# LEARN ACCESSIBILITY BY BUILDING A QUIZ

## Introduction:

Accessibility is making your webpage easy for all people to use – even people with disabilities.

In this course, you'll build a quiz webpage. You'll learn accessibility tools such as keyboard shortcuts, ARIA attributes, and design best practices.

## Step 1:

Welcome to the first part of the Accessibility Quiz. As you are becoming a seasoned HTML and CSS developer, we have started you off with the basic boilerplate.

Start this accessibility journey by providing a `lang` attribute to your `html` element. This will assist screen readers in identifying the language of the page.

## Step 2:

You may be familiar with the `meta` element already; it is used to specify information about the page, such as the title, description, keywords, and author.

Give your page a `meta` element with an appropriate `charset` value.

The `charset` attribute specifies the character encoding of the page, and, nowadays, `UTF-8` is the only encoding supported by most browsers.

## Step 3:

Continuing with the meta elements, a viewport definition tells the browser how to render the page. Including one betters visual accessibility on mobile, and improves *SEO* (search engine optimization).

Add a viewport definition with a content attribute detailing the width and initial-scale of the page.

### Step 4:

Another important meta element for accessibility and SEO is the description definition. The value of the content attribute is used by search engines to provide a description of your page.

Add a meta element with the name attribute set to description, and give it a useful content attribute.

### Step 5:

Lastly, in the head, the title element is useful for screen readers to understand the content of a page. Furthermore, it is an important part of *SEO*.

Give your page a title that is descriptive and concise.

### Step 6:

Navigation is a core part of accessibility, and screen readers rely on you to provide the structure of your page. This is accomplished with semantic HTML elements.

Add a header and a main element to your page.

The header element will be used to introduce the page, as well as provide a navigation menu.

The main element will contain the core content of your page.

## Step 7:

Within the header, provide context about the page by nesting one img, h1, and nav element.

The img should point to https://cdn.freecodecamp.org/platform/universal/fcc_primary.svg, have an id of logo, and have an alt text of freeCodeCamp.

The h1 should contain the text HTML/CSS Quiz.

## Step 8:

A useful property of an *SVG* (scalable vector graphics) is that it contains a path attribute which allows the image to be scaled without affecting the resolution of the resultant image.

Currently, the img is assuming its default size, which is too large. CSS has a max function which returns the largest of a set of comma-separated values. For example:

Example Code:

```
img {

  width: max(250px, 25vw);

}
```

In the above example, the width of the image will be 250px if the viewport width is less than 1000 pixels. If the viewport width is greater than 1000 pixels, the width of the image will be 25vw. This is because 25vw is equal to 25% of the viewport width.

Scale the image using its id as a selector, and setting the width to be the maximum of 10rem or 18vw.

### Step 9:

As described in the freeCodeCamp Style Guide, the logo should retain an aspect ratio of 35 / 4, and have padding around the text.

First, change the background-color to #0a0a23 so you can see the logo. Then, use the aspect-ratio property to set the desired aspect ratio to 35 / 4. Finally, add a padding of 0.4rem all around.

### Step 10:

Make the header take up the full width of its parent container, set its height to 50px, and set the background-color to #1b1b32. Then, set the display to use *Flexbox*.

### Step 11:

Change the h1 font color to #f1be32, and set the font size to min(5vw, 1.2em).

### Step 12:

To enable navigation on the page, add an unordered list with the following three list items:

- INFO
- HTML
- CSS

The list items text should be wrapped in anchor tags.

### Step 13:

The child combinator selector `>` is used between selectors to target only elements that match the second selector and are a direct child of the first selector.

This can be helpful when you have deeply nested elements and want to control the scope of your styling.

Use the `>` selector to target the unordered list elements within the `nav` elements, and use *Flexbox* to evenly space the children.

### Step 14:

As this is a quiz, you will need a form for users to submit answers. You can semantically separate the content within the form using `section` elements.

Within the `main` element, create a form with three nested `section` elements. Then, make the form submit to `https://freecodecamp.org/practice-project/accessibility-quiz`, using the correct method.

### Step 15:

To increase the page accessibility, the `role` attribute can be used to indicate the purpose behind an element on the page to assistive technologies. The `role` attribute is a part of the *Web Accessibility Initiative* (WAI), and accepts preset values.

Give each of the `section` elements the `region` role.

### Step 16:

Every `region` role requires a label, which helps screen reader users understand the purpose of the region. One method for adding a label is to add a heading element inside the region and then reference it with the `aria-labelledby` attribute.

Add the following `aria-labelledby` attributes to the `section` elements:

- `student-info`
- `html-questions`
- `css-questions`

Then, within each `section` element, nest one `h2` element with an `id` matching the corresponding `aria-labelledby` attribute. Give each `h2` suitable text content.

### Step 17:

Typeface plays an important role in the accessibility of a page. Some fonts are easier to read than others, and this is especially true on low-resolution screens.

Change the font for both the `h1` and `h2` elements to `Verdana`, and use another web-safe font in the sans-serif family as a fallback.

Also, add a `border-bottom` of `4px solid #dfdfe2` to `h2` elements to make the sections distinct.

### Step 18:

To be able to navigate within the page, give each anchor element an `href` corresponding to the `id` of the `h2` elements.

### Step 19:

Filling out the content of the quiz, below `#student-info`, add three `div` elements with a `class` of `info`.

Then, within each `div` nest one `label` element, and one `input` element.

### Step 20:

It is important to link each `input` to the corresponding `label` element. This provides assistive technology users with a visual reference to the input.

This is done by giving the `label` a `for` attribute, which contains the `id` of the `input`.

This section will take a student's name, email address, and date of birth. Give the `label` elements appropriate `for` attributes, as well as text content. Then, link the `input` elements to the corresponding `label` elements.

### Step 21:

Keeping in mind best-practices for form inputs, give each `input` an appropriate `type` and `name` attribute. Then, give the first `input` a `placeholder` attribute.

### Step 22:

Even though you added a `placeholder` to the first `input` element in the previous lesson, this is actually not a best-practice for accessibility; too often, users confuse the placeholder text with an actual input value - they think there is already a value in the input.

Remove the placeholder text from the first `input` element, relying on the `label` being the best-practice.

### Step 23:

Within the second `section` element, add two `div` elements with a class of `question-block`.

Then, within each `div.question-block` element, add one `h3` element with text of incrementing numbers, starting at `1`, and one `fieldset` element with a class of `question`.

### Step 24:

The question numbers are not descriptive enough. This is especially true for visually impaired users. One way to get around such an issue, without having to add visible text to the element, is to add text only a screen reader can read.

Append a `span` element with a `class` of `sr-only` to each of the `h3` elements.

### Step 25:

Within the first `span` element, add the text `Question`.

In the second `span` element, add the text `Question`.

### Step 26:

The `.sr-only` text is still visible. There is a common pattern to visually hide text for only screen readers to read.

This pattern is to set the following CSS properties:

Example Code:

```
position: absolute;

width: 1px;

height: 1px;

padding: 0;
```

```
margin: -1px;

overflow: hidden;

clip: rect(0, 0, 0, 0);

white-space: nowrap;

border: 0;
```

Use the above to define the `.sr-only` CSS rule.

### Step 27:

Each `fieldset` will contain a true/false question.

Within each `fieldset`, nest one `legend` element, and one `ul` element with two options.

### Step 28:

Give each `fieldset` an adequate `name` attribute. Then, give both unordered lists a `class` of `answers-list`.

Finally, use the `legend` to caption the content of the `fieldset` by placing a true/false question as the text content.

### Step 29:

To provide the functionality of the true/false questions, we need a set of inputs which do not allow both to be selected at the same time.

Within each list element, nest one `label` element, and within each `label` element, nest one `input` element with the appropriate `type`.

### Step 30:

Add an `id` to all of your radio `inputs` so you can link your labels to them. Give the first one an `id` of `q1-a1`. Give the rest of them `ids` of `q1-a2`, `q2-a1`, and `q2-a2`, respectively.

### Step 31:

Although not required for `label` elements with a nested `input`, it is still best-practice to explicitly link a `label` with its corresponding `input` element.

Now, add a `for` attribute to each of your four `labels` that links the `label` to its corresponding radio `input`.

### Step 32:

Give the `label` elements text such that the `input` comes before the text. Then, give the `input` elements a `value` matching the text.

The text should either be `True` or `False`.

### Step 33:

If you click on the radio inputs, you might notice both inputs within the same true/false fieldset can be selected at the same time.

Group the relevant inputs together such that only one input from a pair can be selected at a time.

### Step 34:

To prevent unnecessary repetition, target the before pseudo-element of the h3 element, and give it a content property of "Question #".

### Step 35:

The final section of this quiz will contain a dropdown, and a text box.

Begin by nesting a div with a class of formrow, and nest four div elements inside of it, alternating their class attributes with question-block and answer.

### Step 36:

Within the div.question-block elements, nest one label element, and add a CSS related question to the label text.

### Step 37:

Within the first div.answer element, nest one required select element with three option elements.

Give the first option element a value of "", and the text Select an option. Give the second option element a value of yes, and the text Yes. Give the third option element a value of no, and the text No.

### Step 38:

Link the first label element to the select element, and give the select element a name attribute.

### Step 39:

Nest one `textarea` element within the second `div.answer` element, and set the number of rows and columns it has.

### Step 40:

As with the other `input` and `label` elements, link the `textarea` to its corresponding `label` element, and give it a `name` attribute.

### Step 41:

Do not forget to give your `form` a submit button with the text `Send`.

### Step 42:

Two final semantic HTML elements for this project are the `footer` and `address` elements. The `footer` element is a container for a collection of content that is related to the page, and the `address` element is a container for contact information for the author of the page.

After the `main` element, add one `footer` element, and nest one `address` element within it.

### Step 43:

Within the `address` element, add the following:

Example Code:

```
freeCodeCamp<br />

San Francisco<br />

California<br />

USA
```

The br tags will allow each part of the address to be on its own line and are useful for presenting address elements properly.

### Step 44:

The address element does not have to contain a physical geographical location. It can be used to provide a link to the subject.

Wrap a link around the text freeCodeCamp, and set its location to https://freecodecamp.org.

### Step 45:

Back to styling the page. Select the list elements within the navigation bar, and give them the following styles:

Example Code:

color: #dfdfe2;

margin: 0 0.2rem;

padding: 0.2rem;

display: block;

### Step 46:

On the topic of visual accessibility, contrast between elements is a key factor. For example, the contrast between the text and the background of a heading should be at least 4.5:1.

Change the font color of all the anchor elements within the list elements to something with a contrast ratio of at least 7:1.

### Step 47:

To make the navigation buttons look more like typical buttons, remove the underline from the anchor tags.

Then, create a new selector targeting the navigation list elements so that when the cursor hovers over them, the background color and text color are switched, and the cursor becomes a pointer.

### Step 48:

Tidy up the `header`, by using *Flexbox* to put space between the children, and vertically center them.

Then, fix the `header` to the top of the viewport.

### Step 49:

When the screen width is small, the `h1` does not wrap its text content how it should. Align the text for the `h1` element in the center.

Then, give the `main` padding such that the `Student Info` section header can be fully seen.

### Step 50:

On small screens, the unordered list in the navigation bar overflows the right side of the screen.

Fix this by using *Flexbox* to wrap the `ul` content. Then, set the following CSS properties to correctly align the text:

Example Code:

```
align-items: center;

padding-inline-start: 0;
```

```
margin-block: 0;

height: 100%;
```

### Step 51:

Set the width of the `section` elements to `80%` of their parent container. Then, use margins to center the `section` elements, adding `10px` to the bottom margin.

Also, ensure the `section` elements cannot be larger than `600px` in width.

### Step 52:

Replace the top margin of the `h2` elements with `60px` of top padding.

### Step 53:

Add padding to the top and left of the `.info` elements, and set the other values to `0`.

### Step 54:

Give the `.formrow` elements top margin, and left and right padding. The other padding values should be `0`.

Then, increase the font size for all `input` elements.

### Step 55:

To make the first section look more inline, target only the `input` elements within `.info` elements, and set their `width` to `50%`, and left-align their text.

### Step 56:

Target all `label` elements within `.info` elements, and set their `width` to `10%`, and make it so they do not take up less than `55px`.

### Step 57:

To align the input boxes with each other, create a new ruleset that targets all `input` and `label` elements within an `.info` element and set the `display` property to `inline-block`.

Also, align the `label` element's text to the right.

### Step 58:

To neaten the `.question-block` elements, set the following CSS properties:

Example Code:

```
text-align: left;

display: block;

width: 100%;

margin-top: 20px;

padding-top: 5px;
```

### Step 59:

Make the h3 elements appear as a higher priority, with the following CSS properties:

Example Code:

```
margin-top: 5px;

padding-left: 15px;

font-size: 1.375rem;
```

### Step 60:

It is useful to see the default border around the fieldset elements, during development. However, it might not be the style you want.

Remove the border and bottom padding on the .question elements.

### Step 61:

While ul/li elements are great at providing bullets for list items, your radio buttons don't need them. You can control what the bullets look with the list-style property. For example you can turn your bullets into circles with the following:

Example Code:

```
ul {

  list-style: circle;

}
```

Remove the default styling for the .answers-list items by setting its style to none, and remove the unordered list padding.

### Step 62:

Give the submit button a freeCodeCamp-style design, with the following CSS properties:

Example Code:

```
display: block;

margin: 40px auto;

width: 40%;

padding: 15px;

font-size: 1.438rem;

background: #d0d0d5;

border: 3px solid #3b3b4f;
```

### Step 63:

Set the `footer` background color to `#2a2a40`, and use *Flexbox* to horizontally center the text.

### Step 64:

Now, we cannot read the text. Target the `footer` and the anchor element within to set the font color to a color of adequate contrast ratio.

### Step 65:

Horizontally center all the text within the `address` element, and add some padding.

### Step 66:

Clicking on the navigation links should jump the viewport to the relevant section. However, this jumping can be disorienting for some users.

Select all elements, and set the `scroll-behavior` to `smooth`.

### Step 67:

Finally, certain types of motion-based animations can cause discomfort for some users. In particular, people with vestibular disorders have sensitivity to certain motion triggers.

The `@media` at-rule has a media feature called `prefers-reduced-motion` to set CSS based on the user's preferences. It can take one of the following values:

- `reduce`
- `no-preference`

Example Code:

```
@media (feature: value) {

  selector {

    styles

  }

}
```

Wrap the style rule that sets `scroll-behavior: smooth` within a `@media` at-rule with the media feature `prefers-reduced-motion` having `no-preference` set as the value.

Well done. You have completed the Accessibility Quiz practice project.