sitepoint

JUMP START CSS

BY LOUIS LAZARIS

Jump Start CSS

by Louis Lazaris

Product Manager: Simon Mackie
English Editor: Paul Fitzpatrick
Technical Editor: Rachel Andrew
Cover Designer: Alex Walker

Notice of Rights

Notice of Liability

Trademark Notice

Published by SitePoint Pty. Ltd.

48 Cambridge Street Collingwood
VIC Australia 3066
Web: www.sitepoint.com
Email: business@sitepoint.com

ISBN 978-0-9874674-4-7 (print)

ISBN 978-0-9874674-5-4 (ebook)
Printed and bound in the United States of America

About Louis Lazaris

Louis Lazaris is a web designer and blogger who has been creating and coding websites for more than a decade. You can find him on Twitter[1] or you can read more on CSS on his website, Impressive Webs[2].

About SitePoint

SitePoint specializes in publishing fun, practical, and easy-to-understand content for web professionals. Visit http://www.sitepoint.com/ to access our blogs, books, newsletters, articles, and community forums. You'll find a stack of information on JavaScript, PHP, Ruby, mobile development, design, and more.

About Jump Start

Jump Start books provide you with a rapid and practical introduction to web development languages and technologies. Typically around 150 pages in length, they can be read in a weekend, giving you a solid grounding in the topic and the confidence to experiment on your own.

[1] https://twitter.com/ ImpressiveWebs
[2] http://www.impressivewebs.com/

Table of Contents

Preface

Do you remember your first educational toy? One of the first that many children get is the big, chunky puzzle—four to ten easy-to-grip pieces that fit into spaces on a board.

This is one of the first experiences a child has with matching shapes to corresponding spaces, helping them develop their shape recognition skills.

My wife thinks I was never given one of these puzzles. Every time I put the dishes away, the plastic food storage containers stump me. I end up trying to put medium containers into small containers and square containers into round ones. It's as if, in her words, I never got the proper training as a child. I pretend she's just joking, but maybe she's right—I really can't remember.

If you've purchased this little book, in terms of CSS knowledge, you're a lot like an infant with its first shapes puzzle. I hope to teach you as much as possible, as quickly as possible, about the fundamentals of CSS. And I hope later in life you'll look back and be thankful that you took the time to "learn your shapes."

Who Should Read This Book

This book is suitable for beginner level web designers and developers. Some knowledge of HTML is assumed.

Conventions Used

You'll notice that we've used certain typographic and layout styles throughout this book to signify different types of information. Look out for the following items.

Code Samples

Code in this book will be displayed using a fixed-width font, like so:

```
<h1>A Perfect Summer's Day</h1>
<p>It was a lovely day for a walk in the park. The birds
were singing and the kids were all back at school.</p>
```

If the code is to be found in the book's code archive, the name of the file will appear at the top of the program listing, like this:

example.css

```
.footer {
  background-color: #CCC;
  border-top: 1px solid #333;
}
```

If only part of the file is displayed, this is indicated by the word *excerpt*:

example.css *(excerpt)*

```
  border-top: 1px solid #333;
```

If additional code is to be inserted into an existing example, the new code will be displayed in bold:

```
function animate() {
  new_variable = "Hello";
}
```

Also, where existing code is required for context, rather than repeat all it, a ⋮ will be displayed:

```
function animate() {
  ⋮
  return new_variable;
}
```

Some lines of code are intended to be entered on one line, but we've had to wrap them because of page constraints. A ➥ indicates a line break that exists for formatting purposes only, and that should be ignored.

```
URL.open("http://www.sitepoint.com/responsive-web-design-real-user-
➥testing/?responsive1");
```

Tips, Notes, and Warnings

Hey, You!

Tips will give you helpful little pointers.

Ahem, Excuse Me ...

Notes are useful asides that are related—but not critical—to the topic at hand. Think of them as extra tidbits of information.

Make Sure You Always ...

... pay attention to these important points.

Watch Out!

Warnings will highlight any gotchas that are likely to trip you up along the way.

Supplementary Materials

http://www.sitepoint.com/books/jscss1/

The book's website, containing links, updates, resources, and more.

http://www.sitepoint.com/books/jscss1/code.php

The downloadable code archive for this book.

http://www.sitepoint.com/forums/forumdisplay.php?53-css

SitePoint's forums, for help on any tricky web problems.

`books@sitepoint.com`

Our email address, should you need to contact us for support, to report a problem, or for any other reason.

Want to take your learning further?

Thanks for buying this book. We appreciate your support. Do you want to continue learning? You can now get unlimited access to courses and ALL SitePoint books at

Learnable for one low price. Enroll now and start learning today! Join Learnable and you'll stay ahead of the newest technology trends: http://www.learnable.com.

Acknowledgments

Thanks to Simon Mackie and Rachel Andrew for their excellent and practical feedback to help make this book much better than it would have been had I tackled this on my own.

Chapter 1

An Introduction to CSS

Welcome to Jump Start CSS! This book is an introduction to CSS that'll teach you the basics so you can start sprucing up your web pages using Cascading Style Sheets (CSS), the industry standard technology for formatting web pages.

For the most part, this book will not serve as a theoretical source for the topics we'll be discussing—there are plenty of other resources for that. In this brief volume, we'll be focusing on practical information. I'll be showing you, in rapid fashion, what the various aspects of CSS are, and how you can use them to build web pages.

The Sample Project

In order to give you hands-on experience with CSS, this book is centered around a sample project that we'll be building together.

The sample project is a phony website called RecipeFinder. You can access a completed version of that website by visiting http://spbooks.github.io/JSCSS1/ in any web browser.

We're going to take RecipeFinder from mock-up to development. The sample site's look is based on a Photoshop design. Figure 1.1 shows you what it looks like.

Figure 1.1. The website we'll be building in this book

Now, before we get into building our project, let's properly introduce the elements of CSS and how it's used to style web pages.

How are web pages built?

Let's begin by briefly considering what exactly CSS is, and how it relates to a web page. Web pages are built on **content**. Content is what you see when you visit a page. It might include text, photos, graphics, and video. Content is stored using a language called HTML. You've probably heard of it, but here's a very quick overview.

HTML consists of **elements**, many of which have what are called opening and closing **tags**. These instruct web browsers how content (copy, photos, videos, and so on) should be presented on screen. For example:

```
<header>
<h1>RecipeFinder</h1>
</header>
```

In this case, the content that's visible on the page is the text “RecipeFinder.” Everything else you see there (specifically the information inside the angle brackets) is HTML, and it's invisible on the page when viewed in a browser. What it does is to help mark where sections of content begin and end. For this reason **HTML** is what's called a markup language. In fact, it stands for Hyper Text Markup Language.

As mentioned, this is not a book on HTML, but if you want to learn more, two possible resources are SitePoint's online HTML Concepts[1] or the Mozilla Developer Network's Introduction to HTML.[2]

What Is CSS?

CSS stands for Cascading Style Sheets and is a separate, but complementary, language to HTML. CSS is what we use to apply styles to the content on our web page.

Let's use the HTML from the example above to give you a first taste of CSS. Don't worry if this looks foreign to you right now—just become familiar with the look of the code:

```
header {
   color: white;
   background-color: #333;
   font-size: 1.5em;
}
```

What you see above is referred to as a **rule set**. Notice the curly braces that wrap three lines of code. Also, notice that each line inside the curly braces has a colon and a semi-colon. Everything inside the curly braces is called a **declaration block**.

[1] http://reference.sitepoint.com/html/html-concepts
[2] https://developer.mozilla.org/en-US/docs/HTML/Introduction

The portion prior to the first curly brace is what defines which part of the web page we are styling. This is referred to as the **selector**. We'll discuss more on selectors later in this chapter. In this case, our CSS is targeting the `<header>` HTML element.

Each of the three lines in the declaration block is referred to as a—you guessed it—**declaration**. Additionally, each declaration consists of a **property** (the part before the colon) and a **value** (the part after the colon). Finally, each CSS declaration ends with a semi-colon.

What I've shown you here is a very simple example. Other CSS code examples might be more complex, but most are fairly easy to figure out through trial and error—so don't be too intimidated if you come across something you don't recognize.

So what does that code do? Well, we'll get into the specifics on CSS properties later, so hang tight while we continue discussing some further important basics that'll serve as a foundation for the rest of this book.

How do I include CSS in a web page?

CSS can be inserted into a web page in four different ways. Let's take a look at each one, saving the most highly recommended method for last.

Using Inline Styles

Any HTML element on a web page can be styled using **inline styles**. Here's an example, using some of the HTML we've already introduced:

```
<h1 style="color: blue; background-color:
#333;">RecipeFinder</h1>
```

In this case, the CSS is contained inside of an HTML **attribute** called `style`. The attribute tells the browser that what follows inside the quotation marks is CSS. In this example, the styles will only apply to the element to which they're attached (the `<h1>` element in this case). This is a very inefficient way of inserting CSS and should be avoided in almost all circumstances.

Using the `<style>` Element

You can also include CSS in an HTML page using a `<style>` tag, which also requires a closing `</style>` tag:

```
<style>
header {
   color: white;
   background-color: #333;
   font-size: 1.5em;
}
</style>
```

In this example, the styles will apply only to the element(s) targeted in the selector. So, in this instance, the styles will apply to all <header> elements on the page where this `<style>` element appears.

Using `@import` inside a `<style>` element

You also have the option to include CSS in a separate file. It's similar to a text file, but has a file extension of ".css". So a chunk of CSS inside a separate file can be imported into your HTML like this:

```
<style>
@import url(css/style.css);
</style>
```

The `@import` method of including CSS has been known to cause some problems—for example, multiple CSS files loaded via `@import` will often take longer to download[3]—so, in general, you'd do well to avoid using it.

The Best Way: Using the `<link>` Element

The best way to include CSS in a web page is by means of the `<link>` element:

```
<link rel="stylesheet" href="css/style.css">
```

This element, which would be placed in the `<head>` element of your HTML document, is much like `@import` in that it references an external file. This enables you to keep all your CSS code separate from the HTML. In addition, this method doesn't cause some of the issues that can arise when using `@import`. Also, because the styles are not "inline," scattered among the HTML, your CSS will be that much easier to maintain.

[3] http://www.stevesouders.com/blog/2009/04/09/dont-use-import/

Introducing CSS Selectors

As already alluded to, a CSS selector is the part of a CSS rule set that actually selects the content you want to style. Let's look at all the different kinds of selectors available, with a brief description of each.

Universal Selector

The **universal selector** works like a wild card character, selecting all elements on a page. You'll recall, from our brief overview earlier, that every HTML page is built on content placed within HTML tags. Each set of tags represents an element on the page. Look at the following CSS example, which uses the universal selector:

```
* {
   color: green;
   font-size: 20px;
   line-height: 25px;
}
```

The three lines of code inside the curly braces (`color`, `font-size`, and `line-height`) will apply to all elements on the HTML page. As seen here, the universal selector is declared using an asterisk. You can also use the universal selector in combination with other selectors—something we'll discuss a little later in this chapter.

Element Type Selector

Also referred to simply as a "type selector," this selector must match one or more HTML elements of the same name. Thus, a selector of `nav` would match all HTML `<nav>` elements, and a selector of `ul` would match all HTML unordered lists, or `<ul>` elements.

The following example uses an element type selector to match all `<ul>` elements:

```
ul {
   list-style: none;
   border: solid 1px #ccc;
}
```

To put this in some context, here's a section of HTML to which we'll apply the above CSS:

```
<ul>
  <li>Fish</li>
  <li>Apples</li>
  <li>Cheese</li>
</ul>

<div class="example">
  <p>Example paragraph text.</p>
</div>

<ul>
  <li>Water</li>
  <li>Juice</li>
  <li>Maple Syrup</li>
</ul>
```

There are three main elements making up this part of the page: Two `<ul>` elements and a `<div>`. The CSS will apply only to the two `<ul>` elements, and not to the `<div>`. Were we to change the element type selector to use div instead of `ul`, then the styles would apply to the `<div>` and not to the two `<ul>` elements.

Also note that the styles will not apply to the elements *inside* the `<ul>` or `<div>` elements. That being said, some of the styles *may* be inherited by those inner elements—more on this later in the book.

ID Selector

An ID selector is declared using a hash, or pound symbol (#) preceding a string of characters. The string of characters is defined by the developer. This selector matches any HTML element that has an ID attribute with the same value as that of the selector, but minus the hash symbol.

Here's an example:

```
#container {
   width: 960px;
   margin: 0 auto;
}
```

This CSS uses an ID selector to match an HTML element such as:

```
<div id="container"></div>
```

In this case, the fact that this is a `<div>` element doesn't matter—it could be any kind of HTML element. As long as it has an ID attribute with a value of `container`, the styles will apply.

An ID element on a web page should be unique. That is, there should only be a single element on any given page with an ID of `container`. This makes the ID selector quite inflexible, because the styles used in the ID selector rule set can be used only once per page.

If there happens to be more than one element on the page with the same ID, the styles will still apply, but the HTML on such a page would be invalid from a technical standpoint, so you'll want to avoid doing this.

In addition to the problems of inflexibility, ID selectors also have the problem of very high specificity—an issue we'll be talking about later in this chapter.

Class Selector

The class selector is the most useful of all CSS selectors. It's declared with a dot preceding a string of one or more characters. Just as is the case with an ID selector, this string of characters is defined by the developer. The class selector also matches all elements on the page that have their class attribute set to the same value as the class, minus the dot.

Take the following rule set:

```
.box {
   padding: 20px;
   margin: 10px;
   width: 240px;
}
```

These styles will apply to the following HTML element:

```
<div class="box"></div>
```

The same styles will also apply to any other HTML elements that have a class attribute with a value of `box`. Having multiple elements on a single page with the same

class attribute is beneficial, because it allows you to reuse styles, and avoid needless repetition. In addition to this, class selectors have very low specificity—again, more on this later.

Another reason the class selector is a valuable ally is that HTML allows multiple classes to be added to a single element. This is done by separating the classes in the HTML class attribute using spaces. Here's an example:

```
<div class=”box box-more box-extended”></div>
```

Descendant Combinator

The descendant selector or, more accurately, the descendant **combinator** lets you combine two or more selectors so you can be more specific in your selection method. For example:

```
#container .box {
   float: left;
   padding-bottom: 15px;
}
```

This declaration block will apply to all elements that have a class of `box` that are inside an element with an ID of `container`. It's worth noting that the `.box` element doesn't have to be an immediate child: there could be another element wrapping `.box`, and the styles would still apply.

Look at the following HTML:

```
<div id="container">
  <div class="box"></div>

  <div class="box-2"></div>
</div>

<div class="box"></div>
```

If we apply the CSS in the previous example to this section of HTML, the only element that'll be affected by those styles is the first `<div>` element that has a class of `box`. The `<div>` element that has a class of `box-2` *won't* be affected by the styles.

Similarly, the second `<div>` element with a class of `box` won't be affected because it's not inside an element with an ID of `container`.

You should be careful when using the descendant combinator in your CSS. This kind of selector, while making your CSS a little easier to read, can unnecessarily restrict your styles to a specific context—in this case, the styles are restricted to boxes inside of `#container`—which can make your code inflexible.

Child Combinator

A selector that uses the child combinator is similar to a selector that uses a descendant combinator, except it only targets immediate child elements:

```
#container > .box {
   float: left;
   padding-bottom: 15px;
}
```

This is the same code from the descendant combinator example, but instead of a space character, we're using the greater-than symbol (or right angle bracket.)

In this example, the selector will match all elements that have a class of `box` and that are *immediate children* of the `#container` element. That means, unlike the descendant combinator, there can't be another element wrapping `.box`—it has to be a direct child element.

Here's an HTML example:

```
<div id="container">
  <div class="box"></div>

  <div>
    <div class="box"></div>
  </div>
</div>
```

In this example, the CSS from the previous code example will apply only to the first `<div>` element that has a class of `box`. As you can see, the second `<div>` element with a class of `box` is inside another `<div>` element. As a result, the styles will not apply to that element, even though it too has a class of `box`.