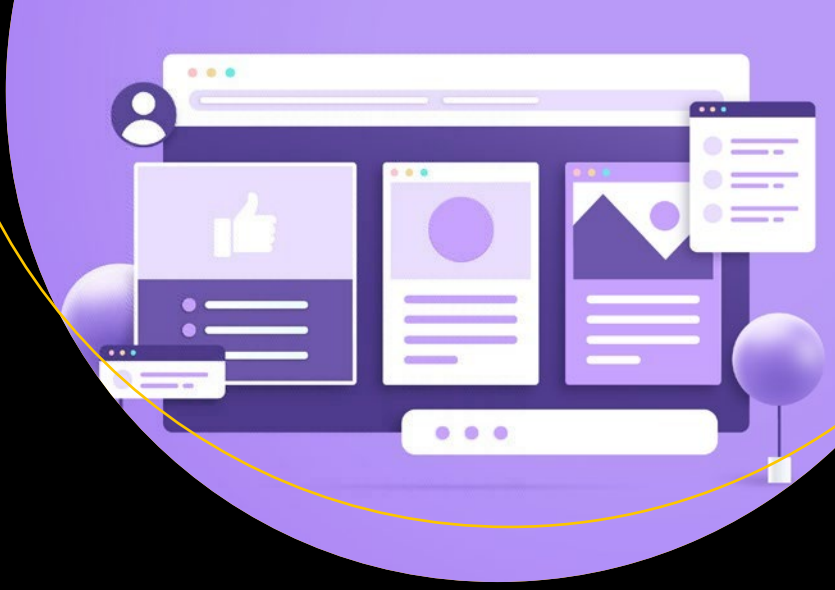

Modern CSS

Master the Key Concepts of CSS for Modern Web Development

Joe Attardi

Apress®

Modern CSS

Master the Key Concepts of CSS for Modern Web Development

Joe Attardi

Apress®

Modern CSS

Joe Attardi
Billerica, MA, USA

ISBN-13 (pbk): 978-1-4842-6293-1 ISBN-13 (electronic): 978-1-4842-6294-8
https://doi.org/10.1007/978-1-4842-6294-8

Managing Director, Apress Media LLC: Welmoed Spahr
Acquisitions Editor: Louise Corrigan
Development Editor: James Markham
Coordinating Editor: Nancy Chen

Cover designed by eStudioCalamar

Cover image designed by Freepik (www.freepik.com)

Distributed to the book trade worldwide by Springer Science+Business Media New York, 1 New York Plaza, New York, NY 10004. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail orders-ny@springer-sbm.com, or visit www.springeronline.com. Apress Media, LLC is a California LLC and the sole member (owner) is Springer Science + Business Media Finance Inc (SSBM Finance Inc). SSBM Finance Inc is a **Delaware** corporation.

For information on translations, please e-mail booktranslations@springernature.com; for reprint, paperback, or audio rights, please e-mail bookpermissions@springernature.com.

Apress titles may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Print and eBook Bulk Sales web page at http://www.apress.com/bulk-sales.

Any source code or other supplementary material referenced by the author in this book is available to readers on GitHub via the book's product page, located at www.apress.com/9781484262931. For more detailed information, please visit http://www.apress.com/source-code.

Printed on acid-free paper

To Liz and Benjamin – you are my whole world.

Table of Contents

About the Author

Joe Attardi is a software engineer specializing in front-end development. He has over 15 years' experience working with JavaScript, HTML, and CSS and has worked extensively with front-end technologies such as Angular and React. He currently works at Salesforce and has worked in the past with companies such as Dell and Nortel. He is also the author of *Using Gatsby and Netlify CMS*, an Apress title. He lives in the Boston area with his wife and son. You can find him on Twitter at @JoeAttardi.

About the Technical Reviewer

Alexander Nnakwue has a background in Mechanical Engineering from the University of Ibadan, Nigeria, and has been a front-end developer for over 3 years working on both web and mobile technologies. He also has experience as a technical author, writer, and reviewer. He enjoys programming for the Web, and occasionally, you can also find him playing soccer. He was born in Benin City and is currently based in Lagos, Nigeria.

Acknowledgments

I'd like to start by thanking my wonderful wife, Liz, for her constant love and encouragement throughout the whole writing process – and for understanding when I locked myself away in solitude to write. And my little toddler, Benjamin, for giving me much-needed breaks from writing for play time.

Thanks to all my friends and family for always supporting and encouraging my interest in computers and technology.

This book began its life as a self-published work, and I'd like to thank Apress for making it what it is today. I'd also like to thank the awesome team at Apress – Louise Corrigan, Nancy Chen, and Jim Markham – for guiding me through the process every step of the way. I appreciate their patience with me as a first-time author.

Thanks to Alexander Nnakwue, the technical reviewer for this book, for his time and excellent feedback, helping make this book even better.

Special thanks also to Stephanie Eckles and Ellie Baker, two extremely talented CSS experts who reviewed some of the chapters in the previous, self-published version of this book.

Introduction

In this book, we will take a tour of modern CSS. Whether you're brand new to CSS or you have some experience and need a refresher, this book will have something for you.

However, this book will not teach you color theory or good design techniques. The intent of this book is to give you a strong foundation with the various CSS technologies.

In Chapter 1, we'll start at the very beginning and talk about what CSS is and how it works. We'll explore the DOM, the CSSOM, and the render tree as well as take a quick detour to look at CSS preprocessors (though we won't cover them further in the book).

In Chapter 2, we will tackle CSS selectors. These are critical to understand. Selectors determine what CSS styles are applied to what elements. We'll also explore the concept of specificity.

Once we've laid the groundwork, we'll start to talk about CSS concepts in Chapter 3 like the box model, units, colors, and overflow. We'll also look at CSS custom properties, better known as variables.

We'll finally start applying styles in Chapter 4, where we'll look at borders, box shadows, and opacity. We will see several ways to hide an element on the page.

In Chapter 5, we'll learn all about backgrounds and gradients (which are actually a type of background image).

Chapter 6 deals with the important topic of styling text. We'll learn about text styles and layout, as well as how to use web fonts.

We'll see how to lay out and position elements in Chapter 7. This covers the different positions such as `static`, `relative`, `absolute`, `fixed`, and `sticky`. Also, in this chapter, we'll see the topic of stacking contexts and Z-index, which often trip up even experienced developers.

In Chapter 8, we'll cover CSS transforms. This allows you to apply transformations such as rotation, scale, and skew to elements. We'll also see a few examples of creating shapes with CSS.

Transforms can be combined with transitions, which is one topic of Chapter 9, to create all kinds of interesting effects. Transitions can be applied to transforms or a slew of other CSS properties. Chapter 9 also covers animations, which takes the concepts of transitions to the next level.

Chapter 10 is dedicated to the flexible box layout, or flexbox, which is a powerful one-dimensional layout tool that has excellent browser support. With flexbox, we can finally easily center a `div`!

Chapter 11 is a gentle introduction to responsive design techniques. While it is not an exhaustive guide – entire books have been written on the subject – it lays a good foundation, covering topics such as media queries and fluid typography.

Finally, we save the best for last. Chapter 12 is all about CSS Grid, the latest and greatest layout tool in the CSS toolbox. It doesn't have great Internet Explorer support, but all of the other major browsers have full support for it.

In Chapter 13 we'll see some other topics for further learning, such as CSS methodologies like BEM and OOCSS, as well as utility-first CSS and Houdini, the future of CSS.

Let's get started!

CHAPTER 1

Introduction to CSS

Chances are that you already have *some* idea of what CSS is, or else you probably wouldn't have been interested in this book. But let's start at the beginning, to make sure we're all on the same page.

CSS stands for Cascading Style Sheets. It's a language for specifying how an HTML document is displayed. Without CSS, every website would just be Times New Roman with tiny buttons. It's capable of much more than styling text, however. CSS lets us define entire layouts and position elements and even perform animations.

Style sheets are self-explanatory, but what is a *cascading* style sheet? Because more than one style rule could apply to a given HTML element, there needs to be some way to determine which rule should apply in the event of a conflict. The styles "cascade" from less specific to more specific selectors, and the most specific rule wins. Specificity is an important concept in CSS, and we will discuss that in more detail later.

A bit of history

Before CSS, support for styling in HTML was limited. The style information was included in the HTML markup. For example, the font face, size, and color were specified with the `font` tag and several different attributes, as shown in Listing 1-1.

Listing 1-1. Styling HTML with the font tag

```
<font face="Arial, sans-serif"
      color="blue"
      size="12">
  Hello world!
</font>
```


J. Attardi, *Modern CSS*, https://doi.org/10.1007/978-1-4842-6294-8_1

This resulted in a tight coupling between the semantics and the presentation of a document, which made websites harder to maintain. There were some basic layout options, such as the `center` tag. For more advanced layouts, the only real option was to use HTML tables. Tables were meant to display tabular data. One disadvantage of using tables for layout is that someone using a screen reader will have a very hard time navigating the page.

There were several other proposals for style sheets for HTML documents as well, but CSS was first proposed at a conference in Chicago in 1994. The following year saw the birth of the World Wide Web Consortium (W3C), and the initial version of the CSS standard was published in late 1996. In the years since, CSS has evolved into a powerful tool for styling HTML documents.

Anatomy of a CSS rule

A CSS style sheet consists of rules. CSS rules target HTML elements by using selectors that describe the elements to which the styles should be applied. As we will see later, elements can be selected in many ways.

Rule syntax

A rule consists of a selector followed by a block of CSS properties contained inside curly braces. The properties consist of a property and value separated by a colon and are delimited with semicolons. A value may be a single value or a collection of multiple values, depending on the property.

Every element in the document that is matched by the selector has the properties in the CSS rule applied to it. An example of a CSS rule is shown in Figure 1-1.

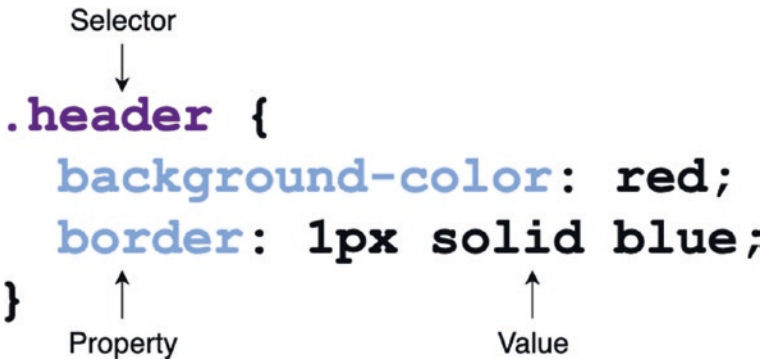

Figure 1-1. *The structure of a CSS rule*

This rule targets any element with the class `header` (more on classes later). Any element with this class will have a red background and a 1-pixel solid blue border.

In the previous example, `background-color` and `border` are CSS *properties*. The border width is specified as `1px`. `px` is a CSS *unit*. There are many units including `em`, `rem`, and `%`. We will learn more about the different CSS units later.

Property conflicts

If the same property is used more than once in a given rule, the last definition in the rule wins. An example of this is shown in Listing 1-2.

Listing 1-2. A CSS rule with conflicting properties

```
.header {
  background-color: red;
  background-color: blue;
}
```

In Listing 1-2, the element with the class `header` will have a blue background because it is the last one in the rule. The second `background-color` property overrides the first.

Comments

CSS can also contain comments, inside and outside of rules, as shown in Listing 1-3.

Listing 1-3. A CSS style sheet with comments

```
/* This is a comment outside of a rule */

.header {
  /* This is a comment inside of a rule */
  background-color: red;
}
```

At-rules

An *at-rule* is a special CSS rule that acts as a directive controlling the behavior of CSS. It is called an at-rule because it starts with the "at" sign (`@`). Here are some examples of at-rules:

- `@charset`: Defines the character encoding used in the CSS file.
- `@import`: Imports, or includes, the contents of another style sheet.
- `@media`: Defines a media query. We will cover media queries in Chapter 11.
- `@keyframes`: Defines a set of keyframes for a CSS animation. Animations will be covered in Chapter 9.

How CSS is used

There are several ways to use CSS in an HTML document. They all have the end result: a style sheet that is applied to the document.

Inline styles

Every HTML element supports the `style` attribute. Inline styles are specified as CSS properties in the value of the `style` attribute. An inline style does not contain selectors or curly braces; it is simply a collection of CSS properties. An example of an inline style is shown in Listing 1-4.

Listing 1-4. An element with inline styles

```
<div style="background-color: red;">
  Hello world!
</div>
```

The element in Listing 1-4 will have a red background. The properties specified in an element's inline style will apply to that element only. If there are conflicts in the rules that apply to an element, its inline style always takes precedence. For example, if there was a CSS rule somewhere that made all div elements have a blue background, this element's inline style would override that and give it a red background.

Style blocks

CSS rules can also be specified inside a style sheet within the HTML document itself. This is done by adding CSS rules inside of a `style` element. Style blocks are typically added to the document's `head` element. These are full style sheets with selectors and rules. Listing 1-5 shows an example HTML document containing a `style` element.

Listing 1-5. A style block inside an HTML document

```
<!DOCTYPE html>
<html>
  <head>
    <style>
      div {
        background-color: red;
      }
    </style>
  </head>
  <body>
    <div>Hello world!</div>
  </body>
</html>
```

In the preceding document, all `div` elements will have a red background.

External style sheets

Lastly, CSS rules can also be listed in a style sheet file with a `.css` extension. This style sheet is then referenced in the `head` of the HTML document using a `link` element. In the following example, using the code from Listing 1-6, all `div` elements in the document will have a red background. The linked CSS file is shown in Listing 1-7.

Listing 1-6. An HTML document referencing an external style sheet

```
<!DOCTYPE html>
<html>
  <head>
    <link rel="stylesheet" href="/path/to/file.css">
```

```
  </head>
  <body>
    <div>Hello world!</div>
  </body>
</html>
```

Listing 1-7. A simple external CSS file

```
div {
  background-color: red;
}
```

Browser support

The CSS features discussed in this book are well supported in modern browsers: recent versions of Chrome, Firefox, Edge, and Safari. Some features have limited or no support in older browsers such as Internet Explorer 11. These compatibility issues will be called out where applicable.

Some features are supported but only with vendor-specific prefixes, but this is rare with modern browsers. For example, several years ago, the `@keyframes` rule (which we'll explore in Chapter 9) was still somewhat experimental, so in order to support it, prefixes had to be used. This is shown in Listing 1-8.

Listing 1-8. Using vendor prefixes

```
@keyframes spin {
  from { transform: rotate(0); }
  to { transform: rotate(360deg); }
}

@-moz-keyframes spin {
  from { -moz-transform: rotate(0); }
  to { -moz-transform: rotate(360deg); }
}
```

```
@-webkit-keyframes spin {
  from { -webkit-transform: rotate(0); }
  to { -webkit-transform: rotate(360deg); }
}
```

The main disadvantage of vendor prefixes is that they require duplication, as shown in Listing 1-8. For each block, the same keyframes have to be specified.

More recent browser versions tend to use experimental feature flags rather than vendor prefixes. These are special configuration flags that are exposed in an advanced configuration interface where experimental features can be turned on and off.

If you still need to support older browsers that use prefixes, there are tools such as Autoprefixer which lets you write prefix-free CSS. The tool then parses the CSS and generates a new style sheet containing the duplicated vendor-prefixed rules and properties.

Web resources

There are many resources and references that are useful when using CSS. Here are a few of the best.

CanIUse.com

The website CanIUse.com (https://caniuse.com) is a great resource for finding out the browser support of a given feature. This site maintains a database of up-to-date browser support information for different CSS features. Figure 1-2 shows a screenshot of an example query.

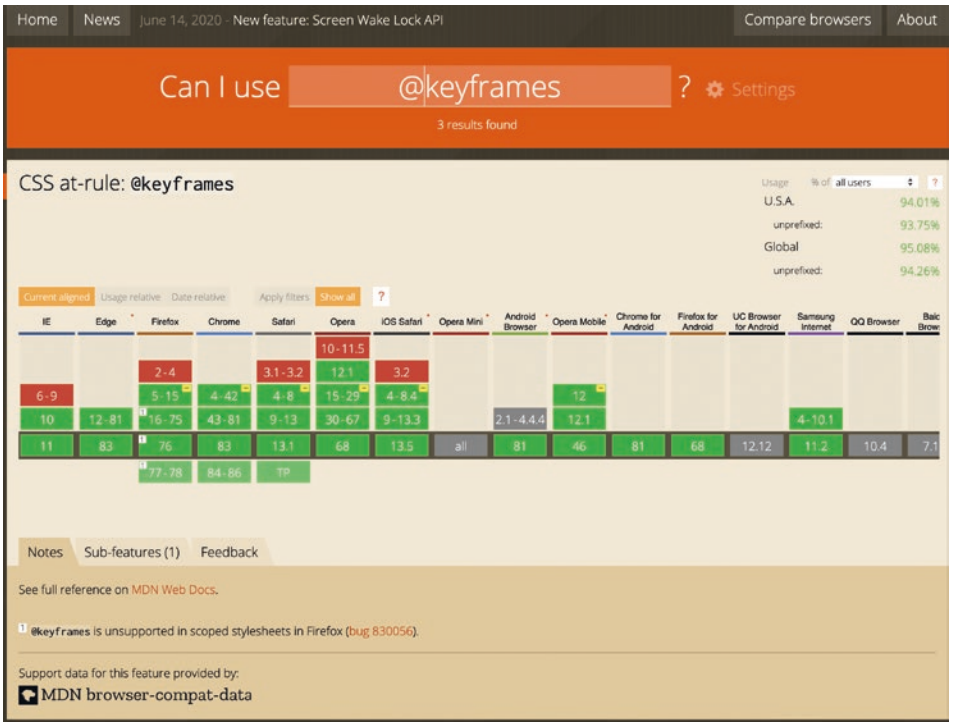

Figure 1-2. *Screenshot of CanIUse.com*

Mozilla Developer Network

Another useful resource is the Mozilla Developer Network (`https://developer.mozilla.org`), or MDN for short. MDN has a complete reference to HTML, CSS, JavaScript, and more. It is an exhaustive reference to all CSS properties.

CSS preprocessors

CSS is not without its limitations and pain points. CSS preprocessors, such as Sass, LESS, and Stylus, provide additional features not found in plain CSS. These tools provide an extended, or even completely different, syntax for writing CSS rules. When you build the application, the preprocessor takes your style sheets and converts them to plain CSS, ready to use in the browser.

We won't cover CSS preprocessors in this book beyond this section, but here is a quick overview if you are interested in using them.

Nested rules

CSS rules don't support nesting. This means that a CSS rule with a selector cannot appear inside another CSS rule. Most preprocessors, however, allow this. For example, the Sass code in Listing 1-9 has nested rules.

Listing 1-9. Nested selectors in Sass

```
.header {
  background-color: red;

  h1 {
    font-size: 24px;
  }
}
```

The inner `h1` selector will only match `h1` elements that are a descendant of an element that matches the outer rule, which would be an element with the class `header`. The equivalent CSS for this nested rule would look like Listing 1-10.

Listing 1-10. The equivalent CSS

```
.header {
  background-color: red;
}

.header h1 {
  font-size: 24px;
}
```

Variables

If you are supporting modern browsers, this is not as compelling of a reason to use a preprocessor, because recent versions of Edge, Firefox, Chrome, and Safari all support native CSS variables. However, if you need to support older browsers such as IE11, this can be a useful feature.

In Sass, for example, variables are declared and referenced starting with a `$` character, as shown in Listing 1-11.

Listing 1-11. Sass variable syntax

```
$header-color: red;

.header {
  background-color: $header-color;
}
```

Mixins

A mixin allows you to write a set of CSS properties and values, then apply that entire set of properties to another CSS rule without having to repeat all the code. If you have to support older browsers that expect vendor prefixes on some properties, this can be useful.

For example, the Flexible Box Layout Module, or flexbox, is supported on all modern browsers. If you need to support older browsers, they may require vendor prefixes. Your CSS might look something like Listing 1-12.

Listing 1-12. Vendor prefixes for flexbox

```
.header {
  display: -webkit-flex;
  display: -ms-flexbox;
  display: flex;
}
```

The given code will need to be repeated for every element using a flexbox layout. This can be simplified by creating a flexbox mixin in Sass, shown in Listing 1-13.

Listing 1-13. Sass flexbox mixin

```
@mixin flexbox {
  display: -webkit-flex;
  display: -ms-flexbox;
  display: flex;
}
```

```
.header {
  @include flexbox;
}
```

Mixins are very useful for cutting down on duplicated code. They can even take arguments to customize the resulting CSS.

How CSS works in the browser

Now, let's take a look at how the browser renders a page with CSS.

The Document Object Model (DOM)

The Document Object Model, or DOM, is a data structure in the browser. It is a tree of objects that represent the elements in the document and their structure and hierarchy. This tree is composed of DOM nodes. The DOM is created by reading the HTML markup, tokenizing it, parsing it, and finally creating the object hierarchy that makes up the DOM.

Consider the example HTML document shown in Listing 1-14.

Listing 1-14. A simple HTML document

```
<html>
  <body>
    <h1>Hello World</h1>
    <div>
      <h2>Subtitle</h2>
      <p>Hello world!</p>
    </div>
  </body>
</html>
```

The corresponding DOM tree is shown in Figure 1-3.

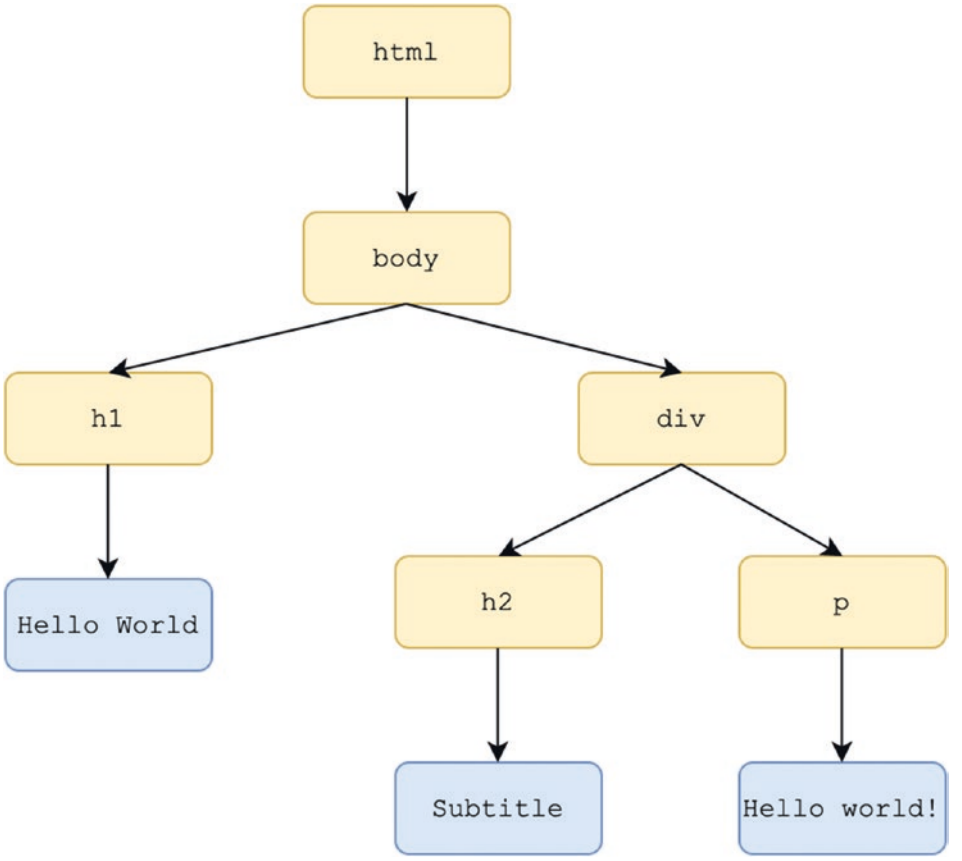

Figure 1-3. *The DOM tree corresponds to the HTML document*

The CSS Object Model (CSSOM)

Similar to the DOM, there is also a CSS Object Model, or CSSOM. This is another tree structure that represents the hierarchy of styles in the document. While they are both tree structures, the CSSOM is a separate structure from the DOM.

Listing 1-15 contains a CSS style sheet meant to be applied to the HTML document in Listing 1-14.

Listing 1-15. CSS style sheet

```
body {
  font-size: 16px;
}
```