Assignment 4: Accessible HTML

Due: February 12 @ 3:30p.

For this activity, we will develop an accessible web document. In doing so, we will practice several aspects of creating accessible documents.

TL;DR: Create an HTML web page that features a "blog post" written by you - this could be a story, a review, or instructions on how to do something. This page should have a structured post with several headings, an image with a description, an accessible table, a form, and a pop-up message (with ARIA tags) for when the form is submitted. You should verify that your HTML page is accessible via the [WAVE accessibility evaluation tool](http://wave.webaim.org).

For most of this activity, you can consult the [TeachAccess tutorial] (https://teachaccess.github.io/tutorial).

# Detailed instructions

## 1. Create a new directory for Assignment 4

Don't overwrite your old files from previous assignments.

## 2. Write a blog post

Think about a blog post you would want to write. Each person should write their own. This doesn't need to be anything too elaborate - it should have a few paragraphs. It could be telling a story, sharing a recipe, reviewing a product, teaching someone how to do something, or whatever else. Your post will include, among other things, a header with subsections, an image, a table, and a form. It would be wise to spend 5 minutes or so thinking through an idea for this before you start working on the rest of the project.

## 3. Construct your HTML document.

Your page should have the following items. You are certainly free to include extra stuff, but you don't need to.

* A site menu for your fake blog, using the appropriate ARIA landmark tags. This could include the kinds of links you would expect to see on a blog: home, archive, about this blog, etc. Or add another links if you prefer. You don't actually need to create the other pages; instead, you can have the links point back to the URL of the page you are creating. For this, you should use ARIA roles to label both the main content and the navigation menu (but you don't need to label anything else).
* A top level heading, and several sub-headings for your blog post. You should use header tags for the headings, and paragraph tags ```<p></p>``` for paragraphs of text.
* An image. You can use your own image, or link to another image online. If you did not create the original image yourself (i.e., your own photo or artwork), you should indicate somewhere on the page where the image is from, and link back to the source. Your image should include an accessible description. Your image should have alternative text.
* A form with appropriate labels. How to do this is mostly described in the "Forms" section of the TeachAccess tutorial. One simple idea for the form would be to submit comments: this form might ask for the commenter's name, email address, and comment text, with a button to post the comment. You don't actually need to post the comment to the web site - for that to work, you would need to write application code that runs on the server and uploads the comment to the database. We won't do that for this assignment. Instead, we will have an alert appear on the screen when the user presses a button on the form. Here you should place your form inside an HTML ```<div>``` tag, not a ```<form>``` tag. The reason for this is that any time a button is clicked inside a form tag, the browser reloads the current page. There are other ways to fix this, but using a div is the easiest solution for now.

It's not required, but you might also want to add a table to your page. Learning how to create properly labeled tables in your HTML pages is very useful.

Note that our goal here is to create an accessible document. This means that we should follow the best practices we have learned to create a document that has the correct semantic representation.

You may want to use CSS to style how your blog looks. This assignment is a great time to explore how to do cool stuff with CSS.

## 4. Add an ARIA alert

This last step is a little bit tricky. If you get stuck here, you might want to skip ahead and do the rest of the assignment, and come back to this at the end.

For this step, you will create an "alert" message that pops up when the user presses a button on the form. You can pick what the message is. It might give the user feedback such as saying "Comment uploaded", or it could present an error message.

For this, we will use the ARIA alert role. ARIA is the web accessibility standard that covers, among other things, how to describe changes that happen to a page after it has been loaded.

Making this ARIA alert function requires three steps.

First, we must create our alert in our HTML code. The most straightforward way to do this is to use a ```<div>``` tag that contains the error message. This HTML tag needs an ARIA label that indicates that that part of the web page represents an alert; the ```div``` tag should have an attribute ```role="alert"```. An example of how this works is shown on [this Mozilla page](https://developer.mozilla.org/en-US/docs/Web/Accessibility/ARIA/ARIA\_Techniques/Using\_the\_alert\_role): our approach is very similar to Example 4 on that page.

Note: We will test our ARIA alerts in a screen reader during the next assignment. For now, follow the instructions via TeachAccess - it's OK if your code does not work on all browsers currently.

Second, you should use a CSS class to make this alert look different than the rest of the document, such as making the text color red. We did this in our previous assignments, so you should be able to build off of what you previously have done to make your alert look like it's an alert. If you want to experiment more with CSS, [this Mozilla CSS tutorial] (https://developer.mozilla.org/en-US/docs/Web/CSS) is a good place to start. Because we want the alert to be hidden when the page loads, we should add the CSS attribute ```display: none``` to our alert. Previously we hid tabs using JavaScript when the page loaded, but here we will use CSS to hide the alert so that it is never shown until the button is pressed.

Third, we will use JavaScript to show the alert when the user presses the button. This sounds scary, but this is how we built our tabs in the previous assignments. Our goal here is to only show the alert when the user needs to see it. In this case, that is when they press a button on the form. The cool thing here is that, because we have used ARIA to indicate that our alert contains important information, a visually impaired user's screen reader will read out the alert text when the alert is displayed. As in Assignments 1 and 2, we will import the JQuery library into our HTML page, and use the JQuery ```show()``` function show the alert when the button is clicked.

## 5.Verify your page's accessibility

Upload your blog post to Github Pages and check its accessibility using [WAVE](http://wave.webaim.org). You may need to go through this step several times to solve accessibility errors.

Note that WAVE will probably tell you that your document's language is not defined. The way to fix that is to add the attribute ```lang``` with value "en" (for English) to your ```html``` tag, like so: ```<html lang="en">```

# What to turn in

The link to your blog post. When entered in the browser, this link should take the user directly to your blog post.