sitepoint

CSS MASTER

BY TIFFANY B. BROWN

ORGANIZED, EFFICENT, POWERFUL–CSS DONE RIGHT!

Summary of Contents

6. CSS Transforms

sitepoint

CSS MASTER

BY TIFFANY B. BROW

CSS Master

by Tiffany B. Brown

Product Manager: Simon Mackie
Technical Reviewer: Rachel Andrew
English Editor: Ralph Mason
Cover Designer: Alex Walker

Published by SitePoint Pty. Ltd.

48 Cambridge Street Collingwood
VIC Australia 3066

Web: www.sitepoint.com
Email: business@sitepoint.com

About Tiffany B. Brown

Tiffany B. Brown is a freelance web developer and writer based in Los Angeles, California. She has worked on the web for more than a decade, for a mix of media companies and agencies. Brown is also a co-author of SitePoint's "Jump Start: HTML5." Before founding her consultancy, Webinista, Inc, she was part of the Opera Software Developer Relations & Tools team. Now she offers web development and consulting services to agencies and small design teams.

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.

To Molly H.

Preface

CSS has grown from a language for formatting documents into a robust language for designing web applications. Its syntax is easy to learn, making CSS a great entry point for those new to programming. Indeed, it's often the second language that developers learn, right behind HTML.

The simplicity of CSS is deceptive, however. It belies the complexity of the box model, stacking contexts, specificity, and the cascade. It's tough to develop interfaces that work across a variety of screen sizes and with an assortment of input mechanisms. CSS mastery lies in understanding these concepts and how to mitigate them.

Mastering CSS development also means learning how to work with new tools such as linters, optimizers, and preprocessors. Linters inspect your code for potential trouble spots. Preprocessors make writing and organizing CSS easier. Optimizers improve CSS quality, and reduce the number of bytes delivered to the browser. And of course, there's the question of CSS architecture: which selectors to use, how to modularize files, and how to prevent selector creep.

CSS is also growing in its capabilities. Until now, we've had to use clunky methods such as float, or weighty JavaScript libraries to create the kinds of layouts made possible with the flexbox and multi-column layout modules. Three-dimensional effects were impossible—or required images—before th rise of CSS transforms. What's on the horizon is even more exciting.

It's really a fascinating time to be a front-end developer. My hope is that you'll come away from this book with a better sense of how CSS works and how to write it well.

Who Should Read This Book

This book is for intermediate-level CSS developers, as it assumes a fair amount of experience with HTML and CSS. No time is spent covering the basics of CSS syntax. Coverage of CSS concepts such as the box model and positioning are included to illuminate tricky concepts for the experienced developer. They're not meant as an introduction for beginners. Experience with JavaScript/DOM Scripting is helpful, but not necessary.

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 is 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.cs

```
.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 *(excerp*

```
  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";
}
```

Where existing code is required for context, rather than repeat all of it, … will be displayed:

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

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

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

Tips, Notes, and Warnings

Tip: Hey, You!

Tips provide helpful little pointers.

Note: 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.

Important: Make Sure You Always …

… pay attention to these important points.

Warning: Watch Out!

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

Supplementary Materials

https://www.sitepoint.com/premium/books/csspro1

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

https://github.com/spbooks/csspro1/

The downloadable code archive for this book.

http://community.sitepoint.com/

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 choosing to buy a SitePoint book. Would you like to continue learning? You can now gain unlimited access to ALL SitePoint books and courses plus high-quality books from our selected partners at SitePoint Premium. Enroll now and start learning today!

Selectors

CSS rules are matched to elements with **selectors**. There are a number of ways to do this, and you're probably familiar with most of them. Element type, class name, ID, and attribute selectors are all well-supported and widely used.

The Selectors Level 3 and Level 4 specifications introduced several new selectors. In some cases, these are new variations of existing types. In other cases, they are new features of the language.

In this chapter, we'll look at the current browser landscape for CSS selectors, with a focus on newer selectors. This includes new attribute selectors and combinators, and a range of new pseudo-classes. the section *Choosing Selectors Wisely*, we look at the concept of specificity.

This chapter stops short of being a comprehensive look at all selectors—that could be a book unto itself. Instead, we'll focus on selectors with good browser support that are likely to be useful in your current work. Some material may be old hat, but it's included for context.

Tip: Browser Coverage for Selectors

A comprehensive look at the current state of browser support for selectors can be found at CSS4-Selectors.

Combinators

Combinators are character sequences that express a relationship between the selectors on either side of it. Using a combinator creates what's known as a complex selector. **Complex selectors** can, in some cases, be the most concise way to define styles.

You should be familiar with most of these combinators:

- descendant combinator, or whitespace character
- child combinator, or `>`
- adjacent sibling combinator, or `+`
- general sibling combinator, or `~`

Let's illustrate each of these combinators. We'll use them to add styles to the HTML form shown in Figure 1.1.

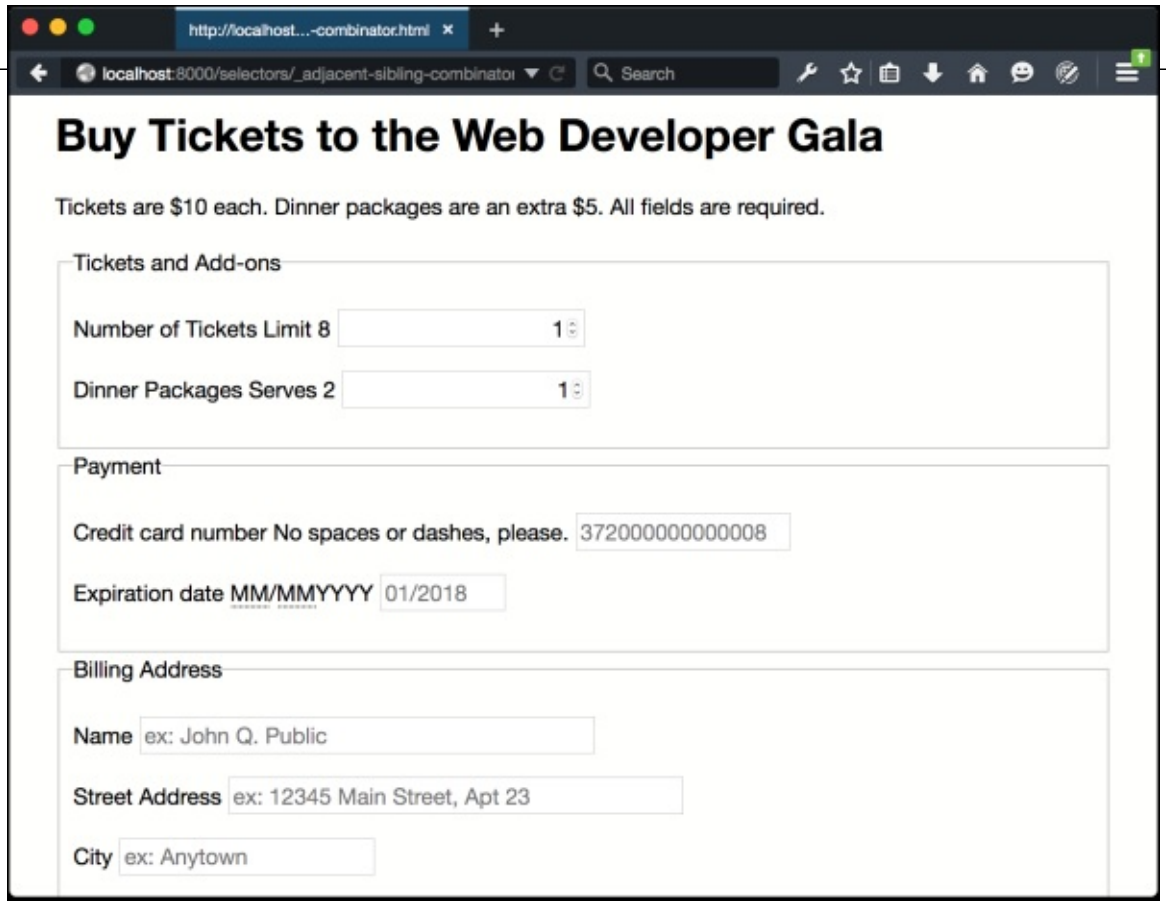

Figure 1.1. Our HTML form that we'll style using combinators

This form was created using the following chunk of HTML:

```
<form method="GET" action="/processor">
    <h1>Buy Tickets to the Web Developer Gala</h1>
    <p>Tickets are $10 each. Dinner packages are an extra $5. All
↵ fields are required.</p>
    <fieldset>
        <legend>Tickets and Add-ons</legend>

        <p>
            <label for="quantity">Number of Tickets</label>
            <span class="help">Limit 8</span>
            <input type="number" value="1" name="quantity"
↵ id="quantity" step="1" min="1" max="8">
    </p>

        <p>
            <label for="quantity">Dinner Packages</label>
            <span class="help">Serves 2</span>
            <input type="number" value="1" name="quantity"
↵ id="quantity" step="1" min="1" max="8">
        </p>

    </fieldset>
    <fieldset>
        <legend>Payment</legend>
        <p>
            <label for="ccn">Credit card number</label>
            <span class="help">No spaces or dashes, please.</span>
```

```
            <input type="text" id="ccn" name="ccn" placeholder=
↵"372000000000008" maxlength="16" size="16">
        </p>
        <p>
            <label for="expiration">Expiration date</label>
            <span class="help"><abbr title="Two-digit month">MM
↵</abbr>/<abbr title="Four-digit Year">MM</abbr>YYYY</span>
            <input type="text" id="expiration" name="expiration"
↵placeholder="01/2018" maxlength="7" size="7">
        </p>

    </fieldset>
    <fieldset>
        <legend>Billing Address</legend>
        <p>
            <label for="name">Name</label>
            <input type="text" id="name" name="name" placeholder=
↵"ex: John Q. Public" size="40">
        </p>
        <p>
            <label for="street_address">Street Address</label>
            <input type="text" id="name" name="name" placeholder=
↵"ex: 12345 Main Street, Apt 23" size="40">
        </p>

        <p>
            <label for="city">City</label>
            <input type="text" id="city" name="city" placeholder=
↵"ex: Anytown">
        </p>

        <p>
            <label for="state">State</label>
            <input type="text" id="state" name="state" placeholder=
↵"CA" maxlength="2" pattern="[A-W]{2}" size="2">
        </p>

        <p>
            <label for="zip">ZIP</label>
            <input type="text" id="zip" name="zip" placeholder=
↵"12345" maxlength="5" pattern="0-9{5}" size="5">
        </p>
    </fieldset>

    <button type="submit">Buy Tickets!</button>
</form>
```

The Descendant Combinator

You’re probably quite familiar with the descendant combinator. It’s been around since the early days of CSS (though it was without a type name until CSS2.1). It’s widely used and widely supported.

The **descendant combinator** is just a whitespace character. It separates the parent selector from its descendant, following the pattern *A B*, where *B* is an element contained by *A*. Let's add some CSS to our markup from above and see how this works:

01-selectors/descendent-combinator.html *(excerp*

```
form h1 {
    color: #009;
}
```

We've just changed the color of our form title, the result of which can be seen in Figure 1.2.

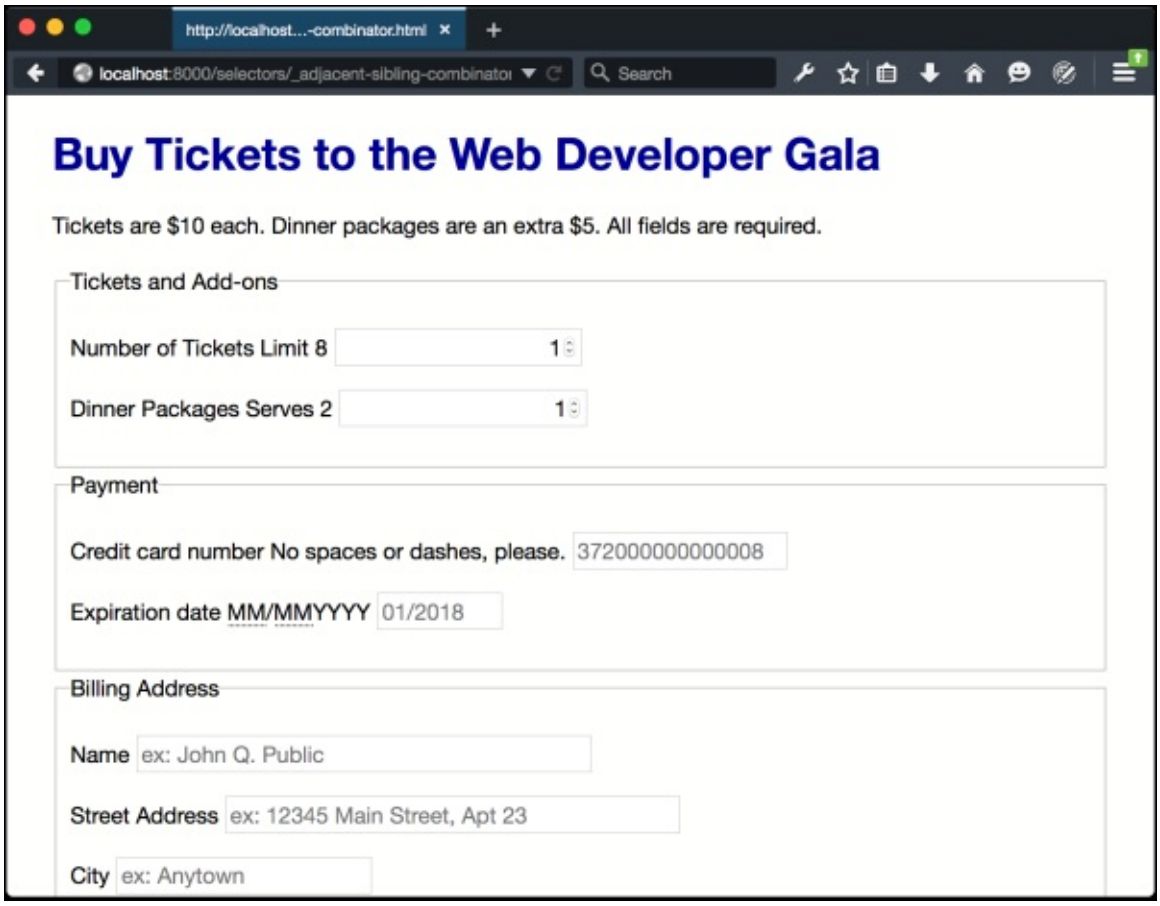

Figure 1.2. The effect of a descendant combinator

Let's add some more CSS, this time to increase the size of our pricing message ("Tickets are $10 each"):

01-selectors/descendent-combinator.html *(excerp*

```
form p {
    font-size: 22px;
}
```

There's a problem with this selector, however, as you can see in Figure 1.3. We've actually increased the size of the text in *all* of our form's paragraphs, which isn't what we want. How can we fix this? Let's try the child combinator.

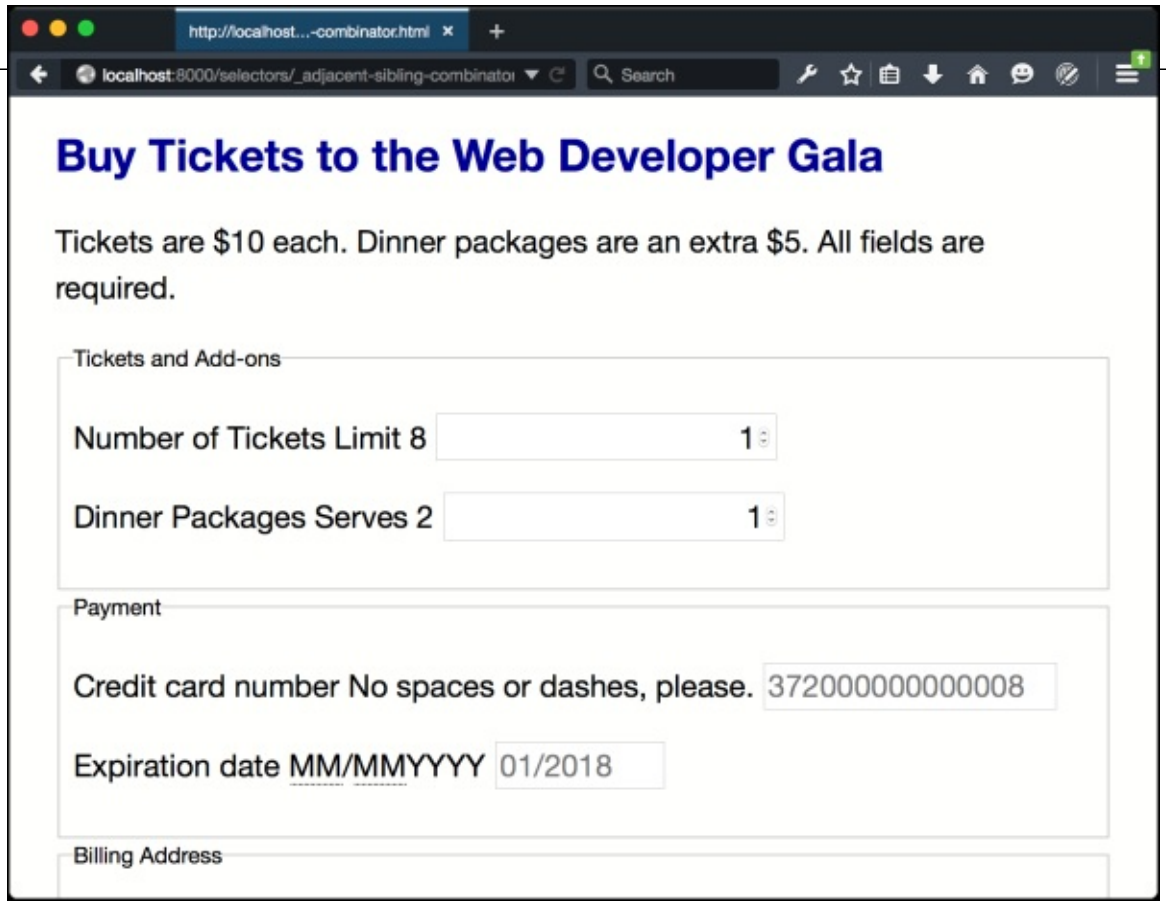

Figure 1.3. Oops! Our selector is too broad

The Child Combinator

In contrast to the descendant combinator, the **child combinator** (>) selects only the *immediate children* of an element. It follows the pattern *A* > *B*, matching any element *B* where *A* is the immedia ancestor.

If elements were people, to use an analogy, the child combinator would match the child of the mothe element. But the descendant combinator would also match her grandchildren, and great-grandchildre Let's modify our previous selector to use the child combinator:

01-selectors/child-combinator.html *(excer*

```
form > p {
    font-size: 22px;
}
```

Now only the direct children of `article` are affected, as shown in Figure 1.4.

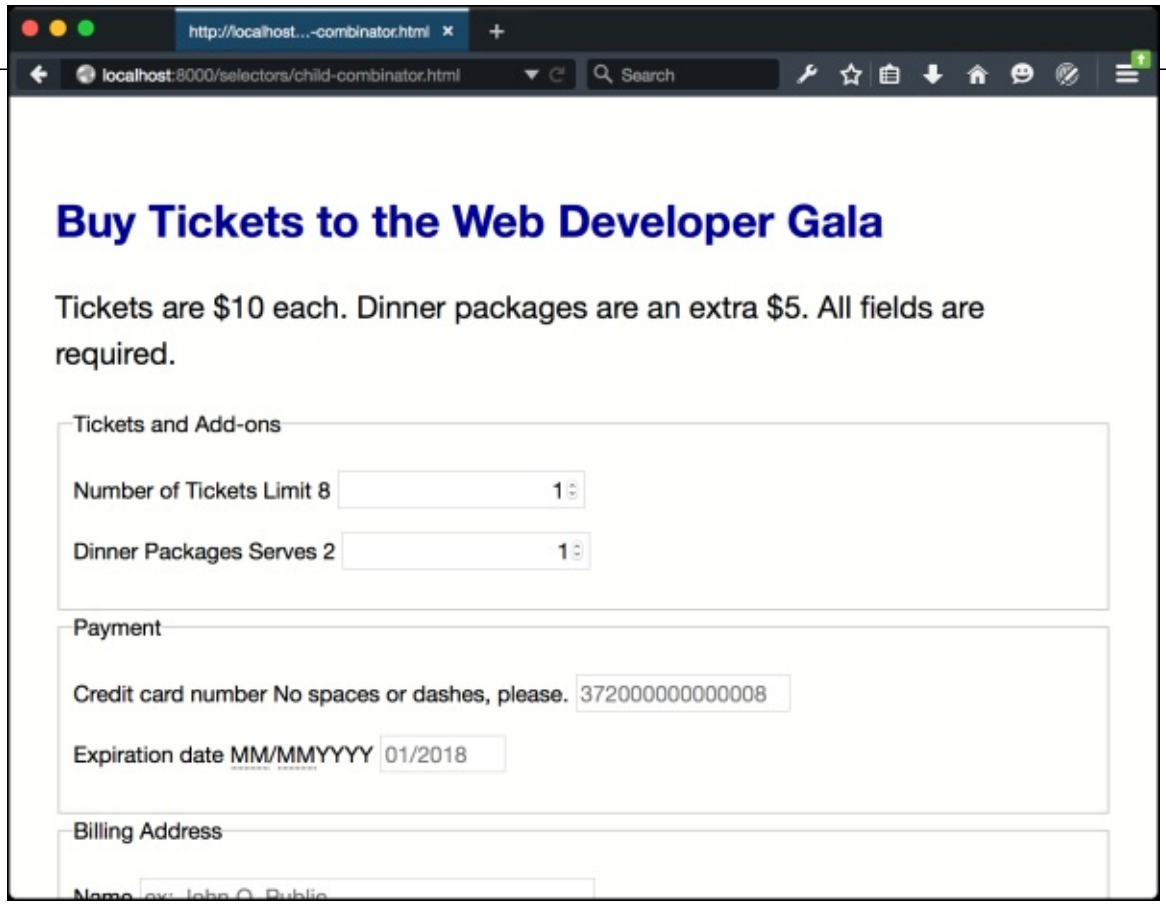

Figure 1.4. The effect of the child combinator

The Adjacent Sibling Combinator

With the **adjacent sibling combinator** (+), we can select elements that follow each other and have th same parent. It follows the pattern *A* + *B*. Styles will be applied to *B* elements that are *immediately* preceded by *A* elements.

Let's go back to our example. Notice that our labels and inputs sit next to each other. That means we can use the adjacent sibling combinator to make them sit on separate lines:

```
label + input {
    display: block;
    clear: both;
}
```

You can see the results in Figure 1.5.

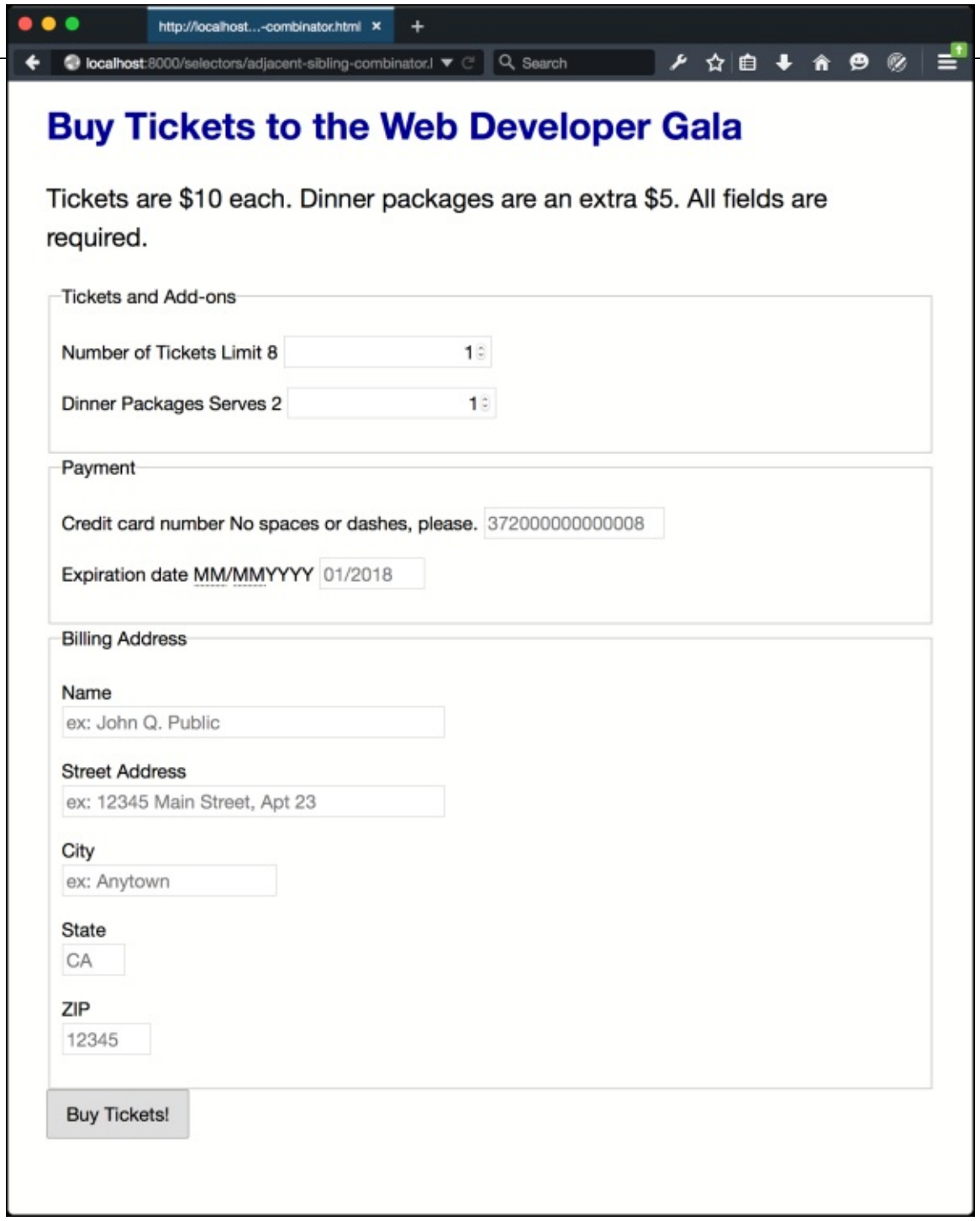

Figure 1.5. Adjacent combinator to the rescue

Let’s look at another example that combines the universal selector (*) with a type selector:

01-selectors/adjacent-sibling-combinator.html *(excerp*

```
* + fieldset {
    margin: 5em 0;
}
```

This example adds a `5em` margin to the top and bottom of every `fieldset` element, shown in Figure 1.6. Since we’re using the universal selector, there’s no need to worry about whether the previous element is another `fieldset` or `p` element.