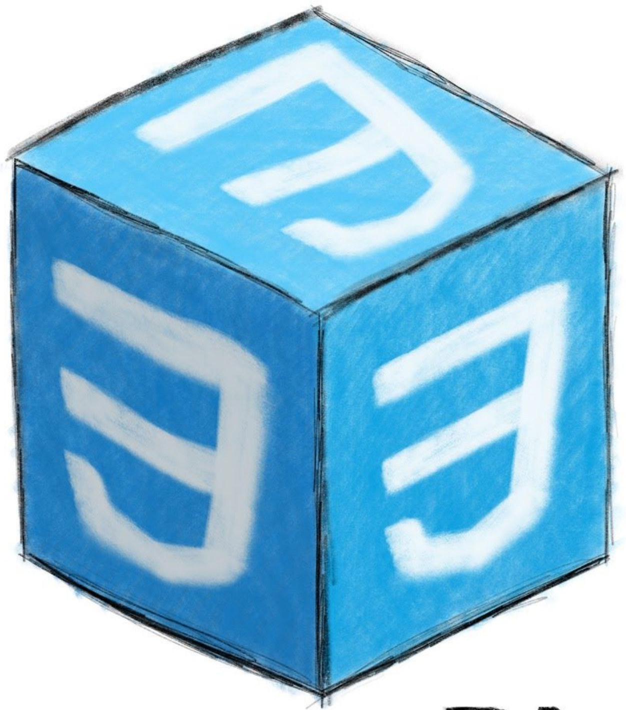
# CSS



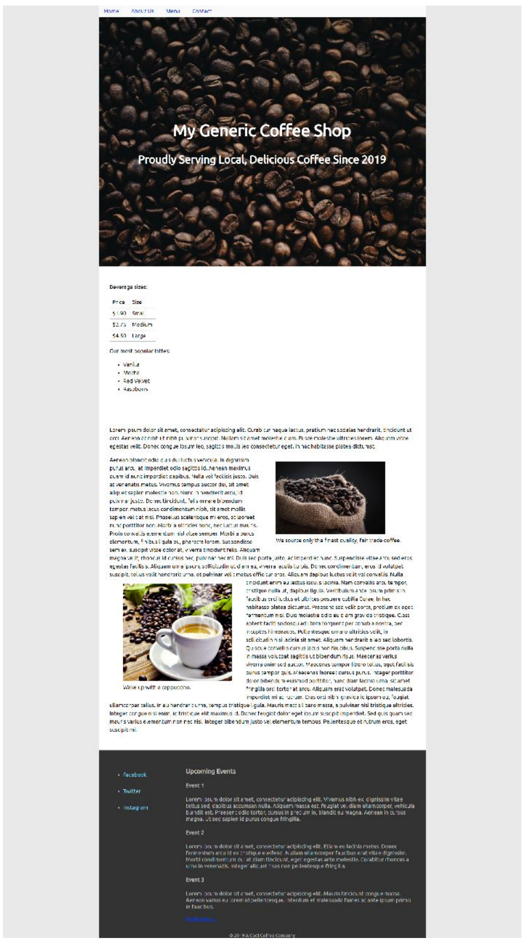
# Beginner Blocks

CSS Beginner Blocks
Version 1.0
Written by Jared York

# Introduction

*CSS Beginner Blocks* is a course designed to help you learn practical CSS that you would read and write everyday.  What is CSS you might ask?  CSS is the abbreviation for *Cascading Style Sheets*.  The language is used everywhere on the Web.  Anywhere you see colored sections, text, spacing, margins, and all sorts of other things spicing up the appearance of a website, it's extremely likely the site is using CSS to affect those items.

Enough with the introduction, let's just dive in!  To start, download the starter files here.  Extract the ZIP file, and place the *cssbeginnerblocks* folder where you want your project to live.  Personally, I will be add the folder to where local web server serves files on my computer.  For our purposes a web server is not necessary, but websites are stored on a server when deployed in production environments.  If you open the index.html file in your browser, you should see a pretty bland (or minimalist, depending on your perspective) page.
In this course we will be spicing up a generic coffee shop website:



Beverage sizes:

| Price | Size |
|---|---|
| $1.90 | Small |
| $2.75 | Medium |
| $4.50 | Large |

Our most popular lattes:

- Vanilla
- Mocha
- Red Velvet
- Raspberry

Lorem ipsum dolor sit amet, consectetur adipiscing elit. [illegible]

There will not be a review of the HTML in this course. It is expected that you have grasped the basics of HTML already. If not, no worries, just download my course, “HTML Beginner Blocks”, available here to bring you up to speed. When you’re ready, fantastic, let’s proceed!

To begin, we will need to add a couple tags to the <*head*> element of the index.html file. On our page, we will be loading a Google Web Font. Specifically, we will be utilizing the Ubuntu font. Just under the title tags, add the following line:

```
<link href=”https://fonts.googleapis.com/css?family=Ubuntu” rel=”stylesheet”>
```

We will be defining our CSS styles in an external file, and we will also use the link element to link the CSS file to our page. Below the above line, add another link element:

```
<link rel=”stylesheet” type=”text/css” href=”css/styles.css”>
```

Next, we have to go back to talking about HTML attributes. For CSS, there are two HTML attributes we care about: *id*, and *class*. To start, the *id* attribute will allow you to reference a specific element with a CSS rule. The *class* attribute also allows you to specify an ID of sorts for referencing in CSS, but it allows multiple elements to share the same class. The *id* attribute takes precedence over the *class* attribute when the browser parses CSS. If you take a look in your HTML file, you should see many elements containing the *class* attribute. Next, open the *styles.css* file which can be found in the *css/* directory.

One problem with styling websites is accounting for how each browser renders HTML and CSS. Many websites include some sort of “reset” file which helps standardize the appearance of their website across all browsers. However for the sake of this course, we will just add a bit of basic code to our styles.css file:

```
html, body {
	margin: 0;
	padding: 0;
}
```

Now let’s start add a bit of color (though not much). Add the following style after the above code:

```
body {
	background-color: #e9e9e9;
}
```

The above code will turn the body of the website to a light grey color. It will set the background of the whole site.

Most all websites contain some sort of main “wrapper” element within the *body* tags of the HTML file. Many websites only allow the wrapper to grow a width like 960px or 1024px. Then the wrapper is usually horizontally centered on the page. Essentially, this centers all of the content and gives the appearance of margins on both sides of the content. In our styles.css file, let’s add the following code for our wrapper:

```
.wrap {
	background: white;
	max-width: 960px;
	margin: 0 auto;
}
```

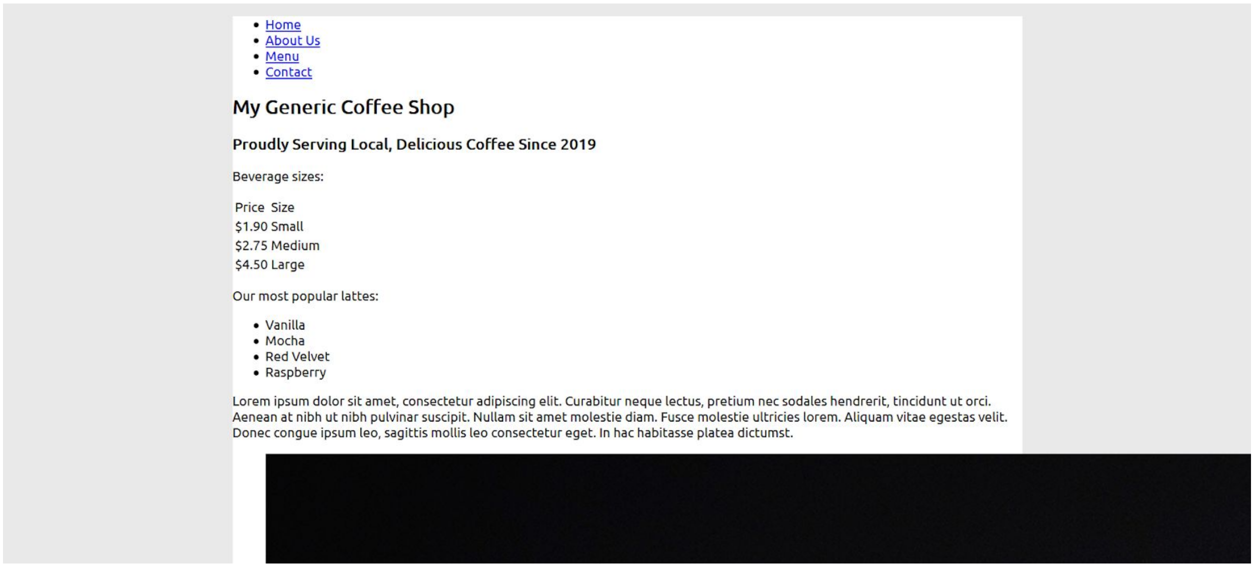
We can also set the font of our website using the *font-family* property. With this property, you can list multiple font families in the order you want them to fall back on. In other words, if the font you chose couldn’t be loaded for some reason, load the next one specified, etc. Let’s put this into practice by adding the following to the bottom of our wrapper style:

```
font-family: ‘Ubuntu’, sans-serif;
```

Now, if the Ubuntu font couldn’t be loaded (for example if the user was offline and can’t access Google Web Fonts), then it would fall back to whatever default sans serif font is installed on the system. Obviously, you could add more fonts to the property like so:

```
font-family: ‘Ubuntu’, Helvetica, Verdana, sans-serif;
```

At this point if you navigate to the index.html file in your browser, you should see the following result:



- Home
- About Us
- Menu
- Contact

My Generic Coffee Shop

Proudly Serving Local, Delicious Coffee Since 2019

Beverage sizes:

| Price | Size |
|---|---|
| $1.90 | Small |
| $2.75 | Medium |
| $4.50 | Large |

Our most popular lattes:

- Vanilla
- Mocha
- Red Velvet
- Raspberry

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Curabitur neque lectus, pretium nec sodales hendrerit, tincidunt ut orci. Aenean at nibh ut nibh pulvinar suscipit. Nullam sit amet molestie diam. Fusce molestie ultricies lorem. Aliquam vitae egestas velit. Donec congue ipsum leo, sagittis mollis leo consectetur eget. In hac habitasse platea dictumst.

We should now see our new font! Great! Notice if we look at the blue links at the top, we will want to arrange all those links into a row to form our navigation bar. To do so, add the following to the styles.css file:

```
.nav-main {
        background: #fafafa;
        margin: 0;
        padding: 0;
}
```

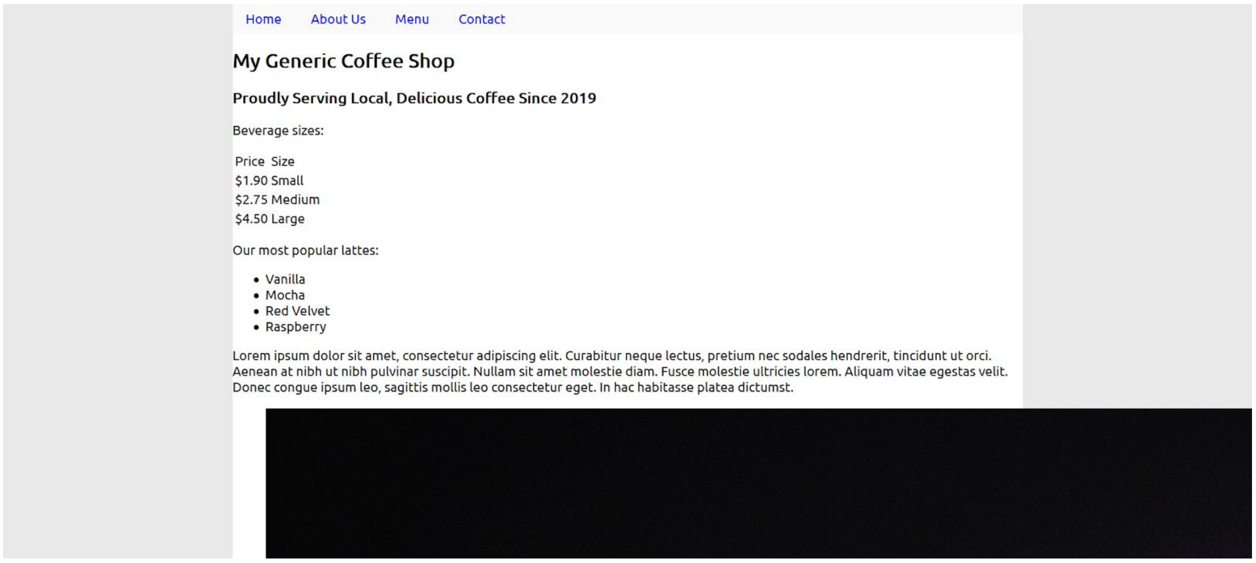
The list items will also have to be styled in order to allow them to be side by side with each other. Add the following after the above code:

```
.nav-main li {
        display: inline;
}
```

Finally, there are some properties we want to give to the navigation links themselves. Add the following after the .nav-main li rule:

```
.nav-main a {
        display: inline-block;
        padding: 8px 16px 8px 16px;
        text-decoration: none;
}
```

If we take another look at the index.html file in the browser, we should now see the navigation bar coming together:



My Generic Coffee Shop

Proudly Serving Local, Delicious Coffee Since 2019

Beverage sizes:

| Price | Size |
|---|---|
| $1.90 | Small |
| $2.75 | Medium |
| $4.50 | Large |

Our most popular lattes:

- Vanilla
- Mocha
- Red Velvet
- Raspberry

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Curabitur neque lectus, pretium nec sodales hendrerit, tincidunt ut orci. Aenean at nibh ut nibh pulvinar suscipit. Nullam sit amet molestie diam. Fusce molestie ultricies lorem. Aliquam vitae egestas velit. Donec congue ipsum leo, sagittis mollis leo consectetur eget. In hac habitasse platea dictumst.

If we look back at the image of what the final result should be, you will notice a large image showing coffee beans and the heading, “My Generic Coffee Shop.” In HTML this is represented as a <*div*> element with the class, “site-header-wrapper.” Add a new style rule to the bottom of our styles.css file:

```
.site-header-wrapper {
        display: flex;
        justify-content: center;
        align-items: center;
        color: white;
        min-height: calc(100vh - 32px);
        background-image: url(../images/jarrod-winkler-1348990-unsplash.jpg);
        background-size: cover;
        text-align: center;
}
```

Note how we’re setting the *display* property. By setting the display property to *flex*, we will be utilizing CSS flexbox for this element. CSS flexbox is quite useful for more easily aligning child elements. The next property we set, *justify-content* will align child elements horizontally. In this case, we will center align child elements. The next property *align-items* will in our case, help with centering child elements vertically. Because of this, we’re able to center the heading, “My Generic Coffee Shop,” and the subheading, “Proudly Serving Local, Delicious Coffee Since 2019.” The next property, *color* allows us to set the text color of child elements. For our use, we want to set both headings to be white. Since the headings are children within elements of the class *site-header-wrapper*, they will inherit this color. Next we're setting the *min-height* property. This will allow the image to take up almost all of the user's screen except for the navigation bar (roughly 32 pixels in height) at the top. We can actually have the browser calculate the value of a property adding the functional notation, *calc* . After adding a parenthesis to *calc*, you can add an expression to evaluate. In our case, the expression we have added is *100vh - 32px*. The unit *vh* stands for virtual height. Any number with the *vh* unit after it will represent that percentage height of the screen (hopefully that makes sense). For example if you add 5vh, that represents 5% of the height of the user's screen. Since we've added *100vh*, that represents the total height of the user's screen. In effect, we're calculating the height of the user's screen subtracting 32 pixels (to make room for the navigation bar.) So once the user starts scrolling down, they will see the rest of the homepage. Next, we're setting the *background-image* property. We can define a background image by adding another functional notation, *url*. There are more properties we can add to adjust our background image. We have specified the *background-size* property allowing us to change the background size. In our case, we will want the background image to cover the entire *site-header-wrapper* element without stretching the background. Using the *cover* value for the *background-size* property will allow the background to scale up proportionally. The next property *text-align* will center the text in child elements, since we've assigned the *text-align* property to *center*. With that, we should be finished messing around with the element used as the header wrapper. The next thing we want to do is change the font size of the two headings inside our header wrapper. Add the following code to change the font sizes for our header headings accordingly:

```
.site-header-text h2 {
	font-size: 3em;
}

.site-header-text h3 {
	font-size: 2em;
}
```

At this point, we can now focus on the content of our page. There are a few housekeeping styles we have to add first. The first is adding padding to articles, as well as increasing the line height to make text more readable inside <*article*> elements. Add the following after the previous styles:

```
article {
	padding: 32px;
	line-height: 1.5em;
}
```

We will also want to constrain the size of article images to stay within the width of the wrapper element:

```
article img {
	max-width: 100%;
}
```

I do want to point out, whenever you specify a style with two style selectors with a space in between, it will look for the second selector within the first. For example, in the above code, the parser will look for any *<img>* elements within any *<article>* elements. This can be super useful and works great in a pinch (or if you're feeling lazy like me, and don't like specifying more classes than you need. ;) )

Now what if we want some images left aligned, and some images right aligned? No problem! Let's add two style rules for classes associated with both left aligned images and right aligned images. Add the following style rules:

```
figure.float-left {
	float: left;
}

figure.float-right {
	float: right;
}
```

Perhaps I should quickly explain what the *<figure>* element is before explaining why there's no space between CSS selectors. The *<figure>* element is pretty much just a container for an image and corresponding caption. Now, if you have two selectors side by side without a space, in this case, the parser looks for figures with the class specified directly after. In the above code, the parser will look for figure elements with the *float-left* and *float-right* classes. Next, we will want to make the footer look better. To do so, add the following style rule:

```
.site-footer {
	color: #c1c1c1;
	background: #333;
}
```

The really neat thing about a lot of CSS properties is there are shortened versions in many cases. Instead of using *background-color* to specify background color, you can simply use the *background* property as we do in the code above.

Taking a look back at the image displaying the final result, you will see the social media links listed on the left side of the footer, and upcoming events on the right side. We can utilize flexbox again to put these two sections next to each other in the footer. Add the following to style the *wrap-footer* class:

```
.wrap-footer {
        display: flex;
        flex-direction: row;
        justify-content: space-around;
        padding: 32px;
}
```

To style the footer links section, add the following code:

```
.site-footer-links {
        width: 25%;
        line-height: 3em;
}

.site-footer-links a {
        Color: #85dcff;
        text-decoration: none;
}

.site-footer-events {
        width: 75%;
}
```

The above code will ensure the links section is only 25% the width of the footer container, and we also increase the line-height to 3em to space out the links. To finish up the main style rules of our stylesheet, we'll add some more miscellaneous styles. While not too noticeable, the copyright text on the bottom of website footers is pretty important. However, this text obviously isn't as important to the reader as the main content. Because of this, we will decrease the font size of the copyright text a little bit:

```
.copyright {
        margin: 0;
        padding: 0.5me;
        font-size: 0.85em;
        text-align: center;
}
```

If we view our index.html file again, you'll see the article images span across the width of the site wrapper. This defeats the purpose of images with the left or right align classes. To fix this, we can add a style for the class *img-small*:

```
.img-small {
        max-width: 320px;
```

```
	width: 100%;
	height: auto;
}
```

Now, any element with the *img-small* class will constrain the width to a maximum of 320 pixels. If the screen is somehow less than 320 pixels, it will adopt the width of the wrapper.

May have also noticed the table containing the beverage sizes and prices in the beginning preview image. Each cell in that table has a light gray border around it. To add this, let's add styles affecting the <*table*> element and the <*td*> element:

```
table {
	border-collapse: collapse;
}

td {
	padding-top: 4px;
	padding-right: 8px;
	padding-bottom: 4px;
	padding-left: 8px;
	border: 1px solid #b1b1b1;
}
```

Okay, the site is styled now. But what about mobile? With nearly 90% of people owning smartphones in developed companies according to the ITU in 2018, mobile friendly websites are more critical than ever.

How do you add this compatibility you may ask? One way to do so is utilizing CSS media queries. Add the following to the bottom of your styles.css files:

```
@media (max-width: 640px) {

}
```

Any style rules inside the curly braces will be used up to a screen width of 640 pixels. These styles will override any styles defined previously if the screen is less than the *max-width* provided. On mobile devices, we want the wrapper to span the entire width of the screen. To do so, add the following within the media query:

```
.wrap {
	width: 100%;
}
```

Also, we want to change the navigation bar items to be listed vertically instead of being displayed in a row. To do so, change the display property to block on list item elements within the element with the *nav-main* class:

```
.nav-main li {
	display: block;
}
```

We also want to decrease the article padding since there isn't much room to display actual content on mobile devices:

```
article {
	padding: 16px;
}
```

We also want to change our header image *min-height* property for small screen sizes. Add the following to make the header image account for the height of the navigation links section on the top of the page:

```
.site-header-wrapper {
	min-height: calc(100vh - 188px);
}
```

Let's also make the footer stack links section and the upcoming events section instead of displaying them side by side:

```
.wrap-footer {
	flex-direction: column;
	padding: 16px;
}
```

Finally, let's set the width of the upcoming events section to the screen width on mobile:

```
.site-footer-events {
	width: 100%;
}
```

If we navigate to our page again, we should see a result matching the image shown in the beginning of this course! You now know the fundamentals of utilizing CSS with your HTML. Fantastic!

The next natural step is to learn JavaScript if you haven't already. I have written the course, "JavaScript Beginner Blocks," to help developers in this effort. That course can be downloaded for free here.

You can find the rest of my courses here. If you have any questions, comments, and feedback at all, I'd love to hear it! I'm always happy to help other developers out.

If you found this course valuable, and would like news regarding future courses I release, be sure to fill out the form.