that I'm looking for news. I can start to type news.

So I filtered down to the news heading. I can now hit Enter, and I jumped right to that heading. I can continue reading or navigating from that place within the page. You can start to see here how the structure of those headings search become pretty important as you explore just the heading levels. All of the form controls on the page, landmarks or regions that I talk about, Header, Nav, Main, things like that. You can see all of those landmarks and whether they're useful. All of the buttons.

Images, so this will show the alternative text for just those images. And you're viewing these in isolation of the images, but you can see if they make sense. If you saw something that was blank or like a file name or something like that, that would probably indicate an issue.

Window slots, that's something that VoiceOver does to guess at what it thinks might be the main content area. If you happen to find any headings or any regions, VoiceOver guesses as to what might be the main content area, which might be helpful. And back to links.

So those are some of the structural things you can navigate, just to get a sense as to whether that-- whether those things have been defined within the page. So pretty basic stuff. You can get some pretty good feedback on accessibility with pretty basic screen reader testing. Again, we have articles on the WebAIM site to help get you started with those primary screen readers. As I mentioned before, don't worry about pronunciation, how it might read particular things.

And then user testing. We don't recommend doing accessibility testing with users with disabilities. And you might be saying, why would I not do accessibility testing with those users? Isn't that the primary audience? What we recommend instead is user testing and include users with disabilities, a little kind of a nuance difference there.

User testing focuses on the broad experience. It focuses on processes and tasks. Accessibility testing, usually we look for instances of inaccessibility. And so to bring somebody in to say, is this alt text appropriate? They may not understand the broader context.

But if you have them, say, go to this page, select this product, put it in your cart, check out, that takes them through an entire process. You can identify instances of inaccessibility along the way, but you're ensuring that the broader user experience is going to be accessible to them, and friendly and efficient.

So that's a little philosophical difference but focused on the user experience-- with users, generally and include people with disabilities to help inform that, with the understanding that sometimes users with disabilities can't tell you that something's inaccessible, because it's inaccessible. Or they may not know what they're actually missing, because they didn't get it. So just be mindful of that.

Also, people with disabilities are just like everybody else. They have their own levels of expertise, their own personal opinions so just treat it that way. If a screen reader user says it should be this way, that doesn't always mean that they're right. And it might be just their personal preference.

That's why we have guidelines. We have best practices to help inform that. But a good user testing can be really insightful as to how efficient it actually is, how accessible it is to your actual end users.

So this is just a few things to consider with accessibility testing. We do have on the WebAIM site-- we have a whole bunch of articles, everything that I've covered today, and lots, lots more are available on the WebAIM site. We have articles on all sorts of things. We didn't get to document accessibility. We have very extensive documentation on Word, PowerPoint, PDF.

We also just launched an online document accessibility course, so four week, online course if you're wanting more intensive, self-paced, four week, online course on document, Word, PowerPoint, and PDF, accessibility. Also, I want to highlight just two other things under resources. We have a lot of useful resources.

We have our WCAG checklist. We have a quick reference for testing web accessibility that goes through a lot of the things I just covered. These are printable in a one page PDF, just basic things to consider for testing. We have one on principles of accessibility. We have an infographic for web accessibility for designers, some considerations there, and just a whole bunch more there.

And then finally, we have a fairly active community around accessibility. We have an active email discussion list, people all over the world. So if you have a question about accessibility, it can be a great place to go to look. We have online archives as well. You can search since 1999. We have tens of thousands of conversations in our archives about accessibility.

We also have a monthly newsletter. So if you want to just keep up to speed on what's happening in accessibility, you might consider our monthly newsletter, subscribing there.

Thank you all. Are there any questions? Brains are full. They're mush. I told you it would be a little bit of a wild ride, a lot of information. Hopefully, it's giving you enough that you can go back and start to build really cool accessibility stuff. But thanks for having me.

[APPLAUSE]