

# LEARN BASIC CSS BY BUILDING A CAFE MENU

## Introduction

CSS tells the browser how to display your webpage. You can use CSS to set the color, font, size, and other aspects of HTML elements.

In this course, you'll learn CSS by designing a menu page for a cafe webpage.

## Step 1:

In this project, you will learn the basics of CSS (Cascading Style Sheets) by building a cafe menu. CSS is the language used to style an HTML document. It describes how HTML elements should be displayed on the screen.

As you learned in the last few steps of the Cat Photo App, there is a basic structure needed to start building your web page. Every HTML document should have a `DOCTYPE` declaration and `html` element. The `DOCTYPE` tells the browser which version of HTML the document is in. And the `html` element represents the root element which contains all other elements.

Example Code:

```
<!DOCTYPE html>
<html lang="en">
<!--all other elements go here-->
</html>
```

Add the `<!DOCTYPE html>` tag, and an `html` element with a `lang` attribute of `en`.

## Step 2:

Add a `head` element within the `html` element, so you can add a `title` element. The `title` element's text should be `Cafe Menu`.

## Step 3:

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 with an attribute named `charset` set to the value `utf-8` to tell the browser how to encode characters for the page.

## Step 4:

To prepare to create some actual content, add a `body` element below the `head` element.

## Step 5:

It's time to add some menu content. Add a `main` element within the existing `body` element. It will eventually contain pricing information about coffee and desserts offered by the cafe.

## Step 6:

The name of the cafe is `CAMPER CAFE`. Add an `h1` element within your `main` element. Give it the name of the cafe in capitalized letters to make it stand out.

## Step 7:

To let visitors know the cafe was founded in 2020, add a `p` element below the `h1` element with the text `Est. 2020`.

## Step 8:

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.

## Step 9:

Create an `h2` element in the `section` element and give it the text `Coffee`.

## Step 10:

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.

## Step 11:

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

Example Code:

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

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

## Step 12:

In the previous step, you used a `type selector` to style the `h1` element. Center the content of the `h2` and the `p` elements by adding a new type selector for each one to the existing `style` element.

## Step 13:

You now have three type selectors with the exact same styling. You can add the same group of styles to many elements by creating a list of selectors. Each selector is separated with commas like this:

Example Code:

```
selector1, selector2 {  
    property: value;  
}
```

Delete the three existing type selectors and replace them with one selector list that centers the text for the `h1`, `h2`, and `p` elements.

## Step 14:

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.

### Step 15:

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.

### Step 16:

Now you need to link the `styles.css` file, so the styles will be applied again. Inside the `head` element, add a `link` element. Give it a `rel` attribute with the value of `"stylesheet"` and a `href` attribute with the value of `"styles.css"`.

### Step 17:

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:

Example Code:

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

## Step 18:

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.

## Step 19:

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 able to read the text.

## Step 20:

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`.

Inside the opening `div` tag, add the `id` attribute with a value of `menu`.

## Step 21:

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.

You can use the `id` selector to target a specific element with an `id` attribute. An `id` selector is defined by placing the hash symbol `#` directly in front of the element's `id` value. For example, if an element has the `id` of `cat` then you would target that element like this:

Example Code:

```
#cat {  
  
    width: 250px;  
  
}
```

Use the `#menu` selector to give your element a width of `300px`.

## Step 22:

Comments in CSS look like this:

Example Code:

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

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

## Step 23:

Now use the existing `#menu` selector to set the background color of the `div` element to be `burlywood`.

## Step 24:

Now it's easy to see that the text is centered inside the `#menu` element. Currently, the width of the `#menu` 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`).

## Step 25:

Next, you want to center the `#menu` 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 `#menu` element within the `body` element.

## Step 26:

So far you have been using type and id selectors to style elements. However, it is more common to use a different selector to style your elements.

A class selector is defined by a name with a dot directly in front of it, like this:

Example Code:

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

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

## Step 27:

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



## Step 28:

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)`.

## Step 29:

It's looking good. 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.

## Step 30:

`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`.

## Step 31:

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.

Example Code:

Caramel Macchiato 3.75

Pumpkin Spice 3.50

Hazelnut 4.00

Mocha 4.50

### Step 32:

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.

### Step 33:

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

### Step 34:

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.

### Step 35:

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

### Step 36:

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. To do that, start by adding a `class` attribute with the value `item` to the first `article` element under the `Coffee` heading.

### Step 37:

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:

Example Code:

```
.item p { }
```

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

### Step 38:

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.

### Step 39:

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.

### Step 40:

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, use the backspace key on your keyboard to move the `p` element with the class `price` next to the `p` element with the class `flavor` so that they are on the same line in the editor. Make sure there is no space between the two elements.

### Step 41:

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

### Step 42:

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.

### Step 43:

Next, position the other `p` elements to be on the same line with no space between them.

### Step 44:

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

### Step 45:

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%.

### Step 46:

You will come back to styling the menu in a few steps, but for now, go ahead and add a second `section` element below the first for displaying the desserts offered by the cafe.

### Step 47:

Add an `h2` element in the new section and give it the text `Desserts`.

### Step 48:

Add an empty `article` element under the `Desserts` heading. Give it a `class` attribute with the value `item`.

### Step 49:

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

## Step 50:

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` element's `class` attribute.

## Step 51:

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.

The CSS rule for the `flavor` class already sets the properties you want. Add the `dessert` class as an additional selector for this CSS rule.

## Step 52:

Below the dessert you just added, add the rest of the desserts and prices using three more `article` elements, each with two nested `p` elements. Each element should have the correct dessert and price text, and all of them should have the correct classes.

Example Code:

Cherry Pie 2.75

Cheesecake 3.00

Cinnamon Roll 2.50

### Step 53:

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`.

### Step 54:

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

### Step 55:

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`.

### Step 56:

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.

### Step 57:

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.

### Step 58:

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 using a single selector so that these elements' text use `Impact` font.

### Step 59:

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.

### Step 60:

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

### Step 61:

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



## Step 62:

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`.

## Step 63:

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

## Step 64:

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`.

## Step 65:

Add a second `p` element below the one with the link and give it the text `123 Free Code Camp Drive`.

## Step 66:

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

First, add an `hr` element between the `p` element with the class `established` and the first `section` element.

## Step 67:

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

Change the height of the `hr` element to be `3px`.

## Step 68:

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

## Step 69:

Notice the gray 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.

## Step 70:

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`.

### Step 71:

Go ahead and add another `hr` element between the `main` element and the `footer` element.

### Step 72:

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.

### Step 73:

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

Use the existing selector that targets all the `p` elements nested in elements with the class named `item` and set their top and bottom margin to be `5px`.

### Step 74:

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`.

### Step 75:

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.

## Step 76:

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

## Step 77:

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`.

## Step 78:

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

## Step 79:

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.

## Step 80:

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 `gray` when a user has visited the link.

### Step 81:

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.

### Step 82:

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.

### Step 83:

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.

### Step 84:

The menu text `CAMPER CAFE` has a different space from the top than the address's space 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.

## Step 85:

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`.

## Step 86:

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.

## Step 87:

Now apply the `address` class to the `p` element containing the street address `123 Free Code Camp Drive`.

## Step 88:

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`.

## Step 89:

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.

## Step 90:

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`.

## Step 91:

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.