**Lesson 1**

**Building Your First Web Page**

If you can, imagine a time before the invention of the Internet. Websites didn’t exist, and books, printed on paper and tightly bound, were your primary source of information. It took a considerable amount of effort—and reading—to track down the exact piece of information you were after.

Today you can open a web browser, jump over to your search engine of choice, and search away. Any bit of imaginable information rests at your fingertips. And chances are someone somewhere has built a website with your exact search in mind.

Within this book I’m going to show you how to build your own websites using the two most dominant computer languages—HTML and CSS.

Before we begin our journey to learn how to build websites with HTML and CSS, it is important to understand the differences between the two languages, the syntax of each language, and some common terminology.

￼￼￼￼￼￼￼￼￼￼￼￼￼￼￼￼

## What Are HTML & CSS?[#what-are-html-and-css](http://learn.shayhowe.com/html-css/building-your-first-web-page/#what-are-html-and-css)

HTML, HyperText Markup Language, gives content structure and meaning by defining that content as, for example, headings, paragraphs, or images. CSS, or Cascading Style Sheets, is a presentation language created to style the appearance of content—using, for example, fonts or colors.

The two languages—HTML and CSS—are independent of one another and should remain that way. CSS should not be written inside of an HTML document and vice versa. As a rule, HTML will always represent content, and CSS will always represent the appearance of that content.

With this understanding of the difference between HTML and CSS, let’s dive into HTML in more detail.

## Understanding Common HTML Terms[#common-html-terms](http://learn.shayhowe.com/html-css/building-your-first-web-page/#common-html-terms)

While getting started with HTML, you will likely encounter new—and often strange—[terms](http://www.scriptingmaster.com/html/HTML-terms-glossary.asp). Over time you will become more and more familiar with all of them, but the three common HTML terms you should begin with are elements, tags, and attributes.

### Elements

Elements are designators that define the structure and content of objects within a page. Some of the more frequently used elements include multiple levels of headings (identified as <h1> through <h6> elements) and paragraphs (identified as the <p> element); the list goes on to include the <a>, <div>, <span>, <strong>, and <em> elements, and many more.

Elements are identified by the use of less-than and greater-than angle brackets, < >, surrounding the element name. Thus, an element will look like the following:

|  |  |
| --- | --- |
| 1  2 | <a> |

### Tags

The use of less-than and greater-than angle brackets surrounding an element creates what is known as a tag. Tags most commonly occur in pairs of opening and closing tags.

An opening tag marks the beginning of an element. It consists of a less-than sign followed by an element’s name, and then ends with a greater-than sign; for example, <div>.

A closing tag marks the end of an element. It consists of a less-than sign followed by a forward slash and the element’s name, and then ends with a greater-than sign; for example, </div>.

The content that falls between the opening and closing tags is the content of that element. An anchor link, for example, will have an opening tag of <a> and a closing tag of </a>. What falls between these two tags will be the content of the anchor link.

So, anchor tags will look a bit like this:

|  |  |
| --- | --- |
| 1  2 | <a>...</a> |

### Attributes

Attributes are properties used to provide additional information about an element. The most common attributes include the id attribute, which identifies an element; the class attribute, which classifies an element; the src attribute, which specifies a source for embeddable content; and the href attribute, which provides a hyperlink reference to a linked resource.

Attributes are defined within the opening tag, after an element’s name. Generally attributes include a name and a value. The format for these attributes consists of the attribute name followed by an equals sign and then a quoted attribute value. For example, an <a> element including an href attribute would look like the following:

|  |  |
| --- | --- |
| 1  2 | <a href="http://shayhowe.com/">Shay Howe</a> |

#### Common HTML Terms Demo

The preceding code will display the text “Shay Howe” on the web page and will take users to http://shayhowe.com/ upon clicking the “Shay Howe” text. The anchor element is declared with the opening <a> and closing </a> tags encompassing the text, and the hyperlink reference attribute and value are declared with href="http://shayhowe.com" in the opening tag.

**Fig 1**

HTML syntax outline including an element, attribute, and tag

Now that you know what HTML elements, tags, and attributes are, let’s take a look at putting together our first web page. If anything looks new here, no worries—we’ll decipher it as we go.

## Setting Up the HTML Document Structure[#html-document-structure](http://learn.shayhowe.com/html-css/building-your-first-web-page/#html-document-structure)

HTML documents are plain text documents saved with an .html file extension rather than a .txt file extension. To begin writing HTML, you first need a plain text editor that you are comfortable using. Sadly this does not include Microsoft Word or Pages, as those are rich text editors. Two of the more popular plain text editors for writing HTML and CSS are Dreamweaver and Sublime Text. Free alternatives also include Notepad++ for Windows and TextWrangler for Mac.

All HTML documents have a required structure that includes the following declaration and elements: <!DOCTYPE html>, <html>, <head>, and <body>.

The document type declaration, or <!DOCTYPE html>, informs web browsers which version of HTML is being used and is placed at the very beginning of the HTML document. Because we’ll be using the latest version of HTML, our document type declaration is simply <!DOCTYPE html>. Following the document type declaration, the <html> element signifies the beginning of the document.

Inside the <html> element, the <head> element identifies the top of the document, including any metadata (accompanying information about the page). The content inside the <head> element is not displayed on the web page itself. Instead, it may include the document title (which is displayed on the title bar in the browser window), links to any external files, or any other beneficial metadata.

All of the visible content within the web page will fall within the <body> element. A breakdown of a typical HTML document structure looks like this:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | <!DOCTYPE html>  <html lang="en">  <head>  <meta charset="utf-8">  <title>Hello World</title>  </head>  <body>  <h1>Hello World</h1>  <p>This is a web page.</p>  </body>  </html> |

#### HTML Document Structure Demo

The preceding code shows the document beginning with the document type declaration, <!DOCTYPE html>, followed directly by the <html> element. Inside the <html> element come the <head> and <body> elements. The <head> element includes the character encoding of the page via the <meta charset="utf-8"> tag and the title of the document via the <title> element. The <body> element includes a heading via the <h1> element and a paragraph via the <p> element. Because both the heading and paragraph are nested within the <body> element, they are visible on the web page.

When an element is placed inside of another element, also known as nested, it is a good idea to indent that element to keep the document structure well organized and legible. In the previous code, both the <head> and <body> elements were nested—and indented—inside the <html> element. The pattern of indenting for elements continues as new elements are added inside the <head> and <body> elements.

#### Self-Closing Elements

In the previous example, the <meta> element had only one tag and didn’t include a closing tag. Fear not, this was intentional. Not all elements consist of opening and closing tags. Some elements simply receive their content or behavior from attributes within a single tag. The <meta> element is one of these elements. The content of the previous <meta> element is assigned with the use of the charset attribute and value. Other common selfclosing elements include

* <br>
* <embed>
* <hr>
* <img>
* <input>
* <link>
* <meta>
* <param>
* <source>
* <wbr>

The structure outlined here, making use of the <!DOCTYPE html> document type and <html>, <head>, and <body> elements, is quite common. We’ll want to keep this document structure handy, as we’ll be using it often as we create new HTML documents.

#### Code Validation

No matter how careful we are when writing our code, we will inevitably make mistakes. Thankfully, when writing HTML and CSS we have validators to check our work. The W3C has built both [HTML](http://validator.w3.org/) and [CSS](http://jigsaw.w3.org/css-validator/) validators that will scan code for mistakes. Validating our code not only helps it render properly across all browsers, but also helps teach us the best practices for writing code.

## In Practice[#practice-1](http://learn.shayhowe.com/html-css/building-your-first-web-page/#practice-1)

As web designers and front-end developers, we have the luxury of attending a number of great conferences dedicated to our craft. We’re going to make up our own conference, Styles Conference, and build a website for it throughout the following lessons. Here we go!

1. Let’s open our text editor, create a new file named index.html, and save it to a location we won’t forget. I’m going to create a folder on my Desktop named “styles- conference” and save this file there; feel free to do the same.
2. Within the index.html file, let’s add the document structure, including the <!DOCTYPE html> document type and the <html>, <head>, and <body> elements.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | <!DOCTYPE html>  <html lang="en">  <head>  </head>  <body>  </body>  </html> |

1. Inside the <head> element, let’s add <meta> and <title> elements. The <meta> element should include the proper charset attribute and value, while the <title> element should contain the title of the page—let’s say “Styles Conference.”

|  |  |
| --- | --- |
| 1  2  3  4  5 | <head>  <meta charset="utf-8">  <title>Styles Conference</title>  </head> |

1. Inside the <body> element, let’s add <h1> and <p> elements. The <h1> element should include the heading we wish to include—let’s use “Styles Conference” again—and the <p> element should include a simple paragraph to introduce our conference.

|  |  |
| --- | --- |
| 1  2  3  4  5 | <body>  <h1>Styles Conference</h1>  <p>Every year the brightest web designers and front-end developers descend on Chicago to discuss the latest technologies. Join us this August!</p>  </body> |

1. Now it’s time to see how we’ve done! Let’s go find our index.html file (mine is within the “styles-conference” folder on my Desktop). Double-clicking this file or dragging it into a web browser will open it for us to review.

**Fig 1**

Our first steps into building our Styles Conference website

Let’s switch gears a bit, moving away from HTML, and take a look at CSS. Remember, HTML will define the content and structure of our web pages, while CSS will define the visual style and appearance of our web pages.

## Understanding Common CSS Terms[#common-css-terms](http://learn.shayhowe.com/html-css/building-your-first-web-page/#common-css-terms)

In addition to HTML terms, there are a few common [CSS terms](http://www.impressivewebs.com/css-terms-definitions/) you will want to familiarize yourself with. These terms include selectors, properties, and values. As with the HTML terminology, the more you work with CSS, the more these terms will become second nature.

### Selectors

As elements are added to a web page, they may be styled using CSS. A selector designates exactly which element or elements within our HTML to target and apply styles (such as color, size, and position) to. Selectors may include a combination of different qualifiers to select unique elements, all depending on how specific we wish to be. For example, we may want to select every paragraph on a page, or we may want to select only one specific paragraph on a page.

Selectors generally target an attribute value, such as an id or class value, or target the type of element, such as <h1> or <p>.

Within CSS, selectors are followed with curly brackets, {}, which encompass the styles to be applied to the selected element. The selector here is targeting all <p> elements.

|  |  |
| --- | --- |
| 1  2 | p { ... } |

### Properties

Once an element is selected, a property determines the styles that will be applied to that element. Property names fall after a selector, within the curly brackets, {}, and immediately preceding a colon, :. There are numerous properties we can use, such as background, color, font-size, height, and width, and new properties are often added. In the following code, we are defining the color and font-size properties to be applied to all <p> elements.

|  |  |
| --- | --- |
| 1  2  3  4  5 | p {  color: ...;  font-size: ...;  } |

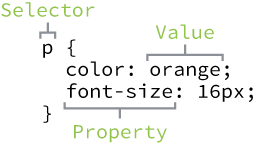
### Values

So far we’ve selected an element with a selector and determined what style we’d like to apply with a property. Now we can determine the behavior of that property with a value. Values can be identified as the text between the colon, :, and semicolon, ;. Here we are selecting all <p> elements and setting the value of the color property to be orange and the value of the font-size property to be 16 pixels.

|  |  |
| --- | --- |
| 1  2  3  4  5 | p {  color: orange;  font-size: 16px;  } |

To review, in CSS our rule set begins with the selector, which is immediately followed by curly brackets. Within these curly brackets are declarations consisting of property and value pairs. Each declaration begins with a property, which is followed by a colon, the property value, and finally a semicolon.

It is a common practice to indent property and value pairs within the curly brackets. As with HTML, these indentations help keep our code organized and legible.

**Fig 1**

CSS syntax outline including a selector, properties, and values

Knowing a few common terms and the general syntax of CSS is a great start, but we have a few more items to learn before jumping in too deep. Specifically, we need to take a closer look at how selectors work within CSS.

## Working with Selectors[#working-with-selectors](http://learn.shayhowe.com/html-css/building-your-first-web-page/#working-with-selectors)

Selectors, as previously mentioned, indicate which HTML elements are being styled. It is important to fully understand how to use selectors and how they can be leveraged. The first step is to become familiar with the different types of selectors. We’ll start with the most common selectors: type, class, and ID selectors.

### Type Selectors

Type selectors target elements by their element type. For example, should we wish to target all division elements, <div>, we would use a type selector of div. The following code shows a type selector for division elements as well as the corresponding HTML it selects.

###### CSS

|  |  |
| --- | --- |
| 1  2 | div { ... } |

###### HTML

|  |  |
| --- | --- |
| 1  2  3 | <div>...</div>  <div>...</div> |

### Class Selectors

Class selectors allow us to select an element based on the element’s class attribute value. Class selectors are a little more specific than type selectors, as they select a particular group of elements rather than all elements of one type.

Class selectors allow us to apply the same styles to different elements at once by using the same class attribute value across multiple elements.

Within CSS, classes are denoted by a leading period, ., followed by the class attribute value. Here the class selector will select any element containing the class attribute value of awesome, including both division and paragraph elements.

###### CSS

|  |  |
| --- | --- |
| 1  2 | .awesome { ... } |

###### HTML

|  |  |
| --- | --- |
| 1  2  3 | <div class="awesome">...</div>  <p class="awesome">...</p> |

### ID Selectors

ID selectors are even more precise than class selectors, as they target only one unique element at a time. Just as class selectors use an element’s class attribute value as the selector, ID selectors use an element’s id attribute value as a selector.

Regardless of which type of element they appear on, id attribute values can only be used once per page. If used they should be reserved for significant elements.

Within CSS, ID selectors are denoted by a leading hash sign, #, followed by the id attribute value. Here the ID selector will only select the element containing the id attribute value of shayhowe.

###### CSS

|  |  |
| --- | --- |
| 1  2 | #shayhowe { ... } |

###### HTML

|  |  |
| --- | --- |
| 1  2 | <div id="shayhowe">...</div> |

### Additional Selectors

Selectors are extremely powerful, and the selectors outlined here are the most common selectors we’ll come across. These selectors are also only the beginning. Many more [advanced selectors](http://learn.shayhowe.com/advanced-html-css/complex-selectors/) exist and are readily available. When you feel comfortable with these selectors, don’t be afraid to look into some of the more advanced selectors.

All right, everything is starting to come together. We add elements to a page inside our HTML, and we can then select those elements and apply styles to them using CSS. Now let’s connect the dots between our HTML and CSS, and get these two languages working together.

## Referencing CSS[#referencing-css](http://learn.shayhowe.com/html-css/building-your-first-web-page/#referencing-css)

In order to get our CSS talking to our HTML, we need to reference our CSS file within our HTML. The best practice for referencing our CSS is to include all of our styles in a single external style sheet, which is referenced from within the <head> element of our HTML document. Using a single external style sheet allows us to use the same styles across an entire website and quickly make changes sitewide.

#### Other Options for Adding CSS

Other options for referencing CSS include using internal and inline styles. You may come across these options in the wild, but they are generally frowned upon, as they make updating websites cumbersome and unwieldy.

To create our external CSS style sheet, we’ll want to use our text editor of choice again to create a new plain text file with a .css file extension. Our CSS file should be saved within the same folder, or a subfolder, where our HTML file is located.

Within the <head> element of the HTML document, the <link> element is used to define the relationship between the HTML file and the CSS file. Because we are linking to CSS, we use the rel attribute with a value of stylesheet to specify their relationship. Furthermore, the href (or hyperlink reference) attribute is used to identify the location, or path, of the CSS file.

Consider the following example of an HTML document <head> element that references a single external style sheet.

|  |  |
| --- | --- |
| 1  2  3  4 | <head>  <link rel="stylesheet" href="main.css">  </head> |

In order for the CSS to render correctly, the path of the href attribute value must directly correlate to where our CSS file is saved. In the preceding example, the main.css file is stored within the same location as the HTML file, also known as the root directory.

If our CSS file is within a subdirectory or subfolder, the href attribute value needs to correlate to this path accordingly. For example, if our main.css file were stored within a subdirectory named stylesheets, the href attribute value would be stylesheets/main.css, using a forward slash to indicate moving into a subdirectory.

At this point our pages are starting to come to life, slowly but surely. We haven’t delved into CSS too much, but you may have noticed that some elements have default styles we haven’t declared within our CSS. That is the browser imposing its own preferred CSS styles for those elements. Fortunately we can overwrite these styles fairly easily, which is what we’ll do next using CSS resets.

## Using CSS Resets[#using-css-resets](http://learn.shayhowe.com/html-css/building-your-first-web-page/#using-css-resets)

Every web browser has its own default styles for different elements. How Google Chrome renders headings, paragraphs, lists, and so forth may be different from how Internet Explorer does. To ensure cross-browser compatibility, CSS resets have become widely used.

CSS resets take every common HTML element with a predefined style and provide one unified style for all browsers. These resets generally involve removing any sizing, margins, paddings, or additional styles and toning these values down. Because CSS cascades from top to bottom—more on that soon—our reset needs to be at the very top of our style sheet. Doing so ensures that those styles are read first and that all of the different web browsers are working from a common baseline.

There are a bunch of different resets available to use, all of which have their own fortes. One of the most popular resets is [Eric Meyer’s reset](http://meyerweb.com/eric/tools/css/reset/), which has been adapted to include styles for the new HTML5 elements.

If you are feeling a bit more adventurous, there is also [Normalize.css](http://necolas.github.io/normalize.css/), created by Nicolas Gallagher. Normalize.css focuses not on using a hard reset for all common elements, but instead on setting common styles for these elements. It requires a stronger understanding of CSS, as well as awareness of what you’d like your styles to be.

#### Cross-Browser Compatibility & Testing

As previously mentioned, different browsers render elements in different ways. It’s important to recognize the value in cross-browser compatibility and testing. Websites don’t need to look exactly the same in every browser, but they should be close. Which browsers you wish to support, and to what degree, is a decision you will need to make based on what is best for your website.

In all there are a handful of things to be on the lookout for when writing CSS. The good news is that anything is possible, and with a little patience we’ll figure it all out.

## In Practice[#practice-2](http://learn.shayhowe.com/html-css/building-your-first-web-page/#practice-2)

Picking back up where we last left off on our conference website, let’s see if we can add in a bit of CSS.

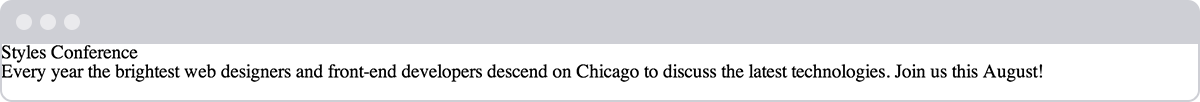
1. Inside of our “styles-conference” folder, let’s create a new folder named “assets.” We’ll store all of the assets for our website, such as our style sheets, images, videos, and so forth, in this folder. For our style sheets, let’s go ahead and add another folder named “stylesheets” inside the “assets” folder.
2. Using our text editor, let’s create a new file named main.css and save it within the “stylesheets” folder we just created.
3. Looking at our index.html file in a web browser, we can see that the <h1> and <p> elements each have default CSS styles. Specifically, they each have a unique font size and spacing around them. Using Eric Meyer’s reset, we can tone down these styles, allowing each of them to be styled from the same base. To do this let’s head over to [Eric’s website](http://meyerweb.com/eric/tools/css/reset/), copy his reset, and paste it at the top of our main.css file.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48 | /\* http://meyerweb.com/eric/tools/css/reset/ 2. v2.0 | 20110126  License: none (public domain)  \*/  html, body, div, span, applet, object, iframe,  h1, h2, h3, h4, h5, h6, p, blockquote, pre,  a, abbr, acronym, address, big, cite, code,  del, dfn, em, img, ins, kbd, q, s, samp,  small, strike, strong, sub, sup, tt, var,  b, u, i, center,  dl, dt, dd, ol, ul, li,  fieldset, form, label, legend,  table, caption, tbody, tfoot, thead, tr, th, td,  article, aside, canvas, details, embed,  figure, figcaption, footer, header, hgroup,  menu, nav, output, ruby, section, summary,  time, mark, audio, video {  margin: 0;  padding: 0;  border: 0;  font-size: 100%;  font: inherit;  vertical-align: baseline;  }  /\* HTML5 display-role reset for older browsers \*/  article, aside, details, figcaption, figure,  footer, header, hgroup, menu, nav, section {  display: block;  }  body {  line-height: 1;  }  ol, ul {  list-style: none;  }  blockquote, q {  quotes: none;  }  blockquote:before, blockquote:after,  q:before, q:after {  content: '';  content: none;  }  table {  border-collapse: collapse;  border-spacing: 0;  } |

1. With our main.css file starting to take shape, let’s connect it to our index.html file. Opening the index.html file in our text editor, let’s add the <link> element within our <head> element, just after the <title> element.
2. Because we’ll be referencing a style sheet within the <link> element, let’s add the relation attribute, rel, with a value of stylesheet.
3. We also want to include a hyperlink reference, using the href attribute, to our main.css file. Remember, our main.css file is saved within the “stylesheets” folder, which is inside the “assets” folder. Therefore, the href attribute value, which is the path to our main.css file, needs to be assets/stylesheets/main.css.

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | <head>  <meta charset="utf-8">  <title>Styles Conference</title>  <link rel="stylesheet" href="assets/stylesheets/main.css">  </head> |

Time to check out our work and see if our HTML and CSS are getting along. Now opening our index.html file (or refreshing the page if it’s already opened) within a web browser should show slightly different results than before.

**Fig 1**

Our Styles Conference website with a CSS reset

### Demo & Source Code

Below you may view the Styles Conference website in its current state, as well as download the source code for the website in its current state.

[**View the Styles Conference Website**](http://learn.shayhowe.com/practice/building-your-first-web-page/index.html) or [Download the Source Code](http://learn.shayhowe.com/practice/building-your-first-web-page.zip) (Zip file)

## Summary[#summary](http://learn.shayhowe.com/html-css/building-your-first-web-page/#summary)

So far, so good! We’ve taken a few big steps in this lesson.

Just think, you now know the basics of HTML and CSS. As we continue and you spend more time writing HTML and CSS, you’ll become much more comfortable with the two languages.

To recap, so far we’ve covered the following:

* The difference between HTML and CSS
* Getting acquainted with HTML elements, tags, and attributes
* Setting up the structure of your first web page
* Getting acquainted with CSS selectors, properties, and values
* Working with CSS selectors
* Referencing CSS in your HTML
* The value of CSS resets

Now let’s take a closer look at HTML and learn a little about semantics.

###### Lesson 2

# Getting to Know HTML

With our introduction to HTML and CSS complete, it’s time to dig a little deeper into HTML and examine the different components that make up this language.

In order to start building websites, we need to learn a little about which HTML elements are best used to display different types of content. It’s also important to understand how elements are visually displayed on a web page, as well as what different elements mean semantically.

Using the proper element for the job goes a long way, and we’ll want to make well-informed decisions in the process.

## Semantics Overview[#semantics-overview](http://learn.shayhowe.com/html-css/getting-to-know-html/#semantics-overview)

So what exactly are semantics? [Semantics within HTML](http://boagworld.com/dev/semantic-code-what-why-how/) is the practice of giving content on the page meaning and structure by using the proper element. Semantic code describes the value of content on a page, regardless of the style or appearance of that content. There are several benefits to using semantic elements, including enabling computers, screen readers, search engines, and other devices to adequately read and understand the content on a web page. Additionally, semantic HTML is easier to manage and work with, as it shows clearly what each piece of content is about.

Moving forward, as new elements are introduced, we’ll talk about what those elements actually mean and the type of content they best represent. Before we do that, though, let’s look at two elements—<div>s and <span>s—that actually don’t hold any semantic value. They exist for styling purposes only.

## Identifying Divisions & Spans[#divs-and-spans](http://learn.shayhowe.com/html-css/getting-to-know-html/#divs-and-spans)

Divisions, or <div>s, and <span>s are HTML elements that act as containers solely for styling purposes. As generic containers, they do not come with any overarching meaning or semantic value. Paragraphs are semantic in that content wrapped within a <p> element is known and understood as a paragraph. <div>s and <span>s do not hold any such meaning and are simply containers.

#### Block vs. Inline Elements

Most elements are either block- or inline-level elements. What’s the difference?

Block-level elements begin on a new line, stacking one on top of the other, and occupy any available width. Block-level elements may be nested inside one another and may wrap inline-level elements. We’ll most commonly see block-level elements used for larger pieces of content, such as paragraphs.

Inline-level elements do not begin on a new line. They fall into the normal flow of a document, lining up one after the other, and only maintain the width of their content. Inline-level elements may be nested inside one another; however, they cannot wrap block-level elements. We’ll usually see inline-level elements with smaller pieces of content, such as a few words.

Both <div>s and <span>s, however, are extremely valuable when building a website in that they give us the ability to apply targeted styles to a contained set of content.

A <div> is a block-level element that is commonly used to identify large groupings of content, and which helps to build a web page’s layout and design. A <span>, on the other hand, is an inline-level element commonly used to identify smaller groupings of text within a block-level element.

We’ll commonly see <div>s and <span>s with class or id attributes for styling purposes. Choosing a class or id attribute value, or name, requires a bit of care. We want to choose a value that refers to the content of an element, not necessarily the appearance of an element.

For example, if we have a <div> with an orange background that contains social media links, our first thought might be to give the <div> a class value of orange. What happens if that orange background is later changed to blue? Having a class value of orange no longer makes sense. A more sensible choice for a class value would be social, as it pertains to the contents of the <div>, not the style.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | <!-- Division -->  <div class="social">  <p>I may be found on...</p>  <p>Additionally, I have a profile on...</p>  </div>  <!-- Span -->  <p>Soon we'll be <span class="tooltip">writing HTML</span> with the best of them.</p> |

#### Comments within HTML & CSS

The previous code includes exclamation points within the HTML, and that’s all right. Those are not elements, those are comments.

HTML and CSS give us the ability to leave comments within our code, and any content wrapped within a comment will not be displayed on the web page. Comments help keep our files organized, allow us to set reminders, and provide a way for us to more effectively manage our code. Comments become especially useful when there are multiple people working on the same files.

HTML comments start with <!-- and end with -->. CSS comments start with /\* and end with \*/.

## Using Text-Based Elements[#text-based-elements](http://learn.shayhowe.com/html-css/getting-to-know-html/#text-based-elements)

Many different forms of media and content exist online; however, text is predominant. Accordingly, there are a number of different elements for displaying text on a web page. For now we’ll focus on the more popular elements, including headings, paragraphs, bold text to show importance, and italics for emphasis. Later, within Lesson 6, “[Working with Typography](http://learn.shayhowe.com/html-css/working-with-typography/),” we’ll take a closer look at how to style text.

### Headings

Headings are block-level elements, and they come in six different rankings, <h1> through <h6>. Headings help to quickly break up content and establish hierarchy, and they are key identifiers for users reading a page. They also help search engines to index and determine the content on a page.

Headings should be used in an order that is relevant to the content of a page. The primary heading of a page or section should be marked up with an <h1> element, and subsequent headings should use <h2>, <h3>, <h4>, <h5>, and <h6> elements as necessary.

Each heading level should be used where it is semantically valued, and should not be used to make text bold or big—there are other, better ways to do that.

Here is an example of HTML for all the different heading levels and the resulting display on a web page.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | <h1>Heading Level 1</h1>  <h2>Heading Level 2</h2>  <h3>Heading Level 3</h3>  <h4>Heading Level 4</h4>  <h5>Heading Level 5</h5>  <h6>Heading Level 6</h6> |

#### Headings Demo

### Paragraphs

Headings are often followed by supporting paragraphs. Paragraphs are defined using the <p> block-level element. Paragraphs can appear one after the other, adding information to a page as desired. Here is example of how to set up paragraphs.

|  |  |
| --- | --- |
| 1  2  3  4 | <p>Steve Jobs was a co-founder and longtime chief executive officer at Apple. On June 12, 2005, Steve gave the commencement address at Stanford University.</p>  <p>In his address Steve urged graduates to follow their dreams and, despite any setbacks, to never give up&ndash;advice which he sincerely took to heart.</p> |

#### Paragraphs Demo

### Bold Text with Strong

To make text bold and place a strong importance on it, we’ll use the <strong> inline-level element. There are two elements that will bold text for us: the <strong> and <b> elements. It is important to understand the [semantic difference](http://html5doctor.com/i-b-em-strong-element/) between the two.

The <strong> element is semantically used to give strong importance to text, and is thus the most popular option for bolding text. The <b> element, on the other hand, semantically means to stylistically offset text, which isn’t always the best choice for text deserving prominent attention. We have to gauge the significance of the text we wish to set as bold and to choose an element accordingly.

Here are the two HTML options for creating bold text in action:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | <!-- Strong importance -->  <p><strong>Caution:</strong> Falling rocks.</p>  <!-- Stylistically offset -->  <p>This recipe calls for <b>bacon</b> and <b>baconnaise</b>.</p> |

#### Bold Text with Strong Demo

### Italicize Text with Emphasis

To italicize text, thereby placing emphasis on it, we’ll use the <em> inline-level element. As with the elements for bold text, there are two different elements that will italicize text, each with a slightly different semantic meaning.

The <em> element is used semantically to place a stressed emphasis on text; it is thus the most popular option for italicizing text. The other option, the <i> element, is used semantically to convey text in an alternative voice or tone, almost as if it were placed in quotation marks. Again, we will need to gauge the significance of the text we want to italicize and choose an element accordingly.

Here’s the HTML code for italicizing:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | <!-- Stressed emphasis -->  <p>I <em>love</em> Chicago!</p>  <!-- Alternative voice or tone -->  <p>The name <i>Shay</i> means a gift.</p> |

#### Italicize Text with Emphasis Demo

These text-level elements are quite handy for bringing our content to life. In addition to these, there are structurally based elements. Whereas text-based elements identify headings and paragraphs, structural elements identify groupings of content such as headers, articles, footers, and so forth. Let’s take a look.

## Building Structure[#building-structure](http://learn.shayhowe.com/html-css/getting-to-know-html/#building-structure)

For the longest time the structure of a web page was built using divisions. The problem was that divisions provide no semantic value, and it was fairly difficult to determine the intention of these divisions. Fortunately HTML5 introduced new [structurally based elements](http://dev.opera.com/articles/new-structural-elements-in-html5/), including the <header>, <nav>, <article>, <section>, <aside>, and <footer> elements.

All of these new elements are intended to give meaning to the organization of our pages and improve our structural semantics. They are all block-level elements and do not have any implied position or style. Additionally, all of these elements may be used multiple times per page, so long as each use reflects the proper semantic meaning.

Let’s roll up our sleeves and take a closer look.

**Fig 2**

One possible example of HTML5 structural elements giving meaning to the organization of our pages

### Header

The <header> element, like it sounds, is used to identify the top of a page, article, section, or other segment of a page. In general, the <header> element may include a heading, introductory text, and even navigation.

|  |  |
| --- | --- |
| 1  2 | <header>...</header> |

#### <header> vs. <head> vs. <h1> through <h6> Elements

It is easy to confuse the <header> element with the <head> element or the heading elements, <h1> through <h6>. They all have different semantic meanings and should be used according to their meanings. For reference…

The <header> element is a structural element that outlines the heading of a segment of a page. It falls within the <body> element.

The <head> element is not displayed on a page and is used to outline metadata, including the document title, and links to external files. It falls directly within the <html> element.

Heading elements, <h1> through <h6>, are used to designate multiple levels of text headings throughout a page.

### Navigation

The <nav> element identifies a section of major navigational links on a page. The <nav> element should be reserved for primary navigation sections only, such as global navigation, a table of contents, previous/next links, or other noteworthy groups of navigational links.

Most commonly, links included within the <nav> element will link to other pages within the same website or to parts of the same web page. Miscellaneous one-off links should not be wrapped within the <nav> element; they should use the anchor element, <a>, and the anchor element alone.

|  |  |
| --- | --- |
| 1  2 | <nav>...</nav> |

### Article

The <article> element is used to identify a section of independent, self-contained content that may be independently distributed or reused. We’ll often use the <article> element to mark up blog posts, newspaper articles, user-submitted content, and the like.

When deciding whether to use the <article> element, we must determine if the content within the element could be replicated elsewhere without any confusion. If the content within the <article> element were removed from the context of the page and placed, for example, within an email or printed work, that content should still make sense.

|  |  |
| --- | --- |
| 1  2 | <article>...</article> |

### Section

The <section> element is used to identify a thematic grouping of content, which generally, but not always, includes a heading. The grouping of content within the <section> element may be generic in nature, but it’s useful to identify all of the content as related.

The <section> element is commonly used to break up and provide hierarchy to a page.

|  |  |
| --- | --- |
| 1  2 | <section>...</section> |

#### Deciding Between <article>, <section>, or <div> Elements

At times it becomes fairly difficult to decide which element—<article>, <section>, or <div>—is the best element for the job based on its semantic meaning. The trick here, as with every semantic decision, is to look at the content.

Both the <article> and <section> elements contribute to a document’s structure and help to outline a document. If the content is being grouped solely for styling purposes and doesn’t provide value to the outline of a document, use the <div> element.

If the content adds to the document outline and it can be independently redistributed or syndicated, use the <article> element.

If the content adds to the document outline and represents a thematic group of content, use the <section> element.

### Aside

The <aside> element holds content, such as sidebars, inserts, or brief explanations, that is tangentially related to the content surrounding it. When used within an <article> element, for example, the <aside> element may identify content related to the author of the article.

We may instinctively think of an <aside> element as an element that appears off to the left or right side of a page. We have to remember, though, that all of the structural elements, including the <aside> element, are block-level elements and as such will appear on a new line, occupying the full available width of the page or of the element they are nested within, also known as their parent element.

|  |  |
| --- | --- |
| 1  2 | <aside>...</aside> |

We’ll discuss how to change the position of an element, perhaps placing it to the right or left of a group of content, in Lesson 5, “[Positioning Content](http://learn.shayhowe.com/html-css/positioning-content/).”

### Footer

The <footer> element identifies the closing or end of a page, article, section, or other segment of a page. Generally the <footer> element is found at the bottom of its parent. Content within the <footer> element should be relative information and should not diverge from the document or section it is included within.

|  |  |
| --- | --- |
| 1  2 | <footer>...</footer> |

With structural elements and text-based elements under our belts, our HTML knowledge is really starting to come together. Now is a good time to revisit our Styles Conference website and see if we can provide it with a little better structure.

## In Practice[#practice-1](http://learn.shayhowe.com/html-css/getting-to-know-html/#practice-1)

Currently, our Styles Conference website lacks real structure—and content for that matter. Let’s take some time to flesh out our home page a bit.

1. Using our existing index.html file, let’s add in a <header> element. Our <header> element should include our existing <h1> element; let’s also add an <h3> element as a tagline to support our <h1> element.

|  |  |
| --- | --- |
| 1  2  3  4  5 | <header>  <h1>Styles Conference</h1>  <h3>August 24&ndash;26th &mdash; Chicago, IL</h3>  </header> |

1. After our <header> element, let’s add a new group of content, using the <section> element, that introduces our conference. We’ll begin this section with a new <h2> element and end it with our existing paragraph.

|  |  |
| --- | --- |
| 1  2  3  4  5 | <section>  <h2>Dedicated to the Craft of Building Websites</h2>  <p>Every year the brightest web designers and front-end developers descend on Chicago to discuss the latest technologies. Join us this August!</p>  </section> |

1. Following the introduction to our conference, let’s add another group of content that teases a few of the pages we’ll be adding, specifically the Speakers, Schedule, and Venue pages. Each of the pages we’re teasing should also reside within its own section and include supporting text.

We’ll group all of the teasers inside a <section> element, and each individual teaser will be wrapped within a <section> element as well. In all, we’ll have three <section> elements inside another <section> element, which is all right.

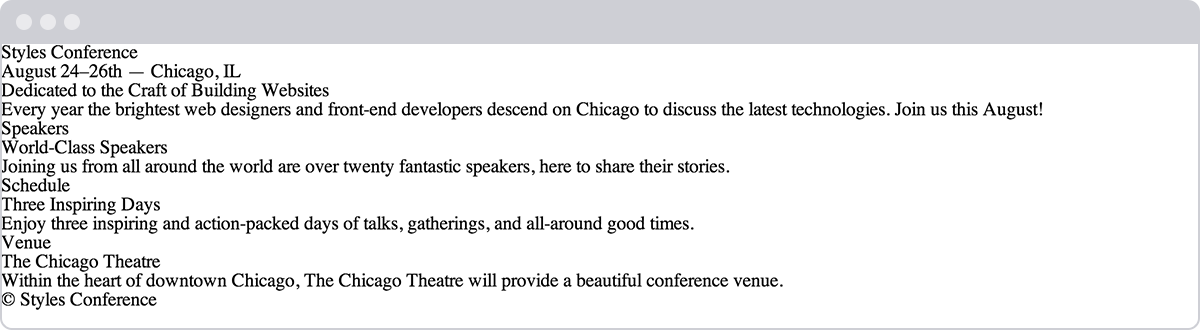
|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | <section>  <section>  <h5>Speakers</h5>  <h3>World-Class Speakers</h3>  <p>Joining us from all around the world are over twenty fantastic speakers, here to share their stories.</p>  </section>  ...  </section> |

1. Lastly, let’s add our copyright within the <footer> element at the end of our page. To do so let’s use the <small> element, which semantically represents side comments and small print—perfect for our copyright.

Generally, content within the <small> element will be rendered as, well, small, but our CSS reset will prevent that from happening.

|  |  |
| --- | --- |
| 1  2  3  4 | <footer>  <small>&copy; Styles Conference</small>  </footer> |

Now we can see our home page beginning to come to life.

**Fig 2**

Our home page after adding more content and structure

#### Encoding Special Characters

The <h3> element within our <header> element, as well as the <small> element within our <footer> element, has some interesting things going on. Specifically, a few special characters within these elements are being encoded.

Special characters include various punctuation marks, accented letters, and symbols. When typed directly into HTML, they can be misunderstood or mistaken for the wrong character; thus they need to be encoded.

Each encoded character will begin with an ampersand, &, and end with a semicolon, ;. What falls between the ampersand and semicolon is a character’s unique encoding, be it a name or numeric encoding.

For example, we would encode the word “resumé” as resum&eacute;. Within our header we have encoded both en and em dashes, and within our footer we have encoded the copyright symbol. For reference, a long list of character encodings may be found at [Copy Paste Character](http://copypastecharacter.com).

With our home page taking shape, let’s take a look at creating hyperlinks so that we may add additional pages and build out the rest of our website.

## Creating Hyperlinks[#creating-hyperlinks](http://learn.shayhowe.com/html-css/getting-to-know-html/#creating-hyperlinks)

Along with text, one of the core components of the Internet is the hyperlink, which provides the ability to link from one web page or resource to another. Hyperlinks are established using the anchor, <a>, inline-level element. In order to create a link from one page (or resource) to another, the href attribute, known as a hyperlink reference, is required. The href attribute value identifies the destination of the link.

For example, clicking the text “Shay,” which is wrapped inside the anchor element with the href attribute value of http://shayhowe.com, will take users to my website.

|  |  |
| --- | --- |
| 1  2 | <a href="http://shayhowe.com">Shay</a> |

#### Creating Hyperlinks Demo

#### Wrapping Block-Level Elements with Anchors

By nature the anchor element, <a>, is an inline element, and, according to web standards, inline-level elements may not wrap block-level elements. With the introduction of HTML5, however, anchor elements specifically have permission to wrap either block-, inline-, or any other level elements. This is a break from the standard convention, but it’s permissible in order to enable entire blocks of content on a page to become links.

### Relative & Absolute Paths

The two most common types of links are links to other pages of the same website and links to other websites. These links are identified by their href attribute values, also known as their paths.

Links pointing to other pages of the same website will have a relative path, which does not include the domain (.com, .org, .edu, etc.) in the href attribute value. Because the link is pointing to another page on the same website, the href attribute value needs to include only the filename of the page being linked to: about.html, for example.

Should the page being linked to reside within a different directory, or folder, the href attribute value needs to reflect this as well. Say the about.html page resides within the pages directory; the relative path would then be pages/about.html.

Linking to other websites outside of the current one requires an absolute path, where the href attribute value must include the full domain. A link to Google would need the href attribute value of http://google.com, starting with http and including the domain, .com in this case.

Here clicking on the text “About” will open the about.html page inside our browser. Clicking the text “Google,” on the other hand, will open http://google.com/ within our browser.

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | <!-- Relative Path -->  <a href="/about.html">About</a>  <!-- Absolute Path -->  <a href="http://www.google.com/">Google</a> |

### Linking to an Email Address

Occasionally we may want to create a hyperlink to our email address—for example, hyperlink text that says “Email Me,” which when clicked opens a user’s default email client and pre-populates part of an email. At a minimum the email address to which the email is being sent is populated; other information such as a subject line and body text may also be included.

To create an [email link](https://yoast.com/guide-mailto-links/), the href attribute value needs to start with mailto: followed by the email address to which the email should be sent. To create an email link to shay@awesome.com, for example, the href attribute value would be mailto:shay@awesome.com.

Additionally, subject, body text, and other information for the email may be populated. To add a subject line, we’ll include the subject= parameter after the email address. The first parameter following the email address must begin with a question mark, ?, to bind it to the hyperlink path. Multiple words within a subject line require that spaces be encoded using %20.

Adding body text works in the same way as adding the subject, this time using the body= parameter in the href attribute value. Because we are binding one parameter to another we need to use the ampersand, &, to separate the two. As with the subject, spaces must be encoded using %20, and line breaks must be encoded using %0A.

Altogether, a link to shay@awesome.com with the subject of “Reaching Out” and body text of “How are you” would require an href attribute value of mailto:shay@awesome.com?subject=Reaching%20Out&body=How%20are%20you.

Here’s the full breakdown:

|  |  |
| --- | --- |
| 1  2 | <a href="mailto:shay@awesome.com?subject=Reaching%20Out&body=How%20are%20you">Email Me</a> |

### Opening Links in a New Window

One feature available with hyperlinks is the ability to determine where a link opens when clicked. Typically, links open in the same window from which they are clicked; however, links may also be opened in new windows.

To trigger the action of opening a link in a new window, use the target attribute with a value of \_blank. The target attribute determines exactly where the link will be displayed, and the \_blank value specifies a new window.

To open http://shayhowe.com/ in a new window, the code would look like this:

|  |  |
| --- | --- |
| 1  2 | <a href="http://shayhowe.com/" target="\_blank">Shay Howe</a> |

### Linking to Parts of the Same Page

Periodically we’ll see hyperlinks that link to part of the same page the link appears on. A common example of these same-page links are “Back to top” links that return a user to the top of a page.

We can create an on-page link by first setting an id attribute on the element we wish to link to, then using the value of that id attribute within an anchor element’s href attribute.

Using the “Back to top” link as an example, we can place an id attribute value of top on the <body> element. Now we can create an anchor element with an href attribute value of #top, pound sign and all, to link to the beginning of the <body> element.

Our code for this same-page link would look like the following:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | <body id="top">  ...  <a href="#top">Back to top</a>  ...  </body> |

Hyperlinks are incredibly useful and have revolutionized how we use the Internet. So far we’ve covered how to link to other pages or websites, as well as how to create email links and links to parts of the same page. Before we go any further, let’s create some links of our own.

## In Practice[#practice-2](http://learn.shayhowe.com/html-css/getting-to-know-html/#practice-2)

It’s time to take Styles Conference from a single-page website to a full-blown website with multiple pages, all of which will be linked together using hyperlinks.

1. We’ll begin by making our “Styles Conference” text inside the <h1> element within our <header> element link to the index.html page.

Because we are already on the index.html page, this may seem a little odd—and rightfully so—but as the header is replicated on other pages, linking back to the home page will make sense.

|  |  |
| --- | --- |
| 1  2  3  4 | <h1>  <a href="index.html">Styles Conference</a>  </h1> |

1. In order to navigate across all of the different pages, we’re going add in a navigation menu, using the <nav> element, within our <header> element. We’ll be creating Speakers, Schedule, Venue, and Register pages to go with our home page, so we should create links for all of them.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | <header>  ...  <nav>  <a href="index.html">Home</a>  <a href="speakers.html">Speakers</a>  <a href="schedule.html">Schedule</a>  <a href="venue.html">Venue</a>  <a href="register.html">Register</a>  </nav>  </header> |

1. Let’s also add the same navigation menu from our <header> element to our <footer> element for convenience.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | <footer>  ...  <nav>  <a href="index.html">Home</a>  <a href="speakers.html">Speakers</a>  <a href="schedule.html">Schedule</a>  <a href="venue.html">Venue</a>  <a href="register.html">Register</a>  </nav>  </footer> |

1. Within the <section> element that introduces our conference, just below our header, we should also include a link to register for the conference. Placing a link below the paragraph will work perfectly.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | <section>  ...  <a href="register.html">Register Now</a>  </section> |

1. We can’t forget to add links to all of the sections teasing our other pages. Inside each section, let’s wrap both the <h3> and <h5> elements within an anchor element linking to the proper page.

We’ll want to make sure we do this for every section accordingly.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | <section>  <section>  <a href="speakers.html">  <h5>Speakers</h5>  <h3>World-Class Speakers</h3>  </a>  <p>Joining us from all around the world are over twenty fantastic speakers, here to share their stories.</p>  </section>  ...  </section> |

1. Now we need to create a handful of new pages. Let’s create speakers.html, schedule.html, venue.html, and register.html files. These files should live within the same folder as the index.html file, and, because we’re keeping them inside the same folder, all of our links should work as expected.

To ensure that all of our pages look the same, let’s make sure that all of these new files have the same document structure and <header> and <footer> elements as the index.html file.

It’s official, we’re no longer working with a single page but indeed a full website.

**Fig 2**

Our home page after all of the different links and navigation have been added

### Demo & Source Code

Below you may view the Styles Conference website in its current state, as well as download the source code for the website in its current state.

[**View the Styles Conference Website**](http://learn.shayhowe.com/practice/getting-to-know-html/index.html) or [Download the Source Code](http://learn.shayhowe.com/practice/getting-to-know-html.zip) (Zip file)

## Summary[#summary](http://learn.shayhowe.com/html-css/getting-to-know-html/#summary)

Semantics, as discussed within this lesson, are essential for providing our HTML with structure and meaning. Moving forward we’ll periodically introduce new elements, all of which will come with their own semantic meaning. It is the meaning of all of these elements that will provide our content with the most value.

Once again, in this lesson we covered the following:

* What semantics are and why they are important
* <div>s and <spans>s, and the difference between block- and inline-level elements
* Which text-based elements best represent the content of a page
* The HTML5 structural elements and how to define the structure and organization of our content and page
* How to use hyperlinks to navigate between web pages or websites

Hopefully you’re starting to feel pretty good about HTML. There is still quite a bit to learn, but the foundation is in place. Next up, we’ll take a deeper look into CSS.

###### Lesson 3

# Getting to Know CSS

CSS is a complex language that packs quite a bit of power.

It allows us to add layout and design to our pages, and it allows us to share those styles from element to element and page to page. Before we can unlock all of its features, though, there are a few aspects of the language we must fully understand.

First, it’s crucial to know exactly how styles are rendered. Specifically, we’ll need to know how different types of selectors work and how the order of those selectors can affect how our styles are rendered. We’ll also want to understand a few common property values that continually appear within CSS, particularly those that deal with color and length.

Let’s look under the hood of CSS to see exactly what is going on.

## The Cascade[#cascade](http://learn.shayhowe.com/html-css/getting-to-know-css/#cascade)

We’ll begin breaking down exactly how styles are rendered by looking at what is known as the cascade and studying a few examples of the cascade in action. Within CSS, all styles cascade from the top of a style sheet to the bottom, allowing different styles to be added or overwritten as the style sheet progresses.

For example, say we select all paragraph elements at the top of our style sheet and set their background color to orange and their font size to 24 pixels. Then towards the bottom of our style sheet, we select all paragraph elements again and set their background color to green, as seen here.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | p {  background: orange;  font-size: 24px;  }  p {  background: green;  } |

Because the paragraph selector that sets the background color to green comes after the paragraph selector that sets the background color to orange, it will take precedence in the cascade. All of the paragraphs will appear with a green background. The font size will remain 24 pixels because the second paragraph selector didn’t identify a new font size.

### Cascading Properties

The cascade also works with properties inside individual selectors. Again, for example, say we select all the paragraph elements and set their background color to orange. Then directly below the orange background property and value declaration, we add another property and value declaration setting the background color to green, as seen here.

|  |  |
| --- | --- |
| 1  2  3  4  5 | p {  background: orange;  background: green;  } |

Because the green background color declaration comes after the orange background color declaration, it will overrule the orange background, and, as before, our paragraphs will appear with a green background.

All styles will cascade from the top of our style sheet to the bottom of our style sheet. There are, however, times where the cascade doesn’t play so nicely. Those times occur when different types of selectors are used and the specificity of those selectors breaks the cascade. Let’s take a look.

## Calculating Specificity[#specificity](http://learn.shayhowe.com/html-css/getting-to-know-css/#specificity)

Every selector in CSS has a specificity weight. A selector’s specificity weight, along with its placement in the cascade, identifies how its styles will be rendered.

In Lesson 1, “[Building Your First Web Page](http://learn.shayhowe.com/html-css/building-your-first-web-page/),” we talked about three different types of selectors: the type, class, and ID selectors. Each of these selectors has a different specificity weight.

The type selector has the lowest specificity weight and holds a point value of 0-0-1. The class selector has a medium specificity weight and holds a point value of 0-1-0. Lastly, the ID selector has a high specificity weight and holds a point value of 1-0-0. As we can see, specificity points are calculated using three columns. The first column counts ID selectors, the second column counts class selectors, and the third column counts type selectors.

What’s important to note here is that the ID selector has a higher specificity weight than the class selector, and the class selector has a higher specificity weight than the type selector.

#### Specificity Points

Specificity points are intentionally hyphenated, as their values are not computed from a base of 10. Class selectors do not hold a point value of 10, and ID selectors do not hold a point value of 100. Instead, these points should be read as 0-1-0 and 1-0-0 respectively. We’ll take a closer look at why these point values are hyphenated shortly, when we combine selectors.

The higher the specificity weight of a selector, the more superiority the selector is given when a styling conflict occurs. For example, if a paragraph element is selected using a type selector in one place and an ID selector in another, the ID selector will take precedence over the type selector regardless of where the ID selector appears in the cascade.

###### HTML

|  |  |
| --- | --- |
| 1  2 | <p id="food">...</p> |

###### CSS

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | #food {  background: green;  }  p {  background: orange;  } |

Here we have a paragraph element with an id attribute value of food. Within our CSS, that paragraph is being selected by two different kinds of selectors: one type selector and one ID selector. Although the type selector comes after the ID selector in the cascade, the ID selector takes precedence over the type selector because it has a higher specificity weight; consequently the paragraph will appear with a green background.

The specificity weights of different types of selectors are incredibly important to remember. At times styles may not appear on elements as intended, and chances are the specificity weights of our selectors are breaking the cascade, therefore our styles are not appearing properly.

Understanding how the cascade and specificity work is a huge hurdle, and we’ll continue to cover this topic. For now, let’s look at how to be a little more particular and intentional with our selectors by combining them. Keep in mind that as we combine selectors, we’ll also be changing their specificity.

## Combining Selectors[#combining-selectors](http://learn.shayhowe.com/html-css/getting-to-know-css/#combining-selectors)

So far we’ve looked at how to use different types of selectors individually, but we also need to know how to use these selectors together. By combining selectors we can be more specific about which element or group of elements we’d like to select.

For example, say we want to select all paragraph elements that reside within an element with a class attribute value of hotdog and set their background color to brown. However, if one of those paragraphs happens to have the class attribute value of mustard, we want to set its background color to yellow. Our HTML and CSS may look like the following:

###### HTML

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | <div class="hotdog">  <p>...</p>  <p>...</p>  <p class="mustard">...</p>  </div> |

###### CSS

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | .hotdog p {  background: brown;  }  .hotdog p.mustard {  background: yellow;  } |

When selectors are combined they should be read from right to left. The selector farthest to the right, directly before the opening curly bracket, is known as the key selector. The key selector identifies exactly which element the styles will be applied to. Any selector to the left of the key selector will serve as a prequalifier.

The first combined selector above, .hotdog p, includes two selectors: a class and a type selector. These two selectors are separated by a single space. The key selector is a type selector targeting paragraph elements. And because this type selector is prequalified with a class selector of hotdog, the full combined selector will only select paragraph elements that reside within an element with a class attribute value of hotdog.

The second selector above, .hotdog p.mustard, includes three selectors: two class selectors and one type selector. The only difference between the second selector and the first selector is the addition of the class selector of mustard to the end of the paragraph type selector. Because the new class selector, mustard, falls all the way to the right of the combined selector, it is the key selector, and all of the individual selectors coming before it are now prequalifiers.

#### Spaces Within Selectors

Within the previous combined selector, .hotdog p.mustard, there is a space between the hotdog class selector and the paragraph type selector but not between the paragraph type selector and the mustard class selector. The use, and omission, of spaces makes a large difference in selectors.

Since there isn’t a space between the paragraph type selector and the mustard class selector that means the selector will only select paragraph elements with the class of mustard. If the paragraph type selector was removed, and the mustard class selector had spaces on both sides of it, it would select any element with the class of mustard, not just paragraphs.

The best practice is to not prefix a class selector with a type selector. Generally we want to select any element with a given class, not just one type of element. And following this best practice, our new combined selector would be better as .hotdog .mustard.

Reading the combined selector from right to left, it is targeting paragraphs with a class attribute value of mustard that reside within an element with the class attribute value of hotdog.

Different types of selectors can be combined to target any given element on a page. As we continue to write different combined selectors, we’ll see their powers come to life. Before we do that, though, let’s take a look at how combining selectors changes a selector’s specificity weight.

### Specificity Within Combined Selectors

When selectors are combined, so are the specificity weights of the individual selectors. These combined specificity weights can be calculated by counting each different type of selector within a combined selector.

Looking at our combined selectors from before, the first selector, .hotdog p, had both a class selector and a type selector. Knowing that the point value of a class selector is 0-1-0 and the point value of a type selector is 0-0-1, the total combined point value would be 0-1-1, found by adding up each kind of selector.

The second selector, .hotdog p.mustard, had two class selectors and one type selector. Combined, the selector has a specificity point value of 0-2-1. The 0 in the first column is for zero ID selectors, the 2 in the second column is for two class selectors, and the 1 in the last column is for one type selector.

Comparing the two selectors, the second selector, with its two classes, has a noticeably higher specificity weight and point value. As such it will take precedence within the cascade. If we were to flip the order of these selectors within our style sheet, placing the higher-weighted selector above the lower-weighted selector as shown here, the appearance of their styles would not be affected due to each selector’s specificity weight.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | .hotdog p.mustard {  background: yellow;  }  .hotdog p {  background: brown;  } |

In general we want to always keep an eye on the specificity weights of our selectors. The higher our specificity weights rise, the more likely our cascade is to break.

## Layering Styles with Multiple Classes[#multiple-classes](http://learn.shayhowe.com/html-css/getting-to-know-css/#multiple-classes)

One way to keep the specificity weights of our selectors low is to be as modular as possible, sharing similar styles from element to element. And one way to be as modular as possible is to layer on different styles by using multiple classes.

Elements within HTML can have more than one class attribute value so long as each value is space separated. With that, we can place certain styles on all elements of one sort while placing other styles only on specific elements of that sort.

We can tie styles we want to continually reuse to one class and layer on additional styles from another class.

Let’s take a look at buttons, for example. Say we want all of our buttons to have a font size of 16 pixels, but we want the background color of our buttons to vary depending on where the buttons are used. We can create a few classes and layer them on an element as necessary to apply the desired styles.

###### HTML

|  |  |
| --- | --- |
| 1  2  3 | <a class="btn btn-danger">...</a>  <a class="btn btn-success">...</a> |

###### CSS

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | .btn {  font-size: 16px;  }  .btn-danger {  background: red;  }  .btn-success {  background: green;  } |

Here you can see two anchor elements, both with multiple class attribute values. The first class, btn, is used to apply a font size of 16 pixels to each of the elements. Then, the first anchor element uses an additional class of btn-danger to apply a red background color while the second anchor element uses an additional class of btn-success to apply a green background color. Our styles here are clean and modular.

Using multiple classes, we can layer on as many styles as we wish, keeping our code lean and our specificity weights low. Much like understanding the cascade and calculating specificity, this is a practice that will take time to fully absorb, but we’ll get better with each lesson.

## Common CSS Property Values[#css-property-values](http://learn.shayhowe.com/html-css/getting-to-know-css/#css-property-values)

We’ve used a handful of common CSS property values already, such as the keyword color values of red and green. You may not have thought too much about them; that’s okay. We’re going to take time now to go over some previously used property values as well as to explore some of the more common property values that we’ll soon be using.

Specifically, we’ll look at property values that relate to colors and length measurements.

### Colors

All color values within CSS are defined on an sRGB (or standard red, green, and blue) color space. Colors within this space are formed by mixing red, green, and blue color channels together, mirroring the way that televisions and monitors generate all the different colors they display. By mixing different levels of red, green, and blue, we can create millions of colors—and find nearly any color we’d like.

Currently there are four primary ways to represent sRGB colors within CSS: keywords, hexadecimal notation, and RGB and HSL values.

#### Keyword Colors

Keyword color values are names (such as red, green, or blue) that map to a given color. These keyword names and their corresponding colors are determined by the CSS specification. Most common colors, along with a few oddities, have keyword names.

A complete list of these keyword names can be found within the [CSS specification](http://www.w3.org/TR/css3-color/).

| **Color** | **Name** | **Hex Values** | **RGB Values** | **HSL Values** |
| --- | --- | --- | --- | --- |
|  | black | #000000 | rgb(0, 0, 0) | hsl(0, 0%, 0%) |
|  | silver | #c0c0c0 | rgb(192, 192, 192) | hsl(0, 0%, 75%) |
|  | gray | #808080 | rgb(128, 128, 128) | hsl(0, 0%, 50%) |
|  | white | #ffffff | rgb(255, 255, 255) | hsl(0, 100%, 100%) |
|  | maroon | #800000 | rgb(128, 0, 0) | hsl(0, 100%, 25%) |
|  | red | #ff0000 | rgb(255, 0, 0) | hsl(0, 100%, 50%) |
|  | purple | #800080 | rgb(128, 0, 128) | hsl(300, 100%, 25%) |
|  | fuchsia | #ff00ff | rgb(255, 0, 255) | hsl(300, 100%, 50%) |
|  | green | #008000 | rgb(0, 128, 0) | hsl(120, 100%, 25%) |
|  | olive | #808000 | rgb(0, 255, 0) | hsl(120, 100%, 50%) |
|  | lime | #00ff00 | rgb(128, 128, 0) | hsl(60, 100%, 25%) |
|  | yellow | #ffff00 | rgb(255, 255, 0) | hsl(60, 100%, 50%) |
|  | navy | #000080 | rgb(0, 0, 128) | hsl(240, 100%, 25%) |
|  | blue | #0000ff | rgb(0, 0, 255) | hsl(240, 100%, 50%) |
|  | teal | #008080 | rgb(0, 128, 128) | hsl(180, 100%, 25%) |
|  | aqua | #00ffff | rgb(0, 255, 255) | hsl(180, 100%, 50%) |

Here we are applying a maroon background to any element with the task class attribute value and a yellow background to any element with the count class attribute value.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | .task {  background: maroon;  }  .count {  background: yellow;  } |

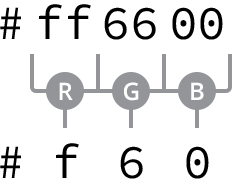
While keyword color values are simple in nature, they provide limited options and thus are not the most popular color value choice.

#### Hexadecimal Colors

Hexadecimal color values consist of a pound, or hash, #, followed by a three- or six- character figure. The figures use the numbers 0 through 9 and the letters a through f, upper or lower case. These values map to the red, green, and blue color channels.

In six-character notation, the first two characters represent the red channel, the third and fourth characters represent the green channel, and the last two characters represent the blue channel. In three-character notation, the first character represents the red channel, the second character represents the green channel, and the last character represents the blue channel.

If in six-character notation the first two characters are a matching pair, the third and fourth characters are a matching pair, and the last two characters are a matching pair, the six-character figure may be shortened to a three-character figure. To do this the repeated character from each pair should be used once. For example, a shade of orange represented by the hexadecimal color #ff6600 could also be written as #f60.

**Fig 3**

Six-character hexadecimal values may be written as three-character hexadecimal values when the red, green, and blue color channels each contain a repeating character

The character pairs are obtained by converting 0 through 255 into a base-16, or hexadecimal, format. The math is a little tricky—and worthy of its own book—but it helps to know that 0 equals black and f equals white.

#### The Millions of Hexadecimal Colors

There are millions of hexadecimal colors, over 16.7 million to be exact. Here’s how…

There are 16 options for every character in a hexadecimal color, 0 through 9 and a through f. With the characters grouped in pairs, there are 256 color options per pair (16 multiplied by 16, or 16 squared).

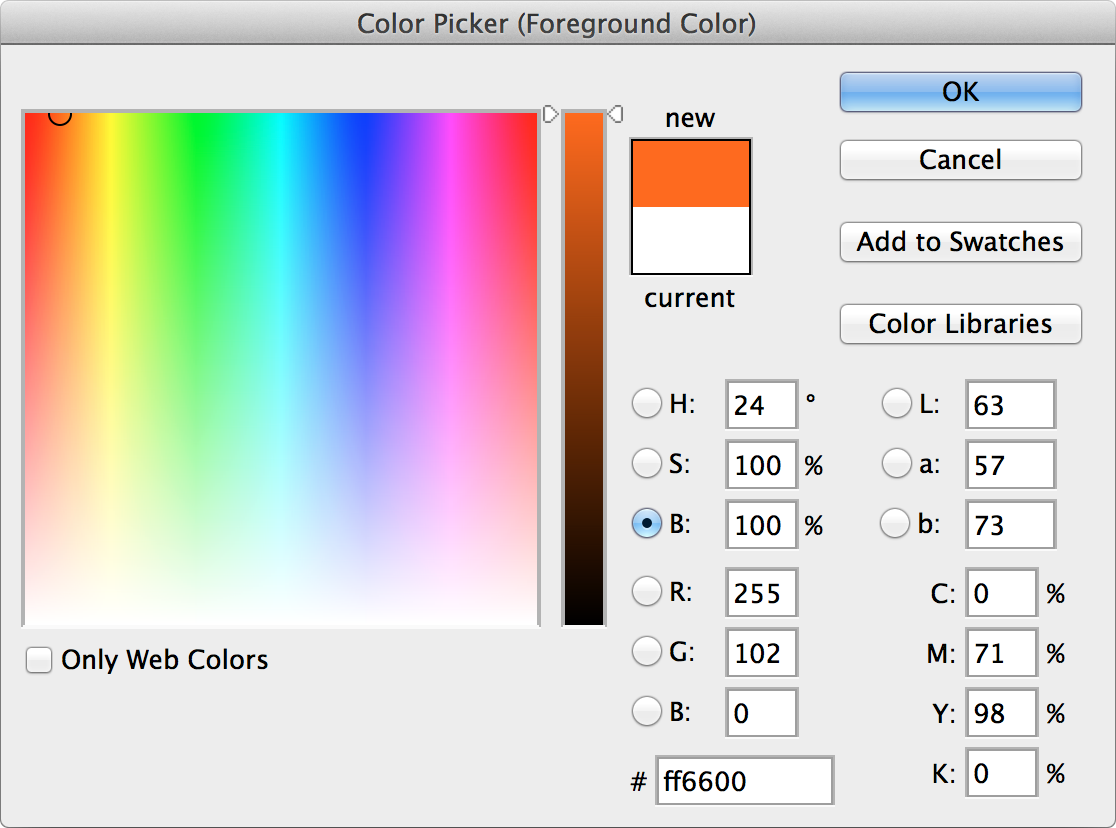
And with three groups of 256 color options we have a total of over 16.7 million colors (256 multiplied by 256 multiplied by 256, or 256 cubed).

To create the same maroon and yellow background colors from before, we could replace the keyword color values with hexadecimal color values, as seen here.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | .task {  background: #800000;  }  .count {  background: #ff0;  } |

Hexadecimal color values have been around for a while, and they have become fairly popular because they offer a large number of color options. They are, however, a little difficult to work with, especially if you’re not too familiar with them. Fortunately Adobe has created [Adobe Kuler](https://kuler.adobe.com/), a free application that provides a color wheel to help us find any color we want and its corresponding hexadecimal value.

Additionally, most image editing applications, such as Adobe Photoshop, provide the capability to locate hexadecimal color values.

**Fig 3**

The color picker tool within Adobe Photoshop displays the hexadecimal and RGB color values

#### RGB & RGBa Colors

RGB color values are stated using the rgb() function, which stands for red, green, and blue. The function accepts three comma-separated values, each of which is an integer from 0 to 255. A value of 0 would be pure black; a value of 255 would be pure white.

As we might expect, the first value within the rgb() function represents the red channel, the second value represents the green channel, and the third value represents the blue channel.

If we were to recreate the shade of orange from before as an RGB color value, it would be represented as rgb(255, 102, 0).

Also, using the same maroon and yellow background colors from before, we could replace the keyword or hexadecimal color values with RGB color values.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | .task {  background: rgb(128, 0, 0);  }  .count {  background: rgb(255, 255, 0);  } |

RGB color values may also include an alpha, or transparency, channel by using the rgba() function. The rgba() function requires a fourth value, which must be a number between 0 and 1, including decimals. A value of 0 creates a fully transparent color, meaning it would be invisible, and a value of 1 creates a fully opaque color. Any decimal value in between 0 and 1 would create a semi-transparent color.

If we wanted our shade of orange to appear 50% opaque, we would use an RGBa color value of rgba(255, 102, 0, .5).

We can also change the opacity of our maroon and yellow background colors. The following code sets the maroon background color to 25% opaque and leaves the yellow background color 100% opaque.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | .task {  background: rgba(128, 0, 0, .25);  }  .count {  background: rgba(255, 255, 0, 1);  } |

RGB color values are becoming more popular, especially due to the ability to create semi-transparent colors using RGBa.

#### HSL & HSLa Colors

HSL color values are stated using the hsl() function, which stands for hue, saturation, and lightness. Within the parentheses, the function accepts three comma-separated values, much like rgb().

The first value, the hue, is a unitless number from 0 to 360. The numbers0through 360 represent the color wheel, and the value identifies the degree of a color on the color wheel.

The second and third values, the saturation and lightness, are percentage values from 0 to 100%. The saturation value identifies how saturated with color the hue is, with 0 being grayscale and 100% being fully saturated. The lightness identifies how dark or light the hue value is, with 0 being completely black and 100% being completely white.

Returning to our shade of orange, as an HSL color value it would be written as hsl(24, 100%, 50%).

Our maroon and yellow background colors can also be stated as HSL color values, as shown here.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | .task {  background: hsl(0, 100%, 25%);  }  .count {  background: hsl(60, 100%, 50%);  } |

HSL color values, like RGBa, may also include an alpha, or transparency, channel with the use of the hsla() function. The behavior of the alpha channel is just like that of the rgba() function. A fourth value between 0 and 1, including decimals, must be added to the function to identify the degree of opacity.

Our shade of orange as an HSLa color set to 50% opaque would be represented as hsla(24, 100%, 50%, .5).

The same 25% opaque maroon background color and 100% opaque yellow background color from before would look like the following as HSLa color values.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | .task {  background: hsla(0, 100%, 25%, .25);  }  .count {  background: hsla(60, 100%, 50%, 1);  } |

The HSL color value is the newest color value available within CSS. Due to its age and support within browsers, though, it isn’t as widely used as the other values.

For the time being, hexadecimal color values remain the most popular as they are widely supported; though when an alpha channel for transparency is needed, RGBa color values are preferred. These preferences may change in the future, but for now we’ll use hexadecimal and RGBa color values.

### Lengths

Length values within CSS are similar to colors in that there are a handful of different types of values for [length](https://developer.mozilla.org/en-US/docs/Web/CSS/length), all of which serve distinct purposes. Length values come in two different forms, absolute and relative, each of which uses different units of measurement.

We’re going to stick to the more common—and more straightforward—values at the moment, as more complex values will provide much more power than we need for now.

#### Absolute Lengths

Absolute length values are the simplest length values, as they are fixed to a physical measurement, such as inches, centimeters, or millimeters. The most popular absolute unit of measurement is known as the pixel and is represented by the px unit notation.

##### Pixels

The pixel is equal to 1/96th of an inch; thus there are 96 pixels in an inch. The exact measurement of a pixel, however, may vary slightly between high-density and low-density viewing devices.

Pixels have been around for quite some time and are commonly used with a handful of different properties. The code here is using pixels to set the font size of all paragraphs to 14 pixels.

|  |  |
| --- | --- |
| 1  2  3  4 | p {  font-size: 14px;  } |

With the changing landscape of viewing devices and their varying screen sizes, pixels have lost some of their popularity. As an absolute unit of measurement, they don’t provide too much flexibility. Pixels are, however, trustworthy and great for getting started. We’re going to lean on them quite a bit as we’re learning the ropes of HTML and CSS.

#### Relative Lengths

In addition to absolute length values, there are also relative length values. Relative length values are a little more complicated, as they are not fixed units of measurement; they rely on the length of another measurement.

##### Percentages

Percentages, represented by the % unit notation, are one of the most popular relative values. Percentage lengths are defined in relation to the length of another object. For example, to set the width of an element to 50%, we have to know the width of its parent element, the element it is nested within, and then identify 50% of the parent element’s width.

|  |  |
| --- | --- |
| 1  2  3  4 | .col {  width: 50%;  } |

Here we’ve set the width of the element with the class attribute value of col to 50%. That 50% will be calculated relative to the width of the element’s parent.

Percentages are extremely helpful for setting the height and width of elements and building out a web page’s layout. We’re going to rely on them often to help us out in these areas.

##### Em

The em unit is also a very popular relative value. The em unit is represented by the em unit notation, and its length is calculated based on an element’s font size.

A single em unit is equivalent to an element’s font size. So, for example, if an element has a font size of 14 pixels and a width set to 5em, the width would equal 70 pixels (14 pixels multiplied by 5).

|  |  |
| --- | --- |
| 1  2  3  4  5 | .banner {  font-size: 14px;  width: 5em;  } |

When a font size is not explicitly stated for an element, the em unit will be relative to the font size of the closest parent element with a stated font size.

The em unit is often used for styling text, including font sizes, as well as spacing around text, including margins and paddings. We’ll explore text a bit more in Lesson 6, “[Working with Typography](http://learn.shayhowe.com/html-css/working-with-typography/).”

There are a lot more absolute and relative units of measurement than those mentioned here. However, these three—pixels, percentages, and em units—are the most popular and the ones we’re going to primarily use.

## Summary[#summary](http://learn.shayhowe.com/html-css/getting-to-know-css/#summary)

Sadly our Styles Conference website lay dormant this lesson. We focused on the foundations of CSS, covering exactly how it works and some common values we’re sure to use.

To briefly recap, within this lesson we’ve discussed the following:

* How style sheets cascade from the top to the bottom of a file
* What specificity is and how we can calculate it
* How to combine selectors to target specific elements or groups of elements
* How to use multiple classes on a single element to layer on different styles for more modular code
* The different color values available to use within CSS, including keyword, hexadecimal, RGB, and HSL values
* The different length values available to use within CSS, including pixels, percentages, and em units

We still have a lot to cover, but the fundamentals are starting to fall into place. Within the next few lessons we’ll continue to dive in to CSS, and our website will really begin to take shape.

###### Lesson 4

# Opening the Box Model

We’ve familiarized ourselves with HTML and CSS; we know what they look like and how to accomplish some of the basics. Now we’re going to go a bit deeper and look at exactly how elements are displayed on a page and how they are sized.

In the process we’ll discuss what is known as the box model and how it works with HTML and CSS. We’re also going to look at a few new CSS properties and use some of the length values we covered in Lesson 3. Let’s begin.

## How Are Elements Displayed?[#how-are-elements-displayed](http://learn.shayhowe.com/html-css/opening-the-box-model/#how-are-elements-displayed)

Before jumping into the box model, it helps to understand how elements are displayed. In Lesson 2 we covered the difference between block-level and inline-level elements. To quickly recap, block-level elements occupy any available width, regardless of their content, and begin on a new line. Inline-level elements occupy only the width their content requires and line up on the same line, one after the other. Block-level elements are generally used for larger pieces of content, such as headings and structural elements. Inline-level elements are generally used for smaller pieces of content, such as a few words selected to be bold or italicized.

### Display

Exactly how elements are displayed—as block-level elements, inline elements, or something else—is determined by the display property. Every element has a default display property value; however, as with all other property values, that value may be overwritten. There are quite a few values for the display property, but the most common are block, inline, inline-block, and none.

We can change an element’s display property value by selecting that element within CSS and declaring a new display property value. A value of block will make that element a block-level element.

|  |  |
| --- | --- |
| 1  2  3  4 | p {  display: block;  } |

A value of inline will make that element an inline-level element.

|  |  |
| --- | --- |
| 1  2  3  4 | p {  display: inline;  } |

Things get interesting with the inline-block value. Using this value will allow an element to behave as a block-level element, accepting all box model properties (which we’ll cover soon). However, the element will be displayed in line with other elements, and it will not begin on a new line by default.

|  |  |
| --- | --- |
| 1  2  3  4 | p {  display: inline-block;  } |

#### Display Inline-Block Demo

#### The Space Between Inline-Block Elements

One important distinction with inline-block elements is that they are not always touching, or displayed directly against one another. Usually a small space will exist between two inline-block elements. This space, though perhaps annoying, is normal. We’ll discuss why this space exists and how to remove it in the next lesson.

Lastly, using a value of none will completely hide an element and render the page as if that element doesn’t exist. Any elements nested within this element will also be hidden.

|  |  |
| --- | --- |
| 1  2  3  4 | div {  display: none;  } |

Knowing how elements are displayed and how to change their display is fairly important, as the display of an element has implications on how the box model is rendered. As we discuss the box model, we’ll be sure to look at these different implications and how they can affect the presentation of an element.

## What Is the Box Model?[#what-is-the-box-model](http://learn.shayhowe.com/html-css/opening-the-box-model/#what-is-the-box-model)

According to the [box model](http://css-tricks.com/the-css-box-model/) concept, every element on a page is a rectangular box and may have width, height, padding, borders, and margins.

That’s worth repeating: **Every element on a page is a rectangular box.**

**Fig 4**

When we look at each element individually, we can see how they are all rectangular, regardless of their presented shapes

Every element on every page conforms to the box model, so it’s incredibly important. Let’s take a look at it, along with a few new CSS properties, to better understand what we are working with.

## Working with the Box Model[#box-model](http://learn.shayhowe.com/html-css/opening-the-box-model/#box-model)

Every element is a rectangular box, and there are several properties that determine the size of that box. The core of the box is defined by the width and height of an element, which may be determined by the display property, by the contents of the element, or by specified width and height properties. padding and then border expand the dimensions of the box outward from the element’s width and height. Lastly, any margin we have specified will follow the border.

Each part of the box model corresponds to a CSS property: width, height, padding, border, and margin.

Let’s look these properties inside some code:

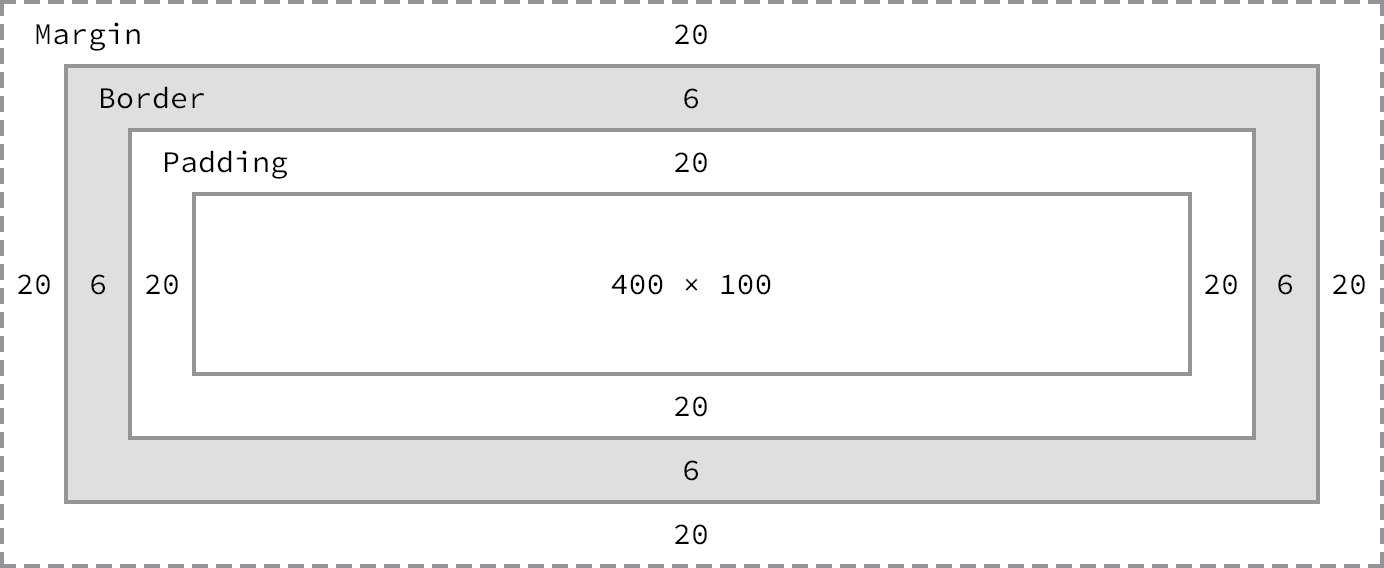
|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | div {  border: 6px solid #949599;  height: 100px;  margin: 20px;  padding: 20px;  width: 400px;  } |

According to the box model, the total width of an element can be calculated using the following formula:

|  |  |
| --- | --- |
| 1  2 | margin-right + border-right + padding-right + width + padding-left + border-left + margin-left |

In comparison, according to the box model, the total height of an element can be calculated using the following formula:

|  |  |
| --- | --- |
| 1  2 | margin-top + border-top + padding-top + height + padding-bottom + border-bottom + margin-bottom |

**Fig 4**

The box model broken down, including a base height and width plus paddings, borders, and margins

Using the formulas, we can find the total height and width of our example code.

* **Width:** 492px = 20px + 6px + 20px + 400px + 20px + 6px + 20px
* **Height:** 192px = 20px + 6px + 20px + 100px + 20px + 6px + 20px

The box model is without question one of the more confusing parts of HTML and CSS. We set a width property value of 400 pixels, but the actual width of our element is 492 pixels. By default the box model is additive; thus to determine the actual size of a box we need to take into account padding, borders, and margins for all four sides of the box. Our width not only includes the width property value, but also the size of the left and right padding, left and right borders, and left and right margins.

So far a lot of these properties might not make a whole lot of sense, and that’s all right. To clarify things, let’s take a close look at all of the properties—width, height, padding, border, and margin—that go into forming the box model.

### Width & Height

Every element has default width and height. That width and height may be 0 pixels, but browsers, by default, will render every element with size. Depending on how an element is displayed, the default width and height may be adequate. If an element is key to the layout of a page, it may require specified width and height property values. In this case, the property values for non-inline elements may be specified.

#### Width

The default width of an element depends on its display value. Block-level elements have a default width of 100%, consuming the entire horizontal space available. Inline and inline-block elements expand and contract horizontally to accommodate their content. Inline-level elements cannot have a fixed size, thus the width and height properties are only relevant to non-inline elements. To set a specific width for a non-inline element, use the width property:

|  |  |
| --- | --- |
| 1  2  3  4 | div {  width: 400px;  } |

#### Height

The default height of an element is determined by its content. An element will expand and contract vertically as necessary to accommodate its content. To set a specific height for a non-inline element, use the height property:

|  |  |
| --- | --- |
| 1  2  3  4 | div {  height: 100px;  } |

#### Sizing Inline-Level Elements

Please keep in mind that inline-level elements will not accept the width and height properties or any values tied to them. Block and inline-block elements will, however, accept the width and height properties and their corresponding values.

### Margin & Padding

Depending on the element, browsers may apply default margins and padding to an element to help with legibility and clarity. We will generally see this with text-based elements. The default margins and padding for these elements may differ from browser to browser and element to element. In Lesson 1 we discussed using a [CSS reset](http://learn.shayhowe.com/html-css/building-your-first-web-page/#using-css-resets) to tone all of these default values down to zero. Doing so allows us to work from the ground up and to specify our own values.

#### Margin

The margin property allows us to set the amount of space that surrounds an element. Margins for an element fall outside of any border and are completely transparent in color. Margins can be used to help position elements in a particular place on a page or to provide breathing room, keeping all other elements a safe distance away. Here’s the margin property in action:

|  |  |
| --- | --- |
| 1  2  3  4 | div {  margin: 20px;  } |

One oddity with the margin property is that vertical margins, top and bottom, are not accepted by inline-level elements. These vertical margins are, however, accepted by block-level and inline-block elements.

#### Padding

The padding property is very similar to the margin property; however, it falls inside of an element’s border, should an element have a border. The padding property is used to provide spacing directly within an element. Here’s the code:

|  |  |
| --- | --- |
| 1  2  3  4 | div {  padding: 20px;  } |

The padding property, unlike the margin property, works vertically on inline-level elements. This vertical padding may blend into the line above or below the given element, but it will be displayed.

#### Margin & Padding on Inline-Level Elements

Inline-level elements are affected a bit differently than block and inline-block elements when it comes to margins and padding. Margins only work horizontally—left and right—on inline-level elements. Padding works on all four sides of inline-level elements; however, the vertical padding—the top and bottom—may bleed into the lines above and below an element.

Margins and padding work like normal for block and inline-block elements.

#### Margin & Padding Declarations

In CSS, there is more than one way to declare values for certain properties. We can use longhand, listing multiple properties and values one after the other, in which each value has its own property. Or we can use shorthand, listing multiple values with one property. Not all properties have a shorthand alternative, so we must make sure we are using the correct property and value structure.

The margin and padding properties come in both longhand and shorthand form. When using the shorthand margin property to set the same value for all four sides of an element, we specify one value:

|  |  |
| --- | --- |
| 1  2  3  4 | div {  margin: 20px;  } |

To set one value for the top and bottom and another value for the left and right sides of an element, specify two values: top and bottom first, then left and right. Here we are placing margins of 10 pixels on the top and bottom of a <div> and margins of 20 pixels on the left and right:

|  |  |
| --- | --- |
| 1  2  3  4 | div {  margin: 10px 20px;  } |

To set unique values for all four sides of an element, specify those values in the order of top, right, bottom, and left, moving clockwise. Here we are placing margins of 10 pixels on the top of a <div>, 20 pixels on the right, 0 pixels on the bottom, and 15 pixels on the left.

|  |  |
| --- | --- |
| 1  2  3  4 | div {  margin: 10px 20px 0 15px;  } |

Using the margin or padding property alone, with any number of values, is considered shorthand. With longhand, we can set the value for one side at a time using unique properties. Each property name (in this case margin or padding) is followed by a dash and the side of the box to which the value is to be applied: top, right, bottom, or left. For example, the padding-left property accepts only one value and will set the left padding for that element; the margin-top property accepts only one value and will set the top margin for that element.

|  |  |
| --- | --- |
| 1  2  3  4  5 | div {  margin-top: 10px;  padding-left: 6px;  } |

When we wish to identify only one margin or padding value, it is best to use the longhand properties. Doing so keeps our code explicit and helps us to avoid any confusion down the road. For example, did we really want to set the top, right, and left sides of the element to have margins of 0 pixels, or did we really only want to set the bottom margin to 10 pixels? Using longhand properties and values here helps to make our intentions clear. When dealing with three or more values, though, shorthand is incredibly helpful.

#### Margin & Padding Colors

The margin and padding properties are completely transparent and do not accept any color values. Being transparent, though, they show the background colors of relative elements. For margins, we see the background color of the parent element, and for padding, we see the background color of the element the padding is applied to.

### Borders

Borders fall between the padding and margin, providing an outline around an element. The border property requires three values: width, style, and color. Shorthand values for the border property are stated in that order—width, style, color. In longhand, these three values can be broken up into the border-width, border-style, and border-color properties. These longhand properties are useful for changing, or overwriting, a single border value.

The width and color of borders can be defined using common CSS units of length and color, as discussed in Lesson 3.

Borders can have [different appearances](http://www.quackit.com/html/codes/html_borders.cfm). The most common style values are solid, double, dashed, dotted, and none, but there are several others to choose from.

Here is the code for a 6-pixel-wide, solid, gray border that wraps around all four sides of a <div>:

|  |  |
| --- | --- |
| 1  2  3  4 | div {  border: 6px solid #949599;  } |

#### Borders Demo

#### Individual Border Sides

As with the margin and padding properties, borders can be placed on one side of an element at a time if we’d like. Doing so requires new properties: border-top, border-right, border-bottom, and border-left. The values for these properties are the same as those of the border property alone: width, style, and color. If we want, we can make a border appear only on the bottom of an element:

|  |  |
| --- | --- |
| 1  2  3  4 | div {  border-bottom: 6px solid #949599;  } |

Additionally, styles for individual border sides may be controlled at an even finer level. For example, if we wish to change only the width of the bottom border we can use the following code:

|  |  |
| --- | --- |
| 1  2  3  4 | div {  border-bottom-width: 12px;  } |

These highly specific longhand border properties include a series of hyphen-separated words starting with the border base, followed by the selected side—top, right, bottom, or left—and then width, style, or color, depending on the desired property.

#### Border Radius

While we’re looking at borders and their different properties, we need to examine the border-radius property, which enables us to round the corners of an element.

The border-radius property accepts length units, including percentages and pixels, that identify the radius by which the corners of an element are to be rounded. A single value will round all four corners of an element equally; two values will round the top-left/bottom-right and top-right/bottom-left corners in that order; four values will round the top-left, top-right, bottom-right, and bottom-left corners in that order.

When considering the order in which multiple values are applied to the border-radius property (as well as the margin and padding properties), remember that they move in a clockwise fashion starting at the top left of an element.

|  |  |
| --- | --- |
| 1  2  3  4 | div {  border-radius: 5px;  } |

#### Border Radius Demo

￼The border-radius property may also be broken out into longhand properties that allow us to change the radii of individual corners of an element. These longhand properties begin with border, continue with the corner’s vertical location (top or bottom) and the corner’s horizontal location (left or right), and then end with radius. For example, to change the top-right corner radius of a <div>, the border-top-right-radius property can be used.

|  |  |
| --- | --- |
| 1  2  3  4 | div {  border-top-right-radius: 5px;  } |

### Box Sizing

Until now the box model has been an additive design. If you set the width of an element to 400 pixels and then add 20 pixels of padding and a border of 10 pixels on every side, the actual full width of the element becomes 460 pixels. Remember, we need to add the width, padding, and border property values together to get the actual, full width of an element.

The box model may, however, be changed to support different calculations. CSS3 introduced the box-sizing property, which allows us to change exactly how the box model works and how an element’s size is calculated. The property accepts three primary values—content-box, padding-box, and border-box—each of which has a slightly different impact on how the box size is calculated.

#### Content Box

The content-box value is the default value, leaving the box model as an additive design. If we don’t use the box-sizing property, this will be the default value for all elements. The size of an element begins with the width and height properties, and then any padding, border, or margin property values are added on from there.

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | div {  -webkit-box-sizing: content-box;  -moz-box-sizing: content-box;  box-sizing: content-box;  } |

#### Browser-Specific Properties & Values

What are all those hyphens and letters on the box-sizing property?

As CSS3 was introduced, browsers gradually began to support different properties and values, including the box-sizing property, by way of vendor prefixes. As parts of the CSS3 specification are finalized and new browser versions are released, these vendor prefixes become less and less relevant. As time goes on, vendor prefixes are unlikely to be a problem; however, they still provide support for some of the older browsers that leveraged them. We may run across them from time to time, and we may even want to use them should we wish to support older browsers.

Vendor prefixes may be seen on both properties and values, all depending on the CSS specification. Here they are shown on the box-sizing property. Browser vendors were free to chose when to use a prefix and when not to. Thus, some properties and values require vendor prefixes for certain browser vendors but not for others.

Moving forward, when a property or value needs a vendor prefix, the prefix will only be used in the introduction of that property or value (in the interest of keeping our code digestible and concise). Do not forget to add the necessary vendor prefixes when you’re actually writing the code.

For reference, the most common vendor prefixes are outlined here:

* Mozilla Firefox: -moz-
* Microsoft Internet Explorer: -ms-
* Webkit (Google Chrome and Apple Safari): -webkit-

#### Padding Box

The padding-box value alters the box model by including any padding property values within the width and height of an element. When using the padding-box value, if an element has a width of 400 pixels and a padding of 20 pixels around every side, the actual width will remain 400 pixels. As any padding values increase, the content size within an element shrinks proportionately.

If we add a border or margin, those values will be added to the width or height properties to calculate the full box size. For example, if we add a border of 10 pixels and a padding of 20 pixels around every side of the element with a width of 400 pixels, the actual full width will become 420 pixels.

|  |  |
| --- | --- |
| 1  2  3  4 | div {  box-sizing: padding-box;  } |

#### Border Box

Lastly, the border-box value alters the box model so that any border or padding property values are included within the width and height of an element. When using the border-box value, if an element has a width of 400 pixels, a padding of 20 pixels around every side, and a border of 10 pixels around every side, the actual width will remain 400 pixels.

If we add a margin, those values will need to be added to calculate the full box size. No matter which box-sizing property value is used, any margin values will need to be added to calculate the full size of the element.

|  |  |
| --- | --- |
| 1  2  3  4 | div {  box-sizing: border-box;  } |

**Fig 4**

Different box-sizing values allow the width of an element—and its box—to be calculated from different areas

#### Picking a Box Size

Generally speaking, the best box-sizing value to use is border-box. The border-box value makes our math much, much easier. If we want an element to be 400 pixels wide, it is, and it will remain 400 pixels wide no matter what padding or border values we add to it.

Additionally, we can easily mix length values. Say we want our box to be 40% wide. Adding a padding of 20 pixels and a border of 10 pixels around every side of an element isn’t difficult, and we can still guarantee that the actual width of our box will remain 40% despite using pixel values elsewhere.

The only drawback to using the box-sizing property is that as part of the CSS3 specification, it isn’t supported in every browser; it especially lacks support in older browsers. Fortunately this is becoming less and less relevant as new browsers are released. Chances are we’re safe to use the box-sizing property, but should we notice any issues, it’s worth looking into which browser those issues are occurring with.

## Developer Tools[#developer-tools](http://learn.shayhowe.com/html-css/opening-the-box-model/#developer-tools)

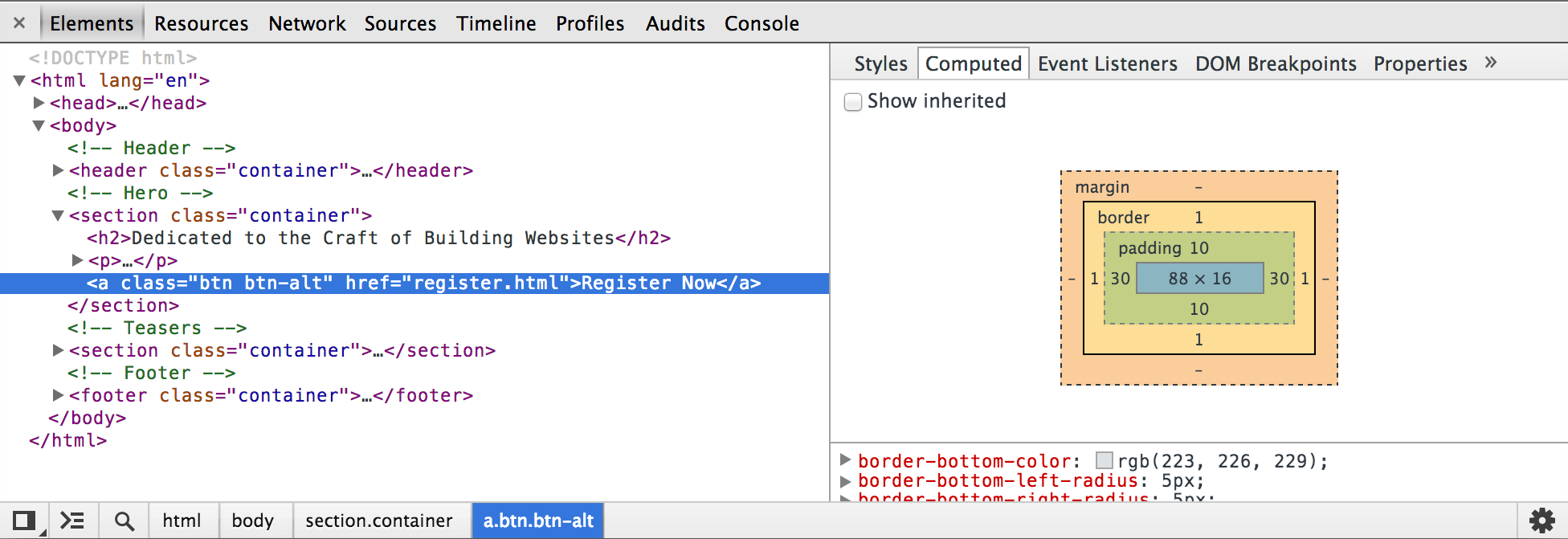
Most browsers have what are known as Developer Tools. These tools allow us to inspect an element on a page, see where that element lives within the HTML document, and see what CSS properties and values are being applied to it. Most of these tools also include a box model diagram to show the computed size of an element.

To see the Developer Tools in Google Chrome, click “View” within the menu bar and navigate to “Developer” and then “Developer Tools.” This loads a drawer at the bottom of the browser window that provides a handful of tools for inspecting our code.

Clicking the magnifying glass at the bottom of this drawer enables us to hover over and then click on different elements on the page to review more information about them.

After selecting an element, we’ll see a handful of tabs on the right-hand side of the Elements panel within our Developer Tools. Selecting the “Computed” tab will show us a breakdown of the box model for our selected element.

Play around with the Developer Tools, be it in Google Chrome, Mozilla Firefox, Apple Safari, or other browsers; there is much to learn from looking at our code. I generally leave the Developer Tools open at all times when writing HTML and CSS. And I frequently inspect the code of other websites to see how they are built, too.

**Fig 4**

The Google Chrome Developer Tools, which help us to inspect the HTML and CSS on any page

The box model is one of the most confusing parts of learning how to write HTML and CSS. It is also one of the most powerful parts of HTML and CSS, and once we have it mastered, most everything else—like positioning content—will come to us fairly easily.

## In Practice[#practice-1](http://learn.shayhowe.com/html-css/opening-the-box-model/#practice-1)

Let’s jump back into our Styles Conference website to center it on the page and add some more content.

1. Let’s start by adjusting our box size to use the border-box version of the box model, which will make sizing all of our elements much easier. Within our main.css file, just below our reset, let’s add a comment to identify the code for what will become our grid and help determine the layout of our website. We’re putting this below our reset so that it falls in the proper position within the cascade.

From there, we can use the universal selector, \*, along with universal pseudo-elements, \*:before and \*:after, to select every imaginable element and change the box-sizing to border-box. Remember, we’re going to want to include the necessary vendor prefixes for the box-sizing property, as it is a relatively new property.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | /\*  ========================================  Grid  ========================================  \*/  \*,  \*:before,  \*:after {  -webkit-box-sizing: border-box;  -moz-box-sizing: border-box;  box-sizing: border-box;  } |

1. Next we’ll want to create a class that will serve as a container for our elements. We can use this container class on different elements to set a common width, center the elements on the page, and apply some common horizontal padding.

Just below our universal selector rule set, let’s create a selector with a class of container. Within this selector let’s set our width to 960 pixels, our left and right padding to 30 pixels, our top and bottom margins to 0, and our left and right margins to auto.

Setting a width tells the browser definitively how wide any element with the class of container should be. Using a left and right margin of auto in conjunction with this width lets the browser automatically figure out equal left and right margins for the element, thus centering it on the page. Lastly, the left and right padding ensures that our content isn’t sitting directly on the edge of the element and provides a little breathing room for the content.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | .container {  margin: 0 auto;  padding-left: 30px;  padding-right: 30px;  width: 960px;  } |

1. Now that we have a container class available to use, let’s go ahead and apply the class of container throughout our HTML to the <header> and <footer> elements on each page, including the index.html, speakers.html, schedule.html, venue.html, and register.html files.

|  |  |
| --- | --- |
| 1  2  3  4 | <header class="container">...</header>  <footer class="container">...</footer> |

1. While we’re at it, let’s go ahead and center the rest of the content on our pages. On the home page, our index.html file, let’s add the class of container to each <section> element on the page, one for our hero section (the section that introduces our conference) and one for our teasers section.

|  |  |
| --- | --- |
| 1  2 | <section class="container">...</section> |

1. Additionally, let’s wrap all of the <h1> elements on each page with a <section> element with the class of container.

|  |  |
| --- | --- |
| 1  2  3  4 | <section class="container">  <h1>...</h1>  </section> |

1. We’ll come back and adjust these elements and classes later, but for now we’re headed in the right direction.
2. Now that all of our content is centered, let’s create some vertical spacing between elements. For starters let’s place a 22-pixel bottom margin on a few of our heading and paragraph elements. We’ll place and comment on these typography styles below our grid styles.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | /\*  ========================================  Typography  ========================================  \*/  h1, h3, h4, h5, p {  margin-bottom: 22px;  } |

1. We intentionally skipped <h2> and <h6> elements, as the design does not call for margins on <h2> elements and as we won’t be using any <h6> elements at this time.
2. Let’s also try our hand at creating a border and some rounded corners. We’ll start by placing a button within the top <section> element on our home page, just below the header.

Previously we added an <a> element within this <section> element. Let’s add the classes of btn and btn-alt to this anchor.

|  |  |
| --- | --- |
| 1  2 | <a class="btn btn-alt">...</a> |

Now let’s create some styles for those classes within our CSS. Below our typography rule set, let’s create a new section of the CSS file for buttons.

To begin let’s add the btn class and apply some common styles that can be shared across all buttons. We’ll want all of our buttons to have a 5-pixel border-radius. They should be displayed as inline-block elements so we can add padding around all four sides without issue; we’ll remove any margin.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | /\*  ========================================  Buttons  ========================================  \*/  .btn {  border-radius: 5px;  display: inline-block;  margin: 0;  } |

We’ll also want to include styles specific to this button, which we’ll do by using the btn-alt class. Here we’ll add a 1-pixel, solid, gray border with 10 pixels of padding on the top and bottom of the button and 30 pixels of padding on the left and right of the button.

|  |  |
| --- | --- |
| 1  2  3  4  5 | .btn-alt {  border: 1px solid #dfe2e5;  padding: 10px 30px;  } |

Using both the btn and btn-alt classes on the same <a> element allows these styles to be layered on, rendering all of the styles on a single element.

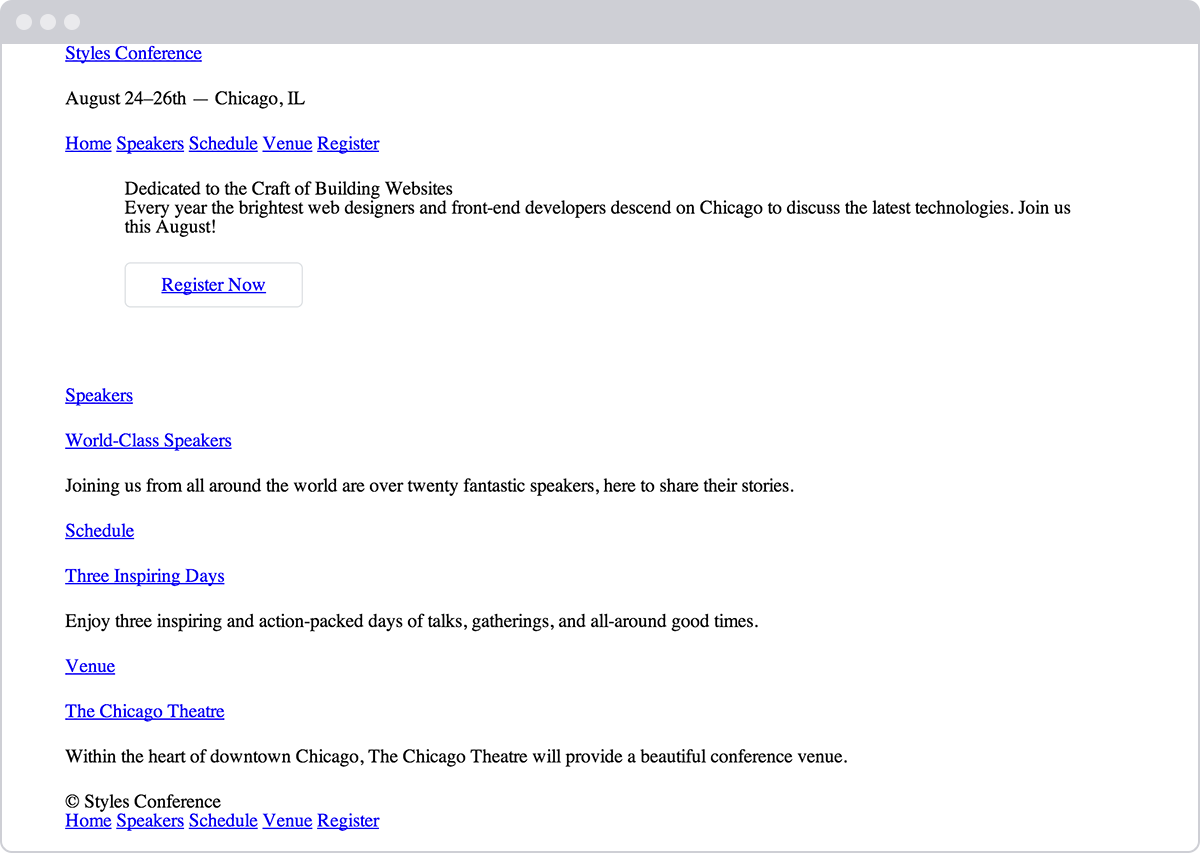
1. Because we’re working on the home page, let’s also add a bit of padding to the <section> element that contains our <a> element with the classes of btn and btn-alt. We’ll do so by adding a class attribute value of hero to the <section> element, alongside the container class attribute value, as this will be the leading section of our website.

|  |  |
| --- | --- |
| 1  2  3  4 | <section class="hero container">  ...  </section> |

1. Next we’ll want to create a new section within our CSS file for home page styles, and, once we’re ready, we’ll use the class of hero to apply padding around all four sides of the <section> element.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | /\*  ========================================  Home  ========================================  \*/  .hero {  padding: 22px 80px 66px 80px;  } |

Our website is starting to come together, especially the home page.

**Fig 4**

Our Styles Conference home page, taking shape after a few updates

### Demo & Source Code

Below you may view the Styles Conference website in its current state, as well as download the source code for the website in its current state.

[**View the Styles Conference Website**](http://learn.shayhowe.com/practice/opening-the-box-model/index.html) or [Download the Source Code](http://learn.shayhowe.com/practice/opening-the-box-model.zip) (Zip file)

#### The Universal Selector

In the first step of this exercise we were introduced to the universal selector. In CSS the asterisk, \*, is the universal selector, which selects every element. Rather than listing every single element imaginable, we can use the asterisk as a catch-all to select all elements for us.

The :before and :after pseudo-elements also mentioned in this step are elements that can be dynamically generated with CSS. We’re not going to be using these elements within our project; however, when using the universal selector it’s a good practice to also include these pseudo-elements in case they should ever appear.

## Summary[#summary](http://learn.shayhowe.com/html-css/opening-the-box-model/#summary)

Take a second and pat yourself on the back. I’ll wait.

Learning all the different parts of the box model is no small feat. These concepts, although briefly introduced, take quite a bit of time to fully master, and we’re on the right path toward doing so.

In brief, within this lesson we talked about the following:

* How different elements are displayed
* What the box model is and why it’s important
* How to change the size, including the height and width, of elements
* How to add margin, padding, and borders to elements
* How to change the box sizing of elements and the effects this has on the box model

Now that we have a better understanding of how elements are displayed and sized, it’s time to move into positioning these elements.

One of the best things about CSS is that it gives us the ability to position content and elements on a page in nearly any imaginable way, bringing structure to our designs and helping make content more digestible.

There are a few different types of positioning within CSS, and each has its own application. In this chapter we’re going to take a look at a few different use cases—creating reusable layouts and uniquely positioning one-off elements—and describe a few ways to go about each.

## Positioning with Floats[#floats](http://learn.shayhowe.com/html-css/positioning-content/#floats)

One way to position elements on a page is with the float property. The float property is pretty versatile and can be used in a number of different ways.

Essentially, the float property allows us to take an element, remove it from the normal flow of a page, and [position it](http://www.smashingmagazine.com/2007/05/01/css-float-theory-things-you-should-know/) to the left or right of its parent element. All other elements on the page will then flow around the floated element. An <img> element floated to the side of a few paragraphs of text, for example, will allow the paragraphs to wrap around the image as necessary.

When the float property is used on multiple elements at the same time, it provides the ability to create a layout by floating elements directly next to or opposite each other, as seen in multiple-column layouts.

The float property accepts a few values; the two most popular values are left and right, which allow elements to be floated to the left or right of their parent element.

|  |  |
| --- | --- |
| 1  2  3  4 | img {  float: left;  } |

### Floats in Practice

Let’s create a common page layout with a header at the top, two columns in the center, and a footer at the bottom. Ideally this page would be marked up using the <header>, <section>, <aside>, and <footer> elements as discussed in Lesson 2, “[Getting to Know HTML](http://learn.shayhowe.com/html-css/getting-to-know-html/).” Inside the <body> element, the HTML may look like this:

|  |  |
| --- | --- |
| 1  2  3  4  5 | <header>...</header>  <section>...</section>  <aside>...</aside>  <footer>...</footer> |

#### Layout without Floats Demo

Here the <section> and <aside> elements, as block-level elements, will be stacked on top of one another by default. However, we want these elements to sit side by side. By floating the <section> to the left and the <aside> to the right, we can position them as two columns sitting opposite one another. Our CSS should look like this:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | section {  float: left;  }  aside {  float: right;  } |

For reference, when an element is floated, it will float all the way to the edge of its parent element. If there isn’t a parent element, the floated element will then float all the way to the edge of the page.

When we float an element, we take it out of the normal flow of the HTML document. This causes the width of that element to default to the width of the content within it. Sometimes, such as when we’re creating columns for a reusable layout, this behavior is not desired. It can be corrected by adding a fixed width property value to each column. Additionally, to prevent floated elements from touching one another, causing the content of one to sit directly next to the content of the other, we can use the margin property to create space between elements.

Here, we are extending the previous code block, adding a margin and width to each column to better shape our desired outcome.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | section {  float: left;  margin: 0 1.5%;  width: 63%;  }  aside {  float: right;  margin: 0 1.5%;  width: 30%;  } |

#### Layout with Floats Demo

#### Floats May Change an Element’s Display Value

When floating an element, it is also important to recognize that an element is removed from the normal flow of a page, and that may change an element’s default display value. The float property relies on an element having a display value of block, and may alter an element’s default display value if it is not already displayed as a block-level element.

For example, an element with a display value of inline, such as the <span> inline-level element, ignores any height or width property values. However, should that inline-level element be floated, its display value will be changed to block, and it may then accept height or width property values.

As we float elements we must keep an eye on how their display property values are affected.

With two columns we can float one column to the left and another to the right, but with more columns we must change our approach. Say, for example, we’d like to have a row of three columns between our <header> and <footer> elements. If we drop our <aside> element and use three <section> elements, our HTML might look like this:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | <header>...</header>  <section>...</section>  <section>...</section>  <section>...</section>  <footer>...</footer> |

To position these three <section> elements in a three-column row, instead of floating one column to the left and one column to the right, we’ll float all three <section> elements to the left. We’ll also need to adjust the width of the <section> elements to account for the additional columns and to get them to sit one next to the other.

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | section {  float: left;  margin: 0 1.5%;  width: 30%;  } |

Here we have three columns, all with equal width and margin values and all floated to the left.

#### Three-column Layout with Floats Demo

### Clearing & Containing Floats

The float property was originally designed to allow content to wrap around images. An image could be floated, and all of the content surrounding that image could then naturally flow around it. Although this works great for images, the float property was never actually intended to be used for layout and positioning purposes, and thus it comes with a few pitfalls.

One of those pitfalls is that occasionally the proper styles will not render on an element that it is sitting next to or is a parent element of a floated element. When an element is floated, it is taken out of the normal flow of the page, and, as a result, the styles of elements around that floated element can be negatively impacted.

Often margin and padding property values aren’t interpreted correctly, causing them to blend into the floated element; other properties can be affected, too.

Another pitfall is that sometimes unwanted content begins to wrap around a floated element. Removing an element from the flow of the document allows all the elements around the floated element to wrap and consume any available space around the floated element, which is often undesired.

With our previous two-column example, after we floated the <section> and <aside> elements, and before we set a width property value on either of them, the content within the <footer> element would have wrapped in between the two floated elements above it, filling in any available space. Consequently, the <footer> element would have sat in the gutter between the <section> and <aside> elements, consuming the available space.

#### Layout without Cleared or Contained Floats Demo

To prevent content from wrapping around floated elements, we need to clear, or contain, those floats and return the page to its normal flow. We’ll proceed by looking at how to clear floats, and then we’ll take a look at how to contain floats.

#### Clearing Floats

Clearing floats is accomplished using the clear property, which accepts a few different values: the most commonly used values being left, right, and both.

|  |  |
| --- | --- |
| 1  2  3  4 | div {  clear: left;  } |

The left value will clear left floats, while the right value will clear right floats. The both value, however, will clear both left and right floats and is often the most ideal value.

Going back to our previous example, if we use the clear property with the value of both on the <footer> element, we are able to clear the floats. It is important that this clear be applied to an element appearing after the floated elements, not before, to return the page to its normal flow.

|  |  |
| --- | --- |
| 1  2  3  4 | footer {  clear: both;  } |

#### Layout with Cleared Floats Demo

#### Containing Floats

Rather than clearing floats, another option is to contain the floats. The outcomes of containing floats versus those of clearing them are nearly the same; however, containing floats does help to ensure that all of our styles will be rendered properly.

To contain floats, the floated elements must reside within a parent element. The parent element will act as a container, leaving the flow of the document completely normal outside of it. The CSS for that parent element, represented by the group class below, is shown here:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | .group:before,  .group:after {  content: "";  display: table;  }  .group:after {  clear: both;  }  .group {  clear: both;  \*zoom: 1;  } |

There’s quite a bit going on here, but essentially what the CSS is doing is clearing any floated elements within the element with the class of group and returning the flow of the document back to normal.

More specifically, the :before and :after pseudo-elements, as mentioned in the Lesson 4 exercise, are dynamically generated elements above and below the element with the class of group. Those elements do not include any content and are displayed as table-level elements, much like block-level elements. The dynamically generated element after the element with the class of group is clearing the floats within the element with the class of group, much like the clear from before. And lastly, the element with the class of group itself also clears any floats that may appear above it, in case a left or right float may exist. It also includes a little trickery to get older browsers to play nicely.

It is more code than the clear: both; declaration alone, but it can prove to be quite useful.

Looking at our two-column page layout from before, we could wrap the <section> and <aside> elements with a parent element. That parent element then needs to contain the floats within itself. The code would look like this:

###### HTML

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | <header>...</header>  <div class="group">  <section>...</section>  <aside>...</aside>  </div>  <footer>...</footer> |

###### CSS

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23 | .group:before,  .group:after {  content: "";  display: table;  }  .group:after {  clear: both;  }  .group {  clear: both;  \*zoom: 1;  }  section {  float: left;  margin: 0 1.5%;  width: 63%;  }  aside {  float: right;  margin: 0 1.5%;  width: 30%;  } |

#### Layout with Contained Floats Demo

The technique shown here for containing elements is know as a “clearfix” and can often be found in other websites with the class name of clearfix or cf. We’ve chosen to use the class name of group, though, as it is representing a group of elements, and better expresses the content.

As elements are floated, it is important to keep note of how they affect the flow of a page and to make sure the flow of a page is reset by either clearing or containing the floats as necessary. Failing to keep track of floats can cause quite a few headaches, especially as pages begin to have multiple rows of multiple columns.

## In Practice[#practice-1](http://learn.shayhowe.com/html-css/positioning-content/#practice-1)

Let’s return to the Styles Conference website to try floating some content.

1. First things first, before we begin floating any elements, let’s provide a way to contain those floats by adding the clearfix to our CSS. Within the main.css file, just below our grid styles, let’s add the clearfix under the class name group, just like before.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | /\*  ========================================  Clearfix  ========================================  \*/  .group:before,  .group:after {  content: "";  display: table;  }  .group:after {  clear: both;  }  .group {  clear: both;  \*zoom: 1;  } |

1. Now that we can contain floats, let’s float the primary <h1> within the <header> element to the left and allow all of the other content in the header to wrap to the right of it.

To do this, let’s add a class of logo to the <h1> element. Then within our CSS, let’s add a new section of styles for the primary header. In this section we’ll select the <h1> element with the logo class and then float it to the left.

###### HTML

|  |  |
| --- | --- |
| 1  2  3  4 | <h1 class="logo">  <a href="index.html">Styles Conference</a>  </h1> |

###### CSS

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | /\*  ========================================  Primary header  ========================================  \*/  .logo {  float: left;  } |

1. While we’re at it, let’s add a little more detail to our logo. We’ll begin by placing a <br> element, or line break, between the word “Styles” and the word “Conference” to force the text of our logo to sit on two lines.

Within the CSS, let’s add a border to the top of our logo and some vertical padding to give the logo breathing room.

###### HTML

|  |  |
| --- | --- |
| 1  2  3  4 | <h1 class="logo">  <a href="index.html">Styles <br> Conference</a>  </h1> |

###### CSS

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | .logo {  border-top: 4px solid #648880;  padding: 40px 0 22px 0;  float: left;  } |

1. Because we floated the <h1> element, we’ll want to contain that float. The closest parent element of the <h1> element is the <header> element, so we’ll want to add the class of group to the <header> element. Doing this applies the clearfix styles we set up earlier to the <header> element.

|  |  |
| --- | --- |
| 1  2  3  4 | <header class="container group">  ...  </header> |

1. The <header> element is taking shape, so let’s take a look at the <footer> element. Much like we did with the <header> element, we’ll float our copyright to the left within the <small> element and let all other elements wrap around it to the right.

Unlike the <header> element, though, we’re not going to use a class directly on the floated element. This time we’re going to apply a class to the parent of the floated element and use a unique CSS selector to select the element and then float it.

Let’s start by adding the class of primary-footer to the <footer> element. Because we know we’ll be floating an element within the <footer> element, we should also add the class of group while we’re at it.

|  |  |
| --- | --- |
| 1  2  3  4 | <footer class="primary-footer container group">  ...  </footer> |

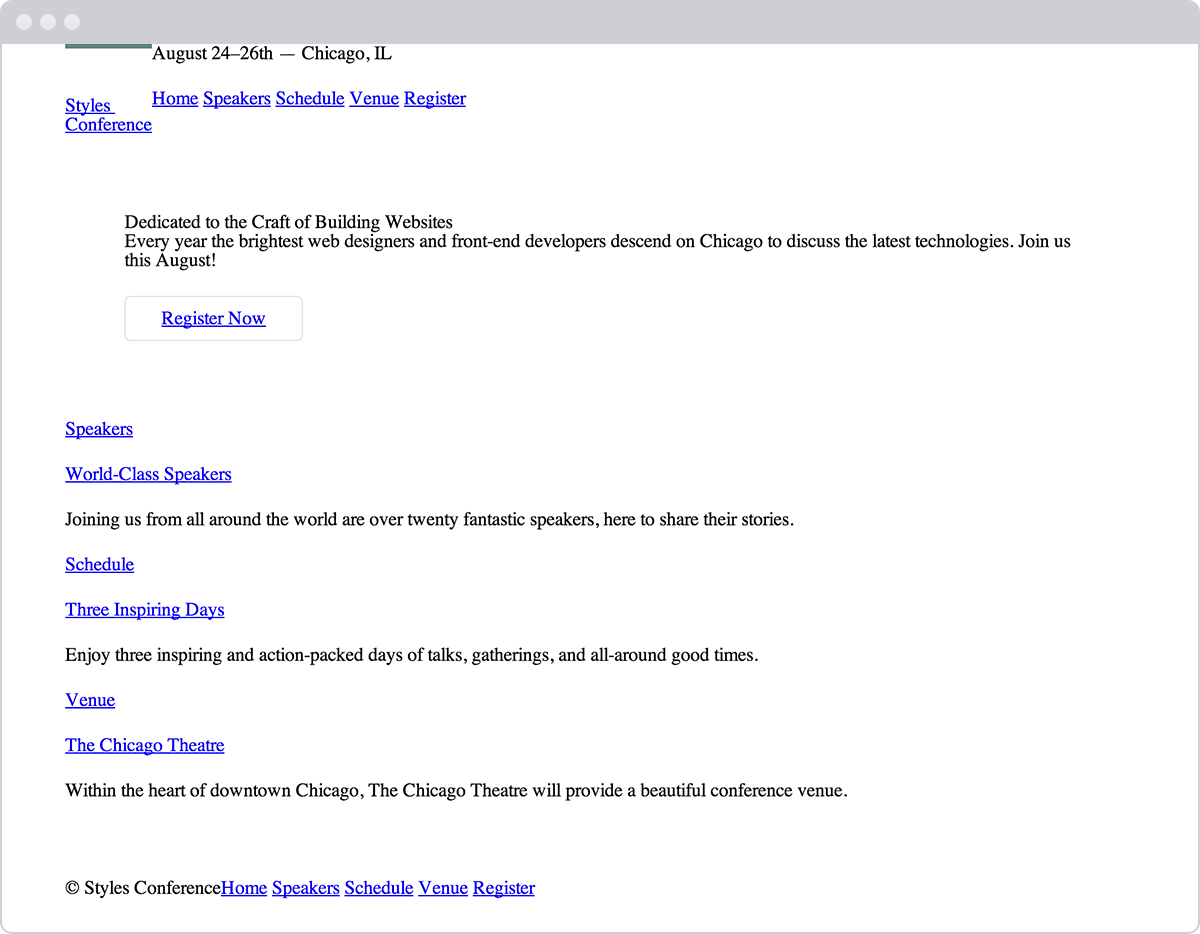
1. Now that the class of primary-footer is on the <footer> element, we can use that class to prequalify the <small> element with CSS. We’ll want to select and float the <small> element to the left. Let’s not forget to create a new section within our main.css file for these primary footer styles.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | /\*  ========================================  Primary footer  ========================================  \*/  .primary-footer small {  float: left;  } |

1. To review, here we are selecting the <small> element, which must reside within an element with the class attribute value of primary-footer, such as our <footer> element, for example.
2. Lastly, let’s put some padding on the top and bottom of the <footer> element to help separate it a little more from the rest of the page. We can do this directly by using the primary-footer class with a class selector.

|  |  |
| --- | --- |
| 1  2  3  4  5 | .primary-footer {  padding-bottom: 44px;  padding-top: 44px;  } |

With all of these changes to the <header> and <footer> elements, we have to be sure to make them on every page, not just the index.html page.

**Fig 5**

With a few floats, the <header> and <footer> elements on our Styles Conference home page are coming together

## Positioning with Inline-Block[#inline-block](http://learn.shayhowe.com/html-css/positioning-content/#inline-block)

In addition to using floats, another way we can position content is by using the display property in conjunction with the inline-block value. The inline-block method, as we’ll discuss, is primarily helpful for laying out pages or for placing elements next to one another within a line.

Recall that the inline-block value for the display property will display elements within a line while allowing them to accept all box model properties, including height, width, padding, border, and margin. Using inline-block elements allows us to take full advantage of the box model without having to worry about clearing any floats.

### Inline-Block in Practice

Let’s take a look at our three-column example from before. We’ll start by keeping our HTML just as it is:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | <header>...</header>  <section>...</section>  <section>...</section>  <section>...</section>  <footer>...</footer> |

Now instead of floating our three <section> elements, we’ll change their display values to inline-block, leaving the margin and width properties from before alone. Our resulting CSS will look like this:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | section {  display: inline-block;  margin: 0 1.5%;  width: 30%;  } |

Unfortunately, this code alone doesn’t quite do the trick, and the last <section> element is pushed to a new row. Remember, because inline-block elements are displayed on the same line as one another, they include a single space between them. When the size of each single space is added to the width and horizontal margin values of all the elements in the row, the total width becomes too great, pushing the last <section> element to a new row. In order to display all of the <section> elements on the same row, the white space between each <section> element must be removed.

#### Inline-Block Elements with White Space Demo

### Removing Spaces Between Inline-Block Elements

There are a number of ways to remove the space between inline-block elements, and some are more complex than others. We are going to focus on two of the easiest ways, both of which happen inside HTML.

The first solution is to put each new <section> element’s opening tag on the same line as the previous <section> element’s closing tag. Rather than using a new line for each element, we’ll end and begin elements on the same line. Our HTML could look like this:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | <header>...</header>  <section>  ...  </section><section>  ...  </section><section>  ...  </section>  <footer>...</footer> |

Writing inline-block elements this way ensures that the space between inline-block elements within HTML doesn’t exist; consequently, the space will not appear when the page is rendered.

#### Inline-Block Elements without White Space Demo

Another way to remove the white space between inline-block elements is to open an HTML comment directly after an inline-block element’s closing tag. Then, close the HTML com- ment immediately before the next inline-block element’s opening tag. Doing this allows inline-block elements to begin and end on separate lines of HTML and “comments out” any potential spaces between the elements. The resulting code would look like this:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | <header>...</header>  <section>  ...  </section><!--  --><section>  ...  </section><!--  --><section>  ...  </section>  <footer>...</footer> |

Neither of these options is perfect, but they are helpful. I tend to favor using comments for better organization, but which option you choose is entirely up to you.

## Creating Reusable Layouts[#reusable-layouts](http://learn.shayhowe.com/html-css/positioning-content/#reusable-layouts)

When building a website, it is always best to write modular styles that may be reused elsewhere, and reusable layouts are high on the list of reusable code. Layouts can be created using either floats or inline-block elements, but which works best and why?

Whether it’s better to use floats or inline-block elements to lay out the structure of a page is open to debate. My approach is to use inline-block elements to create the grid—or layout—of a page and to then use floats when I want content to wrap around a given element (as floats were intended to do with images). Generally, I also find inline-block elements easier to work with.

That said, use whatever works best for you. If you are comfortable with one approach over the other, then go for it.

Currently there are new CSS specifications in the works—specifically flex- and grid- based properties—that will help address how to best lay out pages. Keep an eye out for these methods as they begin to surface.

## In Practice[#practice-2](http://learn.shayhowe.com/html-css/positioning-content/#practice-2)

With a solid understanding of reusable layouts, the time has come to implement one in our Styles Conference website.

1. For the Styles Conference website, we’ll create a three-column reusable layout using inline-block elements. We’ll do so in a way that allows us to have three columns of equal width or two columns with the total width split between them, two-thirds in one and one-third in the other.

To begin, we’ll create classes that define the width of these columns. The two classes we’ll create are col-1-3, for one-third, and col-2-3, for two-thirds. Within the grid section of our main.css file, let’s go ahead and define these classes and their corresponding widths.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | .col-1-3 {  width: 33.33%;  }  .col-2-3 {  width: 66.66%;  } |

1. We’ll want both of the columns to be displayed as inline-block elements. We’ll need to make sure that their vertical alignment is set to the top of each column, too.

Let’s create two new selectors that will share the display and vertical-alignment property styles.

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | .col-1-3,  .col-2-3 {  display: inline-block;  vertical-align: top;  } |

Looking at the CSS again, we’ve created two class selectors, col-1-3 and col-2-3, that are separated with a comma. The comma at the end of the first selector signifies that another selector is to follow. The second selector is followed by the opening curly bracket, {, which signifies that style declarations are to follow. By comma-separating the selectors, we can bind the same styles to multiple selectors at one time.

1. We’ll want to put some space in between each of the columns to help break up the content. We can accomplish this by putting horizontal padding on each of the columns.

This works well; however, when two columns are sitting next to one another, the width of the space between them will be double that of the space from the outside columns to the edge of the row. To balance this we’ll place all of our columns within a grid and add the same padding from our columns to that grid.

Let’s use a class name of grid to identify our grid, and then let’s identify the same horizontal padding for our grid, col-1-3, and col-2-3 classes. With commas separating our selectors again, our CSS looks like this:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | .grid,  .col-1-3,  .col-2-3 {  padding-left: 15px;  padding-right: 15px;  } |

1. When we’re setting up the horizontal padding, we’ll need to be careful. Remember, in the last lesson we created a container element, known by the class of container, to center all of our content on a page within a 960-pixel-wide element. Currently if we were to put an element with the class of grid inside an element with the class of container, their horizontal paddings would add to one another, and our columns would not appear proportionate to the width of the rest of the page.

We don’t want this to happen, so instead, we’ll have to share some of the styles from the container rule set with the grid rule set. Specifically, we’ll need to share the width property and values (to make sure our page stays fixed at 960 pixels wide) and the margin property and values (to center any element with the class of grid on the page).

We’ll accomplish this by breaking up the old container rule set into the following:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | .container,  .grid {  margin: 0 auto;  width: 960px;  }  .container {  padding-left: 30px;  padding-right: 30px;  } |

Now any element with the class of container or grid will be 960 pixels wide and centered on the page. Additionally, we’ve preserved the existing horizontal padding for any element with the class of container by moving it into a new, separate rule set.

1. All right—all of the heavy lifting needed to get our reusable grid styles into place is finished. Now it’s time to work in our HTML and to see how these classes perform.

We’ll begin with the teasers on the home page, within our index.html file, aligning them into three columns. Currently, the teasers are wrapped in a <section> element with the class of container. We’ll want to change that class from container to grid so that we can begin placing columns within it.

|  |  |
| --- | --- |
| 1  2  3  4 | <section class="grid">  ...  </section> |

1. Next, we’ll want to add a class of col-1-3 to each of the <section> elements within the <section> element with the class of grid.

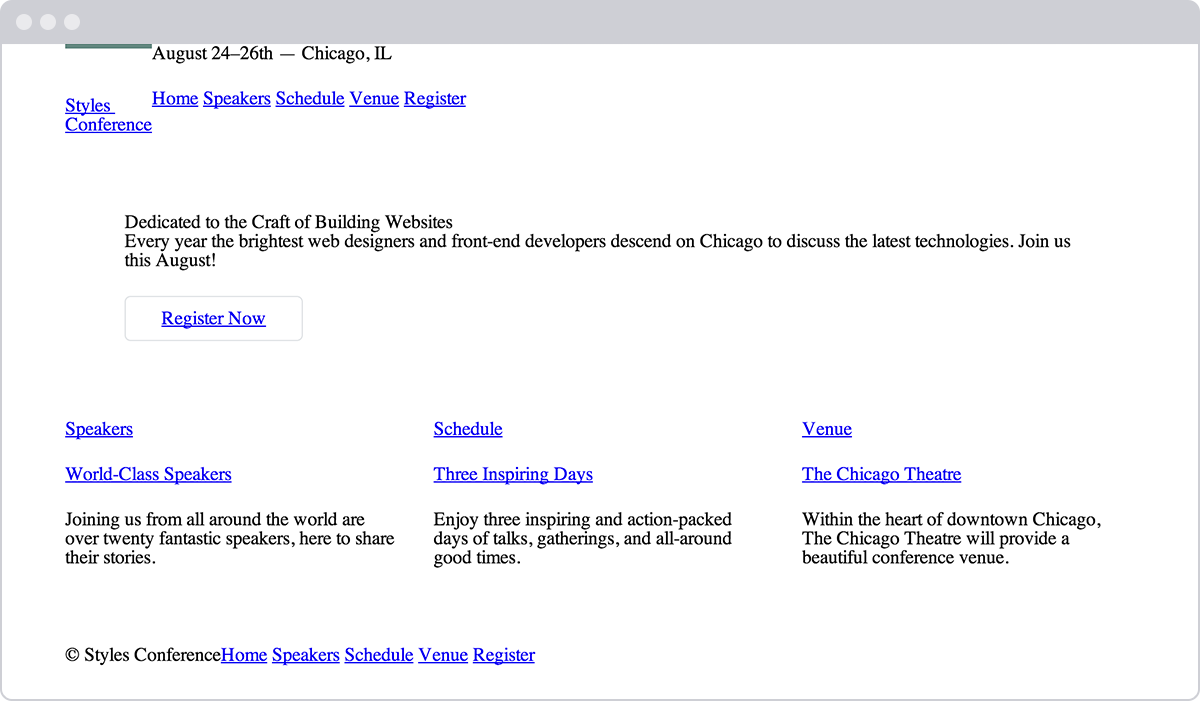
|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | <section class="grid">  <section class="col-1-3">  ...  </section>  <section class="col-1-3">  ...  </section>    <section class="col-1-3">  ...  </section>  </section> |

1. And lastly, because each of our columns is an inline-block element, we’ll want to make sure we remove the empty white space between them. We’ll use comments to do this, and we’ll add a little bit of documentation noting each upcoming section while we’re at it to better organize our code.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22 | <section class="grid">    <!-- Speakers -->    <section class="col-1-3">  ...  </section><!--    Schedule    --><section class="col-1-3">  ...  </section><!--    Venue    --><section class="col-1-3">  ...  </section>  </section> |

1. To review, on line 3 we leave a comment identifying the “Speakers” section to follow. At the end of line 7, we open a comment immediately after the closing </section> tag. Within that comment, on line 9 we identify the “Schedule” section to come. We then close the comment at the beginning of line 11, just before the opening <section> tag. This same comment structure reappears on lines 13 through 17 between the two <section> elements, right before the “Venue” section. In all, we’ve commented out any potential white space between the columns while also using those comments to identify our sections.

We now have a reusable three-column grid that supports multiple arrangements, using both one-third- and two-thirds-width columns. Our home page now has three columns, breaking up all the different teasers.

**Fig 5**

Our Styles Conference home page now includes a three-column layout

### Demo & Source Code

Below you may view the Styles Conference website in its current state, as well as download the source code for the website in its current state.

[**View the Styles Conference Website**](http://learn.shayhowe.com/practice/positioning-content/index.html) or [Download the Source Code](http://learn.shayhowe.com/practice/positioning-content.zip) (Zip file)

## Uniquely Positioning Elements[#uniquely-positioning-elements](http://learn.shayhowe.com/html-css/positioning-content/#uniquely-positioning-elements)

Every now and then we’ll want to [precisely position](http://alistapart.com/article/css-positioning-101) an element, but floats or inline-block elements won’t do the trick. Floats, which remove an element from the flow of a page, often produce unwanted results as surrounding elements flow around the floated element. Inline-block elements, unless we’re creating columns, can be fairly awkward to get into the proper position. For these situations we can use the position property in connection with box offset properties.

The position property identifies how an element is positioned on a page and whether or not it will appear within the normal flow of a document. This is used in conjunction with the box offset properties—top, right, bottom, and left—which identify exactly where an element will be positioned by moving elements in a number of different directions.

By default every element has a position value of static, which means that it exists in the normal flow of a document and it doesn’t accept any box offset properties. The static value is most commonly overwritten with a relative or absolute value, which we’ll examine next.

### Relative Positioning

The relative value for the position property allows elements to appear within the normal flow a page, leaving space for an element as intended while not allowing other elements to flow around it; however, it also allows an element’s display position to be modified with the box offset properties. For example, consider the following HTML and CSS:

###### HTML

|  |  |
| --- | --- |
| 1  2  3  4 | <div>...</div>  <div class="offset">...</div>  <div>...</div> |

###### CSS

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | div {  height: 100px;  width: 100px;  }  .offset {  left: 20px;  position: relative;  top: 20px;  } |

#### Relative Positioning Demo

Here the second <div> element, the element with the class of offset, has a position value of relative and two box offset properties, left and top. This preserves the original position of the element, and other elements are not allowed to move into this space. Additionally, the box offset properties reposition the element, pushing it 20 pixels from the left and 20 pixels from the top of its original location.

With relatively positioned elements, it’s important to know that the box offset properties identify where an element will be moved from given its original position. Thus, the left property with a value of 20 pixels will actually push the element towards the right, from the left, 20 pixels. The top property with a value of 20 pixels, then, will push an element towards the bottom, from the top, 20 pixels.

When we position the element using the box offset properties, the element overlaps the element below it rather than moving that element down as the margin or padding properties would.

### Absolute Positioning

The absolute value for the position property is different from the relative value in that an element with a position value of absolute will not appear within the normal flow of a document, and the original space and position of the absolutely positioned element will not be preserved.

Additionally, absolutely positioned elements are moved in relation to their closest relatively positioned parent element. Should a relatively positioned parent element not exist, the absolutely positioned element will be positioned in relation to the <body> element. That’s quite a bit of information; let’s take a look at how this works inside some code:

###### HTML

|  |  |
| --- | --- |
| 1  2  3  4 | <section>  <div class="offset">...</div>  </section> |

###### CSS

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | section {  position: relative;  }  div {  position: absolute;  right: 20px;  top: 20px;  } |

#### Absolute Positioning Demo

In this example the <section> element is relatively positioned but doesn’t include any box offset properties. Consequently its position doesn’t change. The <div> element with a class of offset includes a position value of absolute. Because the <section> element is the closest relatively positioned parent element to the <div> element, the <div> element will be positioned in relation to the <section> element.

With relatively positioned elements, the box offset properties identify in which direction an element would be moved in relation to itself. With absolutely positioned elements, the box offset properties identify in which direction an element will be moved in relation to its closest relatively positioned parent element.

As a result of the right and top box offset properties, the <div> element will appear 20 pixels from the right and 20 pixels from the top of the <section>.

Because the <div> element is absolutely positioned, it does not sit within the normal flow of the page and will overlap any surrounding elements. Additionally, the original position of the <div> is not preserved, and other elements are able to occupy that space.

Typically, most positioning can be handled without the use of the position property and box offset properties, but in certain cases they can be extremely helpful.

## Summary[#summary](http://learn.shayhowe.com/html-css/positioning-content/#summary)

Learning how to position content within HTML and CSS is a huge step toward mastering the two languages. Add to this the box model, and we’re well on our way to becoming front-end developers.

To review, within this lesson we covered the following:

* What floats are and how to use them to position content
* How to clear and contain floated elements
* How to position content with inline-block elements
* How to remove the white space between inline-block elements
* How to uniquely position content with relatively and absolutely positioned elements

###### Lesson 6

# Working with Typography

The field of [web typography](http://webtypography.net/) has grown substantially over time. There are a couple of different reasons for its rise in popularity; one widely acknowledged reason is the development of a system for embedding our own web fonts on a website.

In the past we were limited to a small number of typefaces that we could use on a website. These typefaces were the most commonly installed fonts on computers, so they were the most likely to render properly on-screen. If a font wasn’t installed on a computer, it wouldn’t render on the website either. Now, with the ability to embed fonts, we have a much larger palette of typefaces to choose from, including those that we add to a website.

While the ability to embed fonts gives us access to countless new typefaces, it’s also important for us to know the basic principles of typography. In this lesson we’re going to take a look at some of these basic principles and how to apply them to our web pages using HTML and CSS.

#### Typeface vs. Font

The terms “typeface” and “font” are often interchanged, causing confusion. Here is a breakdown of exactly what each term means.

A typeface is what we see. It is the artistic impression of how text looks, feels, and reads.

A font is a file that contains a typeface. Using a font on a computer allows the computer to access the typeface.

One way to help clarify the difference between a typeface and a font is to compare them to a [song and an MP3](http://fontfeed.com/archives/font-or-typeface/). A typeface is very similar to a song in that it is a work of art. It is created by an artist or artists and is open to public interpretation. A font, on the other hand, is very similar to an MP3 in that it is not the artistic impression itself, but only a method of delivering the artistic value.

## Adding Color to Text[#color](http://learn.shayhowe.com/html-css/working-with-typography/#color)

Typically one of the first decisions we’ll make when building a website is choosing the primary typeface and text color to be used. While there are a number of other properties that can be changed—size, weight, and so on—the typeface and text color generally have the largest impact on the look and legibility of a page. Getting rid of the browser defaults and using our own typeface and text color immediately begins setting the tone of our page.

The only property we need to set the color of text is the color property. The color property accepts one color value, but in many different formats. These formats, as we discussed in Lesson 3, “[Getting to Know CSS](http://learn.shayhowe.com/html-css/getting-to-know-css/),” include keywords, hexadecimal values, and RGB, RGBa, HSL, and HSLa values. Hexadecimal values are the most prevalent, as they provide the most control with the least amount of effort.

Let’s take a look at the CSS required to change the color of all the text within the <html> element on a page:

|  |  |
| --- | --- |
| 1  2  3  4 | html {  color: #555;  } |

## Changing Font Properties[#font-properties](http://learn.shayhowe.com/html-css/working-with-typography/#font-properties)

CSS offers a lot of different properties for editing the look and feel of text on a page. These properties fit into two categories: font-based properties and text-based properties. Most of these properties will be prefaced with either font-\* or text-\*. To begin we’ll discuss the font-based properties.

### Font Family

The font-family property is used to declare which font—as well as which fallback or substitute fonts—should be used to display text. The value of the font-family property contains multiple font names, all comma separated.

The first declared font, starting from the left, is the primary font choice. Should the first font be unavailable, alternative fonts are declared after it in order of preference from left to right.

Font names consisting of two or more words need to be wrapped in quotation marks. Additionally, the last font should be a keyword value, which will use the system default font for the specified type, most commonly either sans-serif or serif.

The font-family property in action looks like this:

|  |  |
| --- | --- |
| 1  2  3  4 | body {  font-family: "Helvetica Neue", Helvetica, Arial, sans-serif;  } |

In this case, Helvetica Neue is the preferred font to display. If this font is unavailable or not installed on a given device, the next font in the list—Helvetica—will be used, and so on.

### Font Size

The font-size property provides the ability to set the size of text using common length values, including pixels, em units, percentages, points, or font-size keywords.

Here the CSS is setting a font-size of 14 pixels on the <body> element:

|  |  |
| --- | --- |
| 1  2  3  4 | body {  font-size: 14px;  } |

### Font Style

To change text to italics, or to prevent text from being italicized, we’ll use the font-style property. The font-style property accepts four keyword values: normal, italic, oblique, and inherit. Of these four, the most commonly used are italic (sets text to italic) and normal (returns text to its normal style).

The following CSS sets all elements with a class of special to include a font-style of italic:

|  |  |
| --- | --- |
| 1  2  3  4 | .special {  font-style: italic;  } |

### Font Variant

It doesn’t happen often, but occasionally text will need to be set in small capitals, also known as small caps. For this specific case we’ll use the font-variant property. The font-variant property accepts three values: normal, small-caps, and inherit. The most typically seen values are normal and small-caps, which are used to switch typefaces between normal and small caps variants.

To switch all elements with a class of firm, we’ll use a font-variant of small-caps:

|  |  |
| --- | --- |
| 1  2  3  4 | .firm {  font-variant: small-caps;  } |

### Font Weight

Occasionally, we’ll want to style text as bold or to change the specific weight of a typeface. For these cases we’ll use the font-weight property. The font-weight property accepts either keyword or numeric values.

Keyword values include normal, bold, bolder, lighter, and inherit. Of these keyword values, it is recommended to primarily use normal and bold to change text from normal to bold and vice versa. Rather than using the keyword values bolder or lighter, it’s better to use a numeric value for more specific control.

In practice, here’s the CSS to set the font-weight to bold for any element with the class of daring:

|  |  |
| --- | --- |
| 1  2  3  4 | .daring {  font-weight: bold;  } |

The numeric values 100, 200, 300, 400, 500, 600, 700, 800, and 900 pertain specifically to typefaces that have multiple weights. The order of these weights starts with the thinnest weight, 100, and scales up to the thickest weight, 900. For reference, the keyword value of normal maps to 400 and the keyword bold maps to 700; thus, any numeric value below 400 will be fairly thin, and any value above 700 will be fairly thick.

Changing the font-weight to 600 for any element with the class of daring now renders that text as semibold—not quite as thick as the bold keyword value from before:

|  |  |
| --- | --- |
| 1  2  3  4 | .daring {  font-weight: 600;  } |

#### Typeface Weights

Before using a numeric value, we need to check and see whether the typeface we are using comes in the weight we’d like to use. Attempting to use a weight that’s not available for a given typeface will cause those styles to default to the closest value.

For example, the Times New Roman typeface comes in two weights: normal, or 400, and bold, or 700. Attempting to use a weight of 900 will default the typeface to the closest related weight, 700 in this case.

### Line Height

Line height, the distance between two lines of text (often referred to as leading) is declared using the line-height property. The line-height property accepts all general length values, which we covered in Lesson 3, “[Getting to Know CSS](http://learn.shayhowe.com/html-css/getting-to-know-css/).”

The best practice for legibility is to set the line-height to around one and a half times our font-size property value. This could be quickly accomplished by setting the line-height to 150%, or just 1.5. However, if we’re working with a baseline grid, having a little more control over our line-height using pixels may be preferable.

Looking at the CSS, we’re setting a line-height of 22 pixels within the element, thus placing 22 pixels between each line of text:

|  |  |
| --- | --- |
| 1  2  3  4 | body {  line-height: 22px;  } |

Line height may also be used to vertically center a single line of text within an element. Using the same property value for the line-height and height properties will vertically center the text:

|  |  |
| --- | --- |
| 1  2  3  4  5 | .btn {  height: 22px;  line-height: 22px;  } |

This technique may be seen with buttons, alert messages, and other single-line text blocks.

### Shorthand Font Properties

All of the font-based properties listed earlier may be combined and rolled into one font property and [shorthand value](http://www.impressivewebs.com/css-font-shorthand-property-cheat-sheet/). The font property can accept multiple font-based property values. The order of these property values should be as follows, from left to right: font-style, font-variant, font-weight, font-size, line-height, and font-family.

As a shorthand value, these property values are listed from left to right without the use of commas (except for font names, as the font-family property value uses commas). A forward slash, /, separator is needed between the font-size and line-height property values.

When using this shorthand value, every property value is optional except the font-size and font-family property values. That said, we can include only the font-size and font-family property values in the shorthand value if we wish.

|  |  |
| --- | --- |
| 1  2  3  4 | html {  font: italic small-caps bold 14px/22px "Helvetica Neue", Helvetica, Arial, sans-serif;  } |

### Font Properties All Together

Let’s take a look at an example that uses all these font-based properties together. The following HTML and CSS demonstrates the different possibilities when styling text.

###### HTML

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | <h2><a href="#">I Am a Builder</a></h2>  <p class="byline">Posted by Shay Howe</p>  <p>Every day I see designers and developers working alongside one another. They work intelligently in pursuit of business objectives. They work diligently making exceptional products. They solve real problems and take pride in their work. They are builders. <a href="#">Continue&#8230;</a></p> |

###### CSS

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23 | h2,  p {  color: #555;  font: 13px/20px "Helvetica Neue", Helvetica, Arial, sans-serif;  }  a {  color: #0087cc;  }  a:hover {  color: #ff7b29;  }  h2 {  font-size: 22px;  font-weight: bold;  margin-bottom: 6px;  }  .byline {  color: #9799a7;  font-family: Georgia, Times, "Times New Roman", serif;  font-style: italic;  margin-bottom: 18px;  } |

#### Font Properties Demo

#### CSS Pseudo-Classes

The demonstration here uses the :hover CSS pseudo-class, something we’ve never seen before. For reference, pseudo-classes are keywords that may be added to the end of a selector to style an element when it’s in a unique state.

The :hover pseudo-class styles an element when a user hovers over that element. When used with the <a> element, as shown here, all <a> elements will receive unique styles when they are hovered over. Now our <a> elements will change color upon being hovered over.

## In Practice[#practice-1](http://learn.shayhowe.com/html-css/working-with-typography/#practice-1)

Diving back into our Styles Conference website, let’s start adding some font-based properties.

1. We’ll begin by updating the font on all of our text. To do this, we’ll apply styles to our <body> element. We’ll start with a color, and we’ll also add in font-weight, font-size, line-height, and font-family values by way of the font property and shorthand values.

In an attempt to keep our main.css file as organized as possible, let’s create a new section for these custom styles, placing it just below our reset and above our grid styles.

We need to add the following:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | /\*  ========================================  Custom styles  ========================================  \*/  body {  color: #888;  font: 300 16px/22px "Open Sans", "Helvetica Neue", Helvetica, Arial, sans-serif;  } |

1. In Lesson 4, “[Opening the Box Model](http://learn.shayhowe.com/html-css/opening-the-box-model/),” we began adding some typographic styles, specifically adding a bottom margin to a few different levels of headings and paragraphs. Within the same section of the main.css file, let’s add a color to the level-one through level-four headings.

|  |  |
| --- | --- |
| 1  2  3  4 | h1, h2, h3, h4 {  color: #648880;  } |

1. While we’re at it, let’s also add in font sizes for these different heading levels. Our <h1> and <h2> elements will use fairly large font-size values; consequently, we’ll also want to increase their line-height values to keep the text within these elements legible. For reference, we’ll make their line-height values 44 pixels, double the value of the base line-height set within the <body> element rule set.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | h1 {  font-size: 36px;  line-height: 44px;  }  h2 {  font-size: 24px;  line-height: 44px;  }  h3 {  font-size: 21px;  }  h4 {  font-size: 18px;  } |

1. Our <h5> elements are going to be a little more unique than the rest of our headings. Accordingly, we’re going to change their styles a bit.

We’ll use a different color property value and a slightly smaller font-size for these elements, and we’re going to change the font-weight to 400, or normal.

By default, browsers render headings with a font-weight of bold. Our headings, however, are currently all set to a font-weight of 300. Our reset at the top of our main.css file changed the font-weight to normal, and then our font-weight of 300 within the <body> element rule set changed all headings to a font-weight of 300.

The font-weight of 400 on the <h5> element will actually make it slightly thicker than the rest of our other headings and text.

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | h5 {  color: #a9b2b9;  font-size: 14px;  font-weight: 400;  } |

1. Our reset at the beginning of our style sheet also reset the browser default styles for the <strong>, <cite>, and <em> elements, which we’ll want to add back in. For our <strong> elements we’ll want to set a font-weight of 400, which actually equates to normal, not bold, as the typeface we’re using is thicker than most typefaces. Then, for our <cite> and <em> elements we’ll want to set a font-style of italic.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | strong {  font-weight: 400;  }  cite, em {  font-style: italic;  } |

1. We’re on a roll, so let’s keep going by adding some styles to our anchor elements. Currently they are the browser default blue. Let’s make them the same color as our <h1> through <h4> heading elements. Additionally, let’s use the :hover pseudo-class to change the color to a light gray when a user hovers over an anchor.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | /\*  ========================================  Links  ========================================  \*/  a:hover {  color: #a9b2b9;  }  a {  color: #648880;  } |

1. Now let’s take a look at our <header> element and update our styles there. We’ll begin updating our logo by adding the font-size and line-height properties within the logo rule set. Adding to the existing border-top, float, and padding properties, the new rule set should look like this:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | .logo {  border-top: 4px solid #648880;  float: left;  font-size: 48px;  line-height: 44px;  padding: 40px 0 22px 0;  } |

1. Because we’ve bumped up the size of the logo quite a bit, let’s add a margin to the <h3> element within the <header> element to balance it. We’ll do so by placing a class attribute value of tagline on the <h3> element and then using that class within our CSS to apply the proper margins.

Let’s not forget that the changes to the <h3> element need to happen on every page.

###### HTML

|  |  |
| --- | --- |
| 1  2 | <h3 class="tagline">August 24&ndash;26th &mdash; Chicago, IL</h3> |

###### CSS

|  |  |
| --- | --- |
| 1  2  3  4 | .tagline {  margin: 66px 0 22px 0;  } |

1. After the <h3> element with the class attribute value of tagline comes the <nav> element. Let’s add a class attribute value of primary-nav to the <nav> element and add font-size and font-weight properties to make the navigation stand out against the rest of the header.

###### HTML

|  |  |
| --- | --- |
| 1  2  3  4 | <nav class="primary-nav">  ...  </nav> |

###### CSS

|  |  |
| --- | --- |
| 1  2  3  4  5 | .primary-nav {  font-size: 14px;  font-weight: 400;  } |

1. With the <header> element in slightly better shape, let’s also take a look at our <footer> element. Using the primary-footer class, let’s change the color and font-size for all the text within the <footer> element. Additionally, let’s bump up the font-weight of the <small> element to 400.

Including the existing styles, the styles for our primary footer section should look like this:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | .primary-footer {  color: #648880;  font-size: 14px;  padding-bottom: 44px;  padding-top: 44px;  }  .primary-footer small {  float: left;  font-weight: 400;  } |

1. Let’s update our home page a bit, too. We’ll start with the hero section, increasing the overall line-height of the section to 44 pixels. We’ll also make the text within this section larger, increasing the <h2> element’s font-size to 36 pixels and the <p> element’s font-size to 24 pixels.

We can make all of these changes by using the existing hero class selector and creating new selectors for the <h2> and <p> elements. Our styles for the hero section will now break down in this way:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | .hero {  line-height: 44px;  padding: 22px 80px 66px 80px;  }  .hero h2 {  font-size: 36px;  }  .hero p {  font-size: 24px;  } |

1. Lastly, we have one small issue to fix on our home page. Previously we gave all of our anchor elements a light gray color value when a user hovers over them. This works great, except for within the three teasers on our home page where the anchor element wraps both <h3> and <h5> elements. Because the <h3> and <h5> elements have their own color definition, they are not affected by the :hover pseudo-class styles from before.

Fortunately we can fix this, although it’s going to require a fairly complicated selector. We’ll begin by adding a class attribute value of teaser to all three columns on the home page. We’ll use this class as a qualifying selector shortly.

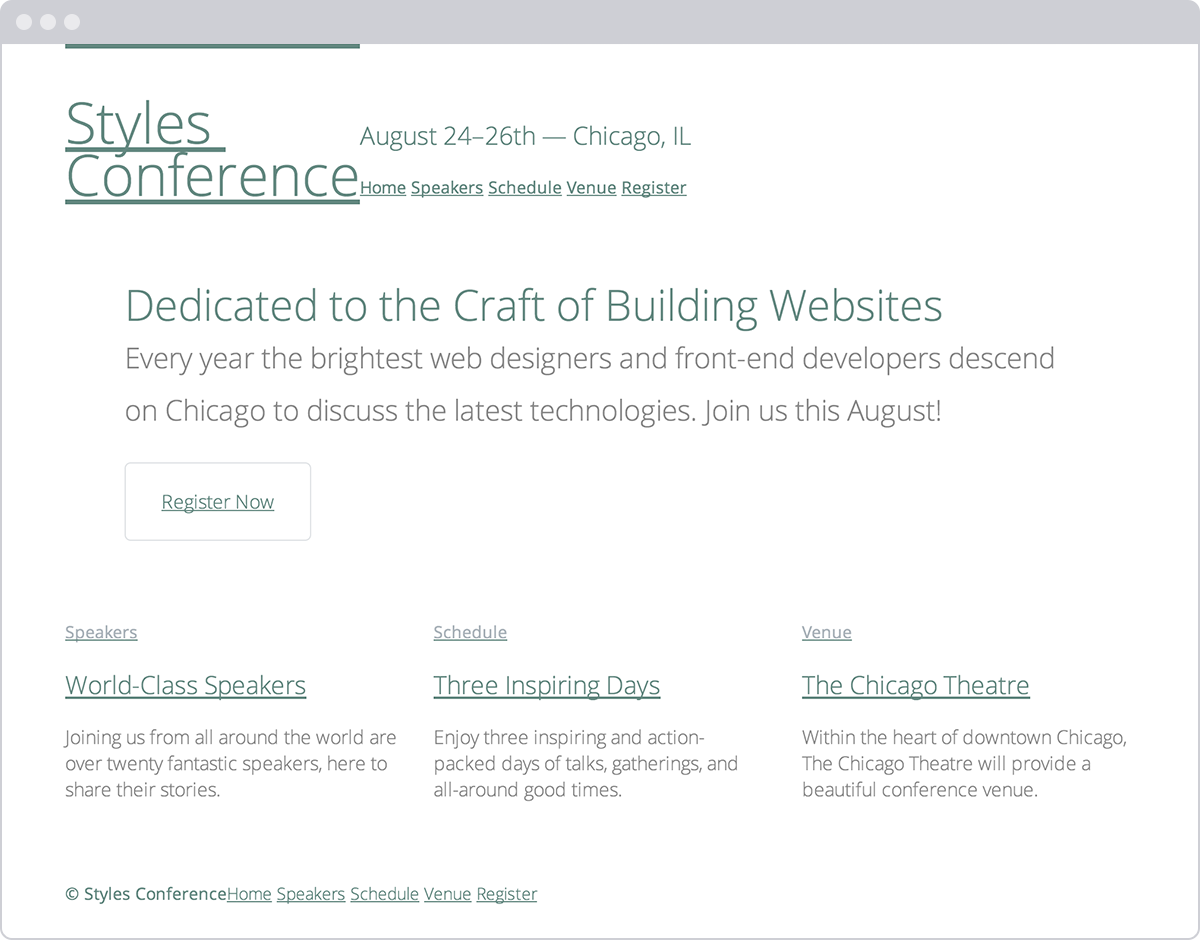
|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | <section class="grid">  <!-- Speakers -->    <section class="teaser col-1-3">  <a href="speakers.html">  <h5>Speakers</h5>  <h3>World-Class Speakers</h3>  </a>  <p>Joining us from all around the world are over twenty fantastic speakers, here to share their stories.</p>  </section>  ...  </section> |

With a qualifying class in place, we’re ready to do some CSS heavy lifting and create a fairly complex selector. We’ll begin our selector with the teaser class, as we only want to target elements within an element with the class of teaser. From there we want to apply styles to elements that reside within anchor elements that are being hovered over; thus we’ll add the a type selector along with the :hover pseudo-class. Lastly, we’ll add the h3 type selector to select the actual <h3> elements we wish to apply styles to.

Altogether, our selector and styles for these <h3> elements will look like this:

|  |  |
| --- | --- |
| 1  2  3  4 | .teaser a:hover h3 {  color: #a9b2b9;  } |

Whew, that was quite a bit. The good news is that our Styles Conference home page is starting to look really nice and is showing a bit of personality.

**Fig 6**

Our Styles Conference website has received quite a bit of love from a handful of font-based properties

## Applying Text Properties[#text-properties](http://learn.shayhowe.com/html-css/working-with-typography/#text-properties)

Knowing how to set the family, size, style, variant, weight, and line height of a font is only half the battle. Additionally we can decide how to align, decorate, indent, transform, and space text. Let’s start with text alignment.

### Text Align

Aligning text is an important part of building a rhythm and flow on a page; we do this using the text-align property. The text-align property has five values: left, right, center, justify, and inherit. All of these values are fairly straightforward; as expected, they align text to the left, right, or center, or they justify text.

The following CSS sets all paragraph text to be center aligned:

|  |  |
| --- | --- |
| 1  2  3  4 | p {  text-align: center;  } |

The text-align property, however, should not be confused with the float property. The text-align values left and right will align text within an element to the left or right, whereas the float values left and right will move the entire element. Sometimes the text-align property will give us the desired outcome, and other times we may need to use the float property.

### Text Decoration

The text-decoration property provides a handful of ways to spruce up text. It accepts the keyword values of none, underline, overline, line-through, and inherit. Use of the text-decoration property varies, but the most popular use is to underline links, which is a default browser style.

Here the CSS styles any element with the class of note with a text-decoration of underline:

|  |  |
| --- | --- |
| 1  2  3  4 | .note {  text-decoration: underline;  } |

Multiple text-decoration values may be applied to an element at once by space-separating each keyword within the value.

### Text Indent

The text-indent property can be used to indent the first line of text within an element, as is commonly seen in printed publications. All common length values are available for this property, including pixels, points, percentages, and so on. Positive values will indent text inward, while negative values will indent text outward.

Here, the CSS indents the text for all <p> elements inward by 20 pixels:

|  |  |
| --- | --- |
| 1  2  3  4 | p {  text-indent: 20px;  } |

### Text Shadow

The text-shadow property allows us to add a shadow or multiple shadows to text. The property generally takes four values, all listed one after the other from left to right. The first three values are lengths, and the last value is a color.

Within the three length values, the first value determines the shadow’s horizontal offset, the second value determines the shadow’s vertical offset, and the third value determines the shadow’s blur radius. The fourth, and last, value is the shadow’s color, which can be any of the color values used within the color property.

The text-shadow property here is casting a 30% opaque black shadow 3 pixels towards the right, 6 pixels down, and blurred 2 pixels off all <p> element text:

|  |  |
| --- | --- |
| 1  2  3  4 | p {  text-shadow: 3px 6px 2px rgba(0, 0, 0, .3);  } |

Using negative length values for the horizontal and vertical offsets allows us to move shadows toward the left and the top.

Multiple text shadows can also be chained together using comma-separated values, adding more than one shadow to the text. Using numerous shadows allows us to place them above and below the text, or in any variation we desire.

#### Box Shadow

The text-shadow property places a shadow specifically on the text of an element. If we’d like to place a shadow on the element as a whole, we can use the box-shadow property.

The box-shadow property works just like the text-shadow property, accepting values for horizontal and vertical offsets, a blur, and a color.

The box-shadow property also accepts an optional fourth length value, before the color value, for the spread of a shadow. As a positive length value, the spread will expand the shadow larger than the size of the element it’s applied to, and as a negative length value the spread will shrink the shadow to be smaller than the size of the element it’s applied to.

Lastly, the box-shadow property may include an optional inset value at the beginning of the value to place the shadow inside an element as opposed to outside the element.

### Text Transform

Similar to the font-variant property, there is the text-transform property. While the font-variant property looks for an alternate variant of a typeface, the text-transform property will change the text inline without the need for an alternate typeface. The text-transform property accepts five values: none, capitalize, uppercase, lowercase, and inherit.

The capitalize value will capitalize the first letter of each word, the uppercase value will capitalize every letter, and the lowercase value will make every letter lowercase. Using none will return any of these inherited values back to the original text style.

The following CSS sets all <p> element text to appear in all uppercase letters:

|  |  |
| --- | --- |
| 1  2  3  4 | p {  text-transform: uppercase;  } |

### Letter Spacing

Using the letter-spacing property, we can adjust the space (or tracking) between the letters on a page. A positive length value will push letters farther apart from one another, while a negative length value will pull letters closer together. The keyword value none will return the space between letters back to its normal size.

Using a relative length value with the letter-spacing property will help ensure that we maintain the correct spacing between letters as the font-size of the text is changed. It is, however, always a good idea to double-check our work.

With the CSS here, all of the letters within our <p> elements will appear .5 em closer together:

|  |  |
| --- | --- |
| 1  2  3  4 | p {  letter-spacing: -.5em;  } |

### Word Spacing

Much like the letter-spacing property, we can also adjust the space between words within an element using the word-spacing property. The word-spacing property accepts the same length values and keywords as the letter-spacing property. Instead of spacing letters apart, though, the word-spacing property applies those values between words.

Here every word within a <p> element will be spaced .25 em apart.

|  |  |
| --- | --- |
| 1  2  3  4 | p {  word-spacing: .25em;  } |

### Text Properties All Together

Let’s revisit our blog teaser demonstration from before, this time adding in a few text-based properties on top of our font-based properties.

###### HTML

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | <h2><a href="#">I Am a Builder</a></h2>  <p class="byline">Posted by Shay Howe</p>  <p class="intro">Every day I see designers and developers working alongside one another. They work intelligently in pursuit of business objectives. They work diligently making exceptional products. They solve real problems and take pride in their work. They are builders. <a href="#">Continue&#8230;</a></p> |

###### CSS

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37 | h2,  p {  color: #555;  font: 13px/20px "Helvetica Neue", Helvetica, Arial, sans-serif;  }  a {  color: #0087cc;  }  a:hover {  color: #ff7b29;  }  h2 {  font-size: 22px;  font-weight: bold;  letter-spacing: -.02em;  margin-bottom: 6px;  }  h2 a {  text-decoration: none;  text-shadow: 2px 2px 1px rgba(0, 0, 0, .2);  }  .byline {  color: #9799a7;  font-family: Georgia, Times, "Times New Roman", serif;  font-style: italic;  margin-bottom: 18px;  }  .intro {  text-indent: 15px;  }  .intro a {  font-size: 11px;  font-weight: bold;  text-decoration: underline;  text-transform: uppercase;  } |

#### Text Properties Demo

## In Practice[#practice-2](http://learn.shayhowe.com/html-css/working-with-typography/#practice-2)

With text-based properties under our belts, let’s jump back into our Styles Conference website and put them to work.

1. Currently every link on the page is underlined, which is the default style for anchor elements. This style is a little overbearing at times, though, so we’re going to change it up a bit.

Adding to our links section within our main.css file, we’ll begin by removing the underline from all anchor elements by way of the text-decoration property. Next, we’ll select all anchor elements that appear within a paragraph element and give them a bottom border.

We could use the text-decoration property instead of the border-bottom property to underline all the links within each paragraph; however, by using the border-bottom property we have more control over the underline’s appearance. Here, for example, the underline will be a different color than the text itself.

Our links section, which includes our previous hover styles, should look like this:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | a {  color: #648880;  text-decoration: none;  }  a:hover {  color: #a9b2b9;  }  p a {  border-bottom: 1px solid #dfe2e5;  } |

1. Going back to our <h5> elements from before, which have slightly different styles than the rest of the headings, let’s make them all uppercase using the text-transform property. Our new <h5> element styles should look like this:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | h5 {  color: #a9b2b9;  font-size: 14px;  font-weight: 400;  text-transform: uppercase;  } |

1. Let’s revisit our <header> element to apply additional styles to our navigation menu (to which we previously added the primary-nav class attribute value). After the existing font-size and font-weight properties, let’s add some slight letter-spacing and change our text to all uppercase via the text-transform property.

Our styles for the <nav> element with the primary-nav class attribute value should now look like this:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | .primary-nav {  font-size: 14px;  font-weight: 400;  letter-spacing: .5px;  text-transform: uppercase;  } |

1. Previously, we floated our logo to the left within the <header> element. Now our tagline sits directly to the right of the logo; however, we’d like it to appear all the way to the right of the <header> element, flush right.

We need to add the text-align property with a value of right to the <h3> element with the class attribute value of tagline to get the tagline to sit all the way to the right.

When added to the existing margin property, our new styles for the <h3> element with the class attribute value of tagline will look like this:

|  |  |
| --- | --- |
| 1  2  3  4  5 | .tagline {  margin: 66px 0 22px 0;  text-align: right;  } |

1. We’d also like our navigation menus, both in the <header> and <footer> elements, to sit flush right. Because both the <header> and <footer> elements have child elements that are floated to the left, we can use the same approach as we did with our tagline.

The floated elements within the <header> and <footer> elements are taken out of the normal flow of the page, and this causes other elements to wrap around them. In this specific case, our navigation menus are the elements wrapping around the floated elements.

Because we’ll be sharing the same styles across both navigation menus, we’ll give them each the class of nav. Our <header> element will now look like this:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | <header class="container group">  <h1 class="logo">...</h1>  <h3 class="tagline">...</h3>  <nav class="nav primary-nav">  ...  </nav>  </header> |

And our <footer> element will now look like this:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | <footer class="primary-footer container group">  <small>...</small>  <nav class="nav">  ...  </nav>  </footer> |

Let’s not forget, changes to our <header> and <footer> elements need to be made on every page.

1. With the nav class in place on both navigation menus, let’s create a new section within our main.css file to add shared navigation styles. We’ll begin by adding the text-align property with a value of right to a nav class rule set. We’ll expand these styles later on, but this will serve as a great foundation.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | /\*  ========================================  Navigation  ========================================  \*/  .nav {  text-align: right;  } |

1. While we’re adding the text-align property to a few different elements, let’s also add the text-align property with a value of center to our hero class selector rule set. For reference, these styles, including our existing line-height and padding properties, are located within the home page section of our main.css file.

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | .hero {  line-height: 44px;  padding: 22px 80px 66px 80px;  text-align: center;  } |

Our Styles Conference now has some serious style. (Bad joke, sorry.) Seriously, though, all of our styles are coming along quite well, and our website is progressing.

**Fig 6**

Our Styles Conference website is coming along quite well after adding a few text-based properties

## Using Web-Safe Fonts[#web-safe-fonts](http://learn.shayhowe.com/html-css/working-with-typography/#web-safe-fonts)

By default there are a few fonts that are pre-installed on every computer, tablet, smart-phone, or other web-browsing-capable device. Because they’ve been installed on every device, we can use these fonts freely within our websites, knowing that no matter what device is browsing our site, the font will render properly. These fonts have become known as “web-safe fonts.” There are only a handful of them, and the safest of the web-safe fonts are listed here:

* Arial
* Courier New, Courier
* Garamond
* Georgia
* Lucida Sans, Lucida Grande, Lucida
* Palatino Linotype
* Tahoma
* Times New Roman, Times
* Trebuchet
* Verdana

## Embedding Web Fonts[#embedding-web-fonts](http://learn.shayhowe.com/html-css/working-with-typography/#embedding-web-fonts)

We also have the ability to upload fonts to a server and include them on a website via the CSS @font-face at-rule. This capability has done wonders for online typography. Now, more than ever, typography is coming to life online.

Embedding our own web fonts looks a bit like the following CSS. First, we use the @font-face at-rule to identify our font’s name, via the font-family property, as well as the source of our font (the path to the font file containing our chosen font), via the src property. From there we are able to use this font by including its name within any font-family property value.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | @font-face {  font-family: "Lobster";  src: local("Lobster"), url("lobster.woff") format("woff");  }  body {  font-family: "Lobster", "Comic Sans", cursive;  } |

#### Text Properties Demo

Having the ability to embed any typeface on a website does not mean we legally have the authority to do so. Typefaces are works of art, and posting them on our server may allow others to easily steal them. The authority to use a typeface depends on the licensing we’ve been warranted.

Fortunately, the value of using new typefaces online has been recognized, and companies have begun developing ways to license and include new fonts on websites. Some of these companies, like [Typekit](https://typekit.com/) and [Fontdeck](http://fontdeck.com/), work off a subscription model for licensing fonts, while others, like [Google Fonts](https://www.google.com/fonts), license the fonts for free. Before uploading any fonts, let’s make sure we have permission to do so.

## In Practice[#practice-3](http://learn.shayhowe.com/html-css/working-with-typography/#practice-3)

To add a little character to our Styles Conference website, let’s try using a Google Font on our website.

1. Let’s head over to the [Google Fonts website](https://www.google.com/fonts) and search for the font we’d like to use: Lato. Once we’ve found it, let’s proceed with adding it to our collection and following the steps on their website to use the font.

When the time comes to choose which font weights we’d like to use, let’s make sure to select 300 and 400, as we’ve already been using those within our CSS. Let’s also add 100 to the collection for another variation, too.

Google will give us an additional <link> element to include in the <head> element of all of our pages. We’ll place this new <link> element directly below our existing <link> element. The new element will include the proper style sheet reference to Google, which will take care of including a new CSS file with the proper @font-face at-rule necessary for us to use the Lato font.

With the addition of the new <link> element, our <head> element will look like this:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | <head>  <meta charset="utf-8">  <title>Styles Conference</title>  <link rel="stylesheet" href="assets/stylesheets/main.css">  <link rel="stylesheet"  href="http://fonts.googleapis.com/css?family=Lato:100,300,400">  </head> |

1. Once we have added the new <link> element to all of our pages, we are ready to begin using the Lato font. We’ll do so by adding it to our primary font stack within the font property inside our <body> element styles.

Let’s add Lato to the beginning of our font stack to make it "Lato", "Open Sans", "Helvetica Neue", Helvetica, Arial, sans-serif.

Although Lato is a single word, because it is an embedded web font we’ll want to surround it with quotation marks within any CSS reference. Our new <body> element styles will look like this:

|  |  |
| --- | --- |
| 1  2  3  4  5 | body {  color: #888;  font: 300 16px/22px "Lato", "Open Sans", "Helvetica Neue", Helvetica, Arial, sans-serif;  } |

1. Lato should now be up and running, visible in all of our text across the Styles Conference website. Let’s take a closer look at our logo and update it a bit.

Within our logo class selector rule set, we’ll begin by adding the font-weight property with a value of 100 to make the text fairly thin. We’ll also use the text-transform property with a value of uppercase to make all of the letters uppercase, as well as the letter-spacing property with a value of .5 pixels to add a tiny bit of space between each letter within the logo.

Altogether the styles for our logo will look like this:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | .logo {  border-top: 4px solid #648880;  float: left;  font-size: 48px;  font-weight: 100;  letter-spacing: .5px;  line-height: 44px;  padding: 40px 0 22px 0;  text-transform: uppercase;  } |

1. Because we have a font-weight property value of 100 available, let’s also set the paragraph element within our hero section to that weight. We can use the existing selector to do so, and the new rule set will look like this:

|  |  |
| --- | --- |
| 1  2  3  4  5 | .hero p {  font-size: 24px;  font-weight: 100;  } |

Our Styles Conference website has taken quite a few large steps this lesson, and the look and feel of our website is starting to really shine.

**Fig 6**

Our Styles Conference home page after adding the Lato Google web font

### Demo & Source Code

Below you may view the Styles Conference website in its current state, as well as download the source code for the website in its current state.

[**View the Styles Conference Website**](http://learn.shayhowe.com/practice/working-with-typography/index.html) or [Download the Source Code](http://learn.shayhowe.com/practice/working-with-typography.zip) (Zip file)

## Including Citations & Quotes[#citations-quotes](http://learn.shayhowe.com/html-css/working-with-typography/#citations-quotes)

Writing online sometimes involves citing different sources or quotations. All of the different citation and quotation cases can be covered semantically in HTML using the <cite>, <q>, and <blockquote> elements. Because they are usually distinguished from regular text in appearance, we’ll discuss them here in the typography lesson.

Knowing when to use which element and attribute to properly mark up citations and quotes takes a bit of practice. In general, follow these rules:

* <cite>: Used to reference a creative work, author, or resource
* <q>: Used for short, inline quotations
* <blockquote>: Used for longer external quotations

### Citing a Creative Work

The <cite> inline element is used in HTML to specifically cite a creative work; the element must include either the title of the work, the author’s name, or a URL reference to the work. By default, content wrapped within the <cite> element will appear in italics within the browser.

For additional reference, it helps to include a hyperlink to the original source of the citation when relevant.

Here the book Steve Jobs, by Walter Isaacson, is referenced within the <cite> element. Inside the citation is also a hyperlink to the book.

|  |  |
| --- | --- |
| 1  2 | <p>The book <cite><a href="http://www.amazon.com/Steve-Jobs-Walter-Isaacson/dp/1451648537">Steve Jobs</a></cite> is truly inspirational.</p> |

#### Citing a Creative Work Demo

### Dialogue & Prose Quotation

Quite often, dialogue or prose is quoted inline, within other text. For this purpose, the <q> (or quote) inline element should be applied. The <q> element semantically indicates quoted dialogue or prose and shouldn’t be used for any other purposes.

By default, the browser will insert the proper quotation marks for us and will even change the quotation marks based on the language identified within the lang global attribute.

Here’s an example:

|  |  |
| --- | --- |
| 1  2 | <p>Steve Jobs once said, <q>One home run is much better than two doubles.</q></p> |

### Dialogue & Prose Citation

An optional attribute to include on the <q> element is the cite attribute. The cite attribute acts as a citation reference to the quote in the form of a URL. This attribute doesn’t alter the appearance of the element; it simply adds value for screen readers and other devices. Because the attribute isn’t viewable within the browser, it’s also helpful to provide a hyperlink to this source next to the actual quotation.

Here’s an example:

|  |  |
| --- | --- |
| 1  2 | <p><a href="http://www.businessweek.com/magazine/content/06\_06/b3970001.htm">Steve Jobs</a> once said, <q cite="http://www.businessweek.com/magazine/content/06\_06/b3970001.htm">One home run is much better than two doubles.</q></p> |

#### Dialogue & Prose Citation Demo

### External Quotation

To quote a large block of text that comes from an external source and spans several lines, we’ll use the <blockquote> element. The <blockquote> is a block-level element that may have other block-level elements nested inside it, including headings and paragraphs.

Here’s an example that uses the <blockquote> element:

|  |  |
| --- | --- |
| 1  2  3  4 | <blockquote>  <p>&#8220;In most people&#8217;s vocabularies, design is a veneer. It&#8217;s interior decorating. It&#8217;s the fabric of the curtains, of the sofa. But to me, nothing could be further from the meaning of design. Design is the fundamental soul of a human-made creation that ends up expressing itself in successive outer layers of the product.&#8221;</p>  </blockquote> |

### External Citation

Longer quotes used within the <blockquote> element will often include a citation. This citation may comprise both the cite attribute and the <cite> element.

The cite attribute can be included on the <blockquote> element—in the same way that it was used on the <q> element earlier—to provide a citation reference to the quote in the form of a URL. The <cite> element then can fall after the actual quote itself to specify the original source of the quote, if relevant.

The HTML here outlines an extended quote from Steve Jobs that originally appeared in Fortune magazine. The quotation is marked up using the <blockquote> element with a cite attribute to specify where the quote originally appeared. In the <blockquote> element, the <cite> element, along with an <a> element, provides an additional citation and reference for the quote that is visible to users.

|  |  |
| --- | --- |
| 1  2  3  4  5 | <blockquote cite="http://money.cnn.com/magazines/fortune/fortune\_archive/2000/01/24/272277/index.htm">  <p>&#8220;In most people&#8217;s vocabularies, design is a veneer. It&#8217;s interior decorating. It&#8217;s the fabric of the curtains, of the sofa. But to me, nothing could be further from the meaning of design. Design is the fundamental soul of a human-made creation that ends up expressing itself in successive outer layers of the product.&#8221;</p>  <p><cite>&#8212; Steve Jobs in <a href="http://money.cnn.com/ magazines/fortune/fortune\_archive/2000/01/24/272277/index.htm"> Fortune Magazine</a></cite></p>  </blockquote> |

#### External Citation Demo

## Summary[#summary](http://learn.shayhowe.com/html-css/working-with-typography/#summary)

Learning how to style text is exciting, as our content can begin to convey some emotion. We can also start to play around with the hierarchy of our content, making our website more legible and digestible.

To quickly recap, within this lesson we discussed the following:

* Adding color to our text to enhance it
* Applying font-based properties, including font-family, font-size, font-style, font-weight, and more
* Applying text-based properties, including text-align, text-decoration, text-indent, text-shadow, and more
* The history behind web-safe fonts and how to embed our own web fonts
* How to properly mark up citations and quotations

Sharpening up our text and dabbling a bit with typography has brought our design along quite a way. Next, we’ll bring a little more color to our website by going over backgrounds and gradients.

###### Lesson 7

# Setting Backgrounds & Gradients

Backgrounds have a significant impact on the design of a website. They help create a site’s look and feel, establish groupings, and assign priority, and they have a considerable influence on a website’s usability.

Within CSS, element backgrounds can be a solid color, an image, a gradient, or a combination of these. As we decide how to implement these backgrounds, we should keep in mind that every background contributes to the overall appearance of our website.

In this lesson we’re going to take a look at how to assign different types of backgrounds, including gradients, to elements; we’ll also play around with a handful of CSS3 properties specific to backgrounds.

## Adding a Background Color[#background-color](http://learn.shayhowe.com/html-css/setting-backgrounds-and-gradients/#background-color)

The quickest way to add a background to an element is to add a single-color background using the background or background-color property. The background property accepts colors and images in shorthand form, while the background-color property is used strictly for setting solid background colors. Either property will work, and which one you decide to use depends on your preference as well as the case for which you’re using it.

|  |  |
| --- | --- |
| 1  2  3  4 | div {  background-color: #b2b2b2;  } |

When adding a background color, we have a few options for the values we can use. As with other color values, we can pick from keywords, hexadecimal codes, and RGB, RGBa, HSL, and HSLa values. Most commonly we’ll see hexadecimal values; however, we may occasionally want to use RGBa or HSLa values for transparencies.

#### Transparent Backgrounds

When using an RGBa or HSLa value as a transparent background color, it’s a good idea to provide a fallback color, too, because not all browsers recognize RGBa or HSLa values. And when a browser comes across a value it doesn’t recognize, it will ignore it.

Fortunately, there is an easy way to provide a fallback background. CSS cascades from the top of a file to the bottom of a file; thus, we can use two background-color properties within a single rule set. The first background-color property will use a “safe” background color, such as a hexadecimal value, and the second background-color property will use an RGBa or HSLa value. Here, if a browser understands the RGBa or HSLa value it will render it, and if it doesn’t, it will fall back to the hexadecimal value above it.

|  |  |
| --- | --- |
| 1  2  3  4  5 | div {  background-color: #b2b2b2;  background-color: rgba(0, 0, 0, .3);  } |

## Adding a Background Image[#background-image](http://learn.shayhowe.com/html-css/setting-backgrounds-and-gradients/#background-image)

Besides adding a background color to an element, we can also add a background image. Background images work similarly to background colors; however, they offer a few additional properties to finesse the images. As before, we can use the background property with a shorthand value, or we can use the background-image property outright. No matter which property we use, there must be an image source identified using a url() function.

The url() function value will be the background image’s path, and the familiar rules for creating hyperlink paths apply here. Keep an eye out for different directories, and be sure to show exactly where the image resides. The path will be placed inside parentheses and quoted.

|  |  |
| --- | --- |
| 1  2  3  4 | div {  background-image: url("alert.png");  } |

Adding a background image solely using a url value can provide undesirable results, as by default the background image will repeat horizontally and vertically from the top left of the given element to fill up the element’s background. Thankfully we can use the background-repeat and background-position properties to control how or even whether the image repeats.

### Background Repeat

By default, a background image will repeat indefinitely, both vertically and horizontally, unless otherwise specified. The background-repeat property may be used to change the direction in which a background image is repeated, if repeated at all.

|  |  |
| --- | --- |
| 1  2  3  4  5 | div {  background-image: url("alert.png");  background-repeat: no-repeat;  } |

The background-repeat property accepts four different values: repeat, repeat-x, repeat-y, and no-repeat. The repeat value is the default value and will repeat a background image both vertically and horizontally.

The repeat-x value will repeat the background image horizontally, while the repeat-y value will repeat the background image vertically. Lastly, the no-repeat value will tell the browser to display the background image once—that is, do not repeat it at all.

### Background Position

By default, background images are positioned at the left top corner of an element. However, by using the background-position property, we can control exactly where the background image is placed relative to that corner.

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | div {  background-image: url("alert.png");  background-position: 20px 10px;  background-repeat: no-repeat;  } |

The background-position property requires two values: a horizontal offset (the first value) and a vertical offset (the second value). If only one value is specified, that value is used for the horizontal offset and the vertical offset will default to 50%.

Because we’re moving the background image from the left top corner of the element, length values specifically will be in relation to that corner.

To set a background-position value, we can use the top, right, bottom, and left keywords, pixels, percentages, or any length measurement. Keywords and percentages work very similarly. The keyword value left top is identical to the percentage value 0 0, which will keep an image positioned at the left top corner of the element. The keyword value right bottom is identical to the percentage value 100% 100%, which will position an image in the right bottom corner of the element.

**Fig 7**

Background images are positioned from the left top corner of an element

One advantage of percentages over keywords is the ability to center a background image by using 50% as a value. To center the background image at the top of an element, we can use the value 50% 0. Using pixels for a background-position value is also common, as pixels give us precise control over where our background will be positioned.

### Shorthand Background Image Values

The background-color, background-image, background-position, and background-repeat properties may be rolled up into a shorthand value for the background property alone. The order of these properties as a shorthand background property value may vary, however it commonly falls as background-color, background-image, background-position, and then background-repeat.

|  |  |
| --- | --- |
| 1  2  3  4 | div {  background: #b2b2b2 url("alert.png") 20px 10px no-repeat;  } |

### Background Image Example

In the following example, we’ll use the background property with a shorthand value that includes background-color, background-image, background-position, and background-repeat values.

Please take note that there is both a relative value and an absolute value within the background-position value. The first value, 20 pixels, is the horizontal value, positioning the background-image 20 pixels from the left of the element. The second value, 50%, is the vertical value, which vertically centers the background-image.

A few other properties and values are also included within the notice-success class rule set to further style the alert message.

###### HTML

|  |  |
| --- | --- |
| 1  2  3  4 | <div class="notice-success">  Woo hoo! Congratulations, you did it!  </div> |

###### CSS

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | .notice-success {  background: #67b11c url("tick.png") 20px 50% no-repeat;  border: 2px solid #467813;  border-radius: 5px;  color: #fff;  font-family: "Helvetica Neue", Helvetica, Arial, sans-serif;  padding: 15px 20px 15px 50px;  } |

#### Background Image Demo

## In Practice[#practice-1](http://learn.shayhowe.com/html-css/setting-backgrounds-and-gradients/#practice-1)

Returning to our Styles Conference website, let’s add some background colors. While we do that, we’ll change a few other styles to keep all of our styles working together and to keep all of our content legible.

1. We’ll begin by taking a big step and applying a blue background to the <body> element alongside the existing color and font properties. All of the styles for the <body> element rule set now include the following:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | body {  background: #293f50;  color: #888;  font: 300 16px/22px "Lato", "Open Sans", "Helvetica Neue", Helvetica, Arial, sans-serif;  } |

1. We’ve placed a blue background on the <body> element purposely, as our website will have a few different rows of background colors, and the most frequent background color will be blue.
2. Now that every page on our Styles Conference website includes a blue background, let’s clean up a few areas that will keep that blue background. Specifically, our <header> and <footer> elements will remain blue, as will the hero section on the home page.

Within our <header> and <footer> elements let’s make all of our link colors start as white and then, when hovered over, turn the same green as our headings.

We’ll begin with our <header> element. In order to select all <a> elements within the <header> element, we’ll add a class of primary-header to the <header> element (in addition to the existing container and group classes). Don’t forget, we’ll need to add this class to the <header> elements across all of our pages.

|  |  |
| --- | --- |
| 1  2  3  4 | <header class="primary-header container group">  ...  </header> |

With the primary-header class in place on the <header> element, and the existing primary-footer class in place on the <footer> element, we can add two new rule sets to the bottom of the links section within our main.css file.

The first rule set will select all <a> elements within an element with the class attribute value of primary-header or primary-footer and set their color to white, as defined by comma separating two individual selectors that share the same property and value. The second rule set will select the same <a> elements as before but will change their color to green when a user hovers over them.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | .primary-header a,  .primary-footer a {  color: #fff;  }  .primary-header a:hover,  .primary-footer a:hover {  color: #648880;  } |

1. While we’re making some of our text white, let’s make the text within the hero section of our home page white also, as it will remain on a blue background. We have the existing hero class rule set available to add styles to, so let’s add our white text color there. In all, our hero class rule set should include the following:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | .hero {  color: #fff;  line-height: 44px;  padding: 22px 80px 66px 80px;  text-align: center;  } |

1. Also within the hero section of our home page, let’s clean up some of the button styles. We’ll begin by adding some new properties to our btn class rule set, within the buttons section of our main.css file.

Specifically, let’s set the button text color to white, make sure our cursor is always a pointer, increase the font-weight, add a small amount of letter-spacing, and change our text-transform to uppercase.

In all, our new btn class rule set should look like this:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | .btn {  border-radius: 5px;  color: #fff;  cursor: pointer;  display: inline-block;  font-weight: 400;  letter-spacing: .5px;  margin: 0;  text-transform: uppercase;  } |

We’ll also clean up some of the alternate button styles by way of the btn-alt class rule set. Specifically, let’s make the buttons’ borders white and add hover styles including a white background and blue text color.

With all of the additions, our new btn-alt class rule set should look like this:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | .btn-alt {  border: 1px solid #fff;  padding: 10px 30px;  }  .btn-alt:hover {  background: #fff;  color: #648880;  } |

1. Now that we have all of the areas with blue backgrounds cleaned up, let’s add styles for the rows that have white backgrounds. Let’s create a new section within our main.css file for rows, just below the clearfix section. Within this new rows section, let’s create a new class selector named row.

Within our new row class rule set, let’s add a white background, a minimum width of 960 pixels (to make sure our row elements are always larger than the width of our container or grid elements), and some vertical padding. Altogether our new row section within our main.css file should look like this:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | /\*  ========================================  Rows  ========================================  \*/  .row {  background: #fff;  min-width: 960px;  padding: 66px 0 44px 0;  } |

1. With our row class styles in place, let’s add a row with a white background to our home page. We’ll do this on our teasers section. Currently this area has a <section> element with the class of grid wrapping three additional <section> elements with the classes of teaser and col-1-3.

To add a white background to this section, we’re going to wrap all of these elements in an element with the class of row.

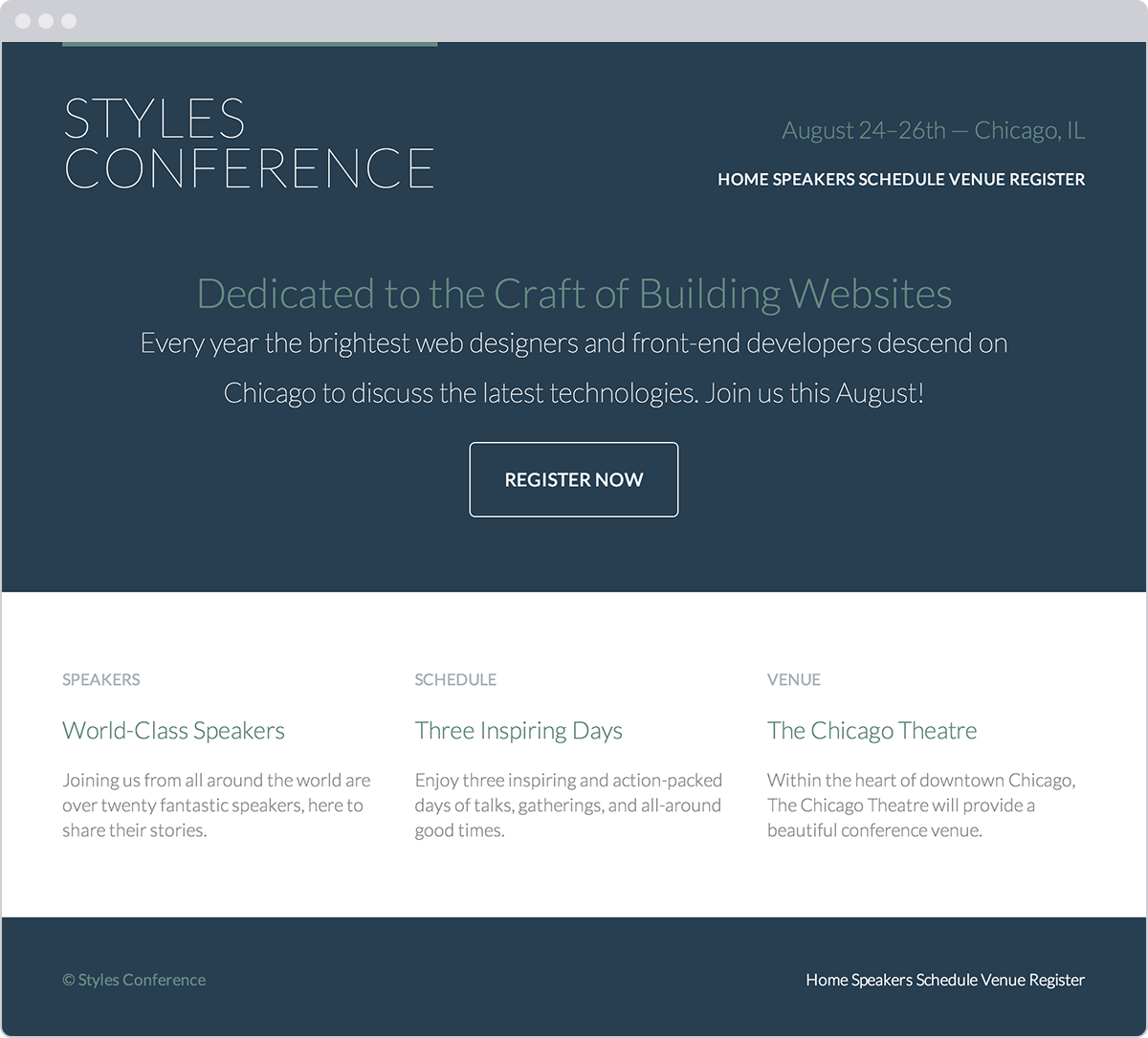
Because we’ll want the entire teasers section wrapped in a <section> element, we’re going to add a new <section> element with the class of row that surrounds the existing <section> element with the class of grid.

Having two <section> elements wrapping the exact same content diminishes semantic value. To correct this we’ll change the second <section> element, the one with the class of grid, to a <div> element. After all, at this point this element is only adding styles, not semantic meaning, and is appropriate as a <div> element.

The structure of our new teasers element should look like this:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24 | <section class="row">  <div class="grid">  <!-- Speakers -->    <section class="teaser col-1-3">  ...  </section><!--    Schedule    --><section class="teaser col-1-3">  ...  </section><!--    Venue    --><section class="teaser col-1-3">  ...  </section>  </div>  </section> |

It is amazing how a few background colors can affect the design of a website. Our Styles Conference website is coming along quite nicely, and our home page is proof.

**Fig 7**

Our Styles Conference website home page after adding some background colors

## Designing Gradient Backgrounds[#gradient-backgrounds](http://learn.shayhowe.com/html-css/setting-backgrounds-and-gradients/#gradient-backgrounds)

Gradient backgrounds were introduced with CSS3, and designers and front-end developers everywhere rejoiced. Although gradient backgrounds do not work in legacy browsers, they are supported by all modern browsers.

Within CSS, gradient backgrounds are treated as background images. We can create a gradient using the background or background-image properties, just like a regular background image. The property value for a gradient background varies depending on what type of gradient we’d like, linear or radial.

#### Gradient Background Vendor Prefixes

In Lesson 4, “[Opening the Box Model](http://learn.shayhowe.com/html-css/opening-the-box-model/),” we discussed adding vendor prefixes to new CSS properties or values so that browsers can support recently developed CSS features. Gradient background values were one of the values that required the use of vendor prefixes. Fortunately, most browsers have since eliminated the need for a vendor prefix in order to render a gradient background; however, it is still worth outlining vendor prefixes to ensure the best support.

At first, as we begin discussing linear gradient backgrounds, we’ll include each of the different vendor prefixes. After that, in the interest of brevity, we’ll omit the different prefixes as we continue to discuss gradient backgrounds, including radial gradient backgrounds.

### Linear Gradient Background

For years designers and developers have been cutting up gradient image files, created using image-processing software, and using them as linear gradient backgrounds on elements. The process worked, but it took a while to implement and was very inflexible. Fortunately those days are gone, and [linear gradient backgrounds](http://dev.opera.com/articles/css3-linear-gradients/) can now be specified within CSS. If a color needs changing, there is no need to reproduce and recut an image and upload it to the server. Now all we need to do is change a quick value within CSS. Beautiful.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | div {  background: #466368;  background: -webkit-linear-gradient(#648880, #293f50);  background: -moz-linear-gradient(#648880, #293f50);  background: linear-gradient(#648880, #293f50);  } |

#### Linear Gradient Background Demo

Linear gradients are identified by using the linear-gradient() function within the background or background-image property. The linear-gradient() function must include two color values, the first of which will be the beginning color value and the second of which will be the ending color value. The browser will then handle the transition between the two colors.

Before any gradient backgrounds are identified, we’ll also put in a default background property with a solid color. The solid color is to be used as a fallback should a browser not support gradient backgrounds.

### Changing the Direction of a Gradient Background

By default, linear gradient backgrounds move from the top to the bottom of an element, transitioning smoothly between the first color value and the second. This direction, however, may be changed with the use of keywords or a degree value stated before any color values.

For example, should we want a gradient to move from the left of an element to the right, we can use the keyword value to right to identify the direction in which the linear gradient should progress. Keyword values may also be combined. If we want the gradient to move from the left top to the right bottom of an element, we can use the keyword value of to right bottom.

|  |  |
| --- | --- |
| 1  2  3  4  5 | div {  background: #466368;  background: linear-gradient(to right bottom, #648880, #293f50);  } |

#### Linear Gradient Background Direction Demo

When we use a diagonal gradient on an element that isn’t exactly square, the background gradient will not proceed directly from one corner to the other. Instead, the gradient will identify the absolute center of the element, place anchors in the perpendicular corners from where it should progress, and then move to the general direction of the corner stated within the value. These corners the gradient moves towards are called “magic corners,” as they are not absolute. Eric Meyer has done a wonderful job of outlining this syntax in his article “[Linear Gradient Keywords](http://meyerweb.com/eric/thoughts/2012/04/26/lineargradient-keywords/)”.

Besides keywords, degree values are also acceptable. If we want our gradient to move to the left top of an element, we can use the degree value of 315deg, or if we want our gradient to move to the right bottom of an element, we can use the degree value of 135deg. This same concept can be applied for any degree value, 0 through 360.

### Radial Gradient Background

While the linear gradient is perfect for a gradient moving from one direction to another, often the need for a radial gradient arises. Radial background gradients work just like linear gradients and share many of the same values. For radial gradients, instead of using the linear-gradient() function within the background or background-image property, we’ll use the radial-gradient() function.

|  |  |
| --- | --- |
| 1  2  3  4  5 | div {  background: #466368;  background: radial-gradient(#648880, #293f50);  } |

#### Radial Gradient Background Demo

Radial gradients work from the inside to the outside of an element. Thus, the first color identified within the radial-gradient() function will sit in the absolute center of the element, and the second color will sit on the outside of an element. The browser will then create the transition between the two colors.

One of the primary differences between radial gradients and linear gradients is that radial gradients can be quite complex, with values for location, size, radius, and so forth. We’ll cover the basics, but please feel free to delve further into [radial gradients](http://dev.opera.com/articles/css3-radial-gradients/), as they provide much more power than is outlined here.

#### CSS3 Gradient Background Generator

Working with CSS3 gradients by hand can be quite difficult at times, especially if you are new to them. Fortunately, a few [CSS3 gradient generators](http://www.cssmatic.com/gradient-generator) have popped up. Each generator works a little differently, and some provide more options than others. If you’re interested, I recommend doing some research to find the right generator for your needs.

### Gradient Color Stops

At a minimum, gradient backgrounds will transition from one color to another; however, we may add multiple colors to a gradient and have the browser transition between all of them. To do this we’ll add color stops to the given gradient function, with commas separating each color stop from the next.

|  |  |
| --- | --- |
| 1  2  3  4  5 | div {  background: #648880;  background: linear-gradient(to right, #f6f1d3, #648880, #293f50);  } |

#### Gradient Color Stops Demo

By default, the browser will position every color stop an equal distance from the next and will transition between them accordingly. If more control over how colors are positioned is desired, a location along the gradient may be identified for each color stop. The location should be declared as a length value and should fall after the color value.

|  |  |
| --- | --- |
| 1  2  3  4  5 | div {  background: #648880;  background: linear-gradient(to right, #f6f1d3, #648880 85%, #293f50);  } |

#### Gradient Color Stop Positions Demo

Unless specified otherwise, the first color stop will be positioned at 0%, and the last color stop will be positioned at 100%.

### Gradient Background Example

Using the same success alert message from before, we’ll swap out the old background image for a linear gradient background image.

For this we’ll include two background properties. The first background property specifies a solid color hexadecimal value, which serves as a fallback should a browser not support linear gradient backgrounds. The second background property includes the linear-gradient() function, which identifies a green gradient background that transitions from the top of the element to the bottom of the element.

###### HTML

|  |  |
| --- | --- |
| 1  2  3  4 | <div class="notice-success">  Woo hoo! Congratulations, you did it!  </div> |

###### CSS

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | .notice-success {  background: #67b11c;  background: linear-gradient(#72c41f, #5c9e19);  border: 2px solid #467813;  border-radius: 5px;  color: #fff;  font-family: "Helvetica Neue", Helvetica, Arial, sans-serif;  padding: 15px 20px;  } |

#### Gradient Background Demo

## In Practice[#practice-2](http://learn.shayhowe.com/html-css/setting-backgrounds-and-gradients/#practice-2)

With gradient backgrounds now in the mix, let’s create a new row for our Styles Conference website, this time using a gradient.

1. We’ll create a new row with a gradient background by using the class of row-alt. Because the new row will share the same min-width property and value as the row class selector, we’ll combine these two selectors.

|  |  |
| --- | --- |
| 1  2  3  4  5 | .row,  .row-alt {  min-width: 960px;  } |

1. Next we’ll want to create new rule sets to apply styles specifically to the row-alt class selector. These new styles will include a gradient background that starts with green and transitions to yellow, from left to right.
2. Using the linear-gradient() function with the appropriate values and vendor prefixes, we’ll add the gradient background to the row-alt class rule set. We’ll also include a single background color before the gradient background as a fallback, just in case a browser doesn’t support gradient backgrounds.
3. Lastly, we’ll also add in some vertical padding. Our updated row section now looks like this:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | .row,  .row-alt{  min-width: 960px;  }  .row {  background: #fff;  padding: 66px 0 44px 0;  }  .row-alt {  background: #cbe2c1;  background: -webkit-linear-gradient(to right, #a1d3b0, #f6f1d3);  background: -moz-linear-gradient(to right, #a1d3b0, #f6f1d3);  background: linear-gradient(to right, #a1d3b0, #f6f1d3);  padding: 44px 0 22px 0;  } |

1. With our row-alt styles in place, let’s put them to use on all of our interior pages. Currently, all of our interior pages have a <section> element with a class of container. Then, inside each <section> element is an <h1> element containing the heading of the page.

We’re going to alter these <section> elements much like we did the teaser <section> element on our home page. We’ll wrap each <section> element with a class of container in a <section> element with the class of row-alt. We’ll then change each <section> element with a class of container to a <div> element for better semantic alignment.

Each of our interior pages should now include the following:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | <section class="row-alt">  <div class="container">  <h1>...</h1>  </div>  </section> |

1. Because we are updating our interior pages, let’s make their introductions, or leads, a little more appealing. We’ll begin by adding a paragraph introducing each page just below the <h1> element in each <section> element with a class of row-alt. Our speakers.html page, for example, may now include the following lead section:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | <section class="row-alt">  <div class="container">  <h1>Speakers</h1>  <p>We&#8217;re happy to welcome over twenty speakers to present on the industry&#8217;s latest technologies. Prepare for an inspiration extravaganza.</p>    </div>  </section> |

1. In addition to inserting the paragraph, let’s also change some of the styles within the lead section. To do this, we’ll add a class of lead to the <div> element that already has a class of container; this can be found nested directly inside the <section> element with a class of row-alt. Our lead section for each interior page will now look like this:

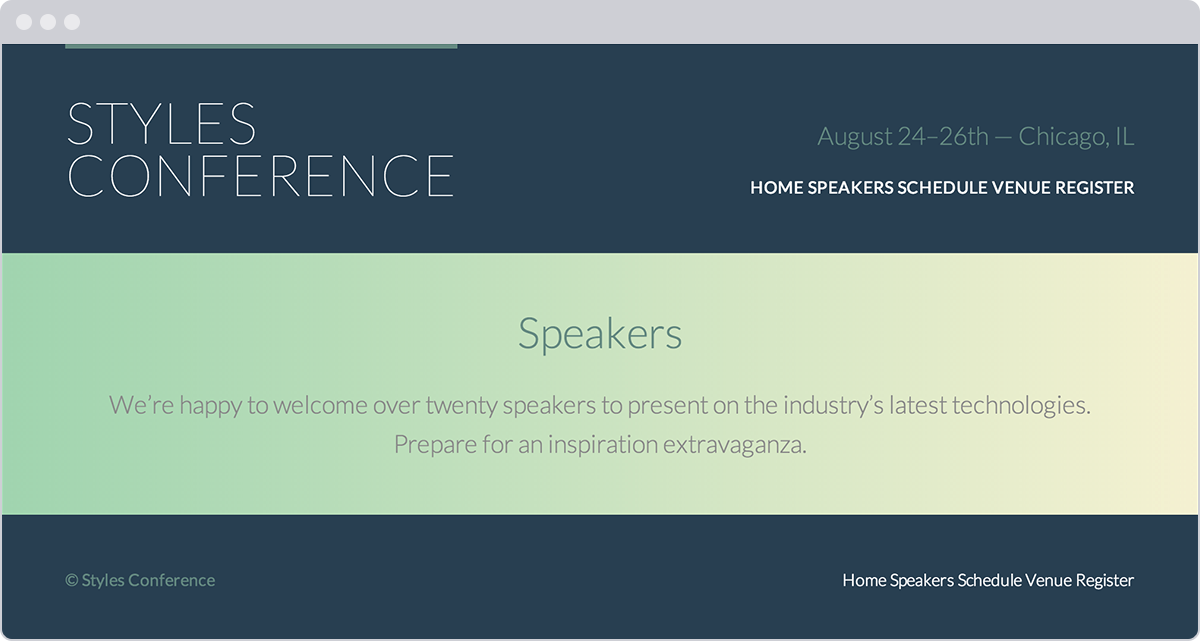
|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | <section class="row-alt">  <div class="lead container">    ...  </div>  </section> |

1. Once the lead class is in place, we’ll center all of the text within these <div> elements. We’ll also increase the font-size and line-height of any paragraphs within these <div> elements.

We’ll create a new section for leads within our main.css file, just below the typography section, and add the following styles:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | /\*  ========================================  Leads  ========================================  \*/  .lead {  text-align: center;  }  .lead p {  font-size: 21px;  line-height: 33px;  } |

The interior pages of our Styles Conference website have now received some long-overdue love in the form of gradient background rows and leads. Make sure to review the code for all of the interior pages to see their newly enhanced content, headings, and paragraphs.

**Fig 7**

The Speakers page of our Styles Conference website, complete with a gradient background row

### Demo & Source Code

Below you may view the Styles Conference website in its current state, as well as download the source code for the website in its current state.

[**View the Styles Conference Website**](http://learn.shayhowe.com/practice/setting-backgrounds-and-gradients/index.html) or [Download the Source Code](http://learn.shayhowe.com/practice/setting-backgrounds-and-gradients.zip) (Zip file)

## Using Multiple Background Images[#multiple-background-images](http://learn.shayhowe.com/html-css/setting-backgrounds-and-gradients/#multiple-background-images)

For the longest time, elements were allowed to have only one background image at a time, which created quite a few constraints when designing a page. Fortunately, with CSS3, we can now use more than one background image on an element by comma-separating multiple background values within a background or background-image property.

The background image value that comes first will be the foremost background image, and the background image that’s listed last will be the rearmost background image. Any value between the first and the last will reside within the middle ground accordingly. Here’s an example of the CSS for a <div> element that uses three background images:

|  |  |
| --- | --- |
| 1  2  3  4 | div {  background: url("foreground.png") 0 0 no-repeat, url("middle-ground.png") 0 0 no-repeat, url("background.png") 0 0 no-repeat;  } |

The preceding code uses a shorthand value for the background property, chaining multiple background image values together. These shorthand values may also be broken up into comma-separated values across the background-image, background-position, and background-repeat properties.

### Multiple Background Images Example

Let’s go back to the success alert message once more to combine both the tick background image and the linear gradient background image.

In order to do so, we’ll include two values within the second background property. The first value, the foremost image, will be the tick image. The second value, the rearmost image, will be the linear gradient. The two values are comma separated.

###### HTML

|  |  |
| --- | --- |
| 1  2  3  4 | <div class="notice-success">  Woo hoo! Congratulations, you did it!  </div> |

###### CSS

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | .notice-success {  background: #67b11c;  background: url("tick.png") 20px 50% no-repeat, linear-gradient(#72c41f, #5c9e19);  border: 2px solid #467813;  border-radius: 5px;  color: #fff;  font-family: "Helvetica Neue", Helvetica, Arial, sans-serif;  padding: 15px 20px 15px 50px;  } |

#### Multiple Background Images Demo

## Exploring New Background Properties[#css3-background-properties](http://learn.shayhowe.com/html-css/setting-backgrounds-and-gradients/#css3-background-properties)

Along with gradient backgrounds and multiple background images, CSS3 also introduced three [new CSS properties](http://css3files.com/background/): background-size, background-clip, and background-origin.

The background-size property allows us to change the size of a background image, while the background-clip and background-origin properties allow us to control where a background image is cropped and where a background image is contained within the element (inside the border or inside the padding, for example).

### CSS3 Background Size

The background-size property allows us to specify a size for a background image. The property accepts a few different values, including length and keyword values.

When using length values, we can specify a width and a height value by using two space-separated values. The first value will set the width of the background image, while the second value will set the height of the background image. It’s important to note that percentage values are in relation to the element’s size, not the background image’s original size.

Consequently, setting a background-size property with a 100% width will make the background image occupy the full width of the element. If a second value isn’t identified after the width, the height value will be automatically set to preserve the aspect ratio of the background image.

The keyword value auto may be used as either the width or height value to preserve the aspect ratio of the background image. For example, if we want to set the height of the background image to be 75% of the height of the element while maintaining the image’s aspect ratio, we can use a background-size property value of auto 75%.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | div {  background: url("shay.jpg") 0 0 no-repeat;  background-size: auto 75%;  border: 2px dashed #9799a7;  height: 240px;  width: 200px;  } |

#### CSS3 Background Size Demo

### Cover & Contain Keyword Values

In addition to length background-size property values, there are also cover and contain keyword values available to the background-size property.

The cover keyword value specifies that the background image will be resized to completely cover an element’s width and height. The background image’s original aspect ratio will be preserved, yet the image will stretch or shrink as necessary to cover the entire element. Often when using the cover keyword value, part of the background image is cut off in order for the image to occupy the full available space of the element.

The contain keyword value, on the other hand, specifies that the background image will be resized to reside entirely contained within an element’s width and height. In doing so the background image’s original aspect ratio will be preserved, but the image will stretch or shrink as necessary to remain within the width and height of the element. In contrast with the cover keyword value, the contain keyword value will always show the full background image; however, oftentimes it will not occupy the full available space of the element.

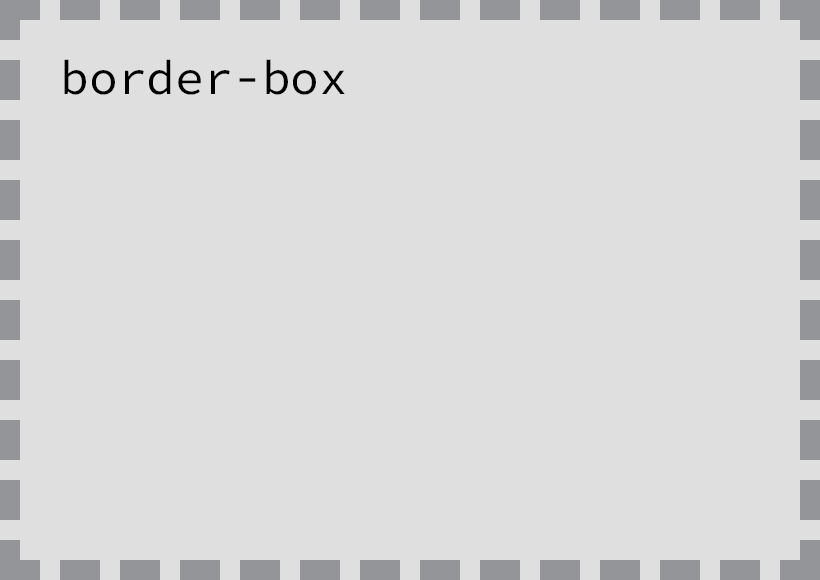
Both the cover and contain keyword values may result in slightly distorted background images, particularly when the images are stretched beyond their original dimensions. We’ll want to keep an eye out for this when using these values, to make sure the resulting styles are satisfactory.

### CSS3 Background Clip & Background Origin

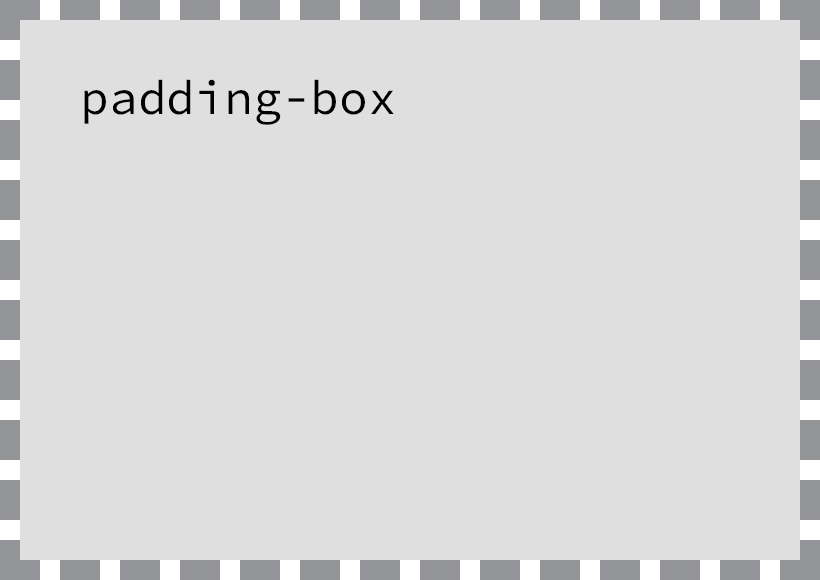
The background-clip property specifies the surface area a background image will cover, and the background-origin property specifies where the background-position should originate. The introduction of these two new properties corresponds with the introduction of three new keyword values: border-box, padding-box, and content-box. Each of these three values may be used for the background-clip and background-origin properties.

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | div {  background: url("shay.jpg") 0 0 no-repeat;  background-clip: padding-box;  background-origin: border-box;  } |

The background-clip property value is set to border-box by default, allowing a background image to extend into the same area as any border. Meanwhile, the background-origin property value is set to padding-box by default, allowing the beginning of a background image to extend into the padding of an element.

**Fig 7**

The border-box value extends the background into the border of an element

**Fig 7**

The padding-box value extends the background into the padding of an element, but the background is contained within any border

**Fig 7**

The content-box value contains the background within the border and padding of an element

We first discussed these keyword values when we covered the box-sizing property back in Lesson 4, “[Opening the Box Model](http://learn.shayhowe.com/html-css/opening-the-box-model/).” The values themselves haven’t changed in meaning, but their functions do change with the use of the different background properties.

## Summary[#summary](http://learn.shayhowe.com/html-css/setting-backgrounds-and-gradients/#summary)

Adding backgrounds and gradients to our pages allows us to bring color to the forefront of our designs. These features also help to define how content is grouped and to improve the layout of our pages as a whole.

To review, this lesson covered the following:

* How to add background colors and images to elements
* CSS gradients, both linear and radial, and how to customize them
* How to apply multiple background images to a single element
* New CSS3 properties that allow us to change the size, surface area, and origin of background images

Adding background colors, gradients, and images brings forth quite a few possibilities to enhance the overall design of our websites. Soon we’ll discuss how to semantically add images (aside from background images) and other media to our pages. But before that, let’s take a look at how to semantically create lists.

###### Lesson 8

# Creating Lists

Lists are a part of everyday life. To-do lists determine what to get done. Navigational routes provide turn-by-turn lists of directions. Recipes provide lists of ingredients and lists of instructions. With a list for nearly everything, it’s easy to understand why they are also popular online.

When we want to use a list on a website, HTML provides three different types to choose from: unordered, ordered, and description lists. Choosing which type of list to use—or whether to use a list at all—comes down to the content and the most semantically appropriate option for displaying that content.

In addition to the three different types of lists available within HTML, there are multiple ways to style these lists with CSS. For example, we can choose what type of marker to use on a list. The marker could be square, round, numeric, alphabetical, or perhaps nonexistent. Also, we can decide if a list should be displayed vertically or horizontally. All of these choices play significant roles in the styling of our web pages.

## Unordered Lists[#unorderd-lists](http://learn.shayhowe.com/html-css/creating-lists/#unorderd-lists)

An unordered list is simply a list of related items whose order does not matter. Creating an unordered list in HTML is accomplished using the unordered list block-level element, <ul>. Each item within an unordered list is individually marked up using the list item element, <li>.

By default, most browsers add a vertical margin and left padding to the <ul> element and precede each <li> element with a solid dot. This solid dot is called the list item marker, and it can be changed using CSS.

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | <ul>  <li>Orange</li>  <li>Green</li>  <li>Blue</li>  </ul> |

#### Unordered Lists Demo

## Ordered Lists[#ordered-lists](http://learn.shayhowe.com/html-css/creating-lists/#ordered-lists)

The ordered list element, <ol>, works very much like the unordered list element; individual list items are created in the same manner. The main difference between an ordered list and an unordered list is that with an ordered list, the order in which items are presented is important.

Because the order matters, instead of using a dot as the default list item marker, an ordered list uses numbers.

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | <ol>  <li>Head north on N Halsted St</li>  <li>Turn right on W Diversey Pkwy</li>  <li>Turn left on N Orchard St</li>  </ol> |

Ordered lists also have unique attributes available to them including start and reversed.

#### Ordered Lists Demo

### Start Attribute

The start attribute defines the number from which an ordered list should start. By default, ordered lists start at 1. However, there may be cases where a list should start at 30 or another number. When we use the start attribute on the <ol> element, we can identify exactly which number an ordered list should begin counting from.

The start attribute accepts only integer values, even though ordered lists may use different numbering systems, such as roman numerals.

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | <ol start="30">  <li>Head north on N Halsted St</li>  <li>Turn right on W Diversey Pkwy</li>  <li>Turn left on N Orchard St</li>  </ol> |

#### Start Attribute Demo

### Reversed Attribute

The reversed attribute, when used on the <ol> element, allows a list to appear in reverse order. An ordered list of five items numbered 1 to 5 may be reversed and ordered from 5 to 1.

The reversed attribute is a Boolean attribute, and as such it doesn’t accept any value. It is either true or false. False is the default value; the value becomes true when the attribute name reversed appears on the <ol> element.

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | <ol reversed>  <li>Head north on N Halsted St</li>  <li>Turn right on W Diversey Pkwy</li>  <li>Turn left on N Orchard St</li>  </ol> |

#### Reversed Attribute Demo

### Value Attribute

The value attribute may be used on an individual <li> element within an ordered list to change its value within the list. The number of any list item appearing below a list item with a value attribute will be recalculated accordingly.

As an example, if the second list item has a value attribute value of 9, the number on that list item marker will appear as if it is the ninth item. All subsequent list items will be numbered upwards from 9.

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | <ol>  <li>Head north on N Halsted St</li>  <li value="9">Turn right on W Diversey Pkwy</li>  <li>Turn left on N Orchard St</li>  </ol> |

#### Value Attribute Demo

## Description Lists[#description-lists](http://learn.shayhowe.com/html-css/creating-lists/#description-lists)

Another type of list seen online (but not as often as unordered or ordered lists) is the description list. Description lists are used to outline multiple terms and their descriptions, as in a glossary, for example.

Creating a description list in HTML is accomplished using the description list block-level element, <dl>. Instead of using a <li> element to mark up list items, the description list requires two block-level elements: the description term element, <dt>, and the description element, <dd>.

A description list may contain numerous terms and descriptions, one after the other. Additionally, a description list may have multiple terms per description, as well as multiple descriptions per term. A single term may have multiple meanings and warrant multiple descriptions. Conversely, a single description may be suitable for multiple terms.

When adding a description list, the <dt> element must come before the <dd> element. The definition term and the description that directly follows it correspond to one another; thus, the order of these elements is important.

By default, the <dl> element will include vertical margins, just like the <ul> and <ol> elements. Additionally, the <dd> element includes a left margin by default.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | <dl>  <dt>study</dt>  <dd>The devotion of time and attention to acquiring knowledge on an academic subject, especially by means of books</dd>  <dt>design</dt>  <dd>A plan or drawing produced to show the look and function or workings of a building, garment, or other object before it is built or made</dd>  <dd>Purpose, planning, or intention that exists or is thought to exist behind an action, fact, or material object</dd>  <dt>business</dt>  <dt>work</dt>  <dd>A person's regular occupation, profession, or trade</dd>  </dl> |

#### Description Lists Demo

## Nesting Lists[#nested-lists](http://learn.shayhowe.com/html-css/creating-lists/#nested-lists)

One feature that makes lists extremely powerful is their ability to be nested. Every list may be placed within another list; they can be nested continually. But the potential to nest lists indefinitely doesn’t provide free rein to do so. Lists should still be reserved specifically for where they hold the most semantic value.

One trick with nesting lists is to know where to begin and end each list and list item. Speaking specifically about unordered and ordered lists, as that is where most nesting will occur, the only element that may reside directly within the <ul> and <ol> elements is the <li> element. To repeat, the only element we can place as a direct child of the <ul> and <ol> elements is the <li> element.

That said, once inside the <li> element, the standard set of elements may be added, including any <ul> or <ol> elements.

To nest a list rather than closing a list item, begin a new list. Once the nested list is complete and closed, close the wrapping list item and continue on with the original list.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | <ol>  <li>Walk the dog</li>  <li>Fold laundry</li>  <li>  Go to the grocery and buy:  <ul>  <li>Milk</li>  <li>Bread</li>  <li>Cheese</li>  </ul>  </li>  <li>Mow the lawn</li>  <li>Make dinner</li>  </ol> |

#### Nesting Lists Demo

Because nesting lists can be a little tricky—and unwanted styles will appear if it’s done incorrectly—let’s quickly review. The <ul> and <ol> elements may contain only <li> elements. The <li> element may contain any normal element as desired; however, the <li> element has to be a direct child of either a <ul> or <ol> element.

It’s also worth noting that as lists are nested inside of other lists, their list item markers will change according to how deeply the list is nested. In the previous example, the unordered list nested within the ordered list uses hollow circles instead of solid discs as the list item marker. This change happens because the unordered list is nested one level into the ordered list.

Fortunately we have control over how these list item markers appear at any level, which we’ll take a look at next.

## List Item Styling[#list-item-styling](http://learn.shayhowe.com/html-css/creating-lists/#list-item-styling)

Unordered and ordered lists use list item markers by default. For unordered lists these are typically solid dots, while ordered lists typically use numbers. [With CSS](http://www.smashingmagazine.com/2009/12/11/styling-html-lists-with-css-techniques-and-resources/) the style and position of these list item markers may be adjusted.

### List Style Type Property

The list-style-type property is used to set the content of a list item marker. The [available values](https://developer.mozilla.org/en-US/docs/Web/CSS/list-style-type) range from squares and decimal numbers all the way to Armenian numbering, and the style may be placed on either the <ul>, <ol>, or <li> elements within CSS.

Any list-style-type property value can be added to either unordered or ordered lists. With this in mind, it is possible to use a numeric list item marker on an unordered list and a nonnumeric marker on an ordered list.

###### HTML

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | <ul>  <li>Orange</li>  <li>Green</li>  <li>Blue</li>  </ul> |

###### CSS

|  |  |
| --- | --- |
| 1  2  3  4 | ul {  list-style-type: square;  } |

#### List Style Type Property Demo

#### List Style Type Values

As previously mentioned, the list-style-type property comes with a handful of different values. The following list outlines these values as well as their corresponding content.

| **List Style Type Value** | **Content** |
| --- | --- |
| none | No list item |
| disc | A filled circle |
| circle | A hollow circle |
| square | A filled square |
| decimal | Decimal numbers |
| decimal-leading-zero | Decimal numbers padded by initial zeros |
| lower-roman | Lowercase roman numerals |
| upper-roman | Uppercase roman numerals |
| lower-greek | Lowercase classical Greek |
| lower-alpha / lower-latin | Lowercase ASCII letters |
| upper-alpha / upper-latin | Uppercase ASCII letters |
| armenian | Traditional Armenian numbering |
| georgian | Traditional Georgian numbering |

### Using an Image as a List Item Marker

There may come a time when the default list-style-type property values are not enough, and we want to customize our own list item marker. Doing so is most commonly accomplished by placing a background image on each <li> element within a list.

The process includes removing any default list-style-type property value and adding a background image and padding to the <li> element.

In detail, the list-style-type property value of none will remove existing list item markers. The background property will identify a background image, along with its position and repeat value, if necessary. And the padding property will provide space to the left of the text for the background image.

###### HTML

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | <ul>  <li>Orange</li>  <li>Green</li>  <li>Blue</li>  </ul> |

###### CSS

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | li {  background: url("arrow.png") 0 50% no-repeat;  list-style-type: none;  padding-left: 12px;  } |

#### Image List Item Marker Demo

### List Style Position Property

By default the list item marker is to the left of the content within the <li> element. This list style positioning is described as outside, meaning all of the content will appear directly to the right, outside of the list item marker. Using the list-style-position property, we can change the default value of outside to inside or inherit.

The outside property value places the list item marker to the left of the <li> element and doesn’t allow any content to wrap below the list item marker. The inside property value (which is rarely seen or used) places the list item marker in line with the first line of the <li> element and allows other content to wrap below it as needed.

###### HTML

|  |  |
| --- | --- |
| 1  2  3  4  5 | <ul>  <li>Cupcakes...</li>  <li>Sprinkles...</li>  </ul> |

###### CSS

|  |  |
| --- | --- |
| 1  2  3  4 | ul {  list-style-position: inside;  } |

#### List Style Position Property Demo

### Shorthand List Style Property

The list style properties discussed thus far, list-style-type and list-style-position, can be combined into one shorthand list-style property value. When using the list-style property, we can use one or all list style property values at a time. The order of these shorthand values should be list-style-type followed by list-style-position.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | ul {  list-style: circle inside;  }  ol {  list-style: lower-roman;  } |

## Horizontally Displaying List[#horizontally-displaying-list](http://learn.shayhowe.com/html-css/creating-lists/#horizontally-displaying-list)

Occasionally we may want to display lists horizontally rather than vertically. Perhaps we want to divide a list into multiple columns, to build a navigational list, or to put a few list items in a single row. Depending on the content and desired appearance, there are a few different ways to display lists as a single line, such as by making the display property value of <li> elements inline or inline-block or by floating them.

### Displaying List

The quickest way to display a list on a single line is to give the <li> elements a display property value of inline or inline-block. Doing so places all the <li> elements within a single line, with a single space between each list item.

If the spaces between each of the <li> elements are troublesome, they may be removed using the same techniques we discussed in Lesson 5, “[Positioning Content](http://learn.shayhowe.com/html-css/positioning-content/).”

More often than not, we’ll use the inline-block property value rather than the inline property value. The inline-block property value allows us to easily add vertical margins and other spacing to the <li> elements, whereas the inline property value does not.

When changing the display property value to inline or inline-block, the list item marker, be it a bullet, number, or other style, is removed.

###### HTML

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | <ul>  <li>Orange</li>  <li>Green</li>  <li>Blue</li>  </ul> |

###### CSS

|  |  |
| --- | --- |
| 1  2  3  4  5 | li {  display: inline-block;  margin: 0 10px;  } |

#### Inline-Block Displaying List Demo

### Floating List

￼Changing the display property value to inline or inline-block is quick; however, it removes the list item marker. If the list item marker is needed, floating each <li> element is a better option than changing the display property.

Setting all <li> elements’ float property to left will horizontally align all <li> elements directly next to each other without any space between them. When we float each <li> element, the list item marker is displayed by default and will actually sit on top of the <li> element next to it. To prevent the list item marker from being displayed on top of other <li> elements, a horizontal margin or padding should be added.

###### HTML

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | <ul>  <li>Orange</li>  <li>Green</li>  <li>Blue</li>  </ul> |

###### CSS

|  |  |
| --- | --- |
| 1  2  3  4  5 | li {  float: left;  margin: 0 20px;  } |

#### Floating List Demo

As when floating any element, this breaks the flow of the page. We must remember to clear our floats—most commonly with the clearfix technique—and return the page back to its normal flow.

### Navigational List Example

We’ll often develop, and find, navigation menus using unordered lists. These lists are commonly laid out as horizontal lists, using either of the two techniques previously mentioned. Here is an example of a horizontal navigation menu marked up using an unordered list with <li> elements displayed as inline-block elements.

###### HTML

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | <nav class="navigation">  <ul>  <li><a href="#">Profile</a></li><!--  --><li><a href="#">Settings</a></li><!--  --><li><a href="#">Notifications</a></li><!--  --><li><a href="#">Logout</a></li>  </ul>  </nav> |

###### CSS

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29 | .navigation ul {  font: bold 11px "Helvetica Neue", Helvetica, Arial, sans-serif;  margin: 0;  padding: 0;  text-transform: uppercase;  }  .navigation li {  display: inline-block;  }  .navigation a {  background: #395870;  background: linear-gradient(#49708f, #293f50);  border-right: 1px solid rgba(0, 0, 0, .3);  color: #fff;  padding: 12px 20px;  text-decoration: none;  }  .navigation a:hover {  background: #314b60;  box-shadow: inset 0 0 10px 1px rgba(0, 0, 0, .3);  }  .navigation li:first-child a {  border-radius: 4px 0 0 4px;  }  .navigation li:last-child a {  border-right: 0;  border-radius: 0 4px 4px 0;  } |

#### Navigational List Demo

## In Practice[#practice-1](http://learn.shayhowe.com/html-css/creating-lists/#practice-1)

Now that we know how to build lists within HTML and CSS, let’s loop back to our Styles Conference website and see where we might be able to use lists.

1. Currently the navigation menus within the <header> and <footer> elements on our pages consist of a handful of anchor elements. These anchor elements could be better organized in an unordered list.

Using an unordered list (via the <ul> element) and list items (via the <li> element) will give structure to our navigation menus. These new elements, however, will display our navigation menus vertically.

We’re going to want to change the display value of our <li> elements to inline-block to get all of them to align in a horizontal row. When we do that, though, we’ll also need to account for the blank space left between each <li> element. Thinking back to Lesson 5, “[Positioning Content](http://learn.shayhowe.com/html-css/positioning-content/),” we know that opening an HTML comment at the end of a <li> element and closing an HTML comment at the beginning of a <li> element will remove this space.

Keeping this in mind, the markup for the navigation menu within our <header> element will now look like this:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | <nav class="nav primary-nav">  <ul>  <li><a href="index.html">Home</a></li><!--  --><li><a href="speakers.html">Speakers</a></li><!--  --><li><a href="schedule.html">Schedule</a></li><!--  --><li><a href="venue.html">Venue</a></li><!--  --><li><a href="register.html">Register</a></li>  </ul>  </nav> |

Along these same lines, the markup for the navigation menu within our <footer> element will now look like this:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | <nav class="nav">  <ul>  <li><a href="index.html">Home</a></li><!--  --><li><a href="speakers.html">Speakers</a></li><!--  --><li><a href="schedule.html">Schedule</a></li><!--  --><li><a href="venue.html">Venue</a></li><!--  --><li><a href="register.html">Register</a></li>  </ul>  </nav> |

Let’s not forget to make these changes in all of our HTML files.

1. With the unordered list in place, let’s make sure the list items align horizontally, and let’s clean up their styles a bit. We’ll use the existing nav class to help target our new styles.

We’ll begin by setting all of the <li> elements within any element with the class attribute value of nav to be displayed inline-block, to include some horizontal margins, and to be vertically aligned to the top of the element.

Additionally, we’ll use the :last-child pseudo-class selector to identify the last <li> element and reset its right margin to 0. Doing so ensures that any horizontal space between the <li> element and the edge of its parent element is removed.

Within our main.css file, below our existing navigation styles, let’s add the following CSS:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | .nav li {  display: inline-block;  margin: 0 10px;  vertical-align: top;  }  .nav li:last-child {  margin-right: 0;  } |

You may be wondering why our unordered list didn’t include any list item markers or default styles. These styles were removed by the reset at the top of our style sheet. If we look at the reset, we’ll see our <ol>, <ul>, and <li> elements all include a margin and padding of 0, and our <ol> and <ul> elements have a list-style value of none.

1. Our navigation menus aren’t the only places we’ll be using lists. We’ll also use them on some of our internal pages, including the Speakers page. Let’s add some speakers to our conference.

Within our speakers.html file just below our lead section, let’s create a new section where we’ll present all of our speakers. Reusing some existing styles, we’ll use a <section> element with a class attribute value of row to wrap all of our speakers and apply a white background and padding behind them. Inside the <section> element, we’ll add a <div> element with a class attribute value of grid to center our speakers on the page and allow us to use multiple columns in doing so.

So far our HTML below the lead section looks like this:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | <section class="row">  <div class="grid">  </div>  </section> |

1. Inside the grid every speaker will be marked up with his or her own <section> element, which will include two columns. The first column will span two-thirds of the <section> element and will be marked up using a <div> element. The second column will span the remaining one-third of the <section> element and will be marked up using an <aside> element, as its content is secondary to the speaker and his or her specific talk.

Using our existing col-2-3 and col-1-3 classes, the outline for a speaker section will look like this:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | <section id="shay-howe">  <div class="col-2-3">  ...  </div><!--  --><aside class="col-1-3">  ...  </aside>  </section> |

There are a few items to notice here. First, each <section> element for each speaker includes an ID attribute with the speaker’s name as the attribute value. Later, when we create the schedule for our conference, these ID attributes will serve as anchors, allowing us to link from the schedule to a speaker’s profile.

Additionally, the closing tag of the <div> element is followed by the opening of an HTML comment, and the opening tag of the <aside> element is preceded by the closing of an HTML comment. Because the column-based classes will display these elements as inline-block elements, we are removing the blank space that will appear between them.

1. Inside the two-thirds column, marked up with the <div> element, we’ll use a few headings and paragraphs to show the speaker’s name, the title and abstract of the talk, and a short biography.

Including this content, a speaker section will look like this:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | <section id="shay-howe">  <div class="col-2-3">  <h2>Shay Howe</h2>  <h5>Less Is More: How Constraints Cultivate Growth</h5>    <p>By setting constraints, we force ourselves...</p>    <h5>About Shay</h5>  <p>As a designer and front-end developer, Shay...</p>    </div><!--  --><aside class="col-1-3">  ...  </aside>  </section> |

1. Within the one-third column, marked up with an <aside> element, we’re going to add a <div> element with a class attribute value of speaker-info. We’ll use a <div> element because we’ll be adding styles to this element soon.

Before getting into any styles, though, let’s add an unordered list within the <div> element that includes as list items some relevant links for the speaker.

Now our HTML for a speaker will look like this:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26 | <section id="shay-howe">  <div class="col-2-3">  <h2>Shay Howe</h2>  <h5>Less Is More: How Constraints Cultivate Growth</h5>  <p>By setting constraints, we force ourselves...</p>  <h5>About Shay</h5>  <p>As a designer and front-end developer, Shay...</p>  </div><!--  --><aside class="col-1-3">  <div class="speaker-info">  <ul>  <li><a href="https://twitter.com/shayhowe">@shayhowe</a></li>  <li><a href="http://learn.shayhowe.com/">learn.shayhowe.com</a></li>  </ul>  </div>  </aside>  </section> |

1. With the <div> element with a class attribute value of speaker-info ready, we can add some styles to it.

We’ll begin by adding a new section within our main.css file for the Speaker page styles. From there, let’s add a 1-pixel solid gray border with a 5-pixel radius around any element that includes the class attribute value of speaker-info.

Next, let’s add a top margin of 88 pixels to position the element on the same vertical line as the first paragraph of the talk description, and let’s also add 22 pixels of vertical padding inside the element to provide room for the nested unordered list.

Lastly, let’s center all of the text within the element.

In all, our CSS for the speaker-info class rule set looks like this:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | /\*  ========================================  Speakers  ========================================  \*/    .speaker-info {  border: 1px solid #dfe2e5;  border-radius: 5px;  margin-top: 88px;  padding: 22px 0;  text-align: center;  } |

Let’s take a minute to review why we’re using a <div> element here and the corresponding styles.

We’re placing a <div> element inside the <aside> element with the class attribute value of col-1-3 because we’ll want the padding inherited from the col-1-3 class to be outside of the border on the <div> element. Before long we’ll be including an image within the <div> element, alongside the unordered list; therefore we created a <div> element as opposed to applying these styles directly to the <ul> element.

1. As we add more and more speakers to the page, we’ll want to ensure that they remain an equal distance apart vertically. To do so, we’ll create a speaker class rule set which includes a bottom margin of 44 pixels, like this:

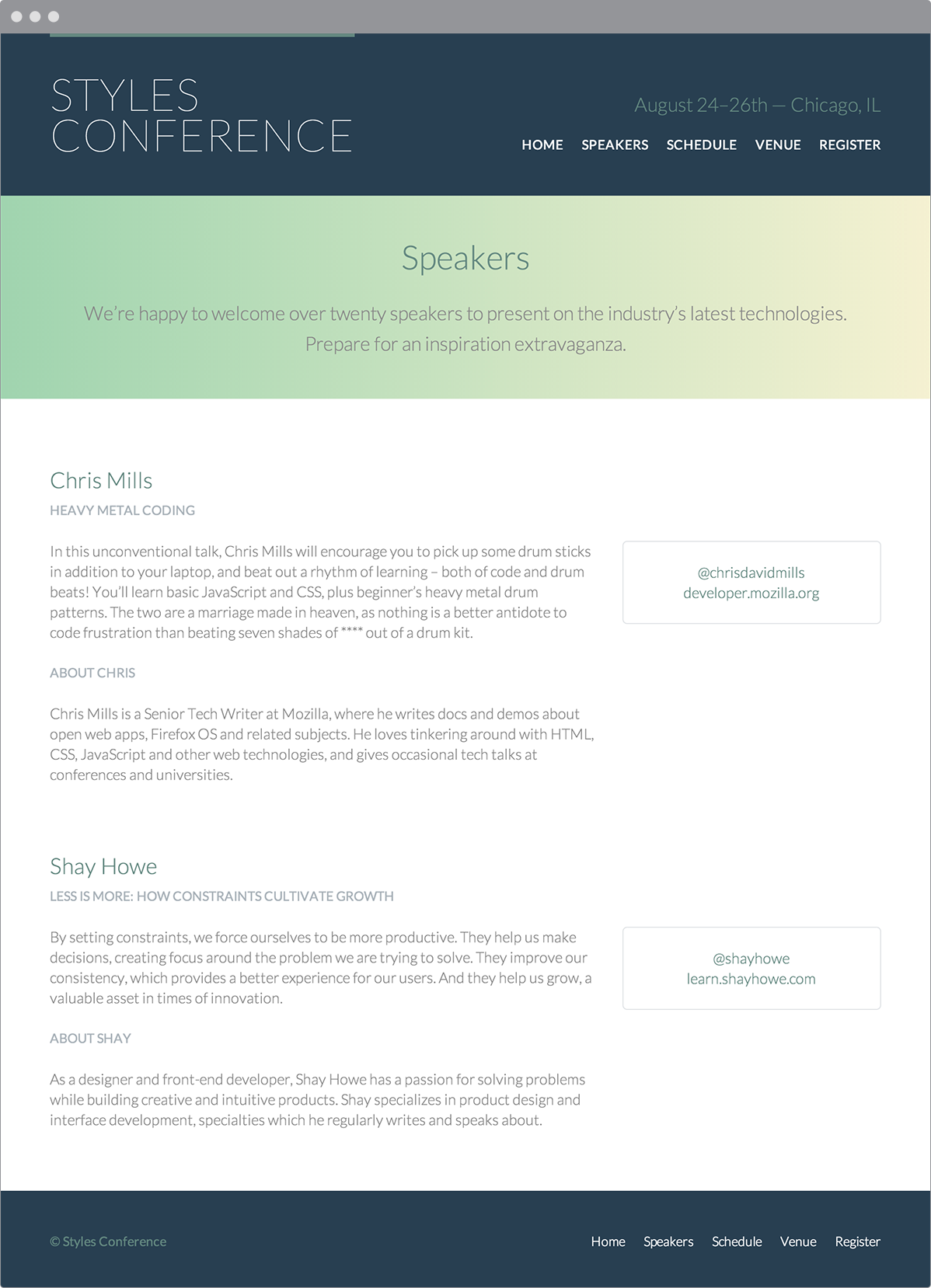
|  |  |
| --- | --- |
| 1  2  3  4 | .speaker {  margin-bottom: 44px;  } |

1. We can then apply this class to the <section> element for each speaker, provided it isn’t the last speaker. We’ll omit this class on the last speaker, as we don’t want to create any unnecessary margins before our <footer> element. With more than one speaker, our layout will look like this:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30 | <section class="row">  <div class="grid">  <section class="speaker" id="chris-mills">  <div class="col-2-3">  ...  </div><!--  --><aside class="col-1-3">  ...  </aside>  </section>    <section id="shay-howe">    <div class="col-2-3">  ...  </div><!--    --><aside class="col-1-3">  ...  </aside>    </section>    </div>  </section> |

1. Notice how the first speaker <section> element, for Chris Mills, includes the class attribute value of speaker, which vertically separates it from the speaker <section> element for myself, Shay Howe. The last speaker <section> element, again for myself, doesn’t include a class attribute value of speaker in order to keep it a proper distance from the <footer> element.

Our navigation menus are now complete, and the Speakers page is taking shape.

**Fig 8**

Our Speakers page after updating our navigation menus and adding speakers

### Demo & Source Code

Below you may view the Styles Conference website in its current state, as well as download the source code for the website in its current state.

[**View the Styles Conference Website**](http://learn.shayhowe.com/practice/creating-lists/index.html) or [Download the Source Code](http://learn.shayhowe.com/practice/creating-lists.zip) (Zip file)

## Summary[#summary](http://learn.shayhowe.com/html-css/creating-lists/#summary)

[Lists](http://alistapart.com/article/taminglists) are used quite commonly in HTML, often in places that might not be obvious or apparent. The key is to use them as semantically as possible and to leverage them where they best fit.

Let’s recap. Within this lesson we covered the following:

* How to create unordered, ordered, and description lists
* How to properly nest lists inside of other lists
* How to change the list item marker style and position
* How to use a background image instead of a list item marker
* How to horizontally display or float lists

Now that we know how to add lists to our pages, let’s add media to our pages, too. In the next chapter we’ll dive into embeddable media such as images, audio, and video.

###### Lesson 9

# Adding Media

We browse the Internet in search of interesting and informative content, which we usually find in the form of plain text. To accompany this plain text, HTML provides ways to embed rich media in the form of images, audio tracks, and videos, as well as to embed content from another web page in the form of an inline frame.

The ability to include images, audio tracks, videos, and inline frames within websites has been around for some time. Browser support for images and inline frames has generally been pretty good. And while the ability to add audio tracks and videos to a website has been around for years, the process has been fairly cumbersome. Fortunately, this process has improved and is much easier with support directly from HTML.

Today, we can freely use images, audio, video, and inline frames knowing that this content is supported across all major browsers.

## Adding Images[#adding-images](http://learn.shayhowe.com/html-css/adding-media/#adding-images)

To add images to a page, we use the <img> inline element. The <img> element is a self-containing, or empty, element, which means that it doesn’t wrap any other content and it exists as a single tag. For the <img> element to work, a src attribute and value must be included to specify the source of the image. The src attribute value is a URL, typically relative to the server where a website is hosted.

In conjunction with the src attribute, the alt (alternative text) attribute, which describes the contents of an image, should be applied. The alt attribute value is picked up by search engines and assistive technologies to help convey the purpose of an image. The alt text will be displayed in place of the image if for some reason the image is not available.

|  |  |
| --- | --- |
| 1  2 | <img src="dog.jpg" alt="A black, brown, and white dog wearing a kerchief"> |

#### Adding Images Demo

See the Pen [Adding Images](http://codepen.io/shayhowe/pen/iBrbe/) by Shay Howe ([@shayhowe](http://codepen.io/shayhowe)) on [CodePen](http://codepen.io).

Image Alternate Text**Fig 9**

The alternate text, “A black, brown, and white dog wearing a kerchief,” shown in place of a missing image

#### ￼Supported Image Formats

Images come in a variety of different file formats, and each browser may support (or not support) different formats. By and large, the most commonly supported formats online are gif, jpg, and png images. Of these, the most widely used formats today are jpg and png. The jpg format provides quality images with high color counts while maintaining a decent file size, ideal for faster load times. The png format is great for images with transparencies or low color counts. We most commonly see jpg images used for photographs and png images used for icons or background patterns.

### Sizing Images

It is important to identify the size of an image in order to tell the browser how large the image should be before the page even loads; thus the browser can reserve space for the image and render the page faster. There are a few different ways to size images so that they work well on a page. One option is to use the width and height attributes directly within the <img> tag in HTML.

Additionally, images may be sized using the width and height properties in CSS. When both the HTML attributes and CSS properties are used, the CSS attributes will take precedence over the HTML attributes.

Specifying either a width or height will cause the other dimension to adjust automatically to maintain the aspect ratio of the image. As an example, if we want an image to be 200 pixels tall but are less specifically concerned about how wide it is, we can set the height to 200 pixels, and the width of the image will adjust accordingly. Setting both a width and height will work also; however, doing so may break the aspect ratio of an image, causing it to appear distorted.

|  |  |
| --- | --- |
| 1  2  3  4  5 | img {  height: 200px;  width: 200px;  } |

#### Sizing Images Demo

See the Pen [Sizing Images](http://codepen.io/shayhowe/pen/kEdwi/) by Shay Howe ([@shayhowe](http://codepen.io/shayhowe)) on [CodePen](http://codepen.io).

While using the width and height attributes directly in HTML provides some semantic value by noting an image’s original size, it can be difficult to manage numerous images that all need to be the same size. In this event, it’s common practice to use CSS to resize the images.

### Positioning Images

We can use a number of different approaches to position images on a web page. By default images are positioned as inline-level elements; however, their positions may be changed using CSS, specifically the float, display, and box model properties, including padding, border, and margin.

#### Inline Positioning Images

The <img> element is by default an inline-level element. Adding an image without any styles to a page will position that image within the same line as the content that surrounds it. Additionally, the height of the line in which an image appears will be changed to match the height of the image, which can create large vertical gaps within that line.

|  |  |
| --- | --- |
| 1  2 | <p>Gatsby is a black, brown, and white hound mix puppy who loves howling at fire trucks and collecting belly rubs. <img src="dog.jpg" alt="A black, brown, and white dog wearing a kerchief"> Although he spends most of his time sleeping he is also quick to chase any birds who enter his vision.</p> |

#### Inline Images Demo

See the Pen [Inline Images](http://codepen.io/shayhowe/pen/BduLm/) by Shay Howe ([@shayhowe](http://codepen.io/shayhowe)) on [CodePen](http://codepen.io).

Leaving images untouched in their default positioning isn’t too common. More often than not, images are displayed as block-level elements or are floated flush to one side.

#### Block Positioning Images

Adding the display property to an image and setting its value to block forces the image to be a block-level element. This makes the image appear on its own line, allowing the surrounding content to be positioned above and below the image.

|  |  |
| --- | --- |
| 1  2  3  4 | img {  display: block;  } |

#### Block Images Demo

See the Pen [Block Images](http://codepen.io/shayhowe/pen/zLKIE/) by Shay Howe ([@shayhowe](http://codepen.io/shayhowe)) on [CodePen](http://codepen.io).

#### Positioning Images Flush Left or Right

Sometimes displaying an image as inline or block, or perhaps even inline-block, isn’t ideal. We may want the image to appear on the left or right side of its containing element, while all of the other content wraps around the image as necessary. To do this, we use the float property with a value of either left or right.

Remembering back to Lesson 5, “[Positioning Content](http://learn.shayhowe.com/html-css/positioning-content/),” we recall that the float property was originally intended to position images to the left or right of a containing element. Now we’ll use it for that original purpose.

Floating an image is a start; however, all other content will align directly against it. To provide spacing around an image, we’ll use the margin property. Additionally, we can use the padding, border, and background properties to build a frame for the image, if desired.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | img {  background: #eaeaed;  border: 1px solid #9799a7;  float: right;  margin: 8px 0 0 20px;  padding: 4px;  } |

#### Floating Images Demo

See the Pen [Floating Images](http://codepen.io/shayhowe/pen/vjdIe/) by Shay Howe ([@shayhowe](http://codepen.io/shayhowe)) on [CodePen](http://codepen.io).

#### When to Use an Image Element vs. a Background Image

There are two primary ways to add images to a web page. One way, as covered here, is to use the <img> element within HTML. Another way is to use the background or background-image property within CSS to assign a background image to an element. Either option will do the job; however, they each have specific use cases.

The <img> element within HTML is the preferred option when the image being used holds semantic value and its content is relevant to the content of the page.

The background or background-image property within CSS is the preferred option when the image being used is part of the design or user interface of the page. As such, it’s not directly relevant to the content of the page.

The <img> element is quite popular, and when it was originally added to the HTML specification it forever changed the way websites were built.

## In Practice[#practice-1](http://learn.shayhowe.com/html-css/adding-media/#practice-1)

Now that we know how to add and position images on a page, let’s take a look at our Styles Conference website and see where we can add a few images.

1. Let’s begin by adding some images to our home page. Specifically, we’ll add an image within each of the teaser sections promoting a few of our pages.

Before we jump into the code, though, let’s create a new folder named “images” within our “assets” folder. Then, within the “images” folder, let’s create another folder named “home” specifically for our home page images. Within the “home” folder we’ll add three images: speakers.jpg, schedule.jpg, and venue.jpg. (For reference, these images may be [downloaded](http://learn.shayhowe.com/practice/images.zip) within a zip file.)

Then, inside our index.html file, each teaser section has an <a> element wrapping both an <h3> and an <h5> element. Let’s move the <h5> element above the <a> element and replace it with an <img> element. The src attribute value for each <img> element will correspond to the folder structure and filename we set up, and the alt attribute value will describe the contents of each image.

The HTML for our first teaser, for the Speakers page, will look like this:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | <section class="teaser col-1-3">  <h5>Speakers</h5>  <a href="speakers.html">  <img src="assets/images/home/speakers.jpg" alt="Professional Speaker">  <h3>World-Class Speakers</h3>  </a>  <p>Joining us from all around the world are over twenty fantastic speakers, here to share their stories.</p>  </section> |

Let’s continue this pattern for both the Schedule and Venue page teasers, too.

1. Now that we’ve added a few images to our home page, we’ll need to clean up their styles a bit and make sure they properly fit into the layout of our page.

Since images are inline-level elements by default, let’s change our images within the teaser sections to block-level elements. Let’s also set their maximum width to 100% to ensure they don’t exceed the width of their respective columns. Changing this width value is important as it allows our images to adjust with the width of the columns as necessary.

Lastly, let’s round the corners of the images slightly and apply 22 pixels of bottom margin to the images, providing a little breathing room.

Once we add these new styles to our existing home page styles (using the teaser class as a qualifying selector for the <img> elements), our CSS will look like this:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | .teaser img {  border-radius: 5px;  display: block;  margin-bottom: 22px;  max-width: 100%  } |

1. Next up, let’s add images of all of the speakers to the Speakers page. We’ll begin by creating a “speakers” folder within our “images” folder and placing images of all of the speakers there.

Within the speakers.html file, let’s add an <img> element within each of the speaker information <aside> elements. Let’s place each <img> element inside the <div> element with the class attribute value of speaker-info, just above the <ul> element.

The src attribute value of each image will correspond to the “speakers” folder we set up and the speaker’s name; the alt attribute value will be the speaker’s name.

The <aside> element for myself, as a speaker, will look like this:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | <aside class="col-1-3">  <div class="speaker-info">  <img src="assets/images/speakers/shay-howe.jpg" alt="Shay Howe">    <ul>  <li><a href="https://twitter.com/shayhowe">@shayhowe</a></li>  <li><a href="http://learn.shayhowe.com/">learn.shayhowe.com</a></li>  </ul>  </div>  </aside> |

This same pattern for adding an image should then be applied to all other speakers.

1. As we did with the images on our home page, we’ll want to apply some styles to the images on the Speakers page.

Let’s begin by applying the border-radius property with a value of 50%, turning our images into circles. From there, let’s set a fixed height of 130 pixels to each image and set them to be vertically aligned to the top of the line they reside within.

With the height and vertical alignment in place, let’s apply vertical margins to the images. Using a negative 66-pixel margin on the top of the images, we’ll pull them slightly out of the <aside> element and make them vertically centered with the top border of the <div> element with a class attribute value of speaker-info. Then, applying a 22-pixel margin on the bottom of the image provides space between the image and the <ul> element below it.

When we add these new styles to our existing Speakers page styles (using the speaker-info class as a qualifying selector for the <img> elements), our CSS will look like this:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | .speaker-info img {  border-radius: 50%;  height: 130px;  margin: -66px 0 22px 0;  vertical-align: top;  } |

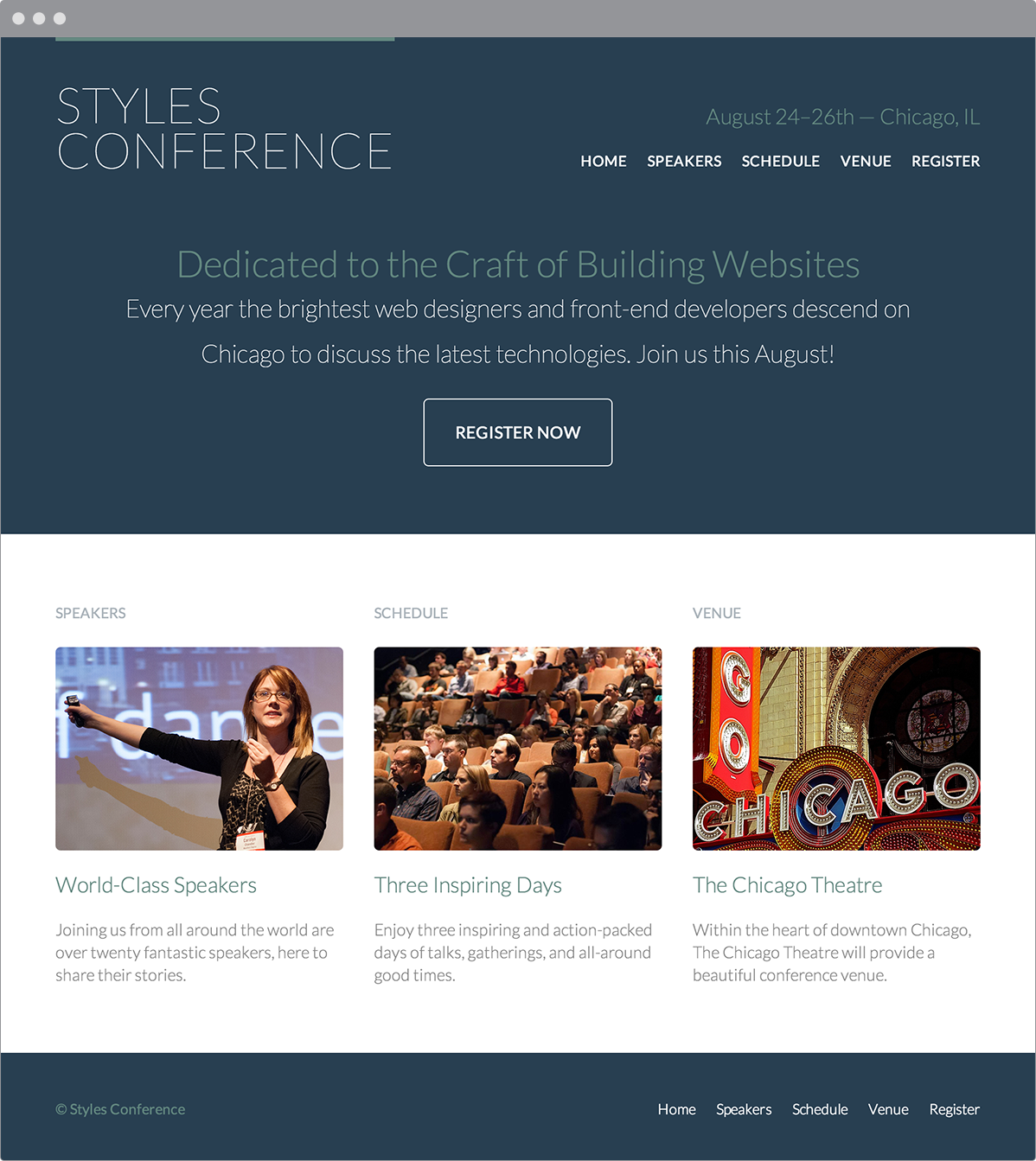
1. Since we are using an aggressive negative margin on the <img> element within the <div> element with a class attribute value of speaker-info, we need to remove the padding on the top of that <div> element.

Previously we were using the padding property with a value of 22px 0, thus placing 22 pixels of padding on the top and bottom and 0 pixels of padding on the left and right of the <div> element. Let’s swap this property and value out for the padding-bottom property, as that’s the only padding we need to identify, and use a value of 22 pixels.

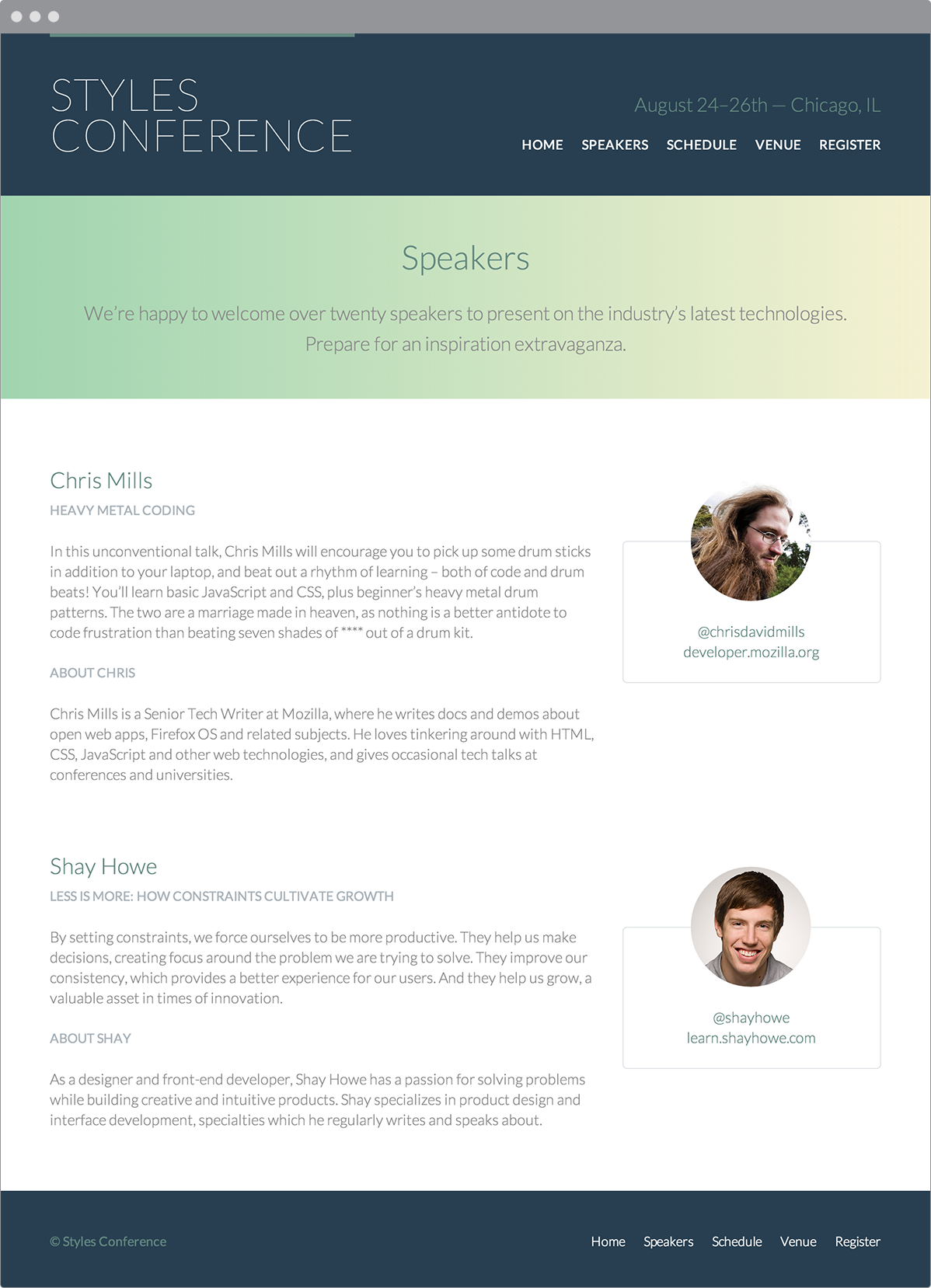
The new speaker-info class rule set looks like this:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | .speaker-info {  border: 1px solid #dfe2e5;  border-radius: 5px;  margin-top: 88px;  padding-bottom: 22px;  text-align: center;  } |

Now both our home and Speaker pages are looking pretty sharp.

**Fig 9**

Our Styles Conference home page after adding images to each section that teases another page

**Fig 9**

Our Styles Conference Speakers page after adding images for each of the speakers

## Adding Audio[#adding-audio](http://learn.shayhowe.com/html-css/adding-media/#adding-audio)

HTML5 provides a quick and easy way to add [audio files](https://developer.mozilla.org/en-US/docs/Web/Guide/HTML/Using_HTML5_audio_and_video) to a website by way of the <audio> [element](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/audio). As with the <img> element, the <audio> element accepts a source URL specified by the src attribute. Unlike the <img> element, though, the <audio> element requires both opening and closing tags, which we’ll discuss soon.

|  |  |
| --- | --- |
| 1  2 | <audio src="jazz.ogg"></audio> |

### Audio Attributes

Several other attributes may accompany the src attribute on the <audio> element; the most popular include autoplay, controls, loop, and preload.

The autoplay, controls, and loop attributes are all Boolean attributes. As Boolean attributes, they don’t require a stated value. Instead, when each is present on the <audio> element its value will be set to true, and the <audio> element will behave accordingly.

By default, the <audio> element isn’t displayed on a page. If the autoplay Boolean attribute is present on the <audio> element, nothing will appear on the page, but the audio file will automatically play upon loading.

|  |  |
| --- | --- |
| 1  2 | <audio src="jazz.ogg" autoplay></audio> |

To display the <audio> element on a page, the controls Boolean attribute is necessary. When it’s applied to the <audio> element, the controls Boolean attribute will display a browser’s default audio controls, including play and pause, seek, and volume controls.

|  |  |
| --- | --- |
| 1  2 | <audio src="jazz.ogg" controls></audio> |

#### Adding Audio Demo

See the Pen [Adding Audio](http://codepen.io/shayhowe/pen/pLriu/) by Shay Howe ([@shayhowe](http://codepen.io/shayhowe)) on [CodePen](http://codepen.io).

When present on the <audio> element, the loop Boolean attribute will cause an audio file to repeat continually, from beginning to end.

Lastly, the preload attribute for the <audio> element helps identify what, if any, information about the audio file should be loaded before the clip is played. It accepts three values: none, auto, and metadata. The none value won’t preload any information about an audio file, while the auto value will preload all information about an audio file. The metadata value sits in between the none and auto values, as it will preload any available metadata information about an audio file, such as the clip’s length, but not all information.

When the preload attribute isn’t present on the <audio> element, all information about an audio file is loaded, as if the value was set to auto. For this reason, using the preload attribute with a value of metadata or none is a good idea when an audio file is not essential to a page. It’ll help to conserve bandwidth and allow pages to load faster.

### Audio Fallbacks & Multiple Sources

At the moment, different browsers support different audio file formats, the three most popular of which are ogg, mp3, and wav. For the best browser support we’ll need to use a handful of audio fallbacks, which will be included inside an <audio> element’s opening and closing tags.

To begin, we’ll remove the src attribute from the <audio> element. Instead, we’ll use the <source> element, with a src attribute, nested inside the <audio> element to define a new source.

Using a <source> element and src attribute for each file format, we can list one audio file format after the other. We’ll use the type attribute to quickly help the browser identify which audio types are available. When a browser recognizes an audio file format it will load that file and ignore all the others.

Because it was introduced in HTML5, some browsers may not support the <audio> element. In this case, we can provide a link to download the audio file after any <source> elements within the <audio> element.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | <audio controls>  <source src="jazz.ogg" type="audio/ogg">  <source src="jazz.mp3" type="audio/mpeg">  <source src="jazz.wav" type="audio/wav">  Please <a href="jazz.mp3" download>download</a> the audio file.  </audio> |

To review the previous code, the <audio> element includes the controls Boolean attribute to ensure the audio player is displayed within browsers that support the element. The <audio> element does not include a src attribute and instead wraps three different <source> elements. Each <source> element includes a src attribute that references a different audio file format and a type attribute to identify the format of the audio file. As a last fallback, if a browser doesn’t recognize any of the audio file formats, the anchor link to download the element will be displayed.

In addition to the <audio> element, HTML5 also introduced the <video> element, which shares quite a few similarities with the <audio> element.

## Adding Video[#adding-video](http://learn.shayhowe.com/html-css/adding-media/#adding-video)

Adding [video in HTML5](http://dev.opera.com/articles/introduction-html5-video/) is very similar to adding audio. We use the <video> element in place of the <audio> element. All of the same attributes (src, autoplay, controls, loop, and preload) and fallbacks apply here, too.

With the <audio> element, if the controls Boolean attribute isn’t specified the audio clip isn’t displayed. With videos, if the controls Boolean attribute is not specified the video will display. However, it is fairly difficult to view unless the autoplay Boolean attribute is also applied. In general, the best practice here is to include the controls Boolean attribute unless there is a good reason not to allow users to start, stop, or replay the video.

Since videos take up space on the page, it doesn’t hurt to specify their dimensions, which is most commonly done with width and height properties in CSS. This helps ensure that the video isn’t too large and stays within the implied layout of a page. Additionally, specifying a size, as with images, helps the browser render videos faster and allows it to allocate the proper space needed for the video to be displayed.

|  |  |
| --- | --- |
| 1  2 | <video src="earth.ogv" controls></video> |

#### Adding Video Demo

For performance reasons this video demo is hosted locally, however you may still [review and edit this code](http://codepen.io/shayhowe/pen/pKHrg) on CodePen.

Please [download](http://html-css-guide.herokuapp.com/assets/courses/html-css-guide/images-audio-video/earth.mp4) the video.

#### Customizing Audio & Video Controls

By default, the <audio> and <video> element controls are determined by each browser independently. Depending on the design of a website, more authority over the look and feel of the media player may be needed. If this is the case, a customized player can be built, but it will require a little JavaScript to work.

Additionally, if a customized player uses the <img> element as a control the value of the alt attribute should explictly state that the image is a control and requires the proper interaction to work.

### Poster Attribute

One additional attribute available for the <video> element is the poster attribute. The poster attribute allows us to specify an image, in the form of a URL, to be shown before a video is played. The example below uses a screen capture from the video as the poster for the Earth video.

|  |  |
| --- | --- |
| 1  2 | <video src="earth.ogv" controls poster="earth-video-screenshot.jpg"></video> |

#### Poster Attribute Demo

For performance reasons this video demo is hosted locally, however you may still [review and edit this code](http://codepen.io/shayhowe/pen/xmcnr/) on CodePen.

Please [download](http://html-css-guide.herokuapp.com/assets/courses/html-css-guide/images-audio-video/earth.mp4) the video.

### Video Fallbacks

As with the <audio> element, video fallbacks are also necessary. The same markup format, with multiple <source> elements for each file type and a plain text fallback, also applies within the <video> element.

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | <video controls>  <source src="earth.ogv" type="video/ogg">  <source src="earth.mp4" type="video/mp4">  Please <a href="earth.mp4" download>download</a> the video.  </video> |

One additional fallback option that could be used in place of a plain text fallback is to use a [YouTube](https://www.youtube.com/) or [Vimeo](https://vimeo.com/) embedded video. These video hosting websites allow us to upload our videos, provide a standard video player, and enable us to embed our videos onto a page using an inline frame.

#### HTML5 Audio & Video File Formats

Browser support for the <audio> and <video> elements varies, as do the file formats required with these elements. Each browser has its own [preferred](https://developer.mozilla.org/en-US/docs/HTML/Supported_media_formats) audio and video file formats.

There are a few tools that help to convert an [audio](http://media.io/) or [video](http://www.mirovideoconverter.com/) file into different formats, and a quick search will provide an abundance of options.

## Adding Inline Frames[#adding-iframes](http://learn.shayhowe.com/html-css/adding-media/#adding-iframes)

Another way to add content to a page is to embed another HTML page within the current page. This is done using an inline frame, or <iframe> element. The <iframe> element accepts the URL of another HTML page within the src attribute value; this causes the content from the embedded HTML page to be displayed on the current page. The value of the src attribute may be a URL relative to the page the <iframe> element appears on or an absolute URL for an entirely external page.

Many pages use the <iframe> element to embed media onto a page from an external website such as Google Maps, YouTube, and others.

|  |  |
| --- | --- |
| 1  2 | <iframe src="https://www.google.com/maps/embed?..."></iframe> |

#### Adding Inline Frames Demo

For security reasons CodePen doesn’t allow iframes within embedded code samples, however you may [review and edit this code](http://codepen.io/shayhowe/pen/zounc) on their website.

The <iframe> element has a few default styles, including an inset border and a width and height. These styles can be adjusted using the frameborder, width, and height HTML attributes or by using the border, width, and height CSS properties.

### Seamless Inline Frames

Pages referenced within the src attribute of an <iframe> element play by their own rules, as they do not inherit any styles or behaviors from the page they are referenced on. Any styles applied to a page that includes an <iframe> element will not be inherited by the page referenced within the <iframe> element. Additionally, links within the page referenced within the <iframe> element will open inside that frame, leaving the page that contains the <iframe> element unchanged.

There will be times when we’ll want to change these behaviors, and the seamless Boolean attribute will allow us to do just that. When present on the <iframe> element, the seamless Boolean attribute allows styles from the page that includes an <iframe> element to be inherited by the page referenced within the <iframe> element. Additionally, the seamless Boolean attribute allows links clicked on a page referenced within an <iframe> element to be opened within the same window as the original page that includes the <iframe> element.

|  |  |
| --- | --- |
| 1  2 | <iframe src="contact.html" seamless></iframe> |

The seamless Boolean attribute is a new attribute introduced in HTML5. Although the browser support for this attribute is growing, it will not work within older browsers. It’s advisable to test the seamless Boolean attribute before using it.

## In Practice[#practice-2](http://learn.shayhowe.com/html-css/adding-media/#practice-2)

Inline frames provide a great way to add dynamic content to a page. Let’s give this a shot by updating our Venue page with some maps.

1. Before adding any maps or inline frames, let’s first prepare our Venue page for a two-column grid. Below the leading section of the page we’ll add a <section> element with the class attribute value of row to identify a new section of the page, and we’ll include some general styles, such as a white background and some vertical padding.

Directly inside this <section> element let’s add a <div> element with the class attribute value of grid. The class of grid centers our content on the page and prepares for the one-third and two-thirds columns to follow.

So far the main section of our venue.html file looks like this:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | <section class="row">  <div class="grid">  ...  </div>  </section> |

1. Within the <div> element with the class attribute value of grid we’ll have two new sections, one for the conference venue and one for the conference hotel. Let’s add two new <section> elements and give each of these <section> elements a unique class that corresponds to its content. We’ll use these classes to add margins to the bottom of each section.

Our HTML should now look like this:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | <section class="row">  <div class="grid">  <section class="venue-theatre">  ...  </section>  <section class="venue-hotel">  ...  </section>  </div>  </section> |

1. Now that we have a few classes to work with, let’s create a new section within our main.css file for Venue page styles. We’ll add a 66-pixel margin to the bottom of the <section> element with the class attribute value of venue-theatre to insert some space between it and the <section> element below it.

Then, we’ll add a 22-pixel margin to the bottom of the <section> element with the class attribute value of venue-hotel to provide some space between it and the <footer> element below it.

The new venue section within the main.css file looks like the following:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | /\*  ========================================  Venue  ========================================  \*/  .venue-theatre {  margin-bottom: 66px;  }  .venue-hotel {  margin-bottom: 22px;  } |

The <section> element with the class attribute value of venue-hotel has a smaller bottom margin than the <section> element with the class attribute value of venue-theatre because it sits next to the padding from the bottom of the <section> element with the class attribute of row. Adding that margin and padding together gives us the same value as the bottom margin on the <section> element with the class attribute value of venue-theatre.

1. Now it’s time to create the two columns within each of the new <section> elements. We’ll start by adding a <div> element with a class attribute value of col-1-3 to establish a one-third column. After it we’ll add an <iframe> element with a class attribute value of col-2-3 to establish a two-thirds column.

Keeping in mind that the column classes make both the <div> and <iframe> elements inline-block elements, we need to remove the empty space that will appear between them. To do so we’ll open an HTML comment directly after the closing <div> tag, and we’ll close the HTML comment immediately before the opening <iframe> tag.

In all, our HTML for the columns looks like this:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21 | <section class="row">  <div class="grid">  <section class="venue-theatre">    <div class="col-1-3"></div><!--  --><iframe class="col-2-3"></iframe>    </section>    <section class="venue-hotel">  <div class="col-1-3"></div><!--  --><iframe class="col-2-3"></iframe>    </section>  </div>  </section> |

1. Within each of the <div> elements with a class attribute value of col-1-3 let’s add the venue’s name within an <h2> element, followed by two <p> elements. In the first <p> element let’s include the venue’s address, and in the second <p> element let’s include the venue’s website (within an anchor link) and phone number.

Within each of the paragraphs, let’s use the line-break element, <br>, to place breaks within the address and in between the website and phone number.

For the <section> element with the class attribute value of venue-theatre, the HTML looks like this:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | <section class="venue-theatre">  <div class="col-1-3">  <h2>Chicago Theatre</h2>  <p>175 N State St <br> Chicago, IL 60601</p>  <p><a href="http://www.thechicagotheatre.com/">thechicagotheatre.com</a> <br> (312) 462-6300</p>  </div><!--  --><iframe class="col-2-3"></iframe>  </section> |

The same pattern shown here for the theatre should also be applied to the hotel (using, of course, the proper address, website, and phone number).

1. We can search for these addresses in [Google Maps](https://www.google.com/maps/). Once we locate an address and create a customized map, we have the ability to embed that map into our page. Following the instructions on Google Maps for how to share and embed a map will provide us with the HTML for an <iframe> element.

Let’s copy the HTML—<iframe> element, src attribute, and all—onto our page where our existing <iframe> element resides. We’ll do this for each location, using two different <iframe> elements.

In copying over the <iframe> element from Google Maps we need to make sure we preserve the class attribute and value, col-2-3, from our existing <iframe> element. We also need to be careful not to harm the HTML comment that closes directly before our opening <iframe> tag.

Looking directly at the <section> element with the class attribute value of venue-theatre again, the HTML looks like this:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | <section class="venue-theatre">  <div class="col-1-3">  <h2>Chicago Theatre</h2>  <p>175 N State St <br> Chicago, IL 60601</p>  <p><a href="http://www.thechicagotheatre.com/">thechicagotheatre.com</a> <br> (312) 462-6300</p>  </div><!--  --><iframe class="col-2-3" src="https://www.google.com/maps/embed?pb=!1m5!3m3!1m2!1s0x880e2ca55810a493%3A0x4700ddf60fcbfad6!2schicago+theatre!5e0!3m2!1sen!2sus!4v1388701393606"></iframe>  </section> |

1. Lastly, we’ll want to make sure that both <iframe> elements that reference Google Maps share the same height. To do this, we’ll create a new class, venue-map, and apply it to each of the <iframe> elements alongside the existing col-2-3 class attribute value.

The HTML for the <section> element with the class attribute value of venue-theatre now looks like this:

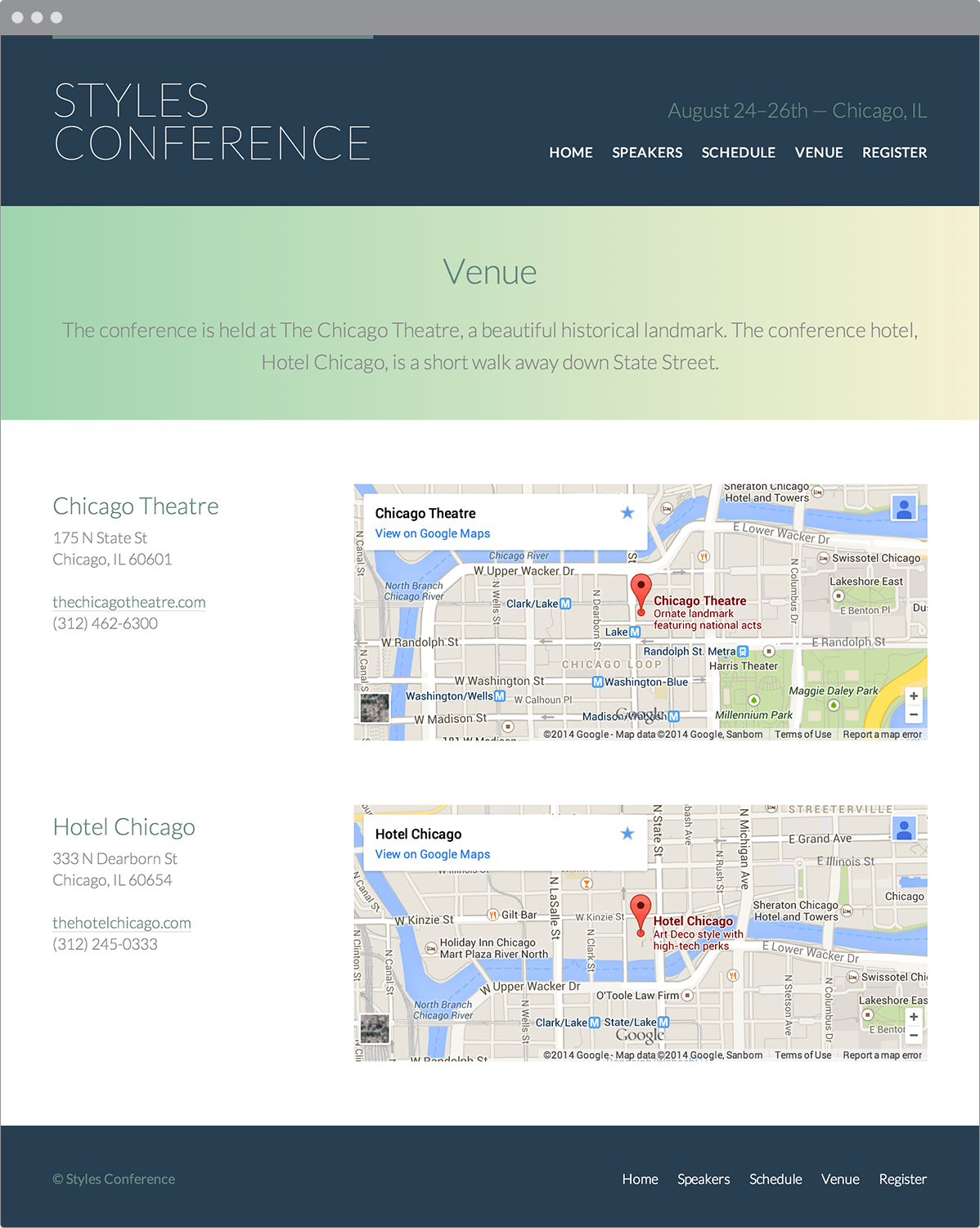
|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | <section class="venue-theatre">  <div class="col-1-3">  <h2>Chicago Theatre</h2>  <p>175 N State St <br> Chicago, IL 60601</p>  <p><a href="http://www.thechicagotheatre.com/">thechicagotheatre.com</a> <br> (312) 462-6300</p>  </div><!--  --><iframe class="venue-map col-2-3" src="https://www.google.com/maps/embed?pb=!1m5!3m3!1m2!1s0x880e2ca55810a493%3A0x4700ddf60fcbfad6!2schicago+theatre!5e0!3m2!1sen!2sus!4v1388701393606"></iframe>  </section> |

Once the venue-map class is applied to each <iframe> element, let’s create the venue-map class rule set within our main.css file. It includes the height property with a value of 264 pixels.

The venue-map class rule set looks like this:

|  |  |
| --- | --- |
| 1  2  3  4 | .venue-map {  height: 264px;  } |

We now have a Venue page, complete with maps for the different locations of our conference.

**Fig 9**

Our Styles Conference Venue page, which now includes inline frames

### Demo & Source Code

Below you may view the Styles Conference website in its current state, as well as download the source code for the website in its current state.

[**View the Styles Conference Website**](http://learn.shayhowe.com/practice/adding-media/index.html) or [Download the Source Code](http://learn.shayhowe.com/practice/adding-media.zip) (Zip file)

## Semantically Identifying Figures & Captions[#figures-and-captions](http://learn.shayhowe.com/html-css/adding-media/#figures-and-captions)

With HTML5 also came the introduction of the <figure> and <figcaption> elements. These [elements](http://html5doctor.com/the-figure-figcaption-elements/) were created to semantically mark up self-contained content or media, commonly with a caption. Before HTML5 this was frequently done using an ordered list. While an ordered list worked, the markup was not semantically correct.

### Figure

The <figure> block-level element is used to identify and wrap self-contained content, often in the form of media. It may surround images, audio clips, videos, blocks of code, diagrams, illustrations, or other self-contained media. More than one item of self-contained content, such as multiple images or videos, may be contained within the <figure> element at a time. If the <figure> element is moved from the main portion of a page to another location (for example, the bottom of the page), it should not disrupt the content or legibility of the page.

|  |  |
| --- | --- |
| 1  2  3  4 | <figure>  <img src="dog.jpg" alt="A black, brown, and white dog wearing a kerchief">  </figure> |

#### Figure Demo

See the Pen [Figure](http://codepen.io/shayhowe/pen/gtGpj/) by Shay Howe ([@shayhowe](http://codepen.io/shayhowe)) on [CodePen](http://codepen.io).

### Figure Caption

To add a caption or legend to the <figure> element, the <figcaption> element is used. The <figcaption> may appear at the top of, bottom of, or anywhere within the <figure> element; however, it may only appear once. When it’s used, the <figcaption> element will serve as the caption for all content within the <figure> element.

Additionally, the <figcaption> element may replace an <img> element’s alt attribute if the content of the <figcaption> element provides a useful description of the visual content of the image.

|  |  |
| --- | --- |
| 1  2  3  4  5 | <figure>  <img src="dog.jpg">  <figcaption>A beautiful black, brown, and white hound dog wearing kerchief.</figcaption>  </figure> |

#### Figure Caption Demo

See the Pen [Figure Caption](http://codepen.io/shayhowe/pen/KsItj/) by Shay Howe ([@shayhowe](http://codepen.io/shayhowe)) on [CodePen](http://codepen.io).

Not all forms of media need to be included within a <figure> element or include a <figcaption> element; only those that are self-contained and belong together as a group.

## Summary[#summary](http://learn.shayhowe.com/html-css/adding-media/#summary)

Alongside text, media is one of the largest parts of the web. Use of images, audio, and video has only grown over recent years, and it isn’t likely to slow down. Now we know how to incorporate these forms of media into our designs and how we can use them to enrich the content on our websites.

Within this lesson we covered the following:

* The best ways to add images, audio clips, videos, and inline frames to a page
* Different ways to position images in different situations
* How to provide audio and video fallbacks for older browsers
* Common attributes available to audio clips and videos
* The seamless attribute, which allows us to make inline frames behave as if they are part of the page they are referenced from
* The semantic way to mark up self-contained content, including media

We’re coming into the homestretch of learning HTML and CSS, with only a few more components left to introduce. Next on the list are forms.

###### Lesson 10

# Building Forms

Forms are an essential part of the Internet, as they provide a way for websites to capture information from users and to process requests, and they offer controls for nearly every imaginable use of an application. Through controls or fields, forms can request a small amount of information—often a search query or a username and password—or a large amount of information—perhaps shipping and billing information or an entire job application.

We need to know how to build forms in order to acquire user input. In this lesson we’ll discuss how to use HTML to mark up a form, which elements to use to capture different types of data, and how to style forms with CSS. We won’t get too deep into how information from a form is processed and handled on the back end of a website. Form processing is a deeper topic, outside the realm of this book; for now we’ll stick to the creation and styling of forms.

## Initializing a Form[#form](http://learn.shayhowe.com/html-css/building-forms/#form)

To add a [form](http://htmldog.com/guides/html/beginner/forms/) to a page, we’ll use the <form> element. The <form> [element](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/form) identifies where on the page control elements will appear. Additionally, the <form> element will wrap all of the elements included within the form, much like a <div> element.

|  |  |
| --- | --- |
| 1  2  3  4 | <form action="/login" method="post">  ...  </form> |

A handful of different attributes can be applied to the <form> element, the most common of which are action and method. The action attribute contains the URL to which information included within the form will be sent for processing by the server. The method attribute is the HTTP method browsers should use to submit the form data. Both of these <form> attributes pertain to submitting and processing data.

## Text Fields & Textareas[#text-and-textareas](http://learn.shayhowe.com/html-css/building-forms/#text-and-textareas)

When it comes to gathering text input from users, there are a few different elements available for obtaining data within forms. Specifically, text fields and textareas are used for collecting text- or string-based data. This data may include passages of text content, passwords, telephone numbers, and other information.

### Text Fields

One of the primary elements used to obtain text from users is the <input> element. The <input> [element](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/Input) uses the type attribute to define what type of information is to be captured within the control. The most popular type attribute value is text, which denotes a single line of text input.

Along with setting a type attribute, it is best practice to give an <input> element a name attribute as well. The name attribute value is used as the name of the control and is submitted along with the input data to the server.

|  |  |
| --- | --- |
| 1  2 | <input type="text" name="username"> |

#### Text Fields Demo

The <input> element is self-contained, meaning it uses only one tag and it does not wrap any other content. The value of the element is provided by its attributes and their corresponding values.

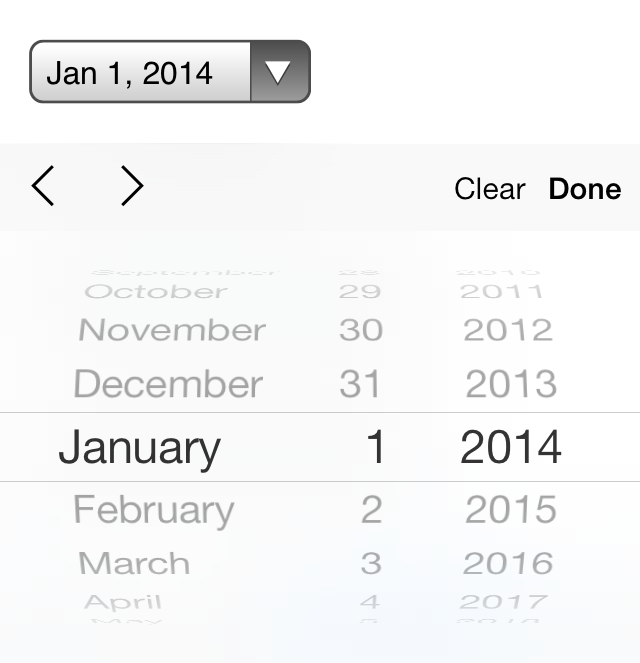
Originally, the only two text-based type attribute values were text and password (for password inputs); however, HTML5 brought along a handful of [new](http://diveinto.html5doctor.com/forms.html) type attribute values.

These values were added to provide clearer semantic meaning for inputs as well as to provide better controls for users. Should a browser not understand one of these HTML5 type attribute values, it will automatically fall back to the text attribute value. Below is a list of the new HTML5 input types.

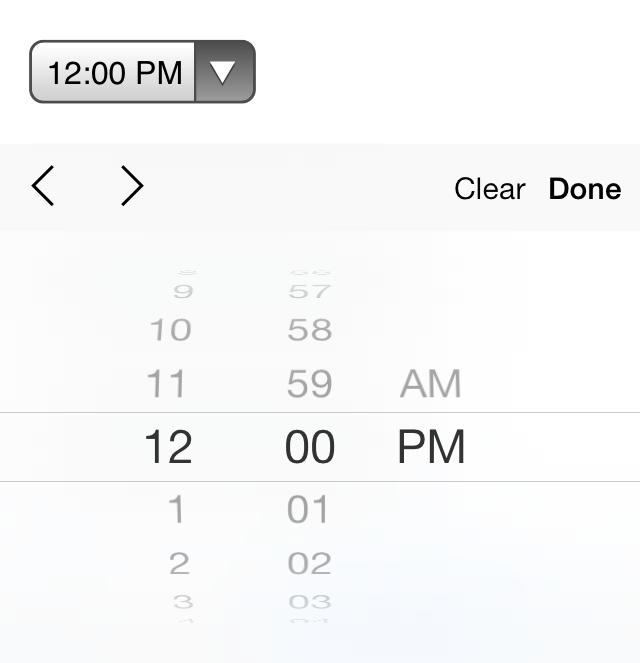
* color
* date
* datetime
* email
* month
* number
* range
* search
* tel
* time
* url
* week

The following <input> elements show a few of these HTML5 type attribute values in use; the following figures show how these unique values may look within iOS. Notice how the different values provide different controls, all of which make gathering input from users easier.

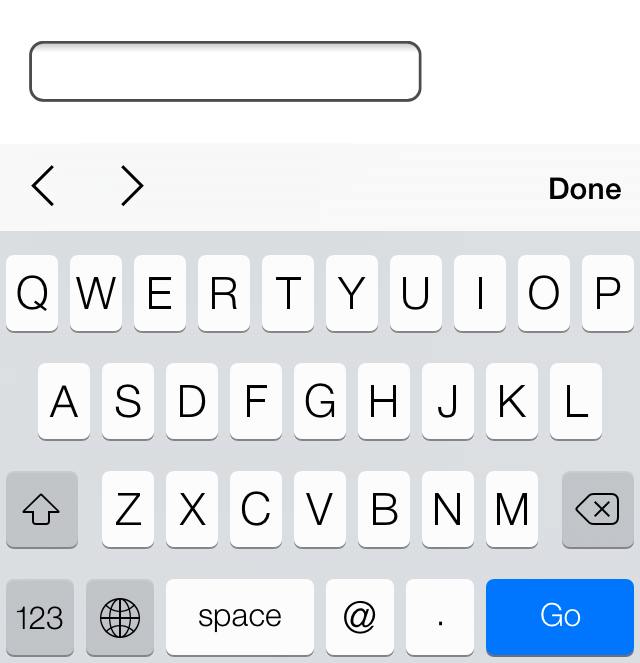
|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | <input type="date" name="birthday">  <input type="time" name="game-time">  <input type="email" name="email-address">  <input type="url" name="website">  <input type="number" name="cost">  <input type="tel" name="phone-number"> |

**Fig 10**

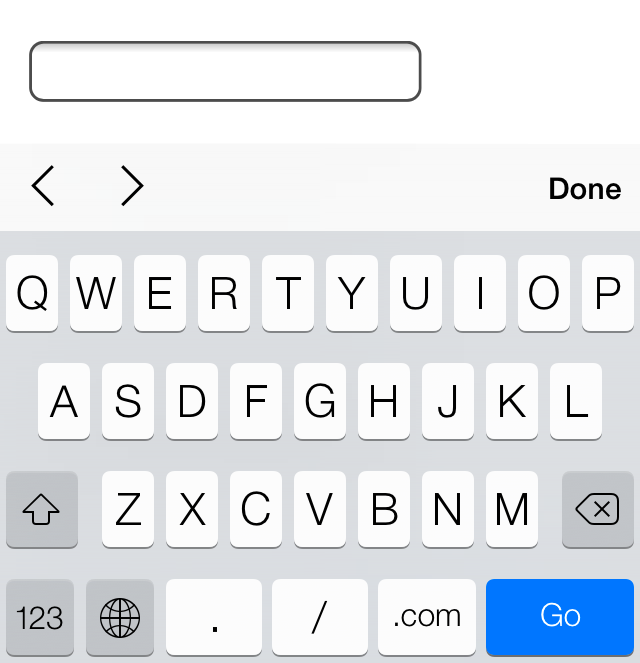
iOS7 controls for an <input> element with a type attribute value of date

**Fig 10**

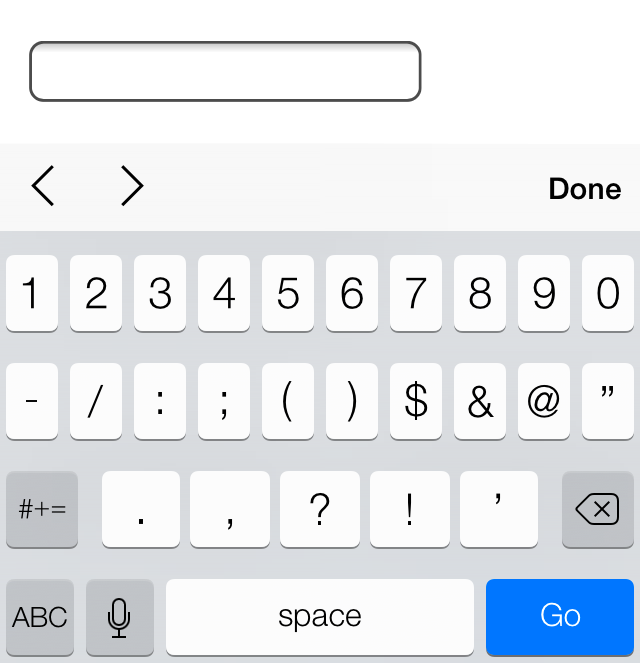
iOS7 controls for an <input> element with a type attribute value of time

**Fig 10**

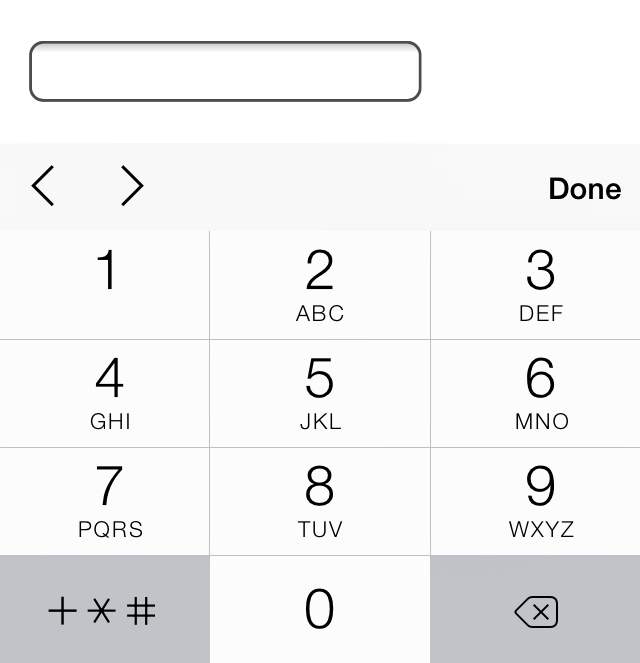
iOS7 controls for an <input> element with a type attribute value of email

**Fig 10**

iOS7 controls for an <input> element with a type attribute value of url

**Fig 10**

iOS7 controls for an <input> element with a type attribute value of number

**Fig 10**

iOS7 controls for an <input> element with a type attribute value of tel

### Textarea

Another element that’s used to capture text-based data is the <textarea> element. The <textarea> element differs from the <input> element in that it can accept larger passages of text spanning multiple lines. The <textarea> element also has start and end tags that can wrap plain text. Because the <textarea> element only accepts one type of value, the type attribute doesn’t apply here, but the name attribute is still used.

|  |  |
| --- | --- |
| 1  2 | <textarea name="comment">Add your comment here</textarea> |

#### Textarea Demo

The <textarea> element has two sizing attributes: cols for width in terms of the average character width and rows for height in terms of the number of lines of visible text. The size of a textarea, however, is more commonly identified using the width and height properties within CSS.

## Multiple Choice Inputs & Menus[#multple-choice-inputs](http://learn.shayhowe.com/html-css/building-forms/#multple-choice-inputs)

Apart from text-based input controls, HTML also allows users to select data using multiple choice and drop-down lists. There are a few different options and elements for these form controls, each of which has distinctive benefits.

### Radio Buttons

Radio buttons are an easy way to allow users to make a quick choice from a small list of options. Radio buttons permit users to select one option only, as opposed to multiple options.

To create a radio button, the <input> element is used with a type attribute value of radio. Each radio button element should have the same name attribute value so that all of the buttons within a group correspond to one another.

With text-based inputs, the value of an input is determined by what a user types in; with radio buttons a user is making a multiple choice selection. Thus, we have to define the input value. Using the value attribute, we can set a specific value for each <input> element.

Additionally, to preselect a radio button for users we can use the Boolean attribute checked.

|  |  |
| --- | --- |
| 1  2  3  4 | <input type="radio" name="day" value="Friday" checked> Friday  <input type="radio" name="day" value="Saturday"> Saturday  <input type="radio" name="day" value="Sunday"> Sunday |

#### Radio Buttons Demo

### Check Boxes

Check boxes are very similar to radio buttons. They use the same attributes and patterns, with the exception of checkbox as their type attribute value. The difference between the two is that check boxes allow users to select multiple values and tie them all to one control name, while radio buttons limit users to one value.

|  |  |
| --- | --- |
| 1  2  3  4 | <input type="checkbox" name="day" value="Friday" checked> Friday  <input type="checkbox" name="day" value="Saturday"> Saturday  <input type="checkbox" name="day" value="Sunday"> Sunday |

#### Check Boxes Demo

### Drop-Down Lists

Drop-down lists are a perfect way to provide users with a long list of options in a practi- cal manner. A long column of radio buttons next to a list of different options is not only visually unappealing, it’s daunting and difficult for users to comprehend, especially those on a mobile device. Drop-down lists, on the other hand, provide the perfect format for a long list of choices.

To create a drop-down list we’ll use the <select> and <option> elements. The <select> element wraps all of the menu options, and each menu option is marked up using the <option> element.

The name attribute resides on the <select> element, and the value attribute resides on the <option> elements that are nested within the <select> element. The value attribute on each <option> element then corresponds to the name attribute on the <select> element.

Each <option> element wraps the text (which is visible to users) of an individual option within the list.

Much like the checked Boolean attribute for radio buttons and check boxes, drop-down menus can use the selected Boolean attribute to preselect an option for users.

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | <select name="day">  <option value="Friday" selected>Friday</option>  <option value="Saturday">Saturday</option>  <option value="Sunday">Sunday</option>  </select> |

#### Drop-Down Lists Demo

### Multiple Selections

The Boolean attribute multiple, when added to the <select> element for a standard drop-down list, allows a user to choose more than one option from the list at a time. Additionally, using the selected Boolean attribute on more than one <option> element within the menu will preselect multiple options.

The size of the <select> element can be controlled using CSS and should be adjusted appropriately to allow for multiple selections. It may be worthwhile to inform users that to choose multiple options they will need to hold down the Shift key while clicking to make their selections.

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | <select name="day" multiple>  <option value="Friday" selected>Friday</option>  <option value="Saturday">Saturday</option>  <option value="Sunday">Sunday</option>  </select> |

#### Multiple Selections Demo

## Form Buttons[#form-buttons](http://learn.shayhowe.com/html-css/building-forms/#form-buttons)

After a user inputs the requested information, buttons allow the user to put that infor- mation into action. Most commonly, a submit input or submit button is used to process the data.

### Submit Input

Users click the submit button to process data after filling out a form. The submit button is created using the <input> element with a type attribute value of submit. The value attribute is used to specify the text that appears within the button.

|  |  |
| --- | --- |
| 1  2 | <input type="submit" name="submit" value="Send"> |

#### Submit Input Demo

### Submit Button

As an <input> element, the submit button is self-contained and cannot wrap any other content. If more control over the structure and design of the input is desired—along with the ability to wrap other elements—the <button> element may be used.

The <button> element performs the same way as the <input> element with the type attribute value of submit; however, it includes opening and closing tags, which may wrap other elements. By default, the <button> element acts as if it has a type attribute value of submit, so the type attribute and value may be omitted from the <button> element if you wish.

Rather than using the value attribute to control the text within the submit button, the text that appears between the opening and closing tags of the <button> element will appear.

|  |  |
| --- | --- |
| 1  2  3  4 | <button name="submit">  <strong>Send Us</strong> a Message  </button> |

#### Submit Button Demo

## Other Inputs[#other-inputs](http://learn.shayhowe.com/html-css/building-forms/#other-inputs)

Besides the applications we’ve just discussed, the <input> element has a few other use cases. These include passing hidden data and attaching files during form processing.

### Hidden Input

Hidden inputs provide a way to pass data to the server without displaying it to users. Hidden inputs are typically used for tracking codes, keys, or other information that is not pertinent to the user but is helpful when processing the form. This information is not displayed on the page; however, it can be found by viewing the source code of a page. It should therefore not be used for sensitive or secure information.

To create a hidden input, you use the hidden value for the type attribute. Additionally, include the appropriate name and value attribute values.

|  |  |
| --- | --- |
| 1  2 | <input type="hidden" name="tracking-code" value="abc-123"> |

### File Input

To allow users to add a file to a form, much like attaching a file to an email, use the file value for the type attribute.

|  |  |
| --- | --- |
| 1  2 | <input type="file" name="file"> |

#### File Input Demo

Unfortunately, styling an <input> element that has a type attribute value of file is a tough task with CSS. Each browser has its own default input style, and none provide much control to override the default styling. JavaScript and other solutions can be employed to allow for file input, but they are slightly more difficult to construct.

## Organizing Form Elements[#organizing-form-elements](http://learn.shayhowe.com/html-css/building-forms/#organizing-form-elements)

Knowing how to capture data with inputs is half the battle. Organizing form elements and controls in a usable manner is the other half. When interacting with forms, users need to understand what is being asked of them and how to provide the requested information.

By using labels, fieldsets, and legends, we can better organize forms and guide users to properly complete them.

### Label

Labels provide captions or headings for form controls, unambiguously tying them together and creating an accessible form for all users and assistive technologies. Created using the <label> element, labels should include text that describes the inputs or controls they pertain to.

Labels may include a for attribute. The value of the for attribute should be the same as the value of the id attribute on the form control the label corresponds to. Matching up the for and id attribute values ties the two elements together, allowing users to click on the <label> element to bring focus to the proper form control.

|  |  |
| --- | --- |
| 1  2  3 | <label for="username">Username</label>  <input type="text" name="username" id="username"> |

#### Label Demo

If desired, the <label> element may wrap form controls, such as radio buttons or check boxes. Doing so allows omission of the for and id attributes.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | <label>  <input type="radio" name="day" value="Friday" checked> Friday  </label>  <label>  <input type="radio" name="day" value="Saturday"> Saturday  </label>  <label>  <input type="radio" name="day" value="Sunday"> Sunday  </label> |

#### Labels with Nested Inputs Demo

### Fieldset

Fieldsets group form controls and labels into organized sections. Much like a <section> or other structural element, the <fieldset> is a block-level element that wraps related elements, specifically within a <form> element, for better organization. Fieldsets, by default, also include a border outline, which can be modified using CSS.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | <fieldset>  <label>  Username  <input type="text" name="username">  </label>  <label>  Password  <input type="text" name="password">  </label>  </fieldset> |

#### Fieldset Demo

### Legend

A legend provides a caption, or heading, for the <fieldset> element. The <legend> element wraps text describing the form controls that fall within the fieldset. The markup should include the <legend> element directly after the opening <fieldset> tag. On the page, the legend will appear within the top left part of the fieldset border.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | <fieldset>  <legend>Login</legend>  <label>  Username  <input type="text" name="username">  </label>  <label>  Password  <input type="text" name="password">  </label>  </fieldset> |

#### Legend Demo

## Form & Input Attributes[#form-and-input-attributes](http://learn.shayhowe.com/html-css/building-forms/#form-and-input-attributes)

To accommodate all of the different form, input, and control elements, there are a number of attributes and corresponding values. These attributes and values serve a handful of different functions, such as disabling controls and adding form validation. Described next are some of the more frequently used and helpful attributes.

### Disabled

The disabled Boolean attribute turns off an element or control so that it is not available for interaction or input. Elements that are disabled will not send any value to the server for form processing.

￼Applying the disabled Boolean attribute to a <fieldset> element will disable all of the form controls within the fieldset. If the type attribute has a hidden value, the hidden Boolean attribute is ignored.

|  |  |
| --- | --- |
| 1  2  3  4  5 | <label>  Username  <input type="text" name="username" disabled>  </label> |

#### Disabled Demo

### Placeholder

The placeholder HTML5 attribute provides a hint or tip within the form control of an <input> or <textarea> element that disappears once the control is clicked in or gains focus. This is used to give users further information on how the form input should be filled in, for example, the email address format to use.

|  |  |
| --- | --- |
| 1  2  3  4  5 | <label>  Email Address  <input type="email" name="email-address" placeholder="name@domain.com">  </label> |

#### Placeholder Demo

The main difference between the placeholder and value attributes is that the value attribute value text stays in place when a control has focus unless a user manually deletes it. This is great for pre-populating data, such as personal information, for a user but not for providing suggestions.

### Required

The required HTML5 Boolean attribute enforces that an element or form control must contain a value upon being submitted to the server. Should an element or form control not have a value, an error message will be displayed requesting that the user complete the required field. Currently, error message styles are controlled by the browser and cannot be styled with CSS. Invalid elements and form controls, on the other hand, can be styled using the :optional and :required CSS pseudo-classes.

Validation also occurs specific to a control’s type. For example, an <input> element with a type attribute value of email will require not only that a value exist within the control, but also that it is a valid email address.

|  |  |
| --- | --- |
| 1  2  3  4  5 | <label>  Email Address  <input type="email" name="email-address" required>  </label> |

#### Required Demo

### Additional Attributes

Other form and form control attributes include, but are not limited to, the following. Please feel free to research these attributes as necessary.

* accept
* autocomplete
* autofocus
* formaction
* formenctype
* formmethod
* formnovalidate
* formtarget
* max
* maxlength
* min
* pattern
* readonly
* selectionDirection
* step

## Login Form Example[#login-example](http://learn.shayhowe.com/html-css/building-forms/#login-example)

The following is an example of a complete login form that includes several different elements and attributes to illustrate what we’ve covered so far. These elements are then styled using CSS.

###### HTML

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | <form>  <fieldset class="account-info">  <label>  Username  <input type="text" name="username">  </label>  <label>  Password  <input type="password" name="password">  </label>  </fieldset>  <fieldset class="account-action">  <input class="btn" type="submit" name="submit" value="Login">  <label>  <input type="checkbox" name="remember"> Stay signed in  </label>  </fieldset>  </form> |

###### CSS

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61 | \*,  \*:before,  \*:after {  box-sizing: border-box;  }  form {  border: 1px solid #c6c7cc;  border-radius: 5px;  font: 14px/1.4 "Helvetica Neue", Helvetica, Arial, sans-serif;  overflow: hidden;  width: 240px;  }  fieldset {  border: 0;  margin: 0;  padding: 0;  }  input {  border-radius: 5px;  font: 14px/1.4 "Helvetica Neue", Helvetica, Arial, sans-serif;  margin: 0;  }  .account-info {  padding: 20px 20px 0 20px;  }  .account-info label {  color: #395870;  display: block;  font-weight: bold;  margin-bottom: 20px;  }  .account-info input {  background: #fff;  border: 1px solid #c6c7cc;  box-shadow: inset 0 1px 1px rgba(0, 0, 0, .1);  color: #636466;  padding: 6px;  margin-top: 6px;  width: 100%;  }  .account-action {  background: #f0f0f2;  border-top: 1px solid #c6c7cc;  padding: 20px;  }  .account-action .btn {  background: linear-gradient(#49708f, #293f50);  border: 0;  color: #fff;  cursor: pointer;  font-weight: bold;  float: left;  padding: 8px 16px;  }  .account-action label {  color: #7c7c80;  font-size: 12px;  float: left;  margin: 10px 0 0 20px;  } |

#### Required Demo

## In Practice[#practice-1](http://learn.shayhowe.com/html-css/building-forms/#practice-1)

With an understanding of how to build forms in place, let’s create a registration page for our Styles Conference website so that we can begin to gather interest and sell tickets for the event.

1. Jumping into our register.html file, we’ll begin by following the same layout pattern we used on our Speakers and Venue pages. This includes adding a <section> element with a class attribute value of row just below the registration lead-in section and nesting a <div> element with a class attribute value of grid directly inside the <section> element.

Our code just below the lead-in section for the Register page should look like this:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | <section class="row">  <div class="grid">  ...  </div>  </section> |

As a refresher, the class attribute value of row adds a white background and provides some vertical padding, while the class attribute value of grid centers our content in the middle of the page and provides some horizontal padding.

1. Inside the <div> element with a class attribute value of grid we’re going to create two columns, one covering two-thirds of the page width and one covering one-third of the page width. The two-thirds column will be a <section> element on the left- hand side that tells users why they should register for our conference. The one-third column, then, will be a <form> element on the right-hand side providing a way for users to register for our conference.

We’ll add these two elements, and their corresponding col-2-3 and col-1-3 classes, directly inside the <div> element with a class attribute value of grid. Since both of these elements will be inline-block elements, we need to open a comment directly after the two-thirds column closing tag and then close that comment directly before the one-third column opening tag.

In all, our code should look like this:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | <section class="row">  <div class="grid">  <section class="col-2-3">  ...  </section><!--  --><form class="col-1-3">  ...  </form>  </div>  </section> |

1. Now, inside our two-thirds column let’s add some details about our event and why it’s a good idea for aspiring designers and front-end developers to attend. We’ll do so using a handful of different heading levels (along with their pre-established styles), a paragraph, and an unordered list.

In our <section> element with a class attribute value of col-2-3, the code should look like this:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | <section class="col-2-3">  <h2>Purchase a Conference Pass</h2>  <h5>$99 per Pass</h5>  <p>Purchase your Styles Conference pass using the form to the right. Multiple passes may be purchased within the same order, so feel free to bring a friend or two along. Once your order is finished we&#8217;ll follow up and provide a receipt for your purchase. See you soon!</p>  <h4>Why Attend?</h4>  <ul>  <li>Over twenty world-class speakers</li>  <li>One full day of workshops and two full days of presentations</li>  <li>Hosted at The Chicago Theatre, a historical landmark</li>  <li>August in Chicago is simply amazing</li>  </ul>  </section> |

1. Currently our unordered list doesn’t have any list item markers. All of the browser default styles have been turned off by the CSS reset we added all the way back in Lesson 1. Let’s create some custom styles specifically for this unordered list.

To do so, let’s add a class attribute value of why-attend to the unordered list.

|  |  |
| --- | --- |
| 1  2  3  4 | <ul class="why-attend">  ...  </ul> |

With a class available to add styles to, let’s create a new section for Register page styles at the bottom of our main.css file. Within this section let’s use the class to select the unordered list and add a list-style of square and some bottom and left margins.

The new section at the bottom of our main.css file should look like this:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | /\*  ========================================  Register  ========================================  \*/  .why-attend {  list-style: square;  margin: 0 0 22px 30px;  } |

1. The details section of our registration page is complete, so now it’s time to address our registration form. We’ll start by adding the action and method attributes to the <form> element. Since we haven’t set up our form processing, these attributes will simply serve as placeholders and will need to be revisited.

The code for our <form> element should look like this:

|  |  |
| --- | --- |
| 1  2  3  4 | <form class="col-1-3" action="#" method="post">  ...  </form> |

1. Next, inside the <form> element we’ll add a <fieldset> element. Inside the <fieldset> element we’ll add a series of <label> elements that wrap a given form control.

We want to collect a user’s name, email address, number of desired conference passes, and any potential comments. The name, email address, and number of conference passes are required fields, and we’ll want to make sure we use the appropriate elements and attributes for each form control.

With a mix of different input types, select menus, textareas, and attributes, the code for our form should look like the following:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36 | <form class="col-1-3" action="#" method="post">  <fieldset>  <label>  Name  <input type="text" name="name" placeholder="Full name" required>  </label>  <label>  Email  <input type="email" name="email" placeholder="Email address" required>  </label>  <label>  Number of Passes  <select name="quantity" required>  <option value="1" selected>1</option>  <option value="2">2</option>  <option value="3">3</option>  <option value="4">4</option>  <option value="5">5</option>  </select>  </label>  <label>  Comments  <textarea name="comments"></textarea>  </label>  </fieldset>  <input type="submit" name="submit" value="Purchase">  </form> |

Here we can see each form control nested within a <label> element. The Name form control uses an <input> element with a type attribute value of text, while the Email form control uses an <input> element with a type attribute value of email.

Both the Name and Email form controls include the required Boolean attribute and a placeholder attribute.

The Number of Passes form control uses the <select> element and nested <option> elements. The <select> element itself includes the required Boolean attribute, and the first <option> element includes the selected Boolean attribute.

The Comments form control uses the <textarea> element without any special modifications. And lastly, outside of the <fieldset> element is the submit form control, which is formed by an <input> element with a type attribute value of submit.

1. With the form in place, it’s time to add styles to it. We’ll begin with a few default styles on the <form> element itself and on the <input>, <select>, and <textarea> elements.

Within the register section of our main.css file we’ll want to add the following styles:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | form {  margin-bottom: 22px;  }  input,  select,  textarea {  font: 300 16px/22px "Lato", "Open Sans", "Helvetica Neue", Helvetica, Arial, sans-serif;  } |

We’ll start by placing a 22-pixel margin on the bottom of our form to help vertically space it apart from other elements. Then we’ll add some standard font-based styles—including weight, size, line-height, and family—for all of the <input>, <select>, and <textarea> elements.

By default, every browser has its own interpretation of how the styles for form controls should appear. With this in mind, we have repeated the font-based styles from our <body> element to ensure that our styles remain consistent.

1. Let’s add some styles to the elements within the <fieldset> element. Since we may add additional <fieldset> elements later on, let’s add a class attribute value of register-group to our existing <fieldset> element, and from there we can apply unique styles to the elements nested within it.

|  |  |
| --- | --- |
| 1  2  3  4 | <fieldset class="register-group">  ...  </fieldset> |

1. Once the register-group class attribute value is in place, we’ll add a few styles to the elements nested within the <fieldset> element. These styles will appear in our main.css file, below the existing form styles.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27 | .register-group label {  color: #648880;  cursor: pointer;  font-weight: 400;  }  .register-group input,  .register-group select,  .register-group textarea {  border: 1px solid #c6c9cc;  border-radius: 5px;  color: #888;  display: block;  margin: 5px 0 27px 0;  padding: 5px 8px;  }  .register-group input,  .register-group textarea {  width: 100%;  }  .register-group select {  height: 34px;  width: 60px;  }  .register-group textarea {  height: 78px;  } |

1. You’ll notice that most of these properties and values revolve around the box model, which we covered in Lesson 4. We’re primarily setting up the size of different form controls, ensuring that they are laid out appropriately. Aside from adding some box model styles, we’re adjusting the color and font-weight of a few elements.
2. So far, so good: our form is coming together quite nicely. The only remaining element yet to be styled is the submit button. As it’s a button, we actually have some existing styles we can apply here. If we think back to our home page, our hero section contained a button that received some styles by way of the btn class attribute value.

Let’s add this class attribute value, btn, along with a new class attribute value of btn-default to our submit button. Specifically we’ll use the class name of btn-default since this button is appearing on a white background and will be the default style for buttons moving forward.

|  |  |
| --- | --- |
| 1  2 | <input class="btn btn-default" type="submit" name="submit" value="Purchase"> |

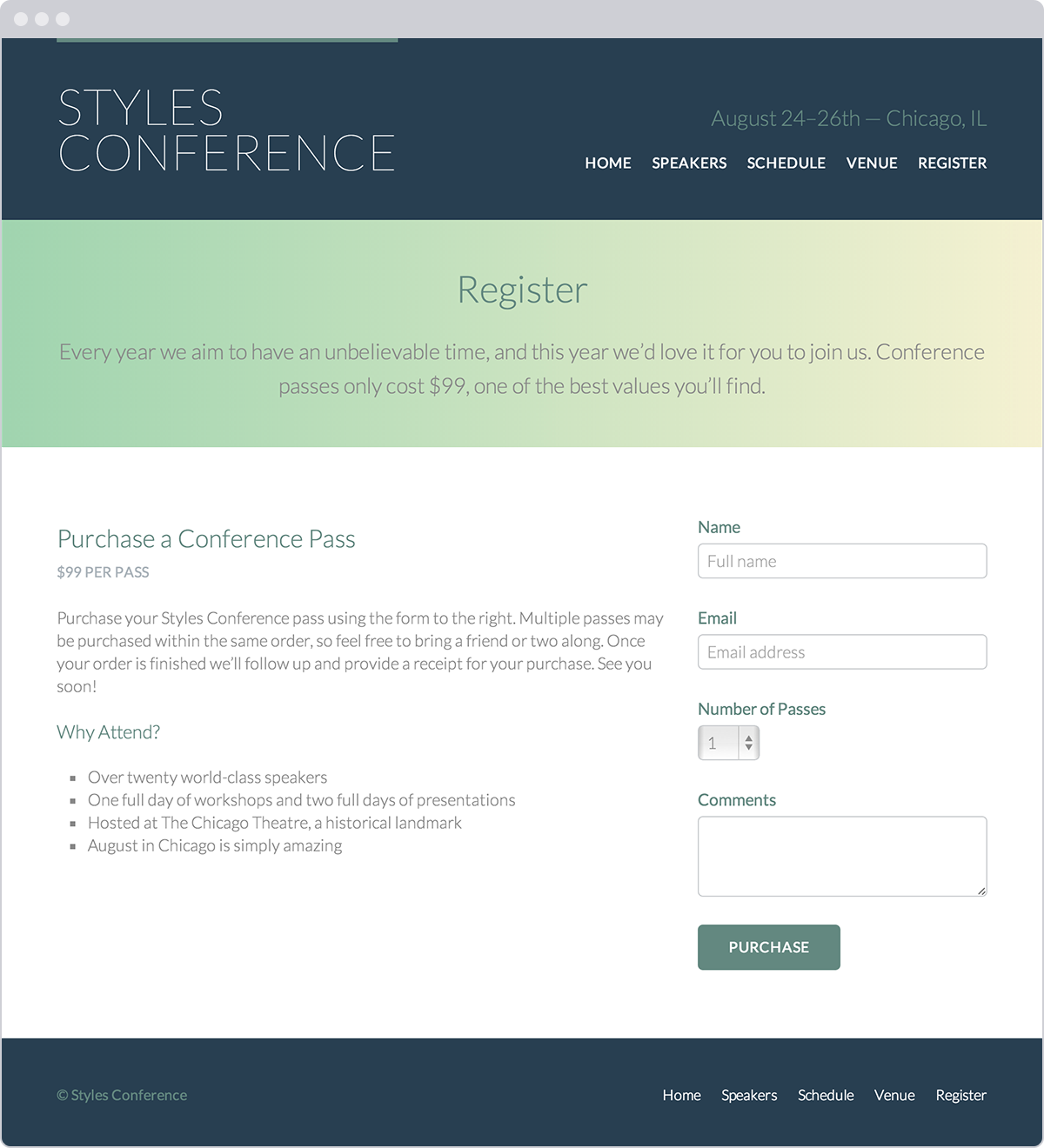
Now our submit button has some shared styles with the button on the home page. We’ll use the btn-default class attribute value to then apply some new styles to our submit button specifically.

Going back to the buttons section of our main.css file, let’s add the following:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | .btn-default {  border: 0;  background: #648880;  padding: 11px 30px;  font-size: 14px;  }  .btn-default:hover {  background: #77a198;  } |

These new styles, which define the size and background of our submit button, are then combined with the existing btn class styles to create the final presentation of our submit button.

Our Register page is finished, and attendees can now begin to reserve their tickets.

**Fig 10**

Our registration page, which includes a form

### Demo & Source Code

Below you may view the Styles Conference website in its current state, as well as download the source code for the website in its current state.

[**View the Styles Conference Website**](http://learn.shayhowe.com/practice/building-forms/index.html) or [Download the Source Code](http://learn.shayhowe.com/practice/building-forms.zip) (Zip file)

## Summary[#summary](http://learn.shayhowe.com/html-css/building-forms/#summary)

Forms play a large role in how users interact with, provide information to, and take action on websites. We’ve taken all the right steps to learn not only how to mark up forms but also how to style them.

To quickly recap, within this lesson we discussed the following:

* How to initialize a form
* Ways to obtain text-based information from users
* Different elements and methods for creating multiple choice options and menus
* Which elements and attributes are best used to submit a form’s data for processing
* How best to organize forms and give form controls structure and meaning
* A handful of attributes that help collect more qualified data

Our understanding of HTML and CSS is progressing quite nicely, and we only have one more component to learn: tables. In the next chapter, we’ll take a look at how to organize and present data with tables.