**HEADER**

**First part of html**

**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.

<html lang="en">

  <head>

    <link rel="stylesheet" href="styles.css" />

  </head>

  <body>

  </body>

</html>

# 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.

  <head>

    <link rel="stylesheet" href="styles.css" />

    <meta charset="utf-8">

  </head>

# 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.

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

# 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="description" content="Free Web tutorials">

# 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.

<title> Sohaib Hassan Quizz</title>

# 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.

<header>

      </header>

    <main>

      </main>

# 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.

<header>

      <!-- context add here -->

      <img src="https://cdn.freecodecamp.org/platform/universal/fcc\_primary.svg" id="logo"/>

      <h1>HTML/CSS QUIZZ</h1>

      <nav></nav>

    </header>

**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:

img {

width: max(250px, 25vw);

}

In this example, img elements will have a minimum width of 250px. And as the viewport grows, the image will grow accordingly to be 25 percent of the viewport width.

Scale the image using its id as a selector, and setting the width to be the maximum of 100px or 18vw.

#logo{

  width:max(100px,18vw);

}

Here we will use a function max to resize our image

# Step 9

As described in the [freeCodeCamp Style Guide](https://design-style-guide.freecodecamp.org/), 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.

#logo {

  width: max(100px, 18vw);

  background-color:#0a0a23;

  aspect-ratio:35/4;

  padding:0.4rem;

}

Note

Here we will design a logo there are a website fot free code camp org which well have a lot of resourses in it about logo to layout even colors everything is available just need your patient and be calm try to visit freecodecamp org style website Thank you!

The link is provided here

[freeCodeCamp Style Guide](https://design-style-guide.freecodecamp.org/),

# 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.

Here we will use a Flexbox property which is very important for understanding in our case it should be a starting let strt

Be calm

header{

  height:50px;

  background-color:#1b1b32;

  display:flexbox;

}

header{

  width:100%;

  height:50px;

  background-color:#1b1b32;

  display:flex;

}

Here we are dealing with header first we design a header then move further

For header it should be a maxumun height of 50px

Bacgroundcolor black

Display for container is flexbox =display:flex;

Here we make a mistake forget width width should must be define with height it will take a full width of the screen

# Step 11

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

Now we are dealing with h1 mean inside header there is an h1 element we design it using different techniques

Min(5vw, 1.2em)

Remember

1em =10px so 12em = 12\*10 = 120px;

h1{

  color:#f1be32;

  font-size:min(5vw, 1.2em);

}

"Happy, happy, joy, joy!"

"Happy, happy, joy, joy!"

# 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.

<nav></nav>

For nav element we should include unorder list

     <nav>

<ul>

  <li><a href="#">INFO</a></li>

    <li><a href="#">HTML</a></li>

      <li><a href="#">CSS</a></li>

      </nav>

NOTE

WE USE A NAV INSDE THE HEADER

"Nice one!"

# Step 13

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

Here we will learn a new topic which is use > this symbol and are called children selector

nav > ul{

  display:flex;

}

Now this will select all the child of nav and inside ul tags which is label in our case

nav > ul{

  display:flex;

  justify-content:space-evenly;

  color:white;

}

Note flex is a container

It is row by default e also set a column felx direction

Justify content space-even is a space between element it have different property

# 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.

Here I will practice on it from start

Lets leave it for now go to another page for main element

It will first task note it will do 5 times minimum

Main Element

After header this part come

We have use section in main element mainly

Let’s start

<main>

<section>

</section>

</main>

This is the second part of html structure we add many section according to our need and demand

Let’s start

# 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.

  <main>

      <form action="https://freecodecamp.org/practice-project/accessibility-quiz" method="post"/>

<section></section>

<section></section>

<section></section>

</form>

    </main>

Okay first thing first we make a three section inside a main element with in main element we create a form

There are two things important

Method: use for post

Action: use for where to submit our form or data

# 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>

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

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

Now you can see here we add a Role=”region”

So basically it is important for web accessibility initiative(WAI)

Accessibility mean

The [quality](https://dictionary.cambridge.org/dictionary/english/quality) of being [able](https://dictionary.cambridge.org/dictionary/english/able) to be [entered](https://dictionary.cambridge.org/dictionary/english/enter) or used by everyone, [including](https://dictionary.cambridge.org/dictionary/english/include) [people](https://dictionary.cambridge.org/dictionary/english/people) who have a [disability](https://dictionary.cambridge.org/dictionary/english/disability):

**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.

 <form method="post" action="https://freecodecamp.org/practice-project/accessibility-quiz">

        <section role="region" aria-labelledby="student-info">

          <h2 id="student-info"> Student Info</h2>

        </section>

       <section role="region" aria-labelledby="html-questions">

          <h2 id="html-questions">Html Questions</h2>

        </section>

  <section role="region" aria-labelledby="css-questions">

          <h2 id="css-questions">CSS Questions</h2>

        </section>

      </form>

"Boom-shakalaka!"

# Step 18

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

 <ul>

          <li><a href="#" id="student-info">INFO</a></li>

           <li><a href="#" id="student-info">HTML</a></li>

            <li><a href="#" id="student-info">CSS</a></li>

        </ul>

<ul>

          <li><a href="#student-info" id="student-info">INFO</a></li>

          <li><a href="#html-questions">HTML</a></li>

          <li><a href="#css-questions">CSS</a></li>

        </ul>

# 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.

<section role="region" aria-labelledby="student-info">

          <h2 id="student-info">Student Info</h2>

          <div class="info">

            <label></label>

            <input>

          </div>

          <div class="info">

            <label></label>

            <input>

          </div>

          <div class="info">

            <label></label>

            <input>

          </div>

        </section>

Note

In the first Section we design a header which have three boxes inside it mean three div element and inside first section there is three label and three input element for collecting user data

Code is

<section role="region" aria-labelledby="student-info">

          <h2 id="student-info">Student Info</h2>

          <div class="info">

            <label></label>

            <input>

          </div>

          <div class="info">

            <label></label>

            <input>

          </div>

          <div class="info">

            <label></label>

            <input>

          </div>

        </section>

"Checkmate!"

Lets move to step 20 but this is the section part mean main part of the layout which comes after the header element lets make it

# 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.

Note

We actually use for attribute inside a label for element for the purposes of connection with input other wise it will be stuck

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

<section role="region" aria-labelledby="student-info">

          <h2 id="student-info">Student Info</h2>

          <div class="info">

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

            <input type="text" required placeholder="Please Enter your Name" id="name" />

          </div>

          <div class="info">

            <label for="email-address">Email</label>

            <input id="email-address" required placeholder="Please Enter your Email"/>

          </div>

          <div class="info">

            <label for="age" >Age</label>

            <input id="age" required placeholder="Age"/>

          </div>

        </section>

Now here we design the ***input attribute with label***

Here we put some ***important information or attribute*** which can help us in making form

For first section we use only a take input in the form of ***Name Email Password or Age***

# 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.

  <section role="region" aria-labelledby="student-info">

          <h2 id="student-info">Student Info</h2>

          <div class="info">

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

            <input id="student-name" type="text" name="Fullname" placeholder="Enter your Name" />

          </div>

          <div class="info">

            <label for="student-email">Email:</label>

            <input id="student-email" type="email" name="email" placeholder="Enter your Email" />

          </div>

          <div class="info">

            <label for="birth-date">D.O.B.:</label>

            <input id="birth-date" type="date" name="Date\_of\_birth" placeholder="Date of Birth"/>

          </div>

        </section>

Here simple put the type of name email and date for inputs

"Even grumpy cat approves!"

# 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.

Here we just remove a placeholder from the input field

"Even grumpy cat approves!"

# 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 elemen

It is very bad with label we append span element inside it

  <label for="birth-date"><span class="sr-only">D.O.B.</span></label>

 <label for="birth-date">D.O.B.<span class="sr-only"></span></label>

# Step 24

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

 <label for="birth-date">D.O.B.<span class="sr-only">(Date of Birth)</span></label>

Here we place a span element and inside we just write a 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

Now here we design the span Element it is important to remember that

Lets start

sr-only{

  position:absolute;

  background:red;

  width:1px;

  height:1px;

  padding:0;

  margin:-1px;

  overflow:hidden;

  clip:rect(0,0,0,0);

  white-space:nowrap;

  border:0;

}

"Standing ovation!"

# Now this is another Section Started for Question

# <section 2></section2>

# 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.

<section role="region" aria-labelledby="html-questions">

          <h2 id="html-questions">HTML</h2>

          <div class="question-block">

            <p>1</p>

            <fieldset class="question"></fieldset>

          </div>

           <div class="question-block">

             <p>2</p>

             <fieldset class="question"></fieldset>

           </div>

        </section>

Now here we make a section 2 inside a section 2 we will ask a questions and for now we just make a two div element with class question-block and nedsted two <P> element and <fieldset> with each block

Fieldset:

Concept: it will create the block mean border and well write text inside it

# Step 27

Each fieldset will contain a true/false question.

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

Inside it we use legend

 <div class="question-block">

            <p>1</p>

            <fieldset class="question">

              <legend></legend>

              <ul>

                <li></li>

                   <li></li>

              </ul>

            </fieldset>

          </div>

          <div class="question-block">

            <p>2</p>

            <fieldset class="question">

              <legend></legend>

              <ul>

                <li></li>

                   <li></li>

              </ul>

            </fieldset>

"You've got the touch!"

# 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.

   <fieldset class="question" name="adequate">

              <legend>True </legend>

              <ul class="answet-list">

                <li></li>

                <li></li>

              </ul>

            </fieldset>

          </div>

          <div class="question-block">

            <p>2</p>

            <fieldset class="question" name="adequate">

              <legend>False</legend>

              <ul class="answer-list">

                <li></li>

                <li></li>

              </ul>

            </fieldset>

Here we place true false insdie legend will show and also gives calluses to ul and fieldset

# 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.

 <fieldset class="question" name="html-question-two">

              <legend>

                A label element nesting an input element is required to have a

                for attribute with the same value as the input's id

              </legend>

              <ul class="answers-list">

                <li>

                  <label> <input type="radio"name="answer"></label>

                </li>

                <li>

                  <label> <input type="radio"name="answer"></label>

                </li>

              </ul>

Here simple we put inside li label and isndie label we put input element

"You've got the touch!"

# 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.

   <ul class="answers-list">

                <li>

                  <label for="q1-a1">

                    <input type="radio" id="q1-a1"/>

                  </label>

                </li>

                <li>

                  <label for="q1-a2">

                    <input type="radio" id="q1-a2"/>

                  </label>

                </li>

              </ul>

            </fieldset>

          </div>

          <div class="question-block">

            <p>2</p>

            <fieldset class="question" name="html-question-two">

              <legend>

                A label element nesting an input element is required to have a

                for attribute with the same value as the input's id

              </legend>

              <ul class="answers-list">

                <li>

                  <label for="q2-a1">

                    <input type="radio" id="q2-a1"/>

                  </label>

                </li>

                <li>

                  <label for="q2-a2">

                    <input type="radio" id="q2-a2" />

                  </label>

                </li>

              </ul>

Here we give id = q1-a1 q1-a2 for first fieldset and for second q2-a1 q2-a2

# 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.

   <ul class="answers-list">

                <li>

                  <label for="q1-a1">

                    <input type="radio" id="q1-a1" value="True"/>

                    True

                  </label>

                </li>

                <li>

                  <label for="q1-a2">

                    <input type="radio" id="q1-a2" value="False" />

                    False

                  </label>

                </li>

              </ul>

            </fieldset>

          </div>

          <div class="question-block">

            <p>2</p>

            <fieldset class="question" name="html-question-two">

              <legend>

                A label element nesting an input element is required to have a

                for attribute with the same value as the input's id

              </legend>

              <ul class="answers-list">

                <li>

                  <label for="q2-a1">

                    <input type="radio" id="q2-a1" value="True" />

                    True

                  </label>

                </li>

                <li>

                  <label for="q2-a2">

                    <input type="radio" id="q2-a2" value="False"/>

                    False

                  </label>

                </li>

              </ul>

Here we just give true false after input tag

# 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.

        <ul class="answers-list">

                <li>

                  <label for="q1-a1">

                    <input type="radio" id="q1-a1" value="true" name="question"/>

                    True

                  </label>

                </li>

                <li>

                  <label for="q1-a2">

                    <input type="radio" id="q1-a2" value="false" name="question"/>

                    False

                  </label>

                </li>

              </ul>

            </fieldset>

          </div>

          <div class="question-block">

            <p>2</p>

            <fieldset class="question" name="html-question-two">

              <legend>

                A label element nesting an input element is required to have a

                for attribute with the same value as the input's id

              </legend>

              <ul class="answers-list">

                <li>

                  <label for="q2-a1">

                    <input type="radio" id="q2-a1" value="true" name="questions"/>

                    True

                  </label>

                </li>

                <li>

                  <label for="q2-a2">

                    <input type="radio" id="q2-a2" value="false" name="questions"/>

                    False

                  </label>

                </li>

              </ul>

Here we just place a name attribute which will select only one radio button at a time

# Step 34

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 #";

}

This CSS rule uses the ::before pseudo-element to insert content before the p element's content. The content property is used to specify the content to be inserted, which in this case is "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.

Now we Design the Final Section of our main element

<section role="region" aria-labelledby="css-questions">

          <h2 id="css-questions">CSS</h2>

          <div class="formrow">

            <div class="question-block answer"></div>

             <div class="question-block answer"></div>

              <div class="question-block answer"></div>

               <div class="question-block answer"></div>

          </div>

        </section>

  <section role="region" aria-labelledby="css-questions">

          <h2 id="css-questions">CSS</h2>

          <div class="formrow">

  <div class="question-block"></div>

  <div class="answer"></div>

  <div class="question-block"></div>

  <div class="answer"></div>

</div>

        </section>

Here one will be question-block and other div will be the answer

"Pikachu chooses you!"

# Step 36

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

 <div class="question-block">

              <label>This is CSS</label>

            </div>

            <div class="answer">

            </div>

            <div class="question-block">

              <label>Background Color </label>

            </div>

            <div class="answer">

Here we just give text to label question block element

# 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.

        <div class="answer">

    <select required>

      <option value="">Select an option</option>

      <option value="yes">Yes</option>

      <option value="no">No</option>

    </select>

  </div>

Here we will place a select and option element inside it for Dropdown we must write in one line

Coding spree!"

# Step 38

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

    </div>

            <div class="answer">

              <select id="optionSelect" name="option" required>

                <option value="">Select an option</option>

                <option value="yes">Yes</option>

                <option value="no">No</option>

              </select>

Here we will give name and id attribute to select and for to label same as id

"Gotta code 'em all!"

# Step 39

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.

 <div class="answer">

              <textarea rows="4" cols="50" placeholder=".....Description here"></textarea>

            </div>

"Cranked it up to 11!"

# Step 40

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

<div class="question-block">

              <label for="description">Do you have any questions:</label>

            </div>

            <div class="answer">

              <textarea rows="5" cols="24" placeholder="Who is flexbox..." name="description"></textarea>

            </div>

 <button type="submit">Send</button>

# 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.

asd

Footer

Last Container of the layout or Website and we add addresses inside it for contact information about author

Syntax

<footer>

<address></address>

</footer>

# 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.

<footer>

  <address></address>

</footer>

"All right!"

# Step 43

Within the address element, add the following:

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.

   <footer>

      <address>

freeCodeCamp<br />

San Francisco<br />

California<br />

USA

      </address>

    </footer>

"We knew you could do it!"

<footer>

      <address>

freeCodeCamp<br />

San Francisco<br />

California<br />

USA

      </address>

    </footer>

# 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.

 <a href="https://freecodecamp.org">freeCodeCamp</a><br />

Here we wrap a freecodecap in a href

"Standing ovation!"

# Step 45

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;

Starting CSS here

…………………………………………………………………………………………………………………………………………………

# Step 45

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;

ul li{

  color:#dfdfe2;

  margin:0 0.2rem;

  padding:0.2rem;

  display:block;

}

We can select by two types

* one is nav> ul >li
* second is nav li

# 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.

li>a{

  color:#dfdfe2;

}

Ratio 7:1 mean contrast of colors with ratio 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.

nav > ul > li:hover{

color:#1b1b32;

background-color:#dfdfe2;;

text-decoration:none;

cursor:pointer;

}

li > a {

  color: inherit;

  text-decoration:none;

}

Here we just make a changes in nav link mean navbar button

* mean child >

# 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.

header {

  width: 100%;

  height: 50px;

  background-color: #1b1b32;

  display: flex;

  justify-content:space-between;

  align-items:center;

  position:fixed;

  top:0;

}

Here is the three changes we do

Position fixed and top o

Which will fix the position and not scrolling with other elements

Align-items:center it will top bottom vertical and left and right space will be center

# 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.

h1 {

  color: #f1be32;

  font-size: min(5vw, 1.2em);

  text-align:center;

}

main{

 padding-top:40px;

}

Here we gives padding-top:40px to see Student info because position fixed hide it

# 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:

align-items: center;

padding-inline-start: 0;

margin-block: 0;

height: 100%;

nav > ul {

  display: flex;

  justify-content: space-evenly;

  align-items: center;

padding-inline-start: 0;

margin-block: 0;

height: 100%;

}

For resposnsive we will use it

flex-wrap:wrap;

# 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.

section{

  width:80%;

  margin:0 auto 10px;

  border:2px solid green;

  max-width:600px;

}

Here we use width 80%

Which will make the box 80% of its parent

Again we center the box with the help of

Margin:auto

Note margin auto will only be use for centering the box only

For text we use text-align center

Max-width:600px;

It means that do not increase the width of the section box above 600px

It will stop then

"All right!"

# Step 52

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

h2 {

  border-bottom: 4px solid #dfdfe2;

  margin-top:0;

  padding-top:60px;

  background-color:red;

}

Here I change the margin to 0 and paddin to 60px px top which will give space in the box from top

The div class=info of the section

We will give padding to it

.info{

  background-color:red;

  padding:1px 0 0 10px;

}

# 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 Passed

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.

.info input{

background:red;

width:50%;

text-align:left;

}

Here I change the input fiend which is inside in info class

# 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.

.info label{

  background:red;

  width:10%;

  min-width: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.

.info label, .info input{

  display:inline-block;

}

.info input {

  width: 50%;

  text-align: left;

}

.info label {

  width: 10%;

  min-width: 55px;

  text-align:right;

}

"To the nines!"

# Step 58

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 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:

ul {

list-style: circle;

}

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

 ul {

  list-style: circle;

}

.answers-list{

  list-style:none;

  padding:0;

}

# Step 62

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;

button{

  display: block;

margin: 40px auto;

width: 40%;

padding: 15px;

font-size: 23px;

background: #d0d0d5;

border: 3px solid #3b3b4f;

}

# Step 63

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

footer{

  color:#2a2a40;

}

# Step 63

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 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.

footer, footer a{

  color:#dfedf2;

}

address{

  padding:10px;

  text-align:center;

}

# 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.

\*{

the scroll-behavior to smooth.

}

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 68

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.

<ul>

          <li><a href="#student-info" accesskey="H">INFO</a></li>

          <li><a href="#html-questions" accesskey="I">HTML</a></li>

          <li><a href="#css-questions" accesskey="C">CSS</a></li>

        </ul>