**Applied Accessibility**

Use Headings to Show Hierarchical Relationships of Content

Headings (h1 through h6 elements) are workhorse tags that help provide structure and labeling to your content. Screen readers can be set to read only the headings on a page so the user gets a summary. This means it is important for the heading tags in your markup to have semantic meaning and relate to each other, not be picked merely for their size values.

Semantic meaning means that the tag you use around content indicates the type of information it contains.

If you were writing a paper with an introduction, a body, and a conclusion, it wouldn't make much sense to put the conclusion as a subsection of the body in your outline. It should be its own section. Similarly, the heading tags in a webpage need to go in order and indicate the hierarchical relationships of your content.

Headings with equal (or higher) rank start new implied sections, headings with lower rank start subsections of the previous one.

As an example, a page with an h2 element followed by several subsections labeled with h4 tags would confuse a screen reader user. With six choices, it's tempting to use a tag because it looks better in a browser, but you can use CSS to edit the relative sizing.

One final point, each page should always have one (and only one) h1 element, which is the main subject of your content. This and the other headings are used in part by search engines to understand the topic of the page.

Jump Straight to the Content Using the main Element

HTML5 introduced several new elements that give developers more options while also incorporating accessibility features. These tags include main, header, footer, nav, article, and section, among others.

By default, a browser renders these elements similar to the humble div. However, using them where appropriate gives additional meaning to your markup. The tag name alone can indicate the type of information it contains, which adds semantic meaning to that content. Assistive technologies can access this information to provide better page summary or navigation options to their users.

The main element is used to wrap (you guessed it) the main content, and there should be only one per page. It's meant to surround the information related to your page's central topic. It's not meant to include items that repeat across pages, like navigation links or banners.

The main tag also has an embedded landmark feature that assistive technology can use to navigate to the main content quickly. If you've ever seen a "Jump to Main Content" link at the top of a page, using the main tag automatically gives assistive devices that functionality.

Wrap Content in the article Element

article is another one of the new HTML5 elements that add semantic meaning to your markup. article is a sectioning element and is used to wrap independent, self-contained content. The tag works well with blog entries, forum posts, or news articles.

Determining whether content can stand alone is usually a judgment call, but you can use a couple of simple tests. Ask yourself if you removed all surrounding context, would that content still make sense? Similarly, for text, would the content hold up if it were in an RSS feed?

Remember that folks using assistive technologies rely on organized, semantically meaningful markup to better understand your work.

Note: The section element is also new with HTML5, and has a slightly different semantic meaning than article. An article is for standalone content, and a section is for grouping thematically related content. They can be used within each other, as needed. For example, if a book is the article, then each chapter is a section. When there's no relationship between groups of content, then use a div.

<div> - groups content <section> - groups related content <article> - groups independent, self-contained content

Make Screen Reader Navigation Easier with the nav Landmark

The nav element is another HTML5 item with the embedded landmark feature for easy screen reader navigation.

This tag is meant to wrap around the main navigation links in your page.

If there are repeated site links at the bottom of the page, it isn't necessary to markup those with a nav tag as well. Using a footer (covered in the next challenge) is sufficient.

Make Screen Reader Navigation Easier with the footer Landmark

Similar to header and nav, the footer element has a built-in landmark feature that allows assistive devices to quickly navigate to it.

It's primarily used to contain copyright information or links to related documents that usually sit at the bottom of a page.

Improve Accessibility of Audio Content with the audio Element

HTML5's audio element gives semantic meaning when it wraps sound or audio stream content in your markup. Audio content also needs a text alternative to be accessible to people who are deaf or hard of hearing.

This can be done with nearby text on the page or a link to a transcript.

The audio tag supports the controls attribute. This shows the browser default play, pause, and other controls, and supports keyboard functionality.

This is a boolean attribute, meaning it doesn't need a value, its presence on the tag turns the setting on.

Here's an example:

<audio id="meowClip" controls>

<source src="audio/meow.mp3" type="audio/mpeg">

<source src="audio/meow.ogg" type="audio/ogg">

</audio>

Note: Multimedia content usually has both visual and auditory components. It needs synchronized captions and a transcript so users with visual and/or auditory impairments can access it. Generally, a web developer is not responsible for creating the captions or transcript, but needs to know to include them.

Improve Chart Accessibility with the figure Element

HTML5 introduced the figure element and the related figcaption. Used together, these items wrap a visual representation (like an image, diagram, or chart) along with its caption.

Wrapping these elements together gives a two-fold accessibility boost by semantically grouping related content and providing a text alternative explaining the figure.

For data visualizations like charts, the caption can be used to briefly note the trends or conclusions for users with visual impairments. Another challenge covers how to move a table version of the chart's data off-screen (using CSS) for screen reader users.

Here's an example - note that the figcaption goes inside the figure tags and can be combined with other elements:

<figure>

<img src="roundhouseDestruction.jpeg" alt="Photo of Camper Cat executing a roundhouse kick">

<br>

<figcaption>

Master Camper Cat demonstrates proper form of a roundhouse kick.

</figcaption>

</figure>

Improve Form Field Accessibility with the label Element

Improving accessibility with semantic HTML markup applies to using both appropriate tag names and attributes. The next several challenges cover some important scenarios using attributes in forms.

The label tag wraps the text for a specific form control item, usually the name or label for a choice. This ties meaning to the item and makes the form more readable.

The for attribute on a label tag explicitly associates that label with the form control and is used by screen readers.

You learned about radio buttons and their labels in a lesson in the Basic HTML section. In that lesson, we wrapped the radio button input element inside a label element along with the label text in order to make the text clickable. Another way to achieve this is by using the for attribute, as explained in this lesson.

The value of the for attribute must be the same as the value of the id attribute of the form control. Here's an example:

<form>

<label for="name">Name:</label>

<input type="text" id="name" name="name">

</form>

Wrap Radio Buttons in a fieldset Element for Better Accessibility

The next form topic covers the accessibility of radio buttons. Each choice is given a label with a for attribute tying to the id of the corresponding item as covered in the last challenge.

Since radio buttons often come in a group where the user must choose one, there's a way to semantically show the choices are part of a set.

The fieldset tag surrounds the entire grouping of radio buttons to achieve this. It often uses a legend tag to provide a description for the grouping, which screen readers read for each choice in the fieldset element.

The fieldset wrapper and legend tag are not necessary when the choices are self-explanatory, like a gender selection. Using a label with the for attribute for each radio button is sufficient.

Here's an example:

<form>

<fieldset>

<legend>Choose one of these three items:</legend>

<input id="one" type="radio" name="items" value="one">

<label for="one">Choice One</label><br>

<input id="two" type="radio" name="items" value="two">

<label for="two">Choice Two</label><br>

<input id="three" type="radio" name="items" value="three">

<label for="three">Choice Three</label>

</fieldset>

</form>

Add an Accessible Date Picker

Forms often include the input field, which can be used to create several different form controls. The type attribute on this element indicates what kind of input element will be created.

You may have noticed the text and submit input types in prior challenges, and HTML5 introduced an option to specify a date field.

Depending on browser support, a date picker shows up in the input field when it's in focus, which makes filling in a form easier for all users.

For older browsers, the type will default to text, so it helps to show users the expected date format in the label or placeholder text just in case.

Here's an example:

<label for="input1">Enter a date:</label>

<input type="date" id="input1" name="input1">

Standardize Times with the HTML5 datetime Attribute

Continuing with the date theme, HTML5 also introduced the time element along with a datetime attribute to standardize times. The time element is an inline element that can wrap a date or time on a page.

A datetime attribute holds a valid format of that date. This is the value accessed by assistive devices. It helps avoid confusion by stating a standardized version of a time, even if it's informally or colloquially written in the text.

Here's an example:

<p>Master Camper Cat officiated the cage match between Goro and Scorpion <time datetime="2013-02-13">last Wednesday</time>, which ended in a draw.</p>

Make Elements Only Visible to a Screen Reader by Using Custom CSS

Have you noticed that all of the applied accessibility challenges so far haven't used any CSS? This shows the importance of using a logical document outline and semantically meaningful tags around your content before introducing the visual design aspect.

However, CSS's magic can also improve accessibility on your page when you want to visually hide content meant only for screen readers. This happens when information is in a visual format (like a chart), but screen reader users need an alternative presentation (like a table) to access the data. CSS is used to position the screen reader-only elements off the visual area of the browser window.

Here's an example of the CSS rules that accomplish this:

.sr-only {

position: absolute;

left: -10000px;

width: 1px;

height: 1px;

top: auto;

overflow: hidden;

}

Note: The following CSS approaches will NOT do the same thing:

display: none; or visibility: hidden; hides content for everyone, including screen reader users

Zero values for pixel sizes, such as width: 0px; height: 0px; removes that element from the flow of your document, meaning screen readers will ignore it

Improve Readability with High Contrast Text

Low contrast between the foreground and background colors can make text difficult to read. Sufficient contrast improves your content's readability, but what exactly does "sufficient" mean?

The Web Content Accessibility Guidelines (WCAG) recommend at least a 4.5 to 1 contrast ratio for normal text.

The ratio is calculated by comparing the relative luminance values of two colors. This ranges from 1:1 for the same color, or no contrast, to 21:1 for white against black, the most substantial contrast. There are many contrast checking tools available online that calculate this ratio for you.

Avoid Colorblindness Issues by Using Sufficient Contrast

Color is a large part of visual design, but its use introduces two accessibility issues. First, color alone should not be used as the only way to convey important information because screen reader users won't see it. Second, foreground and background colors need sufficient contrast so colorblind users can distinguish them.

Previous challenges covered having text alternatives to address the first issue. The last challenge introduced contrast checking tools to help with the second.

The WCAG-recommended contrast ratio of 4.5:1 applies for color use as well as gray-scale combinations.

Colorblind users have trouble distinguishing some colors from others - usually in hue but sometimes lightness as well.

You may recall the contrast ratio is calculated using the relative luminance (or lightness) values of the foreground and background colors.

In practice, the 4.5:1 contrast ratio can be reached by shading (adding black to) the darker color and tinting (adding white to) the lighter color. Darker shades on the color wheel are considered to be shades of blues, violets, magentas, and reds, whereas lighter tinted colors are oranges, yellows, greens, and blue-greens.

Avoid Colorblindness Issues by Carefully Choosing Colors that Convey Information

There are various forms of colorblindness. These can range from a reduced sensitivity to a certain wavelength of light to the inability to see color at all.

The most common form is a reduced sensitivity to detect greens.

For example, if two similar green colors are the foreground and background color of your content, a colorblind user may not be able to distinguish them.

Close colors can be thought of as neighbors on the color wheel, and those combinations should be avoided when conveying important information.

Note: Some online color picking tools include visual simulations of how colors appear for different types of colorblindness. These are great resources in addition to online contrast checking calculators.

Give Links Meaning by Using Descriptive Link Text

Screen reader users have various options for what type of content their device reads. These options include skipping to (or over) landmark elements, jumping to the main content, or getting a page summary from the headings.

Another option is to only hear the links available on a page.

Screen readers do this by reading the link text, or what's between the anchor (a) tags. Having a list of "click here" or "read more" links isn't helpful.

Instead, use brief but descriptive text within the a tags to provide more meaning for these users.

Make Links Navigable with HTML Access Keys

HTML offers the accesskey attribute to specify a shortcut key to activate or bring focus to an element. Adding an accesskey attribute can make navigation more efficient for keyboard-only users.

HTML5 allows this attribute to be used on any element, but it's particularly useful when it's used with interactive ones. This includes links, buttons, and form controls.

Here's an example:

<button accesskey="b">Important Button</button>

Use tabindex to Add Keyboard Focus to an Element

The HTML tabindex attribute has three distinct functions relating to an element's keyboard focus. When it's on a tag, it indicates that the element can be focused on. The value (an integer that's positive, negative, or zero) determines the behavior.

Certain elements, such as links and form controls, automatically receive keyboard focus when a user tabs through a page. It's in the same order as the elements come in the HTML source markup. This same functionality can be given to other elements, such as div, span, and p, by placing a tabindex="0" attribute on them. Here's an example:

<div tabindex="0">I need keyboard focus!</div>

Note: A negative tabindex value (typically -1) indicates that an element is focusable, but is not reachable by the keyboard. This method is generally used to bring focus to content programmatically (like when a div used for a pop-up window is activated), and is beyond the scope of these challenges.

Camper Cat created a new survey to collect information about his users.

He knows input fields automatically get keyboard focus, but he wants to make sure his keyboard users pause at the instructions while tabbing through the items.

Add a tabindex attribute to the p tag and set its value to 0. Bonus - using tabindex also enables the CSS pseudo-class :focus to work on the p tag.

Practical Example:

<style>

  p:focus {

    background-color: yellow;

  }

  </style>

</head>

<body>

  <section>

    <form>

<p tabindex="0">Instructions: Fill in ALL your information then click <b>Submit</b></p>

(the focused “p” will become yellow”)

Use tabindex to Specify the Order of Keyboard Focus for Several Elements

The tabindex attribute also specifies the exact tab order of elements. This is achieved when the attribute's value is set to a positive number of 1 or higher.

Setting a tabindex="1" will bring keyboard focus to that element first. Then it cycles through the sequence of specified tabindex values (2, 3, etc.), before moving to default and tabindex="0" items.

It's important to note that when the tab order is set this way, it overrides the default order (which uses the HTML source).

This may confuse users who are expecting to start navigation from the top of the page. This technique may be necessary in some circumstances, but in terms of accessibility, take care before applying it.

Here's an example:

<div tabindex="1">I get keyboard focus, and I get it first!</div>

<div tabindex="2">I get keyboard focus, and I get it second!</div>