# 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.

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<meta name="description" content="freeCodeCamp Accessibility Quiz practice project" />

# 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, and have an id of logo.

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 it's default size, which is too large. Correctly, scale the image using it's id as a selector, and setting the width to max(100px, 18vw).

# Step 9

As described in the [freeCodeCamp Style Guide](https://design-style-guide.freecodecamp.org/" \t "/home/carlos/Documents\\x/_blank), 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 it's 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

Target unordered list elements within nav elements, and use Flexbox to evenly space the children.

nav > ul {

display: flex;

justify-content: space-evenly;

}

# 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.

<section role="region"></section>

# Step 16

Every region role requires a visible label, which should be referenced by the aria-labelledby attribute.

To the section elements, give the following aria-labelledby attributes:

* 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

Arguably, D.O.B. is 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 label, is to add text only a screen reader can read.

Append a span element with a class of sr-only to the current text content of the third label element.

# Step 24

Within the span element, add the text (Date of Birth).

# Step 25

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:

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 class.

# Step 26

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

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

# 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

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.

Link the label elements with their corresponding input elements.

# Step 31

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 32

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 33

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

**p::before {content: "Question #"}**

# Step 34

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 35

Within the div.question-block elements, nest one label element, and give the label elements text content

# Step 36

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 37

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

# tep 38

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

Then, give the textarea placeholder text describing an example answer.

# Step 39

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

# Step 40

Do not forget to give your form a submit button.

# Step 41

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 42

Within the address element, add the following:

freeCodeCamp<br />

San Francisco<br />

California<br />

USA

# Step 43

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 44

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

color: #dfdfe2; margin: 0 0.2rem; padding: 0.2rem; display: block;

# Step 45

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 46

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***.

nav > ul > li:hover or nav li:hover

# Step 47

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.

justify-content: space-between;

align-items: center;

position: fixed;

top: 0;

# Step 48

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.

main {

padding-top: 25px;

}

# Step 49

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:

align-items: center;

padding-inline-start: 0;

margin-block: 0;

height: 100%;

flex-wrap: wrap;

# Step 50

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.

section {

width: 80%;

max-width: 600px;

margin: 0 auto;

margin-bottom: 10px;

}

# Step 51

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

# Step 52

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

# Step 53

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 54

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 55

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.

To align the input boxes with each other, set the display property to inline-block for all input and label elements within .info elements.

Also, align the text to the right.

You should use either a .info > label, .info > input or .info label, .info input selector.

# Step 57

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

text-align: left;

display: block;

width: 100%;

margin-top: 20px;

padding-top: 5px;

# Step 58

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

margin-top: 5px;

padding-left: 15px;

font-size: 20px;

# Step 59

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 60

Remove the default styling for the list items of .answers-list, and remove the unordered list padding.

list-style: none;

padding: 0;

# Step 61

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

display: block; margin: 40px auto; width: 40%; padding: 15px; font-size: 23px; background: #d0d0d5; border: 3px solid #3b3b4f;

# Step 62

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

footer {

background-color: #2a2a40;

display: flex;

justify-content: center;

}

# Step 63

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.

footer, footer a {

color: #dfdfe2;

}

# Step 64

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

# Step 65

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 66

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

@media (feature: value) {

selector {

styles

}}

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

# Step 67

Finally, the navigation accessibility can be improved by providing keyboard shortcuts.

The accesskey attribute accepts a space-separated list of access keys. For example:

**<button type="submit" accesskey="s">Submit</button>**

Give each of the navigation links a single-letter access key.

Note: It is not always advised to use access keys, but they can be useful

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