CSS Example

by Sophia

WHAT'S COVERED

In this lesson, you will see an example of adding styling to the homepage of a website. We will add styling but also the necessary CSS components to make the page responsive to larger screens. We will cover how to stylize the different sections of a page and how they can be adjusted for larger screens.

Specifically, this lesson will cover the following:

1. Responsive Web Design Example

Recall that in our last example, we set up a webpage with the appropriate HTML semantic elements and divisions to prepare the page to be stylized using CSS. At this point, we will focus on the following:

1. Stylizing the page overall for the mobile layout
2. Adding media queries and CSS for the desktop layout
3. Adjusting any styling issues in the desktop layout

After the page has been stylized, we will use CSS flexbox to arrange the content for our mobile-first approach, and then we will add the media queries to manipulate the layout for larger screens. Let us tackle the different semantic sections one at a time. Then, we will move on to media queries.

1a. Header

Before we get to the header, we will want to set up a few mandatory styles for the page overall, such as the body and the default font size for the HTML element for rem units. First, we consider the universal selector where we adjust the box-sizing property to border-box. Remember that this allows us to add a border to an element without affecting the amount of space taken up by the element:

EXAMPLE CSS universal selector

```
* { box-sizing: border-box; }
```

Next, we consider the HTML element, where we can set the font size for relative units. This allows us to use simple rem values such as 2.4rem for a font size of 24px:

EXAMPLE CSS html selector preparation for responsive web design

```
html { font-size: 62.4%; }
```

Finally, we use the body selector to set the background color for the entire page.

EXAMPLE CSS body selector and background color

```
body { background-color: #FFF8F6; }
```

Now that we have that out of the way, we can begin working on styling the header section of the page. This section will include the logo image; the mandatory <h1> element, which will hold the title of the company; and the navigation menu. Additionally, the entire header section will have a background and font color different from the rest of the page's color. The HTML for the header section is as follows:

EXAMPLE Header HTML

```
 <header>
    <div id="header-upper">
      <a href="index.html">
        <img src="WEBDEV336a_c.png" alt="Good Harvest Bakery Logo" id="logo">
      </a>
      <h1 id="client-name">Good<br>Harvest<br>Bakery</h1>
    </div>

    <nav>
       <ul class="navbar-list">
        <li><a href="index.html">Home</a></li>
        <li><a href="gallery.html">Gallery</a></li>
        <li><a href="about.html">About Us</a></li>
        <li><a href="community.html">Community</a></li>
      </ul>

      <button class="navbar"><i class="fa-solid fa-bars"></i>
        <ul class="navbar-list-mobile">
          <li><a href="index.html">Home</a></li>
          <li><a href="gallery.html">Gallery</a></li>
          <li><a href="about.html">About Us</a></li>
          <li><a href="community.html">Community</a></li>
        </ul>
      </button>
    </nav>

  </header>
```

The image tag for the logo and the h1 elements are given id values of “logo” and “client-name,” respectively, for styling. Next, we include the <nav> semantic tag, which contains two unordered lists containing the navigation hyperlinks. The first is a simple list with a class value of “navbar-list.” The second is a button with the class value of “navbar.” Inside the navbar button, we included a Font Awesome hamburger menu icon using <i> and an unordered list containing the navigation links with the class value of “navbar-list-mobile.”

KEY CONCEPT

The button tag is needed for the dropdown navigation menu to function properly on mobile devices. If you use a division, it will work on a desktop, but it will not operate on mobile devices.

Next, we can add the CSS to stylize the header section, hide the first unordered navigation list, and add the hamburger menu dropdown menu. Remember that hamburger menus are often used on mobile sites to save valuable space by hiding away the navigation menu until it is needed. We will start with the style rules for the header itself, the logo, and the <h1> title; then, we will tackle the mobile navigation menu.

EXAMPLE CSS for the header, logo, and <h1> title

```
header {
  background-color: #B7D7BB;
  display: flex;
  flex-direction: column;
  justify-content: space-between;
  align-items: center;
  padding: 10px 20px;
}

#header-upper {
  display: flex;
  align-items: center;
  flex-direction: row;
}

#logo {
  width: 100%;
  margin-right: auto;
}

h1#client-name {
  font-size: 3.6rem;
  color: #cc6d4d;
  text-align: left;
  flex: 1;
}
```

The header style rule itself sets the background color for the header and converts the container into a flex container using display: flex. The two direct children of the header element are a division for the logo and <h1> elements and then the nav section. We want the logo and the <h1> header to stay together, so they are contained within the division. Furthermore, we want to be able to rearrange the nav menu's position to either be below the logo and <h1> for mobile devices or to the right for desktops. We can use a media query to change the flex-direction for the header, thus changing the location of the nav menu relative to the logo and <h1>.

The container for the logo and the <h1>, "header-upper," also uses flexbox in order to easily arrange the items without a lot of extra CSS styles.

KEY CONCEPT

Flexbox is not just ideal for organizing and arranging multiple siblings; it can also be used to conveniently position a simple set of items, such as the logo and the <h1> title. However, one thing that is always challenging to remember with flexbox is which one properly handles vertical alignment and which one handles horizontal alignment.

Flexbox Property	Description
justify-content	Horizontal alignment: flex-start, flex-end, center, space-around, space-between, space-evenly, etc.
align-items	Vertical alignment: baseline, center, flex-start, flex-end, and stretch
align-content	Position and spacing between rows when using flex-wrap: center, flex-start, flex-end, space-around, space-between, and stretch

Next is the logo id selector, in which case we make the logo take up 100% of its available space. This space is shared with the h1 title because both of them are in the same flexbox container.

Next, we apply styles to the <h1> heading to adjust the color, alignment, and size, and we give it the flex-item property of "flex" and a value of 1. This helps ensure that the <h1> takes up the available space.

Finally, let us take a look at the default styles for the navigation menu. Remember we have two navigation menus, a basic unordered list, and an unordered list wrapped in a button. Here is where we will stylize the actual navigation hyperlinks. Additionally, the basic unordered list will be hidden by default for mobile devices.

EXAMPLE CSS for mobile navigation

```
.navbar a,
.navbar-list a {
  text-decoration: none;
  color: #cc6d4d;
```

```
    margin: 0px;
    font-size: 18px;
    display: inline-block;
    padding: 5px;
  }

  .navbar a:hover,
  .navbar-list a:hover {
    /* Change the background color on hover */
    background-color: #eba27a;
    color: #fff;
  }

  .navbar {
    position: relative;
    margin: 0 auto;
    display: inline-block;
    height: 50px;
    width: 50px;
  }

  nav i {
    font-size: 30px;
    color: #cc6d4d;
  }

  .navbar-list {
    display: none;
    list-style: none;
  }

  .navbar-list li {
    display: inline;
  }



  .navbar-list-mobile {
    list-style: none;
    display: none;
    position: absolute;
    margin: 0;
    padding: 20px;
    background-color: #f1f1f1;
    width: 300px;
    box-shadow: 0px 8px 16px 0px rgba(0, 0, 0, 0.2);
    border-radius: 5px;
    z-index: 1;
    top: 45px;
    left: -140px;
  }
  .navbar:hover .navbar-list-mobile {
    display: block;
  }
```

The first two style rules in the example above stylize the actual anchor elements, the hyperlinks, according to the design plan, including a hover color-change effect. We do this for both menus. The next style rule for ".navbar" configures the visible button for the hamburger menu, and the rule for ".navbar i" simply formats the Font Awesome fa-bars symbol, which serves as our hamburger menu icon.

The next rule, ".navbar-list," sets the list-style property to none and hides the menu using display: none. We hide this menu because it will only be visible on desktop screens. The rule right after for ".navbar-list li" changes the list items' display property to inline, which positions the items horizontally as opposed to the default vertical stack.

The final two rules are used to stylize the dropdown menu and create the dropdown functionality. This way, the actual menu is not visible until the user taps or hovers over the hamburger menu. The ".navbar-list-mobile" style rule points to the unordered list contained within the hamburger button and will appear as a floating section of content. As such, we need to add height and width, a background color, and padding for the dropdown menu that will appear. Additionally, in order for the CSS dropdown menu technique to work, we need to set the element's position attribute to absolute, give it a z-index of 1 so it floats above the page content, and position the dropdown menu using the top and left attributes. By making this element's position absolute, it will be relative to its parent's upper left corner, that is, the upper left corner of the button. Setting a value of 45 pixels to top moves the menu down to just below the button, and setting a value of -140 pixels to left moves the menu to the left of the button so that it appears centered under it.

The final rule creates the trigger that makes the menu appear. This is a compound selector that says that when the mouse hovers over the .navbar button, the .navbar-list-mobile element will have a display value of block. This creates the dropdown effect and displays the navigation menu when the user hovers over or taps on the hamburger menu button.

To make this header responsive for desktops, we will use a media query that will swap the visibility of the two menus, hiding the hamburger menu and making the plain unordered list menu visible. The media query will do more for other sections, but for now we just focus on the header section.

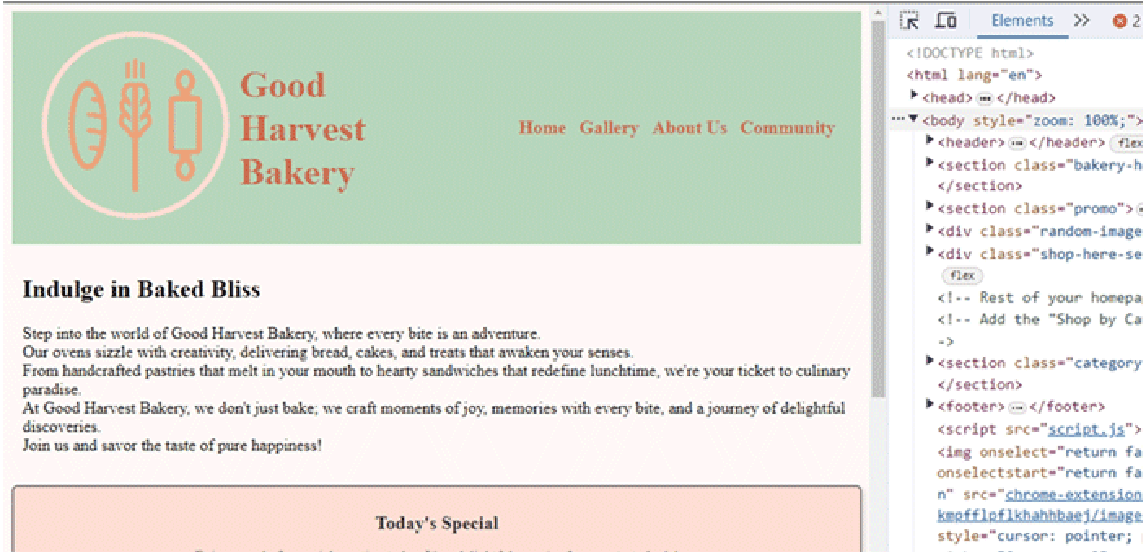
EXAMPLE CSS media query for desktops

```
@media all and (min-width: 800px) {
  header {
    flex-direction: row;
  }

  .navbar-list {
    display: inline-block;
  }

  .navbar {
    display: none;
  }
}
```

The media query above activates its contained style rules when the user's viewport width reaches the minimum threshold of 800 pixels and above. When the user's viewport is wider than 800 pixels, we switch the header's flex-direction to row so that the logo and h2 group will sit to the left of the navigation menu instead of above it. Then, the ".navbar-list" rule makes the plain unordered list menu visible by setting display to inline-block and hides the mobile menu button ".navbar" by setting its display to none. Now, if you open the site in a browser and resize the screen, you should see the two menus swapping places when the width passes the 800-pixel threshold.

You can see the width of your viewport in pixels by opening the Google Chrome Developer Tools and sliding the divider back and forth.

EXAMPLE



Indulge in Baked Bliss

Step into the world of Good Harvest Bakery, where every bite is an adventure.
Our ovens sizzle with creativity, delivering bread, cakes, and treats that awaken your senses.
From handcrafted pastries that melt in your mouth to hearty sandwiches that redefine lunchtime, we're your ticket to culinary paradise.
At Good Harvest Bakery, we don't just bake; we craft moments of joy, memories with every bite, and a journey of delightful discoveries.
Join us and savor the taste of pure happiness!

Today's Special

```
<!DOCTYPE html>
<html lang="en">
 ▶<head> … </head>
…▼<body style="zoom: 100%;">
   ▶<header> … </header> flex
   ▶<section class="bakery-h
    </section>
   ▶<section class="promo">
   ▶<div class="random-image
   ▶<div class="shop-here-se
     flex
    <!-- Rest of your homepa
    <!-- Add the "Shop by Ca
    ->
   ▶<section class="category
    </section>
   ▶<footer> … </footer>
    <script src="script.js">
    <img onselect="return fa
    onselectstart="return fa
    n" src="chrome-extension
    kmpfflpflkhahhbaej/image
    style="cursor: pointer;
```

1b. Shop Here and Categories

The next two sections on the page are the perks and the "Shop by Category." The perks section is a division that contains an h2 heading and a division containing three additional divisions. The three divisions are the containers for the perks, which include an h3 heading and a paragraph.

EXAMPLE HTML content for the perks content

```
<div class="shop-here-section">

    <h2>Why shop here...</h2>

    <div class="perk-box">
      <div class="perk">
        <h3>Fresh Ingredients</h3>
        <p>We use the freshest ingredients to create our delicious baked goods.</p>
      </div>
      <div class="perk">
        <h3>Artisanal Craftsmanship</h3>
        <p>Our products are crafted with artisanal expertise and care.</p>
      </div>
      <div class="perk">
        <h3>Sustainability</h3>
        <p>We are committed to sustainable practices for a better environment.</p>
      </div>
    </div>

  </div>
```

For this section, we will stylize the boxes with rounded corners and a border. Only the division containing the three perk boxes with the class value of “perk-box” will need to be a flex container. This way, we can change the orientation of the perk boxes from vertical to horizontal.

EXAMPLE CSS for the “shop here” section and perk boxes

```
.shop-here-section {
  background-color: #f7f7f7;
  padding: 20px;
  text-align: center;
  display: flex;
  flex-direction: column;
}

.shop-here-section h2 {
  font-size: 2.4rem;
}

.perk {
  display: inline-block;
  background-color: #fff;
  padding: 10px;
  margin: 2px;
  border: 1px solid #ddd;
  border-radius: 5px;
  text-align: center;
  vertical-align: top;
  width: 90%;
}

.perk-box {
  display: flex;
  flex-direction: column;
  align-items: center;
}

.perk h3 {
  font-size: 1.8rem;
}

.perk p {
  font-size: 1.6rem;
}
```

Next, to make the perk boxes responsive, we will add the following style rule to the media query.

EXAMPLE CSS responsive style for perk boxes

```
.perk-box {
  flex-direction: row;
}
```

Remember that the style rule above will be placed within the curly brackets of the @media query.

The “Shop by Category” section will be slightly different. Since we have so many category boxes and each box contains a single word, we can make use of a two-column (or more) layout by using a grid layout in the media query. For the mobile-first default CSS, we will make a simple vertical stack.

EXAMPLE HTML content for “Shop by Category”

```
  <section class="category-section">
    <h2>Shop by Category</h2>
    <div class="categories">
      <div>Breads</div>
      <div>Cakes</div>
      <div>Pastries</div>
      <div>Cookies</div>
      <div>Pies</div>
      <div>Muffins</div>
    </div>
  </section>
```

As you can see, we have a similar setup. A division for the entire “Shop by Category” section contains an h2 heading and then a list of divisions within its own container. We format the section for mobile devices first as follows:

EXAMPLE Default CSS for “Shop by Category”

```
.category-section {
  width: 100%;
  text-align: left;
  margin-bottom: 20px;
}

.category-section h2 {
  font-size: 2.4rem;
}

.categories {
  width: 100%;
  display: flex;
  flex-direction: column;
  text-align: center;
}

.categories div {
  width: calc(50% - 10px);
  margin: 5px auto;
  background-color: #fff;
  padding: 20px;
  border: 1px solid #ddd;
  border-radius: 5px;
  text-align: center;
  font-size: 1.6rem;
}
```

As you can see from the styles above, we stylize the .category-section, the h2, and then set up the .categories container as a flex container with a column direction. Lastly, we stylize the individual category divisions with the typical styles for this design. Note that the value of width in the last style rule uses a special function named calc(). This is a function that can be used within CSS files to perform basic mathematical operations and calculate values dynamically. In this case, we take 50% of the viewport and subtract 10 pixels from that value; the remaining pixels will be the width of the category divisions.

The last step is to make the categories list responsive by applying a grid layout to the .category class and setting the grid up with two columns. This will be done within the curly brackets of the media query.

EXAMPLE Responsive CSS with a grid layout

```
.categories {
    display: grid;
    grid-template-columns: auto auto;
  }

  .categories div {
    width: 75%;
  }
```

To configure the category container as a grid layout, we provide the display property the value of “grid.” Next, we configure the number and width of the columns by using the grid-template-columns property and giving it two values of “auto auto.” Not only does “auto auto” give us two columns, but it will automatically take up an equal amount of available horizontal space. Lastly, we adjust the width of the category boxes to 75% while the media query is active.

1c. Footer

The last section that we need to tackle is the footer. The footer in this case contains two child divisions. The first division is divided into two divisions: the left-half, which will contain the newsletter signup form, and the right-half, which will contain the social media links. The second division will simply contain the text-based footer navigation menu. The footer will not be a flex container; however, the first child division will be a flexbox in order to allow the media query to reorient the newsletter and social media icons.

EXAMPLE HTML content for the footer

```
<footer>

    <div class="newsletter-section">
      <div class="left-half">
        <h2>Sign up for our Newsletter</h2>
        <input type="email" placeholder="Your email" id="subscribe" required>
        <button id="subscribe-button">Subscribe</button>
      </div>

      <div class="right-half">
        <a href="#"><i class="fa-brands fa-facebook"></i></a>
        <a href="#"><i class="fa-brands fa-instagram"></i></a>
        <a href="#"><i class="fa-brands fa-x-twitter"></i></a>
        <a href="#"><i class="fa-brands fa-youtube"></i></a>
      </div>
    </div>


    <div class="footer-links">
      <a href="index.html">Home</a>
      <a href="gallery.html">Gallery</a>
      <a href="about.html">About Us</a>
      <a href="community.html">Community</a>
    </div>
  </footer>
```

KEY CONCEPT

Notice the use of the pound symbol as the value of the href attribute. This is often done for hyperlinks that do not yet have a URL or page to actually point to. In this case, the social media accounts have not been created (or provided by the client as of yet). However, we still want to mock up the hyperlinks and have them behave as a hyperlink normally would, except that it just does not go anywhere. Why not just leave the value of href empty? Doing so causes some browsers to omit the entire attribute, which renders the hyperlink as normal content and not an actual hyperlink.

From the HTML, you can see that the first division of the footer is the newsletter-section, which is divided into the left-half and right-half divisions. We will only make the newsletter-section a flex container.

EXAMPLE CSS for the footer's first section.

```
footer {
  background-color: #f7f7f7;
  /* Add a gray background color */
}

.newsletter-section {
  display: flex;
  padding: 10px;
  justify-content: space-between;
  align-items: center;
  flex-direction: column-reverse;
}

.left-half {
  width: 100%;
  text-align: center;
}

.left-half h2 {
  font-size: 1.8rem;
}

.right-half {
  width: 100%;
  display: flex;
  justify-content: center;
  align-items: center;
}

.right-half a {
  margin: 0 10px;
  text-decoration: none;
}

.right-half img {
  width: 30px;
  /* Smaller icon size */
}
```

As you can see from the styles above, the first style rule simply adds a gray background color to the entire footer. The second rule creates a flex container for the left-half and right-half divisions. For the mobile-first approach, the default style uses flex-direction: column-reverse to stack the social media icons above the newsletter signup form. We use the reversed column because we want to have the social media icons on the right (after) for larger screens and position them above for smaller screens. This style rule also handles the flex-box alignment by setting values for justify-content and align-items.

Next, the left-half and right-half are stylized as desired, but with a width of 100%. Even though they will shrink as needed by being flexed, we still want each of the flex items (newsletter form and social media icons) to take up as much space as they can.

In looking at the .right-half style rule, you will notice that we have made this a flexbox. We primarily did this to have better and more convenient control over how the icons are arranged.

Now let us look at some of the style rules that will be used not just for the newsletter signup form but also for other form elements.

EXAMPLE CSS for buttons and input fields

```
input[type="submit"],
input[type="reset"],
button {
  background-color: #eba27a;
  color: #fff;
  padding: 10px;
  /* Smaller padding */
  border: none;
  border-radius: 5px;
  cursor: pointer;
  width: 25%;
  /* Narrow the button */
}

input[type="submit"]:hover,
input[type="reset"]:hover,
button:hover {
  background-color: #cc6d4d;
}

input[type="text"],
input[type="email"],
input[type="tel"],
textarea {
  width: 50%;
  padding: 10px;
  border: 1px solid #ccc;
  border-radius: 5px;
  font-family: Arial;
  color: #333;
}
```

First, we give some style to our generic button elements, the submit button, and the reset buttons by giving them white text and rounded corners, specifying the cursor type, and using one of the brand/theme colors as the background. Remember that we can use HTML attributes and their value as a selector using the square brackets, attribute name, and value. Next, we apply a hover effect to the same three button selectors, including the :hover pseudoclass. Finally, we format the input fields, including generic text fields, email, telephone, and textarea elements. Although we currently only have the newsletter signup input field, which is an email input element, we will use the other field types on other pages.

MAKE THE CONNECTION

You will use what you learned in this Challenge to complete Touchstone Task 2.2: Adding Style to the Webpages Using CSS. You may notice that this tutorial did not demonstrate all sections of the webpage. You will take what you have learned here and apply it to the other sections of your project site. The process you use to make the “shop here” section, the categories, and the footer sections responsive will mostly be the same for other responsive sections. You will need to ensure that responsive elements are properly nested within their own container configured for a mobile layout. Then, you will add the desktop layout properties for the responsive container to the media query.

SUMMARY

In this lesson, you saw an **responsive web design example** of adding CSS styling to a website homepage. We examined the styling for each of the sections of the homepage, including the **header**, **shop here, categories**, and **footer**. Additionally, we examined how we use media queries to configure and, thus, rearrange the content for larger screens.