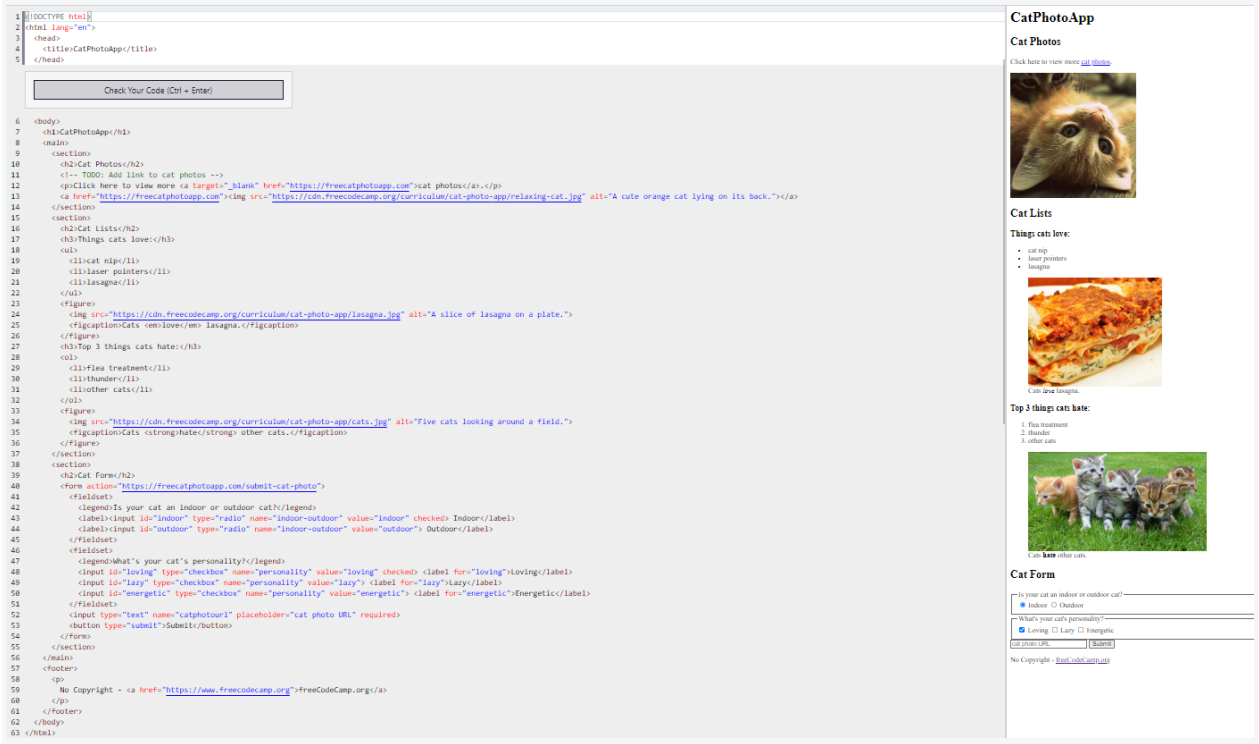



```
<!DOCTYPE html>

<html lang="en">
  <head>
    <title>CatPhotoApp</title>
  </head>
  <body>
    <h1>CatPhotoApp</h1>
    <main>
      <section>
        <h2>Cat Photos</h2>
        <!-- TODO: Add link to cat photos -->
        <p>Click here to view more <a target="_blank"
href="https://freecatphotoapp.com">cat photos</a>.</p>
        <a href="https://freecatphotoapp.com"><img
src="https://cdn.freecodecamp.org/curriculum/cat-photo-app/relaxing-cat.jpg"
alt="A cute orange cat lying on its back."></a>
      </section>
      <section>
        <h2>Cat Lists</h2>
        <h3>Things cats love:</h3>
        <ul>
          <li>cat nip</li>
          <li>laser pointers</li>
```

```
          <li>lasagna</li>
        </ul>
        <figure>
          <img
src="https://cdn.freecodecamp.org/curriculum/cat-photo-app/lasagna.jpg" alt="A
slice of lasagna on a plate.">
          <figcaption>Cats <em>love</em> lasagna.</figcaption>
        </figure>
        <h3>Top 3 things cats hate:</h3>
        <ol>
          <li>flea treatment</li>
          <li>thunder</li>
          <li>other cats</li>
        </ol>
        <figure>
          <img
src="https://cdn.freecodecamp.org/curriculum/cat-photo-app/cats.jpg" alt="Five
cats looking around a field.">
          <figcaption>Cats <strong>hate</strong> other cats.</figcaption>
        </figure>
      </section>
      <section>
        <h2>Cat Form</h2>
        <form action="https://freecatphotoapp.com/submit-cat-photo">
          <fieldset>
            <legend>Is your cat an indoor or outdoor cat?</legend>
            <label><input id="indoor" type="radio" name="indoor-outdoor"
value="indoor" checked> Indoor</label>
            <label><input id="outdoor" type="radio" name="indoor-outdoor"
value="outdoor"> Outdoor</label>
          </fieldset>
          <fieldset>
            <legend>What's your cat's personality?</legend>
            <input id="loving" type="checkbox" name="personality"
value="loving" checked> <label for="loving">Loving</label>
            <input id="lazy" type="checkbox" name="personality" value="lazy">
<label for="lazy">Lazy</label>
            <input id="energetic" type="checkbox" name="personality"
value="energetic"> <label for="energetic">Energetic</label>
          </fieldset>
          <input type="text" name="catphotourl" placeholder="cat photo URL"
required>
          <button type="submit">Submit</button>
```

```
        </form>
      </section>
    </main>
    <footer>
      <p>
        No Copyright - <a
href="https://www.freecodecamp.org">freeCodeCamp.org</a>
      </p>
    </footer>
  </body>
</html>
```

PAGE 17 NEXT PROJECT CAFE MENU
PAGE 36 NEXT
<ELEMENT> ATTRIBUTE=”VALUE” <ELEMENT> ****SOMETIMES NO CLOSING
HTML elements have opening tags like `<h1>` and closing tags like `</h1>`.
I.e. use <>
i.e.: `<h1>CatPhotoApp</h1>`
The `h1(high)` to `h6` (low)heading elements are used to signify the importance of content below them. The lower the number, the higher the importance, so `h2` elements have less importance than `h1` elements. Only use one `h1` element per page and place **lower importance headings below higher importance headings.(appear bold)**

<element attribute=”value”>

Paragraph p elements are used to create paragraph text on websites. (ends with) (appears smaller, less bold)

Commenting allows you to leave messages without affecting the browser display. It also allows you to make code inactive. A comment in HTML starts with `<!--`, contains any number of lines of text, and ends with `-->`. For example, the comment **`<!-- TODO: Remove h1 -->`** contains the text `TODO: Remove h1`.

HTML5 has some elements that identify different content areas. These elements make your HTML easier to read and help with Search Engine Optimization (SEO) and accessibility.
Identify the main section of this page by adding a `<main>` opening tag after the `h1` element, and a `</main>` closing tag after the `p` element.
HTML elements are often nested within other HTML elements.

HTML elements are often nested within other HTML elements. In the previous step you nested the `h2` element, comment and `p` element within the `main` element. A nested element is a child of its parent element.

To make HTML easier to read, indent the `h2` element, the comment, and `p` element exactly two spaces to indicate they are children of the `main` element.

After the `h3` element with the `Things cats love:` text, add an unordered list (`ul`) element. Note that nothing will be displayed at this point.
You can add images to your website by using the `img` element. `img` elements have an opening tag without a closing tag. A tag for an element without a closing tag is known as a self-closing tag.

Add an `img` element below the `p` element. At this point, no image will show up in the browser.

HTML attributes are special words used inside the opening tag of an element to control the element's behavior. The `src` attribute in an `img` element specifies the image's URL (where the image is located). An example of an `img` element using an `src` attribute: `<img src="https://www.your-image-source.com/your-image.jpg">`

All `img` elements should have an `alt` attribute. The `alt` attribute's text is used for screen readers to improve accessibility and is displayed if the image fails to load. For example, `<img src="cat.jpg" alt="A cat">` has an `alt` attribute with the text `A cat`.

You can link to another page with the anchor (`a`) element. For example, `<a href='https://freecodecamp.org'></a>` would link to `freecodecamp.org`.

The HREF is **an attribute of the anchor tag, which is also used to identify sections within a document**. The HREF contains two components: the URL, which is the actual link

A link's text must be placed between the opening and closing tags of an anchor (a) element. For example, `<a href="https://www.freecodecamp.org">click here to go to freeCodeCamp.org</a>` is a link with the text `click here to go to freeCodeCamp.org`.

Turn the words `cat photos` located inside `p` element into a link by replacing the words with the anchor element added previously. The `p` element should show the same text in the browser, but the words `cat photos` should now be a link. There should only be one link showing in the app.

```
<p>Click here to view more <a href="https://freecatphotoapp.com">cat photos</a></p>
```

Add a `target` attribute with the value `_blank` to the anchor (a) element's opening tag, so that the link opens in a new tab.(The target attribute **specifies a name or a keyword that indicates where to display the response that is received after submitting the form)**

- **_blank: It opens the link in a new window.**
- **_self: It is the default value. ...**
- **_parent: It opens the linked document in the parent frameset.**
- **_top: It opens the linked document in the full body of the window.**
- **framename: It opens the linked document in the named frame.**

```
<p>Click here to view more <a target="_blank" href="https://freecatphotoapp.com">cat photos</a>.</p>
```

Turn the image into a link by surrounding it with necessary element tags. Use `https://freecatphotoapp.com` as the anchor's `href` attribute value.
Anchor an image link

```
<a href="https://freecatphotoapp.com"><img src="https://cdn.freecodecamp.org/curriculum/cat-photo-app/relaxing-cat.jpg" alt="A cute orange cat lying on its back."></a>
```

Before adding any new content, you should make use of a `section` element to separate the cat photos content from the future content.

Take all the elements currently located within the `main` element and nest them in a `section` element.

Before adding any new content, you should make use of a `section` element to separate the cat photos content from the future content.

Take all the elements currently located within the `main` element and nest them in a `section` element.

```
<main>
      <section>
      <h2>Cat Photos</h2>
      <!-- TODO: Add link to cat photos -->
      <p>Click here to view more <a target="_blank" href="https://freecatphotoapp.com">cat photos</a>.</p>
      <a href="https://freecatphotoapp.com"><img src="https://cdn.freecodecamp.org/curriculum/cat-photo-app/relaxing-cat.jpg" alt="A cute orange cat lying on its back."></a>
      </section>
    </main>
```

When you add a lower rank heading element to the page, it's implied that you're starting a new subsection.

After the last `h2` element of the second `section` element, add an `h3` element with the text `Things cats love:`.

```
<section>
        <h2>Cat Lists</h2>
      </section>
```

After the `h3` element with the `Things cats love:` text, add an unordered list (`ul`) element. Note that nothing will be displayed at this point.

```
<h2>Cat Lists</h2>
        <h3>Things cats love:</h3>
        <ul>Things cats love:</ul>
```

Use list item (`li`) elements to create items in a list. Here is an example of list items in an unordered list:

```
<ul>
  <li>milk</li>
  <li>cheese</li>
</ul>
```

Nest three list items within the `ul` element to display three things cats love: `cat nip`, `laser pointers` and `lasagna`.

The ul element represents an unordered list of items that, in contrast with ordered lists (ol element), doesn't produce a different meaning if the items order is altered. The items in both, ordered and unordered lists, are represented by the li element.

The `figure` element represents self-contained content and will allow you to associate an image with a caption.

Nest the image you just added within a `figure` element.

```
<figure><img src="https://cdn.freecodecamp.org/curriculum/cat-photo-app/lasagna.jpg" alt="A slice of lasagna on a plate."></figure>
```

A figure caption (`figcaption`) element is used to add a caption to describe the image contained within the `figure` element. For example, `<figcaption>A cute cat</figcaption>` adds the caption `A cute cat`.

After the image nested in the `figure` element, add a `figcaption` element with the text `Cats love lasagna.`

```
<figure>
          <img src="https://cdn.freecodecamp.org/curriculum/cat-photo-app/lasagna.jpg" alt="A slice of lasagna on a plate."><figcaption>Cats love lasagna</figcaption>
        </figure>
```

CatPhotoApp

Cat Photos

Click here to view more cat photos.

Cat Lists

Things cats love:

- cat nip
- laser pointers
- lasagna

Emphasize the word `love` in the `figcaption` element by wrapping it in an emphasis (`em`) element. *italicize*

```
<figcaption>Cats <em>love</em> lasagna.</figcaption>
```

The code for an ordered list (`ol`) is similar to an unordered list, but list items in an ordered list are numbered when displayed.

After the second `section` element's last `h3` element, add an ordered list with these three list items: `flea treatment`, `thunder` and `other cats`.

```
<h3>Top 3 things cats hate:</h3>
        <ol>
            <li>flea treatment</li>
            <li>thunder</li>
            <li>other cats</li>
        </ol>
```

Inside the `figure` element you just added, nest an `img` element with a `src` attribute set to `https://cdn.freecodecamp.org/curriculum/cat-photo-app/cats.jpg`.

To improve accessibility of the image you just added, add an `alt` attribute with the text `Five cats looking around a field.`

```
<img alt="Five cats looking around a
field"src="https://cdn.freecodecamp.org/curriculum/cat-photo-app/cats.jpg">
```

The `strong` element is used to indicate that some text is of strong importance or urgent. (bold)

In the `figcaption` you just added, indicate that `hate` is of strong importance by wrapping it in a `strong` element.

```
<figcaption>Cats <strong>hate</strong> other cats.</figcaption>
```

Now you will add a web form to collect information from users. After the `Cat Form` heading, add a `form` element.

The `action` attribute indicates where form data should be sent. For example, `<form action="/submit-url"></form>` tells the browser that the form data should be sent to the path `/submit-url`.

The `action` attribute indicates where form data should be sent. For example, `<form action="/submit-url"></form>` tells the browser that the form data should be sent to the path `/submit-url`.

Add an `action` attribute with the value `https://freecatphotoapp.com/submit-cat-photo` to the `form` element.

```
<form action="https://freecatphotoapp.com/submit-cat-photo"</form>
```

The `input` element allows you several ways to collect data from a web form. Like `img` elements, `input` elements are self-closing and do not need closing tags.

Nest an `input` element in the `form` element.

```
<form action="https://freecatphotoapp.com/submit-cat-photo">
        <input></form>
```

There are many kinds of inputs you can create using the `type` attribute. You can easily create a password field, reset button, or a control to let users select a file from their computer.
Create a text field to get text input from a user by adding the `type` attribute with the value `text` to the `input` element.

```
<input type="text">
```

In order for a form's data to be accessed by the location specified in the `action` attribute, you must give the text field a `name` attribute and assign it a value to represent the data being submitted. For example, you could use the following syntax for an email address text field: `<input type="text" name="email">`.

Add the `name` attribute with the value `catphotourl` to your text field.

```
<input type="text"name="catphotourl">
```

Placeholder text is used to give people a hint about what kind of information to enter into an input. For example, `<input type="text" placeholder="Email address">`.

Add the placeholder text `cat photo URL` to your `input` element.

```
<input placeholder="cat photo URL"type="text"
name="catphotourl">
```

To prevent a user from submitting your form when required information is missing, you need to add the `required` attribute to an `input` element. There's no need to set a value to the `required` attribute. Instead, just add

the word `required` to the `input` element, making sure there is space between it and other attributes.

```
<input required type="text" name="catphotourl"
placeholder="cat photo URL">
```

Use the `button` element to create a clickable button. For example, `<button>Click Here</button>` creates a button with the text `Click Here`.

Add a `button` element with the text `Submit` below the `input` element. The default behavior of clicking a form button without any attributes submits the form to the location specified in the form's `action` attribute.

```
<button>Submit</button>
```

Cats *love* lasagna.

Top 3 things cats hate:

1. flea treatment
2. thunder
3. other cats

Cats **hate** other cats.

Cat Form

cat photo URL Submit

Even though you added your button below the text input, they appear next to each other on the page. That's because both `input` and `button` elements are inline elements, which don't appear on new lines.

You learned previously that the button submits the form by default, but you can explicitly add the `type` attribute with the value `submit` to make it clearer. Go ahead and do this to specify where this button should submit the form.

```
<button type="submit">Submit</button>
```

You can use radio buttons for questions where you want only one answer out of multiple options.

Here is an example of a radio button with the option of `cat`: `<input type="radio"> cat`. Remember that `input` elements are self-closing.

Before the text input, add a radio button with the option `Indoor`.

```
<input type="radio"> Indoor
          <input type="text" name="catphotourl" placeholder="cat photo URL" required>
```

`label` elements are used to help associate the text for an `input` element with the input element itself (especially for assistive technologies like screen readers). For example, `<label><input type="radio"> cat</label>` makes it so clicking the word `cat` also selects the corresponding radio button.

Nest your `radio` button inside a `label` element.

```
<label><input type="radio"> Indoor</label>
```

The `id` attribute is used to identify specific HTML elements. Each `id` attribute's value must be unique from all other `id` values for the entire page.

Add an `id` attribute with the value `indoor` to the radio button. When elements have multiple attributes, the order of the attributes doesn't matter.

```
<label><input id="indoor" type="radio"> Indoor</label>
```

Nest another radio button with the option `Outdoor` in a new `label` element. The new radio button should be placed after the first one. Also, set its `id` attribute value to `outdoor`.

```
<label><input id="outdoor" type="radio"> Outdoor</label>
```

Notice that both radio buttons can be selected at the same time. To make it so selecting one radio button automatically deselects the other, both buttons must have a `name` attribute with the same value.

Add the `name` attribute with the value `indoor-outdoor` to both radio buttons.

```
          <label><input id="outdoor" type="radio"
name="indoor-outdoor"> Outdoor</label>
<label><input id="outdoor" type="radio"
name="indoor-outdoor"> Outdoor</label>
```

If you select the `Indoor` radio button and submit the form, the form data for the button is based on its `name` and `value` attributes. Since your radio buttons do not have a `value` attribute, the form data will include `indoor-outdoor=on`, which is not useful when you have multiple buttons.

Add a `value` attribute to both radio buttons. For convenience, set the button's `value` attribute to the same value as its `id` attribute.

```
<label><input id="indoor" type="radio"
name="indoor-outdoor" value="indoor"> Indoor</label>
          <label><input id="outdoor" type="radio"
name="indoor-outdoor" value="outdoor"> Outdoor</label>
```

The `fieldset` element is used to group related inputs and labels together in a web form. `fieldset` elements are block-level elements, meaning that they appear on a new line.

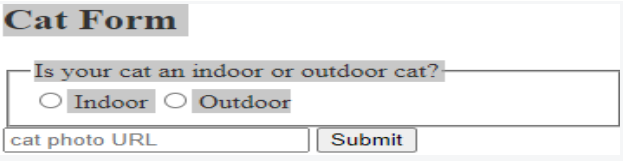
Nest the `Indoor` and `Outdoor` radio buttons within a `fieldset` element, and don't forget to indent the radio buttons.

```
            <fieldset>
        <label><input id="indoor" type="radio" name="indoor-outdoor" value="indoor">
Indoor</label>
        <label><input id="outdoor" type="radio" name="indoor-outdoor" value="outdoor">
Outdoor</label>
        </fieldset>
```

The `legend` element acts as a caption for the content in the `fieldset` element. It gives users context about what they should enter into that part of the form.

Add a `legend` element with the text `Is your cat an indoor or outdoor cat?` above both of the radio buttons.

```
<fieldset>
          <legend>Is your cat an indoor or outdoor cat?</legend>
```



Next, you are going to add some new form `input` elements, so add another `fieldset` element directly below the current `fieldset` element. Add a `legend` element with the text `What's your cat's personality?` inside the second `fieldset` element.

Forms commonly use checkboxes for questions that may have more than one answer. For example, here's a checkbox with the option of `tacos`: `<input type="checkbox"> tacos`.

Under the `legend` element you just added, add an `input` with its `type` attribute set to `checkbox` and give it the option of `Loving`.

```
<legend>What's your cat's personality?</legend>
            <input type="checkbox"> Loving
```

Add an `id` attribute with the value `loving` to the checkbox input.

```
<input id="loving" type="checkbox"> Loving
```

There's another way to associate an `input` element's text with the element itself. You can nest the text within a `label` element and add a `for` attribute with the same value as the `input` element's `id` attribute.

Associate the text `Loving` with the checkbox by only nesting the text `Loving` in a `label` element and place it to the right side of the checkbox `input` element.

```
<input id="loving" type="checkbox"> <label for="loving">Loving</label>
```

Add the `name` attribute with the value `personality` to the checkbox `input` element.

While you won't notice this in the browser, doing this makes it easier for a server to process your web form, especially when there are multiple checkboxes.

```
<input name="personality" id="loving" type="checkbox"> <label for="loving">Loving</label>
```

Add another checkbox after the one you just added. The `id` attribute value should be `lazy` and the `name` attribute value should be the same as the last checkbox.

Also add a `label` element to the right of the new checkbox with the text `Lazy`. Make sure to associate the `label` element with the new checkbox using the `for` attribute.

```
<input id="loving" type="checkbox" name="personality"> <label for="loving">Loving</label>
            <input id="lazy" type="checkbox" name="personality"> <label for="lazy">Lazy</label>
```

Add a final checkbox after the previous one with an `id` attribute value of `energetic`. The `name` attribute should be the same as the previous checkbox.

Also add a `label` element to the right of the new checkbox with text `Energetic`. Make sure to associate the `label` element with the new checkbox.

```
<input id="energetic" type="checkbox" name="personality"> <label
for="energetic">Energetic</label>
```

Like radio buttons, form data for selected checkboxes are `name` / `value` attribute pairs. While the `value` attribute is optional, it's best practice to include it with any checkboxes or radio buttons on the page.

Add a `value` attribute to each checkbox. For convenience, set each checkbox's `value` attribute to the same value as its `id` attribute.

i.e.: `<input value="energetic" id="energetic" type="checkbox" name="personality"> <label for="energetic"> Energetic</label>`

In order to make a checkbox checked or radio button selected by default, you need to add the `checked` attribute to it. There's no need to set a value to the `checked` attribute. Instead, just add the word `checked` to the `input` element, making sure there is space between it and other attributes.

Make the first radio button and the first checkbox selected by default.

```
<label><input checked id="indoor" type="radio" name="indoor-outdoor" value="indoor">
Indoor</label>
<input checked id="loving" type="checkbox" name="personality" value="loving"> <label
for="loving">Loving</label>
```

Now you will add a footer section to the page.

After the `main` element, add a `footer` element.

```
<footer></footer>
```

Nest a `p` element with the text `No Copyright - freeCodeCamp.org` within the `footer` element.

```
<footer><p> No Copyright - freeCodeCamp.org</p>
    </footer>
```

Make the text `freeCodeCamp.org` into a link by enclosing it in an anchor (`a`) element. The `href` attribute should be set to `https://www.freecodecamp.org`.

```
No Copyright - <a href="https://www.freecodecamp.org">
freeCodeCamp.org </a>
```

Notice that everything you've added to the page so far is inside the `body` element. All page content elements that should be rendered to the page go inside the `body` element. However, other important information goes inside the `head` element.

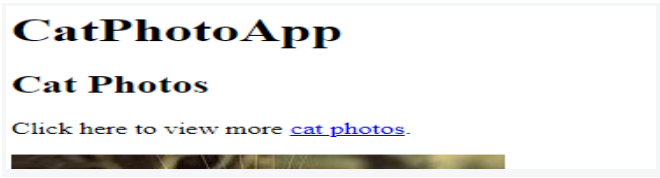
Add a `head` element just above the `body` element.

```
<head>
  </head>
```

The `title` element determines what browsers show in the title bar or tab for the page.

Add a `title` element within the `head` element. Its text should be `CatPhotoApp`.

```
<head><title>CatPhotoApp</title>
  </head>
```



Notice that the entire contents of the page are nested within an `html` element. All other elements must be descendants of this `html` element.

Add the `lang` attribute with the value `en` to the opening `html` tag to specify that the language of the page is English.

```
<html lang="en">
```

All pages should begin with `<!DOCTYPE html>`. This special string is known as a declaration and ensures the browser tries to meet industry-wide specifications.

To complete this project, add this declaration as the first line of the code.

```
<!DOCTYPE html>
<html lang="en">
```

CAFE MENU

BASIC CSS

HTML below CSS after

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Cafe Menu</title>
    <link href="styles.css" rel="stylesheet"/>
  </head>
  <body>
    <div class="menu">
      <header>
```

```
        <h1>CAMPER CAFE</h1>
        <p class="established">Est. 2020</p>
      </header>
      <hr>
      <main>
        <section>
          <h2>Coffee</h2>
          <img
src="https://cdn.freecodecamp.org/curriculum/css-cafe/coffee.jpg" alt="coffee
icon"/>
          <article class="item">
            <p class="flavor">French Vanilla</p><p class="price">3.00</p>
          </article>
          <article class="item">
            <p class="flavor">Caramel Macchiato</p><p class="price">3.75</p>
          </article>
          <article class="item">
            <p class="flavor">Pumpkin Spice</p><p class="price">3.50</p>
          </article>
          <article class="item">
            <p class="flavor">Hazelnut</p><p class="price">4.00</p>
          </article>
          <article class="item">
            <p class="flavor">Mocha</p><p class="price">4.50</p>
          </article>
        </section>
        <section>
          <h2>Desserts</h2>
          <img src="https://cdn.freecodecamp.org/curriculum/css-cafe/pie.jpg"
alt="pie icon"/>
          <article class="item">
            <p class="dessert">Donut</p><p class="price">1.50</p>
          </article>
          <article class="item">
            <p class="dessert">Cherry Pie</p><p class="price">2.75</p>
          </article>
          <article class="item">
            <p class="dessert">Cheesecake</p><p class="price">3.00</p>
          </article>
          <article class="item">
            <p class="dessert">Cinnamon Roll</p><p class="price">2.50</p>
          </article>
        </section>
```

```
      </main>
      <hr class="bottom-line">
      <footer>
        <p>
          <a href="https://www.freecodecamp.org" target="_blank">Visit our
website</a>
        </p>
        <p class="address">123 Free Code Camp Drive</p>
      </footer>
    </div>
  </body>
</html>
```

CSS CODE

```
body {
  background-image:
url(https://cdn.freecodecamp.org/curriculum/css-cafe/beans.jpg);
  font-family: sans-serif;
  padding: 20px;
}

h1 {
  font-size: 40px;
  margin-top: 0;
  margin-bottom: 15px;
}

h2 {
  font-size: 30px;
}

.established {
  font-style: italic;
}

h1, h2, p {
  text-align: center;
}

.menu {
  width: 80%;
```

```
  background-color: burlywood;
  margin-left: auto;
  margin-right: auto;
  padding: 20px;
  max-width: 500px;
}

img {
  display: block;
  margin-left: auto;
  margin-right: auto;
}

hr {
  height: 2px;
  background-color: brown;
  border-color: brown;
}

.bottom-line {
  margin-top: 25px;
}

h1, h2 {
  font-family: Impact, serif;
}

.item p {
  display: inline-block;
  margin-top: 5px;
  margin-bottom: 5px;
  font-size: 18px;
}

.flavor, .dessert {
  text-align: left;
  width: 75%;
}

.price {
  text-align: right;
  width: 25%;
}
```

```css
/* FOOTER */

footer {
  font-size: 14px;
}

.address {
  margin-bottom: 5px;
}

a {
  color: black;
}

a:visited {
  color: black;
}

a:hover {
  color: brown;
}

a:active {
  color: brown;
}
```

SELECTOR { PROPERTY:VALUE; ETC.}

Add a `head` element within the `html` element, so you can add a `title` element. The `title` element's text should be `Cafe Menu`.
The `title` is one of several elements that provide extra information not visible on the web page, but it is useful for search engines or how the page gets displayed.

Inside the `head` element, nest a `meta` element (defines metadata about an HTML document)with an attribute named `charset` set to the value `utf-8` to tell the browser how to encode characters for the page. Note that `meta` elements are self-closing.

```
<head>
   <meta charset="utf-8" <title>Cafe Menu</title>
  </head>
```

To prepare to create some actual content, add a `body` element below the `head` element.
The name of the cafe is `CAMPER CAFE`. Add an `h1` element within your `body` element. Give it the name of the cafe in capitalized letters to make it stand out.

```
<body>
    <h1>CAMPER CAFE</h1>
  </body>
```

Since the `p` element added in the previous step provides supplemental information about the cafe, nest both the `h1` and `p` elements in a `header` element - represents introductory content, typically a group of introductory or navigational aids. It may contain some heading elements but also a logo, a search form, an author name, and other elements

```
<header><h1>CAMPER CAFE</h1>
    <p>Est. 2020</p></header>.
```

It's time to add some menu content. Add a `main` element below the existing `header` element. It will eventually contain pricing information about coffee and desserts offered by the cafe.
There will be two sections on the menu, one for coffees and one for desserts. Add a `section` element within the `main` element so you have a place to put all the coffees available.

```
<main>
      <section></section>
    </main>
```

Up until now, you have been limited regarding the presentation and appearance of the content you create. To start taking control, add a `style` element within the `head` element.

```
<style></style>
```

You can add style to an element by specifying it in the `style` element and setting a property for it like this:

```
element {
 property: value;
}
```

Center your `h1` element by setting its `text-align` property to the value `center`.

```
<style>
    h1 {
        text-align: center;
      }
    </style>
```

In the previous step, you used a type selector to style the `h1` element. Go ahead and center the `h2` and `p` elements with a new type selector for each one. (better way with commas)

```
<style>
     h1 {
       text-align: center;
     }
     h2 {
       text-align: center;
     }
     p {
       text-align: center;
     }
   </style>
```

You now have three type selectors with the exact same styling. You can add the same group of styles to many elements by separating the selectors with commas like this:

```
selector1, selector2 {

  property: value;
}
```

Use a single type selector to center the `h1`, `h2` and `p` elements at the same time.

```
<style>
```

```
      h1, h2, p {
        text-align: center;
      }

    </style>
```

CSS (Cascading Style Sheets) is used to style and layout web pages: use <link

You have styled three elements by writing CSS inside the `style` tags. This works, but since there will be many more styles, it's best to put all the styles in a separate file and link to it.

We have created a separate `styles.css` file for you and switched the editor view to that file. You can change between files with the tabs at the top of the editor.

Start by rewriting the styles you have created into the `styles.css` file. Make sure to exclude the opening and closing `style` tags.

(different tab)
```
h1, h2, p {
        text-align: center;
      }
```

Now that you have the CSS in the `styles.css` file, go ahead and remove the `style` element and all its content. Once it is removed, the text that was centered will shift back to the left.

```
<head>
    <meta charset="utf-8" />
    <title>Cafe Menu</title>
  </head>
```

Now you need to link the `styles.css` file so the styles will be applied again. Nest a self-closing `link` element in the `head` element. Give it a `rel` attribute value `stylesheet`, a `type` attribute value of `text/css`, and an `href` attribute value of `styles.css`.

```
head>
    <meta charset="utf-8" />
    <title>Cafe Menu</title>
    <link href="styles.css" rel="stylesheet" type="text/css" />
  </head>
```

For the styling of the page to look similar on mobile as it does on a desktop or laptop, you need to add a `meta` element with a special `content` attribute.

Add the following within the `head` element:

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

The text is centered again so the link to the CSS file is working. Add another style to the file that changes the `background-color` property to `brown` for the `body` element. (another style link..)

```
h1, h2, p {
  text-align: center;
}
body {
  background-color:brown
}
```

That brown background makes it hard to read the text. Change the `body` element's background color to be `burlywood` so it has some color but you are still be able to read the text.

The `div` element is used mainly for design layout purposes unlike the other content elements you have used so far. Add a `div` element inside the `body` element and then move all the other elements inside the new `div`.

The goal now is to make the `div` not take up the entire width of the page. The CSS `width` property is perfect for this. Create a new type selector in the style sheet that gives your `div` element a width of `300px`. (within link code)

```
div {
  width: 300px
}
```

Comments in CSS look like this:

```
/* comment here */
```

In your style sheet, comment out the the line containing the `background-color` property and value, so you can see the effect of only styling `div` element. This will make the background white again.

```
/* background-color: burlywood; */
```

Now make the background color of the `div` element to be `burlywood`.

```
div {
  width: 300px; background-color: burlywood
}
```

Now it's easy to see that the text is centered inside the `div` element. Currently, the width of the `div` element is specified in pixels (`px`). Change the `width` property's value to be `80%`, to make it 80% the width of its parent element (`body`).

```
div {
  width: 80%;
  background-color: burlywood;
}
```

Next, you want to center the `div` horizontally. You can do this by setting its `margin-left` and `margin-right` properties to `auto`. Think of the margin as invisible space around an element. Using these two margin properties, center the `div` element within the `body` element.

```
div {
  width: 80%;
  background-color: burlywood; margin-left: auto; margin-right: auto
}
```

So far you have been using type selectors to style elements. A class selector is defined by a name with a dot directly in front of it, like this:

```
.class-name {
  styles
}
```

Change the existing `div` selector into a class selector by replacing `div` with a class named `menu`.

```
.menu {
  width: 80%;
  background-color: burlywood;
  margin-left: auto;
  margin-right: auto;
}
```

To apply the class's styling to the `div` element, add a `class` attribute to the `div` element's opening tag and set its value to `menu`.

```
<div class="menu">
```

To apply the class's styling to the `div` element, add a `class` attribute to the `div` element's opening tag and set its value to `menu`.

```
<div class="menu">
```

Since the cafe's main product for sale is coffee, you could use an image of coffee beans for the background of the page.

Delete the comment and its contents inside the `body` type selector. Now add a `background-image` property and set its value to `url(https://cdn.freecodecamp.org/curriculum/css-cafe/beans.jpg)`.

```
body {
  background-image:
url(https://cdn.freecodecamp.org/curriculum/css-cafe/beans.jpg)}
```

Time to start adding some menu items. Add an empty `article` element under the `Coffee` heading. It will contain a flavor and price of each coffee you currently offer.

`article` elements commonly contain multiple elements that have related information. In this case, it will contain a coffee flavor and a price for that flavor. Nest two `p` elements inside your `article` element. The first one's text should be `French Vanilla`, and the second's text `3.00`.

```
<article>
            <p>French Vanilla</p>
```

```
            <p>3.00</p>
        </article>
```

Starting below the existing coffee/price pair, add the following coffee and prices using `article` elements with two nested `p` elements inside each. As before, the first `p` element's text should contain the coffee flavor and the second `p` element's text should contain the price.(duplicate the above for other items)

The flavors and prices are currently stacked on top of each other and centered with their respective `p` elements. It would be nice if the flavor was on the left and the price was on the right.

Add the class name `flavor` to the `French Vanilla` `p` element. (in HTML)

```
<p class="flavor">French Vanilla</p>
```

Using your new `flavor` class as a selector, set the `text-align` property's value to `left`. (Back in CSS)

```
.flavor {text-align: left}
```

Next, you want to align the price to the right. Add a class named `price` to your `p` element that has `3.00` as its text. HTML

```
<p class="price">3.00</p>
```

Now align the text to the `right` for the elements with the `price` class. CSS

```
.price { text-align: right}
```

That is kind of what you want, but now it would be nice if the flavor and price were on the same line. `p` elements are block-level elements, so they take up the entire width of their parent element.

To get them on the same line, you need to apply some styling to the `p` elements, so they behave more like `inline` elements. Add a `class` attribute with the value `item` to the first `article` element under the `Coffee` heading.

The `p` elements are nested in an `article` element with the class attribute of `item`. You can style all the `p` elements nested anywhere in elements with a class named `item` like this: `.item p { }`

Using the above selector, add a `display` property with value `inline-block` so the `p` elements behave more like `inline` elements.

```
.item p {display:inline-block}
```

That's closer, but the price didn't stay over on the right. This is because `inline-block` elements only take up the width of their content. To spread them out, add a `width` property to the `flavor` and `price` class selectors that have a value of `50%` each.

Well that did not work. Styling the `p` elements as `inline-block` and placing them on separate lines in the code creates an extra space to the right of the first `p` element, causing the second one to shift to the next line. One way to fix this is to make each `p` element's width a little less than `50%`.

Change the `width` value to `49%` for each class to see what happens. That worked, but there is still a little space on the right of the price.

You could keep trying various percentages for the widths. Instead, simply move the price `p` element to be on the same line and make sure there is no space between them.

```
<p class="flavor">French Vanilla</p><p class="price">3.00</p>
```

Now go ahead and change both the `flavor` and `price` class' widths to be `50%` again.

Now that you know it works, you can change the remaining `article` and `p` elements to match the first set. Start by adding the class `item` to the other `article` elements.`<article class="item">`

Next, position the other `p` elements to be on the same line with no space between them`<p>Caramel Macchiato</p><p>3.75</p> ETC`

To complete the styling, add the applicable class names `flavor` and `price` to all the remaining `p` elements

```
<article class="item">
            <p class="flavor">Caramel Macchiato</p><p class="price">3.75</p>
          </article>
ETC
```

If you make the width of the page preview smaller, you will notice at some point, some of the text on the left starts wrapping around to the next line. This is because the width of the `p` elements on the left side can only take up `50%` of the space.

Since you know the prices on the right have significantly fewer characters, change the `flavor` class `width` value to be `75%` and the `price` class `width` value to be `25%`.

If you make the width of the page preview smaller, you will notice at some point, some of the text on the left starts wrapping around to the next line. This is because the width of the `p` elements on the left side can only take up `50%` of the space.

Since you know the prices on the right have significantly fewer characters, change the `flavor` class `width` value to be `75%` and the `price` class `width` value to be `25%`.

```
.flavor {
```

```
  text-align: left;
  width: 75%;
}

.price {
  text-align: right;
  width: 25%;
} HAVE TO EQUAL 100?
```

Nest two `p` elements inside your `article` element. The first one's text should be `Donut`, and the second's text `1.50`. Put both of them on the same line making sure there is no space between them.

```
  <article class="item">
            <p >Donut</p >
            <p >1.50</p >
          </article>
```

For the two `p` elements you just added, add `dessert` as the value of the first `p` element's `class` attribute and the value `price` as the second `p` elements `class` attribute.

```
<p class="dessert">Donut</p><p class="price">1.50</p>
```

Something does not look right. You added the correct `class` attribute value to the `p` element with `Donut` as its text, but you have not defined a selector for it.

Since the `flavor` class selector already has the properties you want, just add the `dessert` class name to it.

```
.flavor, .dessert {
  text-align: left;
  width: 75%;
}
```

You can give your menu some space between the content and the sides with various `padding` properties.

Give the `menu` class a `padding-left` and a `padding-right` with the same value `20px`.

```
.menu {

  padding-left: 20px; padding-right: 20px; width: 80%;

  background-color: burlywood;
```

```
  margin-left: auto;

  margin-right: auto;

}
```

That looks better. Now try to add the same `20px` padding to the top and bottom of the menu.

```
.menu { padding-top: 20px; padding-bottom: 20px;

  width: 80%;

  background-color: burlywood;

  margin-left: auto;

  margin-right: auto;

  padding-left: 20px;

  padding-right: 20px; }
```

Since all 4 sides of the menu have the same internal spacing, go ahead and delete the four properties and use a single `padding` property with the value `20px`.

```
.menu {

  width: 80%;

  background-color: burlywood;

  margin-left: auto;

  margin-right: auto;

  padding: 20px

}
```

The current width of the menu will always take up 80% of the `body` element's width. On a very wide screen, the coffee and dessert appear far apart from their prices.

Add a `max-width` property to the `menu` class with a value of `500px` to prevent it from growing too wide.

```
.menu { max-width: 500px; ETC}
```

You can change the `font-family` of text, to make it look different from the default font of your browser. Each browser has some common fonts available to it.

Change all the text in your `body`, by adding a `font-family` property with the value `sans-serif`. This is a fairly common font that is very readable.

```
body {font-family: sans-serif;
  background-image: url(https://cdn.freecodecamp.org/curriculum/css-cafe/beans.jpg);
}
```

It is a bit boring for all the text to have the same `font-family`. You can still have the majority of the text `sans-serif` and make just the `h1` and `h2` elements different using a different selector.

Style both the `h1` and the `h2` elements so that only these elements' text use `Impact` font.`h1, h2 {font-family:Impact }`

You can add a fallback value for the font-family by adding another font name separated by a comma. Fallbacks are used in instances where the initial is not found/available.

Add the fallback font `serif` after the `Impact` font.`h1, h2 {`

```
  font-family: Impact, serif;}
```

Make the `Est. 2020` text italicized by creating an `established` class selector and giving it the `font-style` property with the value `italic`.`.established {`

```
  font-style: italic;}
```

Now apply the `established` class to the `Est. 2020` text

```
<header>
        <h1>CAMPER CAFE</h1>
        <p class="established">Est. 2020</p>
```

```
    </header>
```

The typography of heading elements (e.g. `h1`, `h2`) is set by default values of users' browsers.

Add two new type selectors (`h1` and `h2`). Use the `font-size` property for both, but use the value `40px` for the `h1` and `30px` for the `h2`.

```
h1 {font-size: 40px}
h2 {font-size: 30px}
```

Add a `footer` element below the `main` element, where you can add some additional information. `<footer>  </footer>`

Inside the `footer`, add a `p` element. Then, nest an anchor (`a`) element in the `p` that links to `https://www.freecodecamp.org` and has the text `Visit our website`. `<footer>`

```
      <p> <a href='https://www.freecodecamp.org'</a>Visit our website
    </footer>
```

Add a second `p` element below the one with the link and give it the text `123 Free Code Camp Drive`.`<p><a href="https://www.freecodecamp.org" target="_blank">123 Free Code Camp Drive</a>`

You can use an `hr` element to display a divider between sections of different content.

First, add an `hr` element between the first `header` element and the `main` element. Note that `hr` elements are self closing. `<hr>`

The default properties of an `hr` element will make it appear as a thin light grey line. You can change the height of the line by specifying a value for the `height` property.

Change the height the `hr` element to be `3px`.`hr {height: 3px}`

Change the background color of the `hr` element to `brown` so it matches the color of the coffee beans.`hr {`

```
  height: 3px; background-color: brown}
```

Notice the grey color along the edges of the line. Those edges are known as borders. Each side of an element can have a different color or they can all be the same.

Make all the edges of the `hr` element the same color as the background of it using the `border-color` property.

```
hr {
  height: 3px;
  background-color: brown; border-color: brown}
```

Notice how the thickness of the line looks bigger? The default value of a property named `border-width` is `1px` for all edges of `hr` elements. By changing the border to the same color as the background, the total height of the line is `5px` (`3px` plus the top and bottom border width of `1px`).

Change the `height` property of the `hr` to be `2px`, so the total height of it becomes `4px`.

```
hr {  height: 2px;ETC
```

To create a little more room around the menu, add `20px` of space on the inside of the `body` element by using the `padding` property.

Focusing on the menu items and prices, there is a fairly large gap between each line.

Target all the `p` elements nested in elements with the `class` named `item` and set their top and bottom margin to be `5px`.

```
.item p {margin-top:5px; margin-bottom: 5px;
  display: inline-block;}
```

Using the same style selector in the previous step, make the font size of the items and prices larger by using a value of `18px`.

```
.item p {
  font-size: 18px; price-size: 18px; ETC
```

Changing the `margin-bottom` to `5px` looks great. However, now the space between the `Cinnamon Roll` menu item and the second `hr` element does not match the space between the top `hr` element and the `Coffee` heading.

Add some more space by creating a class named `bottom-line` using `25px` for the `margin-top` property.

```
.bottom-line {margin-top: 25px}
```

Now add the `bottom-line` class to the second `hr` element so the styling is applied. `<hr class="bottom-line">`

Next you are going to be styling the `footer` element. To keep the CSS organized, add a comment at the end of `styles.css` with the text `FOOTER`. `footer {/* FOOTER */}`

Moving down to the `footer` element, make all the text have a value of `14px` for the font size. `footer {font-size: 14px}`

The default color of a link that has not yet been clicked on is typically blue. The default color of a link that has already been visited from a page is typically purple.

To make the `footer` links the same color regardless if a link has been visited, use a type selector for the anchor element (`a`) and use the value `black` for the `color` property. `a {color: black}`

You change properties of a link when the link has actually been visited by using a pseudo-selector that looks like `a:visited { propertyName: propertyValue; }`.

Change the color of the footer `Visit our website` link to be `grey` when a user has visited the link. `a:visited {color:grey;}`

You change properties of a link when the mouse hovers over them by using a pseudo-selector that looks like `a:hover { propertyName: propertyValue; }`.

Change the color of the footer `Visit our website` link to be `brown` when a user hovers over it.`a:hover {color:brown}`

You change properties of a link when the link is actually being clicked by using a pseudo-selector that looks like `a:active { propertyName: propertyValue; }`.

Change the color of the footer `Visit our website` link to be `white` when clicked on. `a:active {color: white}`

To keep with the same color theme you have already been using (black and brown), change the color for when the link is visited to `black` and use `brown` for when the link is actually clicked. ETC
The menu text `CAMPER CAFE` has a different space from the top than the address at the bottom of the menu. This is due to the browser having some default top margin for the `h1` element.

Change the top margin of the `h1` element to `0` to remove all the top margin.`h1 { margin-top:0;`

```
  font-size: 40px;}
```

To remove some of the vertical space between the `h1` element and the text `Est. 2020`, change the bottom margin of the `h1` to `15px`. `h1 { margin-bottom: 15px;ETC`

Now the top spacing looks good. The space below the address at the bottom of the menu is a little bigger than the space at the top of the menu and the `h1` element.

To decrease the default margin space below the address `p` element, create a class selector named `address` and use the value `5px` for the `margin-bottom` property. `.address {margin-bottom: 5px}`
Now apply the `address` class to the `p` element containing the address.
`<p class="address">123 Free Code Camp Drive</p>`
The menu looks good, but other than the coffee beans background image, it is mainly just text.

Under the `Coffee` heading, add an image using the url `https://cdn.freecodecamp.org/curriculum/css-cafe/coffee.jpg`. Give the image an `alt` value of `coffee icon`.

```
<h2><img alt="coffee icon"
src="https://cdn.freecodecamp.org/curriculum/css-cafe/coffee.jpg">Coffee</h2>
```

The image you added is not centered horizontally like the `Coffee` heading above it. `img` elements are "like" inline elements.

To make the image behave like heading elements (which are block-level), create an `img` type selector and use the value `block` for the `display` property and use the applicable `margin-left` and `margin-right` values to center it horizontally.`img {display: block; margin-left:auto; margin-right:auto}`

Add one last image under the `Desserts` heading using the url `https://cdn.freecodecamp.org/curriculum/css-cafe/pie.jpg`. Give the image an `alt` value of `pie icon`.

```
<h2> <img src="https://cdn.freecodecamp.org/curriculum/css-cafe/pie.jpg" alt="pie icon" Desserts</h2>
```

It would be nice if the vertical space between the `h2` elements and their associated icons was smaller. The `h2` elements have default top and bottom margin space, so you could change the bottom margin of the `h2` elements to say `0` or another number.

There is an easier way, simply add a negative top margin to the `img` elements to pull them up from their current positions. Negative values are created using a `-` in front of the value. To complete this project, go ahead and use a negative top margin of `25px` in the `img` type selector. `img { margin-top: -25px;`

LEARN CSS COLORS BY BUILDING A SET OF COLORED MARKERS

As you've seen in the previous projects, webpages should start with a `DOCTYPE html` declaration, followed by an `html` element.

Add a `DOCTYPE html` declaration at the top of the document, and an `html` element after that.

```
<!DOCTYPE html>

<html>

  </html>
```

Within the `head` element, nest a `title` element with the text `Colored Markers`.

```
<head>
    <title>Colored Markers
      </title>
  </head>
```

To tell browsers how to encode characters on your page, set the `charset` to `utf-8`. `utf-8` is a universal character set that includes almost every character from all human languages.

Inside the `head` element, nest a `meta` element with the attribute `charset` set to `utf-8`. Remember that `meta` elements are self-closing, and do not need a closing tag.

```
<head>
    <meta charset="utf-8"
    <title>Colored Markers</title>
  </head>
```

Finally, use a `viewport` `<meta>` tag to make sure your page looks the same on all devices.The viewport is **the user's visible area of a web page**. The viewport varies with the device, and will be smaller on a mobile phone than on a computer screen

Nest a self-closing `meta` element within the `head`. Give it a `name` attribute set to `viewport` and a `content` attribute set to `width=device-width, initial-scale=1.0`.

```
<head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Colored Markers</title>
  </head>
```

Now you're ready to start adding content to the page.

Within the `body`, nest an `h1` element with the text `CSS Color Markers`.

```
<body>
    <h1>CSS Color Markers
      </h1>
  </body>
```

In this project you'll work with an external CSS file to style the page. We've already created a `styles.css` file for you. But before you can use it, you'll need to link it to the page.

Nest a `link` element within the `head`. Give it a `rel` attribute set to `stylesheet`, a `type` attribute set to `text/css`, and an `href` attribute set to `styles.css`.

```
<head>
    <link rel="stylesheet"; type="text/css"; href="styles.css">
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Colored Markers</title>
  </head>
```

Now that your external CSS file is set up, you can start styling the page.

As a reminder, here's how to target a paragraph element and align it to the right:`p {`

```
  text-align: right;
}
```

Create a new CSS rule that targets the `h1` element, and set its `text-align` property to `center`.

```
h1 {
text-align:center;
}
```

Now you'll add some elements that you'll eventually style into color markers.

First, within the `body`, add a `div` element and set its `class` attribute to `container`. Make sure the `div` element is below the `h1` element.

```
<body>
    <h1>CSS Color Markers</h1>
    <div class="container">
      </div>
  </body>
```

Next, within the `div`, add another `div` element and give it a class of `marker`.

```
<div class="container">
      <div class="marker">
        </div>
    </div>
```

It's time to add some color to the page. Remember that one way to add color to an element is to use a color keyword like `black`, `cyan`, or `yellow`.

As a reminder, here's how to target the class `freecodecamp`: `.freecodecamp {`

```


}
```

Create a new CSS rule that targets the class `marker`, and set its `background-color` property to `red`.

```
.marker {
  background-color:red;
}
```

Notice that your marker doesn't seem to have any color. The background color was actually applied, but since the marker `div` element is empty, it doesn't have any width or height by default.

In your `.marker` CSS rule, set the `width` property to `200px` and the `height` property to `25px`.

```
.marker {
  background-color: red; width:200px; height:25px;
}
```

Your marker would look better if it was centered on the page. An easy way to do that is with the `margin` shorthand property.

In the last project, you set the margin area of elements separately with properties like `margin-top` and `margin-left`. The `margin` shorthand property makes it easy to set multiple margin areas at the same time.

To center your marker on the page, set its `margin` property to `auto`. This sets `margin-top`, `margin-right`, `margin-bottom`, and `margin-left` all to `auto`.
i.e.: `margin:auto;`
Now that you've got one marker centered with color, it's time to add the other markers.

In the `container div`, add two more `div` elements and give them each a class of `marker`. (with closing tags=same)

While you have three separate marker `div` elements, they look like one big rectangle. You should add some space between them to make it easier to see each element.

When the shorthand `margin` property has two values, it sets `margin-top` and `margin-bottom` to the first value, and `margin-left` and `margin-right` to the second value.

In your `.marker` CSS rule, set the `margin` property to `10px auto`.

```
.marker {
  width: 200px;
  height: 25px;
  background-color: red;
  margin: 10px auto;
}
```

In school, you might have learned that red, yellow, and blue are primary colors, and learned how to create new colors by mixing those. However, this is an outdated model.

These days, there are two main color models: the additive RGB (red, green, blue) model used in electronic devices, and the subtractive CMYK (cyan,

magenta, yellow, black) model used in print. In this project you'll work with the RGB model.

First, add the `class` `one` to the first marker `div` element

```
<div class="marker one">
      </div>
```

Next, remove the `background-color` property and its value from the `.marker` CSS rule.
Then, create a new CSS rule that targets the class `one` and set its `background-color` property to `red`.

```
.one {background-color:red;}
```

Add the class `two` to the second marker `div`, and the class `three` to the third marker `div`.

```
<div class="marker one">
      </div>
      <div class="marker two">
      </div>
      <div class="marker three">
      </div>
```

Create a CSS rule that targets the class `two` and set its `background-color` property to `green`.

Also, create a separate CSS rule that targets the class `three` and set its `background-color` to `blue`.

```
.two {background-color:green}
.three {background-color: blue}
```

Earlier you learned that the RGB color model is additive. This means that colors begin as black, and change as different levels of red, green, and blue are introduced.

An easy way to see this is with the CSS `rgb` function.

Create a new CSS rule that targets the class `container` and set its `background-color` to black with `rgb(0, 0, 0)`. Black=0 absence of color

```
.container {background-color:rgb(0, 0, 0)}
```

A function is a piece of code that can take an input and perform a specific action. The CSS `rgb` function accepts values, or arguments, for red, green, and blue, and produces a color: `rgb(red, green, blue);`

Each red, green, and blue value is a number from `0` to `255`. `0` means that there's 0% of that color, and is black. `255` means that there's 100% of that color. In the `.one` CSS rule, replace the color keyword `red` with the `rgb` function. For the `rgb` function, set the value for red to `255`, the value for green to `0`, and the value for blue to `0`.

```
.one {

  background-color: rgb(255, 0, 0);

}
```

Notice that the `background-color` for your marker is still red. This is because you set the red value of the `rgb` function to the max of `255`, or 100% red, and set both the green and blue values to `0`.

Now use the `rgb` function to set the other colors.

In the `.two` CSS rule, use the `rgb` function to set the `background-color` to the max value for green, and `0` for the other values. And in the `.three` CSS rule, use the `rgb` function to set the `background-color` to the max value for blue, and `0` for the other values. (brightens the colors

```
.two {

  background-color: rgb(0, 255, 0);

}
```

```
.three {

  background-color: rgb(0, 0, 255);
```


```
}
```

While the red and blue markers look the same, the green one is much lighter than it was before. This is because the `green` color keyword is actually a darker shade, and is about halfway between black and the maximum value for green.

In the `two` CSS rule, set the green value in the `rgb` function to `127` to lower its intensity. (darker=lower)

```
.two {

  background-color: rgb(0, 127, 0);
```


```
}
```

Now add a little more vertical space between your markers and the edge of the `container` element they're in.

In the `.container` CSS rule, use the shorthand `padding` property to add `10px` of top and bottom padding, and set the left and right padding to `0`. This works similarly to the shorthand `margin` property you used earlier.

```
.container {

  background-color: rgb(0, 0, 0); padding:10px; padding-left:0px; padding-right:0px}
```

In the additive RGB color model, primary colors are colors that, when combined, create pure white. But for this to happen, each color needs to be at its highest intensity.

Before you combine colors, set your green marker back to pure green. For the `rgb` function in the `.two` CSS rule, set green back to the max value of `255`.

```
.two {

  background-color: rgb(0, 255, 0);}
```

Now that you have the primary RGB colors, it's time to combine them.

For the `rgb` function in the `.container` rule, set the red, green, and blue values to the max of `255`.

```
.container {

  background-color: rgb(255, 255, 255);

  padding: 10px 0;}
```

Now that you're familiar with secondary colors, you'll learn how to create tertiary colors. Tertiary colors are created by combining a primary with a nearby secondary color.

To create the tertiary color orange, update the `rgb` function in the `.one` CSS rule so that red is at the max value, and set green to `127`.

```
.one {

  background-color: rgb(255, 127, 0);}
```

Notice that, to create orange, you had to increase the intensity of red and decrease the intensity of the green `rgb` values. This is because orange is the combination of red and yellow, and falls between the two colors on the color wheel.

To create the tertiary color spring green, combine cyan with green. Update the `rgb` function in the `.two` CSS rule so that green is at the max value, and set blue to `127`. `.two {`

```
  background-color: rgb(0, 255, 127);}
```

And to create the tertiary color violet, combine magenta with blue. Update the `rgb` function in the `.three` CSS rule so that blue is at the max value, and set red to `127`.

```
.three {
  background-color: rgb(127, 0, 255);}
```

There are three more tertiary colors: chartreuse green (green + yellow), azure (blue + cyan), and rose (red + magenta).

To create chartreuse green, update the `rgb` function in the `.one` rule so that red is at `127`, and set green to the max value.

For azure, update the `rgb` function in the `.two` rule so that green is at `127` and blue is at the max value. And for rose, which is sometimes called bright pink, update the `rgb` function in the `.three` rule so that blue is at `127` and red is at the max value.

Now that you've gone through all the primary, secondary, and tertiary colors on a color wheel, it'll be easier to understand other color theory concepts and how they impact design.

First, in the rules `.one`, `.two`, and `.three`, adjust the values in the `rgb` function so that the `background-color` of each element is set to pure black. Remember that the `rgb` function uses the additive color model, where colors start as black and change as the values of red, green, and blue increase.
i.e.=`rgb(0, 0, 0)`
A color wheel is a circle where similar colors, or hues, are near each other, and different ones are further apart. For example, pure red is between the hues rose and orange.

Two colors that are opposite from each other on the color wheel are called complementary colors. If two complementary colors are combined, they produce gray. But when they are placed side-by-side, these colors produce strong visual contrast and appear brighter.

In the `rgb` function for the `.one` CSS rule, set the red value to the max of `255` to produce pure red. In the `rgb` function for `.two` CSS rule, set the values for green and blue to the max of `255` to produce cyan.

```
.one {
  background-color: rgb(255, 0, 0);}

.two {
  background-color: rgb(0, 255, 255);}
```

Notice that the red and cyan colors are very bright right next to each other. This contrast can be distracting if it's overused on a website, and can make text hard to read if it's placed on a complementary-colored background.

It's better practice to choose one color as the dominant color, and use its complementary color as an accent to bring attention to certain content on the page.

First, in the `h1` rule, use the `rgb` function to set its background color to cyan.

```
h1 {
  text-align: center; background-color: rgb(0, 255, 255);}
```

Next, in the `.one` rule, use the `rgb` function to set the `background-color` to black. And in the `.two` rule, use the `rgb` function to set the `background-color` to red.

```
.one {
  background-color: rgb(0, 0, 0);}

.two {
  background-color: rgb(255, 0, 0);}
```

Notice how your eyes are naturally drawn to the red color in the center? When designing a site, you can use this effect to draw attention to important headings, buttons, or links.

There are several other important color combinations outside of complementary colors

And in the `h1` rule, remove the `background-color` property and value to go back to the default white color.

Now it's time to add other details to the markers, starting with the first one.

In the first marker `div` element, change the class `one` to `red`.

```
<div class="marker red">        </div>
```

Update the `.one` class selector to target the new `red` class.

```
.red {

  background-color: rgb(0, 0, 0);}

.two {

  background-color: rgb(0, 0, 0);}

.three {

  background-color: rgb(0, 0, 0);}
```

Next, change the class `two` to `green` in the second marker `div`, and the class `three` to `blue` in the third marker `div`. (same as above)

Update the CSS class selector `.two` so it targets the new `green` class. And update the `.three` selector so it targets the new `blue` class (same)

```
.red {

  background-color: rgb(255, 0, 0);}

.green {

  background-color: rgb(0, 0, 0);}

.blue {

  background-color: rgb(0, 0, 0);}
```

A very common way to apply color to an element with CSS is with hexadecimal or hex values. While hex values sound complicated, they're really just another form of RGB values.

Hex color values start with a `#` character and take six characters from 0-9 and A-F. The first pair of characters represent red, the second pair represent green, and the third pair represent blue. For example, `#4B5320`.

In the `.green` CSS rule, set the `background-color` property to a hex color code with the values `00` for red, `FF` for green, and `00` blue.

```
.green {

  background-color: #00FF00;}
```

You may already be familiar with decimal, or base 10 values, which go from 0 - 9. Hexadecimal, or base 16 values, go from 0 - 9, then A - F:

`0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F`

With hex colors, `00` is 0% of that color, and `FF` is 100%. So `#00FF00` translates to 0% red, 100% green, and 0% blue, and is the same as `rgb(0, 255, 0)`.

Lower the intensity of green by setting green value of the hex color to `7F`.

```
.green {

  background-color: #007F00;}
```

The HSL color model, or hue, saturation, and lightness, is another way to represent colors.

The CSS `hsl` function accepts 3 values: a number from 0 to 360 for hue, a percentage from 0 to 100 for saturation, and a percentage from 0 to 100 for lightness.

If you imagine a color wheel, the hue red is at 0 degrees, green is at 120 degrees, and blue is at 240 degrees.

Saturation is the intensity of a color from 0%, or gray, to 100% for pure color.

Lightness is how bright a color appears, from 0%, or complete black, to 100%, complete white, with 50% being neutral.

In the `.blue` CSS rule, use the `hsl (HSL)` function to change the `background-color` property to pure blue. Set the hue to `240`, the saturation to `100%`, and the lightness to `50%`.

```
.blue {

  background-color: hsl(240, 100%, 50%);}
```

You've learned a few ways to set flat colors in CSS, but you can also use a color transition, or gradient, on an element.

A gradient is when one color transitions into another. The CSS `linear-gradient` function lets you control the direction of the transition along a line, and which colors are used.

One thing to remember is that the `linear-gradient` function actually creates an `image` element, and is usually paired with the `background` property which can accept an image as a value.

In the `.red` CSS rule, change the `background-color` property to `background`.

```
.red {

  background: linear-gradient(90deg);

}
```

You'll use the `rgb` function for the colors of this gradient.

In the `linear-gradient` function, use the `rgb` function to set the first color argument to pure red.

```
.red {

  background: linear-gradient(90deg, rgb(255, 0, 0));

}
```

You won't see gradient yet because the `linear-gradient` function needs at least two color arguments to work.

In the same `linear-gradient` function, use the `rgb` function to set the second color argument to pure green.

```
.red {

  background: linear-gradient(90deg, rgb(255, 0, 0), rgb(0, 255, 0));}
```

As you can see, the `linear-gradient` function produced a smooth red-green gradient. While the `linear-gradient` function needs a minimum of two color arguments to work, it can accept many color arguments.

Use the `rgb` function to add pure blue as the third color argument to the `linear-gradient` function.

```
.red {

  background: linear-gradient(90deg, rgb(255, 0, 0), rgb(0, 255, 0), rgb(0, 0, 255));}
```

Color-stops allow you to fine-tune where colors are placed along the gradient line. They are a length unit like `px` or percentages that follow a color in the `linear-gradient` function.

For example, in this red-black gradient, the transition from red to black takes place at the 90% point along the gradient line, so red takes up most of the available space:

```
linear-gradient(90deg, red 90%, black);
```

In the `linear-gradient` function, add a `75%` color stop after the first red color argument. Do not add color stops to the other colors arguments.