In this chapter, I’ll cover selectors in some detail. Understanding selectors is essential to writing good CSS (Cascading Style Sheets). A selector offers the ability to apply a set of styles to all elements of the same type. For example, say you wanted to change the color of all your headers (not just one) from green to orange. You can accomplish this with a single line of CSS code. Imagine the alternative—going through the painstaking process of reviewing the markup and changing the color of each header element individually.

I’ll start by introducing the different kinds of selectors before diving into each exercise. Then, I’ll discuss how style conflict resolution works, how you can use the `!important` declaration to troubleshoot style conflicts, and how inheritance impacts your final document. These are important concepts, and I encourage you to study this chapter thoroughly and refer to it as necessary as you proceed through the rest of the book.
Understanding Selectors

Several kinds of selectors exist: element selectors and ID selectors are two of the most common.

Element selectors
An XHTML (eXtensible HTML) document consists of elements and the contents of those elements, so it makes sense you would at least be able to apply styles to those elements. An element selector is simply a selector that applies to a type of element. While working through Chapter 2, “Understanding CSS,” you worked with a number of element selectors.

For example, suppose you wanted to style all paragraphs consistently. The rule to do that would look like this:

```
p {font-family: Arial, Verdana, sans-serif;}
```

The selector `p` just means “apply the declarations in this rule to any `p` element.” Whether the document has one `p` element or one thousand, every one of them will be matched by the element selector `p`. If the document doesn’t have any `p` elements, then the selector won’t match any elements, and the declarations won’t be applied to anything.

Another example would be if you wanted to draw a 3-pixel red border around the outside of all your tables. Again, you would use an element selector:

```
table {border: 3px solid red;}
```

Because they are so basic, element selectors are sometimes referred to as “simple” selectors. This distinguishes them from the more complex selectors you’ll see in the next several sections of this chapter.

ID selectors
Before discussing how to write an ID selector, it’s important to quickly cover what exactly an ID selector matches. XHTML allows any element (except for the `html` element) to be given an `id` attribute, such as this:

```
<div id="example">Here's an example.</div>
```

The value of an element’s `id` attribute, which is usually referred to as the element’s ID, is what’s matched by an ID selector. To match the previous bit of markup, you would write this selector:

```
#example
```

Simple enough. It’s just the ID value you want to match preceded by an octothorpe (#), with no space between the two.

Technically, `#example` means “any element whose `id` attribute has the value `example`.” Thus, the selector would still match the previous markup even if you changed the `div` to a `p`, an `h1`, or anything else. As long as the ID remained, the match would still occur.

If you have a situation where you want to restrict the selector to match only a certain type of element with the ID, then you add the element name before the octothorpe. For example:

```
div#example
```

Octo-what?
Yes, the # symbol is really called an octothorpe. Other common names are the number sign, the pound sign (but the British find this confusing and annoying), the hash symbol (which can confuse the DEA), and the tic-tac-toe symbol (which makes a great deal of sense but isn’t easy to type more than once or twice).

In case you’re curious, the word octothorpe is derived from octo, meaning “eight” in Latin and referring to the eight points that stick out (two on each side), and from thorpe in honor of the famous early 20th-century athlete Jim Thorpe. No kidding!
This now means “any \texttt{div} element whose \texttt{id} attribute has the value \texttt{example}.” If the ID value appears on a different element type, then the selector won’t match it.

The most important fact to know about IDs in XHTML is this: In any given XHTML document, every ID value must be unique. In other words, if you have an element in your document with an ID of \texttt{example}, then no other element anywhere else in that document can have an ID of \texttt{example}.

For this reason, IDs are usually employed to label sections of a page layout. Here’s an example document skeleton showing how you can use IDs in this way:

\begin{verbatim}
<\texttt{div id="masthead">}_</\texttt{div}>
<\texttt{div id="main">}
  <\texttt{div id="content">}_</\texttt{div}>
  <\texttt{div id="sidebar">}_</\texttt{div}>
</\texttt{div}>
<\texttt{div id="footer">}_</\texttt{div}>
\end{verbatim}

Because each area of a layout is going to be unique—most pages don’t have more than one footer, after all—IDs are well suited to this kind of labeling.

Think of an ID like your Social Security number (or the local equivalent). It’s unique to you, and any information pertaining to that number pertains only to you. Thus, if your number is 123-456-7890 and the government announces that the next leader of the free world will be the person with the number 123-456-7890, that’s just like an ID selection. Only one person gets the leadership, just like the unique ID gets the style.

\section*{Class selectors}

Like IDs, you can use classes to “label” elements in a document. Classes are different from IDs in two ways. The primary is that you must add a class to an element via the \texttt{class} attribute, like this:

\begin{verbatim}
<\texttt{li class="first urgent">}<a href="aboutus-ourgoal.html">about us</a></\texttt{li}>
\end{verbatim}

Words appearing in an element’s \texttt{class} attribute are usually referred to simply as the element’s \texttt{class}. As you can see in this example, you can actually have multiple words in an element’s \texttt{class} attributes and address them separately in the style sheet. These words are what are matched by class selectors. To match the previous bit of markup, you would write this selector:

\begin{verbatim}
.first
\end{verbatim}

It’s just the class value you want to match preceded by a period (\texttt{.}), with no space between the two.

Technically, \texttt{.first} means “any element whose \texttt{class} attribute contains the word \texttt{first} in its value.” The actual element type doesn’t matter in this case. If you wanted to restrict the match to just \texttt{p} elements with a class of \texttt{first}, then you’d write this:

\begin{verbatim}
p.first
\end{verbatim}

\section*{VIDEO: id_selectors.mov}

To learn more about ID selectors, check out \texttt{id_selectors.mov} in the \texttt{videos} folder on the CSS HOT CD-ROM.

\section*{TIP: Maintaining ID Uniqueness}

Although it’s clear that having repeated IDs within a document is not permitted, most browsers will let you get away with it. Despite this, you should never repeat IDs within a document. It’s a bad habit and one that browsers might one day punish. It’s also unnecessary, since you can have multiple elements share a common “label” in another way, as you’ll see later in the chapter.
Then add your rules. You could add another class selector, .urgent, and add a different set of rules. It doesn’t matter in which order you list them. (The order styles are in can matter, but I will discuss this later in this chapter.) This isn’t a common need—sometimes called multiclassing in a document—but it can come in handy. A classic example is when you’re creating a portal page, and you have form fields wrapped in a div and each div has its own class. For all the boxes that require user input, such as a box to enter a ZIP code to find the local weather forecast, you could have a class of user allowed, which would allow you to style these particular form fields differently. That’s a situation where multiclassing would come in handy.

Let’s spend some time discussing a few best practices when it comes to class selectors. It can be tempting to “overclass” on the off chance you may need them someday. In the following example, these list items contain links and navigations links, all the list items contain classes of nav links, and the a elements inside them also contain class links.

```html
<div id="masthead">
  <ol id="navlinks">
    <li class="navlink">a class="navlink" href="aboutus-ougool.html">about us</li>
    <li class="navlink">a class="navlink" href="brewing.html">brewing</li>
    <li class="navlink">a class="navlink" href="drinking-instructions.html">drinking</li>
    <li class="navlink">a class="navlink" href="#">products</li>
    <li class="navlink">a class="navlink" href="#">contact us</li>
  </ol>
</div>
```

This is really inefficient. You don’t need to have all this class linking. You could remove all this and just have the plain elements. How are you supposed to select those? That’s coming up later in the chapter, but the short answer is that you use the primary ID on the element <ol id="navlinks"> that encloses all the other elements. It’s an understandable urge to class everything, but it’s an urge that needs to be

### TIP: Maintaining Case Consistency

XHTML defines ID values to be case-sensitive; that is, the capitalization matters. Thus, the ID Example is different than example, and both are different from EXAMPLE. Most browsers do enforce this rule, so always make sure to keep your capitalization consistent. If your XHTML has id="searchBox", then the CSS has to say #searchBox, not #searchbox.
restrained. You risk bloating your document, making it that much harder to maintain. As shown in the illustration here, it's much cleaner.

Descendant Selectors

In this section I'll discuss descendent selectors, but first I'll talk about document structure. I don't exactly mean the order in which you write your elements, but instead the structure that's created by those elements. For example, look at this very (very!) simple XHTML document:

```xml
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title>Simple Document</title>
</head>
<body>
  <h1>Simple Document Title</h1>
  <p>Simple paragraph.</p>
  <ul>
    <li>Simple list item.</li>
    <li>Simple list item.</li>
    <li><em>Simple</em> list item.</li>
    <li>Simple list item.</li>
  </ul>
</body>
</html>
```

The history of tea extends so far into the past that its very origins are shrouded by legend. It is said that Emperor Shen Nung, who ruled China in 2737 BC, used to enjoy relaxing in his garden with a cup of hot water. It was
Every document contains elements—starting with the `html` element. All these elements are connected in parent-child relationships. Most elements have either a parent or a child, or both. For example, the `body` and `head` are the children of the `html` element. Now, you could abstractly represent that document in a lot of ways, but the one that’s most useful in CSS terms is a tree diagram.

Hierarchical data structures in the computer science world are typically represented as a “tree”—it’s the kind of tree that grows downward, more like a root system, but still a tree. The illustration shown here presents a tree view of an XHTML document. At the root of everything is the `html` element. Descending from that root are all the other elements of the document. To one side, you see the `head` and to the other, the `body`. Descending from the `body` are all the elements that eventually get drawn in a browser.

When discussing relationships in document trees, I’ll use four terms often:

**Ancestor:** An element that is on a path between an element and the root element in the document tree.

**Descendant:** The inverse of ancestor; an element on a path that descends from a given element.

**Parent:** A direct ancestor; that is, the element directly above an element in the document tree.

**Child:** A direct descendant; that is, an element directly below an element in the document tree.

Did you notice how I have used the word *descending* throughout this section? I used it intentionally, because it’s key to how you create descendant selectors.
Suppose you want to make the summary paragraphs red. Here’s how:

```css
div p {color: red;}
```

Yes, there’s a space there, and it’s on purpose. This is a descendant selector: It selects elements based on their relationship to a parent element. The way to read this is actually backward: “Select any `p` element that is a descendant of a `div` element.” The space between `div` and `p` is what indicates the relationship “is a descendant of.”

You can figure out whether an element descends from another element by tracing up the document tree. Consider all the paragraphs in your tree.

Put your finger on any `p` element, and trace upward along the lines until you reach the root. If your finger passes over a `div` along the way, then it’s an ancestor of the `p`, and by definition that means the `p` is a descendant of the `div`.

Similarly, if you wanted to put all the `em` (emphasis) elements that are descendants of the `div` in bold, you’d write the following:

```css
div em {font-weight: bold;}
```

Again, trace up the tree from every `em` element, and if there’s a `div` between it and the document tree’s root, then the selector will match it.
EXERCISE

1 Using ID Selectors

In this exercise, you’ll work with ID selectors. Remember, you assign IDs in the markup, and the most important point to bear in mind when it comes to ID selectors is that you can have only one instance of a given ID in a document.

1 Copy the 03_cascade folder from the CSS HOT CD-ROM to your desktop. Double-click the 03_cascade folder to open it so you see the exercise files inside.

2 In a browser, open ex0301.html from the 03_cascade folder you copied to your desktop, as shown in the illustration.
Open ex0301.html with your text or HTML (HyperText Markup Language) editor. Near the beginning of the file, find the following block of code, as shown in the illustration:

```
<style type="text/css">

</style>
```

This is the embedded style sheet you’ll be working with in this exercise. The first step is to add a rule that styles the masthead.
4. After the `<style type="text/css">` line, type the following:

```css
#masthead {background: green;}
```

The rule you just added has a selector that says “select any element whose `id` attribute has a value of `masthead`.” Because the XHTML source has only one such element (remember that IDs have to be unique!), you know that the green background will show up there only.

5. Save `ex0301.html`, switch to your browser, and reload `ex0301.html`.

There’s the green. It’s filling the background of the masthead, just as planned.
Return to your text or HTML editor. Add the following rules to the style sheet:

```css
#navlinks {background: silver;}
#sidebar {border: 1px solid gray;}
```

Here, you've added two rules, each of which selects a different ID value. As before, the actual element types don’t really matter, because ID values are unique. Regardless of whether the element with an ID of `navlinks` is a `div` or an `ul` or an `ol` or a `table`, `#navlinks` will select it. (As it happens, it’s an `ol`.)
Save `ex0301.html`, switch to your browser, and reload `ex0301.html`.

The navigation area now has a silver background, and the sidebar has a border around it. This is kind of nice, in a bold, minimalist fashion, don’t you think? Now, just to prove a point, you’ll add some element names.
Return to your text or HTML editor. Edit the style sheet as follows:

```css
#masthead {background: green;}
ol#navlinks {background: silver;}
p#sidebar {border: 1px solid gray;}
```

Here you’ve added some element names. Now, instead of selecting any element with an ID of `navlinks`, you’re selecting any `ol` element with an ID of `navlinks`; similarly, instead of selecting any element with an ID of `sidebar`, you’re selecting any `p` element with that ID.

**WARNING:**

**Avoiding Trouble with Spaces**

A fairly common mistake is to accidentally put a space between the element and the ID descriptor. Doing so would make the last rule in Step 8 look like this:

```css
ol #navlinks {background: silver;}
```

See that space there before the octothorpe? Its presence *completely* changes the meaning of the selector. Return to your browser, and reload the document—there’s no silver background. The selector no longer refers to an `ol` element with an ID of `navlinks`. Instead, it now will select any element with an ID of `navlinks` that’s a descendant of an `ol` element.

And what does being a descendant of another element mean? You’ll find out all about it later in the section “Understanding Descendant Selectors.”
Save and close ex0301.html, switch to your browser, and reload ex0301.html.

The silver background stays in place, but the sidebar’s border disappears.

That’s because the element with an ID of sidebar is not a p; it’s a div, so the selector fails to match it, and the border style just doesn’t get added. The silver background stayed because the element with an ID of navlinks is indeed an ol, so ol#navlinks matches it. If you wanted to include the sidebar, you would include an element of div#sidebar or just #sidebar.

Congratulations! You finished the first exercise on selectors. In the next exercise, you’ll start working with class selectors.

**NOTE:**

Learning All About Capitalization

If you were to change this selector to ol#NavLinks {background: silver;} and return to the Web page in the browser, you still won’t see a silver background. Why? Because ID names are case-sensitive. In the markup, navlinks is all lowercase. To make sure the selector matches the element, you need to change the markup either in the XHTML or in the selector. It doesn’t matter what kind of capitalization you use with your IDs—all lowercase, all uppercase, or mixed—as long as it is consistent throughout your documents. Some older browsers ignored these inconsistencies, notably earlier versions of Internet Explorer for Windows, but that led to some sloppiness and the need for further corrections down the road. Do not do this. Nowadays, if your capitalization is not consistent, your styles will not be applied to the resulting Web page.
EXERCISE 2 Using Class Selectors

This exercise will allow you to explore the value of class selectors firsthand. Using a class selector, you will be able to apply style changes to one or many unrelated elements in a document, provided they are assigned to a class.

In a browser, open ex0302.html from the 03_cascade folder you copied to your desktop, as shown in the illustration.
Open `ex0302.html` with your text or HTML editor. Near the beginning of the file, find the following block of code:

```html
<style type="text/css">
  /* This is the embedded style sheet where you'll be working in this exercise. The first step is to add a rule that styles based on a class value. */
</style>
```

This is the embedded style sheet where you’ll be working in this exercise. The first step is to add a rule that styles based on a class value.

### WARNING: Avoiding More Trouble with Spaces

As with IDs, a fairly common mistake is to accidentally put a space between the element and the class descriptor. When you’re selecting specific kinds of elements based on their class names, you need to have the element name and class name snuggled right up against that period. Otherwise, the rule will fail.

3 After the `<style type="text/css">` line, type the following:

```
li.first {background: yellow;}
```
4 Save ex0302.html, switch to your browser, and reload ex0302.html. A yellow background appears behind the “about us” link in the nav bar. The rest of the page background has not changed, because the “about us” link is the only li element with a class attribute that contained the word first.

5 Return to your text or HTML editor. Edit the first rule in the style sheet as follows:

```
.first {background: yellow;}
```
Now you’ve removed the element restriction. Now any element with a class of `first` will get a yellow background.

6. Save `ex0302.html`, switch to your browser, and reload `ex0302.html`.

The first paragraph on the page now has a yellow background—and if you were to scroll down in your XHTML document, it would have a class of `first`.

**NOTE:**

**ID Versus Class Selectors**

Earlier, I mentioned there were two primary differences between ID and class selectors. The first was that classes had to be assigned in the markup. Well, remember how IDs have to be unique within a document? Classes do not. This is the second critical difference that exists between IDs and classes. You can apply a given class to as many or as few elements as you want. In fact, you could give every single element in a document the same class. It would be a completely pointless exercise, but it’s certainly possible.

Think of a class like your name. Many people throughout the world share that name with you, right? So if your name is Joe Smith, and if the results of a lottery meant that anyone with the name Joe Smith gets 100 dollars, that’s like a class selection. Everyone in the class Joe Smith is selected (regardless of nationality, ethnicity, age, and so on) and gets a hundred dollars. That’s just like selecting all the elements (regardless of their type) based on a class and applying styles to them all.
7 Return to your text or HTML editor. Insert a new line and add the following rule to your style sheet:

\[
\text{.tue} \{ \text{background: orange;} \}
\]

The rule you just added puts an orange background on elements whose \textit{class} attribute values contain the word \textit{tue}. This is short for Tuesday, and you may guess that this will have an effect on the calendar.

8 Save \texttt{ex0302.html}, switch to your browser, and reload \texttt{ex0302.html}.

You were right! Because all the cells in the calendar table are classed according to the day of the week, all the Tuesday cells—including the “Tue” header at the top—are selected by . \texttt{tue}. 
9. Return to your text or HTML editor. Edit the second rule in the style sheet as follows:

```css
td.tue {background: orange;}
```

Having the orange on the “Tue” header looked kind of ugly, so here you restricted the selector so that it matches only `td` elements with a class of `tue`. Any other elements, including `th` elements, will not be selected no matter what class they have.

10. Save and close `ex0302.html`, switch to your browser, and reload `ex0302.html`.

The orange is confined to the `td` cells.

In this exercise, you learned how to apply single, simple class selectors. In the next exercise, you’ll learn how to use a grouped selector to apply the same styles to a series of elements.
EXERCISE

3 Using Grouped Selectors

So far, you’ve seen only single selectors. In every rule you’ve written, a single selector went with the declaration block. That’s fine for simple rules, but what if you want to apply the same set of styles to a set of elements? You could class them all the same, but that would get messy. It’s much easier to use a grouped selector, which is a comma-separated list of simple selectors in the same rule.

To pick a basic example, suppose you wanted to set a consistent indentation for all three kinds of XHTML lists: unordered, ordered, and definition. You would write this like so:

```css
ul, ol, dl {margin-left: 2em; padding-left: 0;}
```

See how that works? The selector means “select all `ul`, all `ol`, and all `dl` elements.” This will cause the declarations to be applied to all those element types.

You can group any kind of selector, not just element selectors. If you had to set a color for links with a class of `more`, all tables on your page, and a paragraph with an ID of `today`, you’d write the following:

```css
a.more, table, p#today
```

In this exercise, you’ll apply a style to a group of header elements. This is particularly useful when working with text-heavy documents.

1 In a browser, open `ex0303.html` from the `03_cascade` folder you copied to your desktop. It should appear as shown in the illustration.
2. Open ex0303.html with your text or HTML editor. Near the beginning of the file, find the following block of code:

```html
<style type="text/css">
</style>
```

This is the embedded style sheet where you’ll be working for this exercise.

3. After the `<style type="text/css">` line, type the following:

```css
h1 {text-transform: uppercase;}
h2 {text-transform: uppercase;}
h3 {text-transform: uppercase;}
h4 {text-transform: uppercase;}
h5 {text-transform: uppercase;}
h6 {text-transform: uppercase;}
```

You just added a rule that alters the heading elements so their text is all in uppercase letters, regardless of the capitalization of the text in the XHTML source.

4. Save ex0303.html, switch to your browser, and reload ex0303.html.

You’ll notice the headers, from “About Tea: History” to “Tea of the Day,” are now uppercase.
After the `<style type="text/css">` line, delete the header declaration, and type the following instead:

```
h1, h2, h3, h4, h5, h6 {text-transform: uppercase;}
```

This grouped selector radically simplifies the document and, as you will see in the next step, is rendered the same by your browser.
6 Save ex0303.html, switch to your browser, and reload ex0303.html.

7 Return to your text or HTML editor. Add the following rule to the style sheet:

```
th, td {background: #A0C63A;}
```

The rule you've just created selects all `th` (table header) and `td` (table data, more usually called a table cell) elements and fills in a green background color.
8 Save and close ex0303.html, switch to your browser, and reload ex0303.html.

Perhaps that wasn’t the best choice of a background color for the calendar, but it nicely illustrates that tables are often composed of two different kinds of cells, and styling both consistently is a snap using grouped selectors.

In this exercise, you learned how to simplify your style sheet by using grouped selectors:

```css
h1, h2, h3, h4, h5, h6 {text-transform: uppercase;}
```

As far as a browser is concerned, you typed the following:

```css
h1 {text-transform: uppercase;}

h2 {text-transform: uppercase;}

h3 {text-transform: uppercase;}

h4 {text-transform: uppercase;}

h5 {text-transform: uppercase;}

h6 {text-transform: uppercase;}
```

To the browser, it doesn’t matter which of those two approaches you take. The six-rule sequence is treated exactly the same as the one-rule grouped selector. And who wants to do all that typing?
NOTE: Recognizing That Blocks Are Groups, Too

Another instance of grouping in CSS exists, and that’s the declaration block, which I covered in Chapter 2, “Understanding CSS.” Consider this simple little rule:

```css
p {font-style: italic; color: red; margin: 1em;}
```

To a browser, this is exactly like typing the following:

```css
p {font-style: italic;}
p {color: red;}
p {margin: 1em;}
```

Using grouped selectors with a declaration block reaps you even more savings. For example:

```css
h1, h2 {font-weight: bold; color: navy;}
```

That’s equivalent to the following:

```css
h1 {font-weight: bold; }
h1 {color: navy;}
h2 {font-weight: bold;}
h2 {color: navy;}
```

You can see why I’m not using more complicated examples. The expanded lists would go on for pages!
EXERCISE 4 Using Descendant Selectors

In this exercise, you'll use descendant selectors to italicize elements of a Web page and then start to limit exactly where the italics show up on the Web page.

1. In a browser, open ex0304.html from the 03_cascade folder you copied to your desktop, as shown in the illustration here.

A quick glance at the page reveals that none of the text is italicized.
2 Open ex0304.html with your text or HTML editor. Near the beginning of the file, type the following after `<style type="text/css">`:

```
p {font-style: italic;}
```

3 Save ex0304.html, switch to your browser, and reload ex0304.html.

Notice the text is now italic.
4 Return to ex0304.html in your text or HTML editor, and edit the first line of the style sheet (the \( \text{p} \{ \text{font-style: italic;} \} \) line) as follows:

\[
\text{#masthead p} \{ \text{font-style: italic;} \}
\]

One of the ancestors of the \( \text{p} \) element enclosing the date is the \( \text{div} \) that has an ID of \text{masthead}. By adding that ID to the selector, you’re saying that the declaration block of this rule should be applied to any \( \text{p} \) element that \text{descends} from (that is structurally nested inside of) any element with an ID of \text{masthead}.

5 Save ex0304.html, switch to your browser, and reload ex0304.html.

Now, the only italic text is the date.

6 Return to your text or HTML editor. Edit the first line of the style sheet (the \( \text{#masthead p} \{ \text{font-style: italic;} \} \) line) as follows:

\[
\text{#sidebar p} \{ \text{font-style: italic;} \}
\]
You just changed the descendant relationship. Now, only `p` elements that descend from the element with an ID of `sidebar` will be selected.

7 Save `ex0304.html`, switch to your browser, and reload `ex0304.html`.

The date is no longer italic, since it isn’t in the sidebar. Instead, the sidebar paragraph in the “tea of the day” section is being italicized. But what if you want to italicize both of them?

8 Return to your text or HTML editor. Edit the first line of the style sheet (the `#sidebar p {font-style: italic;}` line) as follows:

```css
#masthead p, #sidebar p {font-style: italic;}
```

Here you created a grouped selector of two descendant selectors. This comes out to mean “select any `p` element that descends from an element with an ID of `masthead` and select any `p` element that descends from an element with an ID of `sidebar`.”
In this exercise, thanks to the grouped descendant selectors, both the date and the sidebar paragraphs are italicized. The paragraphs in the main content area of the page are not affected, since none of them is a descendant of the masthead or sidebar element. In the next exercise, I’ll discuss specificity and style conflict resolution, but first I’ll discuss the origins of styles.

Looking for the Sources of Styles

Where does style come from? A combination of three sources. The first, and the one that probably interests you the most, is from the author of the Web page. The second comes from user styles, or reader styles. The person who is viewing your page may have a style sheet associated with the browser they are using and may have the font appear larger on the pages they visit or have all the links underlined. This is a particularly useful feature for users with vision problems, color blindness, and so on. User style sheets are very rare; most people don’t even realize they’re there, but keep it in mind. The third source is common—user agent styles, otherwise known as browser styles. This is a style behavior built into the browser.

The Sources of Style

User Agent styles

User (reader) styles

Author styles
A Really Unstyled Page

You might not think much style is built into a browser, but let's take an “unstyled” page as an example, like the one shown in the illustration here.

Nothing is being applied here by default, except for the hyperlink treatment, the default colors, and the default fonts that come from the browser preferences, which are turned into style rules. But as you can see, paragraphs don’t look like paragraphs, and headings don’t look like headings. The text is all running together. If you look below the last image, that’s table markup. But the browser is displaying the text as if there were no elements at all. This is really unstyled—except for the browser preference settings, which are the most basic (and third and final) source of styles to keep in mind when you’re styling your document.

When the style sheets conflict, and inevitably they do—the user style sheet and the author style sheet, for example—the browser has to pick one of the two. This is usually resolved by the cascade. The first step is to consider the sources—author styles usually trump user styles, but the second step in the resolution process is specificity, which I will discuss in the next exercise.

**VIDEO:** style_sources.mov

To find out more about the sources of styles and how they are resolved in the cascade, check out style_sources.mov in the videos folder on the CSS HOT CD-ROM.
EXERCISE

5 Resolving Style Conflicts with Specificity

In this exercise, I’ll discuss specificity, the heart of conflict resolution in CSS. If you have two rules that absolutely conflict in a single document, specificity is CSS’s way of assigning priority to the rules.

1 In your text or HTML editor, open `ex0305.html` from the `03_cascade` folder you copied to your desktop. Near the beginning of the file, find the following block of code:

```html
<style type="text/css">
</style>
```

2 Type the following after the leading `<style>` tag:

```css
li {background: red;}
li {background: green;}
```

This presents an obvious conflict. Which color is going to appear? You can’t have both. It would be nice if the browser would resolve them, maybe with stripes or candy-cane style, but it doesn’t.

3 Save `ex0305.html`, return to your browser, and reload the page.

The answer is: green! Why? Because green was the last declaration.

4 Return to your text or HTML editor, and modify the `li {background: red;}` line as follows:

```css
body li {background: red;}
```
Save ex0305.html, return to your browser, and reload the page.

The background is now red. The reason for this is specificity. According to the CSS specification, an element selector has a specificity of 1, class selectors have a specificity of 10, and ID selectors have a specificity of 100. When you add descriptors or other attributes to the selector, the specificity increases. The list item li has one element descriptor, li, so the specificity of that is 0,0,1,0, being that there is one element descriptor. The specificity of the body li is 0,0,0,2, because it has two element descriptors. When rules are in conflict, the rule with the selector of the highest specificity "wins." The order of the items no longer matters, when the specificity differs.

5 Save ex0305.html, return to your browser, and reload the page.

Return to your text or HTML editor, and type the following under the last list item (li):

li.first {background: yellow;}

7 Save ex0305.html, return to your browser, and reload the page.

You discover the list item with the class of first is yellow. That's not because it's last but because it has a higher specificity. It has one element descriptor and one class descriptor. The specificity is 0,0,1,0 for the element descriptor and 0,0,1,0 for the class descriptor. 0,0,1,0 is a higher specificity than 0,0,0,2 (for body li). The reason why only the first list item is yellow is because it's the only element that has a class of first.
8  Return to your text or HTML editor, and type the following after the `li.first` list item:

```
li.last {background: yellow;}
```

9  Save `ex0305.html`, return to your browser, and reload the page.
Now, both the first and last list items have yellow backgrounds.

10 Return to your text or HTML editor, and type the following after the `li.first` list item:

```
#navlinks li {background: fuchsia;}
```

So, the specificity of this will be 01 for the ID (navlinks) and 1 for the element (li).

11 Save `ex0305.html`, return to your browser, and reload the page.
All the list items are now fuchsia. The specificity of the `#navlinks` selector outweighs all the others.

12 Return to your text or HTML editor, and modify the `li` list item as follows:

```
li {background: green; font-style: italic;}
```
Save ex0305.html, return to your browser, and reload the page.

The background of the list items is still fuchsia, but the fonts are now all italic. This is because none of the rules has any conflicting font style declarations with higher specificities. The specificity resolution kicks in only when conflicts exist. But you have one more way to affect specificity: using the inline style attribute, which you’ll test in the next step.

14 Return to your text editor, and scroll down through the HTML to the following:

```html
<li class="first"><a href="aboutus-ougoal.html">aboutus</a></li>
```
Modify it as follows, as shown in the illustration:

```html
<li class="first" style="background: cyan;"><a href="aboutus-ougoal.html">aboutus</a></li>
```

The specificity of this style attribute value is 1,0,0,0. That's higher than 0,1,0,1, the specificity of the `#navlinks li` rule.

15 Save and close ex0305.html, return to your browser, and reload the page.

The element with the in-style attribute wins. The first list item background is cyan. The specificity of an inline style attribute always overrides the author style sheet, because these rules start with 1, and rules in the style sheet begin with 0. This is not recommended, because it defeats the purpose of using style sheets at all; however, it is there if you need it.

In this exercise, you explored specificity and how specificity helps resolve declaration conflicts. It is not the most intuitive concept but is important to understanding CSS. In the next exercise, you'll learn how using the `!important` declaration can aid you in finding these conflicts.
EXERCISE 6

Using the !important Declaration

In this exercise, you’ll use the !important declaration to root out and resolve style conflicts.

1. In a browser, open ex0306.html from the 03_cascade folder you copied to your desktop, as shown in the illustration.

2. Open ex0306.html in your text or HTML editor. Near the beginning of the file, find the following block of code:

   <style type="text/css">
   </style>
3 Type `h1 {background: green;}` between the style tags.

4 Save `ex0306.html`, return to your browser, and reload the page.
There’s no change! What’s happening? Let’s find out in the next step.

5 Return to your text or HTML editor, and modify the `h1` declaration as follows:

```css
h1 {background: green !important;}
```
6 Save `ex0306.html`, return to your browser, and reload the page.

Aha! The background of our header is now green. What this tells you is that somewhere in your style sheet, you have a rule that is setting that white background, and it has a higher specificity than the `h1` declaration. In effect, you’re using `!important` as a diagnostic tool. The `!important` declaration overrides all the unimportant declarations, no matter how high the specificity of their selector. No matter what their source is, they will lose out to this `!important` declaration. So now that you know what the problem is, you can dig into the style sheet to resolve it.
7 In a browser, open the file `ex0306.css` (in your text or HTML editor) from the `styles` folder in the `03_cascade` folder you copied to your desktop. Find the following declaration:

```css
#content h1 {
    margin: 0 33% 1.25em -2em; padding: 0.5em 2em;
    background: #FFF; color: #686397;
    font-size: 1.5em;
}
```

This rule determines the background color of the header. You can reduce the specificity either in your external style sheet or in your HTML document. Let’s do the latter; you may not want to override this in every document tied to your style sheet.

---

8 Return to your text or HTML editor, and modify the `h1` declaration as follows:

```css
#content h1 {background: green;}
```

This will increase the specificity of this declaration.
Save and close ex0306.css, return to your browser, and reload the page.

The “about tea” heading is again green—without using !important. You found your conflict and were able to resolve it.

In this exercise, you used the !important declaration to troubleshoot specificity problems. If you used this declaration, reloaded your Web page, and saw no change, you would know the problem was not a specificity conflict. In that case, you may want to check your declaration statement for spelling errors. It happens to me all the time. Also important to note about the !important declaration is that you should not leave it in the pages you publish to the Web. It fixes your problem temporarily, but you might find in the future you want to override the particular style someplace else and therefore will need to use yet another !important because the original problem was never fixed. Never leave !important in your style sheets, if you can avoid it. That’s !important in a nutshell: Any declaration that is important always wins over unimportant declarations. Only when you have two conflicting !important declarations do you return to using the cascade resolution mechanisms such as specificity and declaration order.
EXERCISE

7 Understanding Inheritance

In this exercise, you'll explore a core feature of CSS called inheritance, which is the mechanism by which properties and values are propagated down the document tree. By applying styles to core elements, you can see this propagation in action.

1. In a browser, open ex0307.html from the 03_cascade folder you copied to your desktop, as shown in the illustration here.

2. Open ex0307.html in your text or HTML editor. Add the following line of code between the style tags as follows:

    <style type="text/css">
    body {color: orange;}
    </style>
Save ex0307.html, return to your browser, and reload the page.

Voila! The text changes to orange. You specifically set the body element to be orange but didn’t touch any of the other elements. But the text of the sidebar, for example, is orange. Why?

**NOTE:**

Inheritance and the Document Tree

Computers are notoriously short on intuition, so the computer doesn’t automatically understand your every desire. It needs a mechanism to apply properties throughout elements—that mechanism is inheritance.

The document structure is key to inheritance. When you apply a property to an element like the **body**, that value is propagated down the structure or document tree. It spreads downward from the **body**. Child elements receive properties from their parents. Not all properties are inherited. The exceptions are pretty obvious—borders, margins, and padding, and other properties that cannot be applied to all elements in the same way. The properties that are inherited, such as font color, size, family, are sensible.

You may have noticed in the previous step that the link colors didn’t change. Why? Inheritance works only when there is no competition. Inherited values have no specificity. Not even zero—none at all. Inherited values will always lose to directly assigned values. The color of the links (blue) is coming from the browser style sheet, and that value overrides any inherited properties and values.
4 Return to your text or HTML editor, and type the following after the first (body) rule:

```css
#sidebar {color: teal;}
```

By assigning a specific property value to the `sidebar` element, you will override the inherited properties set up in the first rule in your style sheet.

5 Save `ex0307.html`, return to your browser, and reload the page.

The sidebar text should now appear teal, as shown in the illustration here. This includes all the headings and borders. Why? Well, this property is propagated down the `sidebar` element tree to its descendent elements.

6 Return to your text or HTML editor, and modify the first rule (body) as follows:

```css
body {color: orange !important;}
```

Ordinarily, the `!important` declaration would override any other style rules. Let’s see whether this happens here.
7 Save and close ex0307.html, return to your browser, and reload the page.

There is no change. The links stay blue. As the orange value is inherited, the specificity is lost and the importance is lost.

In this exercise, you learned how the mechanism of inheritance works to propagate property values to descendent elements in a document and how inherited values are overridden. Often times, people will confuse the cascade in CSS with inheritance, assuming that the properties cascade down the document tree. But as you saw previously in this chapter, the cascade is actually the set of conflict resolution mechanisms used to determine how disparate styles are brought together to arrive at a final presentation style sheet. The process of having one style propagate to child elements is inheritance, not the cascade.
EXERCISE

Revealing Unstyled Documents

This exercise will show how to modify the settings of the Firefox browser to reveal documents in their truly unstyled state. This shows how much browser settings really do impact how the user perceives your Web site and the importance they play in the cascade.

1. Mac users, start by opening the **Finder** application. Windows users, proceed directly to Step 6.

2. Ensure Firefox is not running on your computer. Choose **Applications > Firefox**, right-click, and choose **Show Package Contents**.

3. Navigate through **Contents / Mac OS** to the **Res** folder.

4. Choose **Sort by Kind** to view all the CSS files together. The files that most directly relate to Web page presentation are **forms.css**, **html.css**, and **quirk.css**. The **ua.css** file is actually for the user agent.

5. **Ctrl+click** to multiple-select these files, and then drag them to the **Trash**. Minimize your **Finder** window. Mac users, skip the next step. It’s for Windows users only.

6. Windows users, ensure Firefox is not running on your computer; then choose **Start > Search**. Select **Search All Files and Folders** for the file name **html.css**. As shown in the illustration here, it should appear
in C:\Program Files\Mozilla Firefox\res. Delete the file by selecting it and hitting the **Delete** key or by right-clicking and choosing **Delete**. Close the **Search Results** window.

---

7 This step applies to both Windows Mac users. In a browser, open the `ex0308.html` file from the `03_cascade` folder you copied to your desktop, as shown in the illustration here.

This is the Web page without any user agent styles applied. There are still some styles applied, but you'll remove them in the next step.

---

8 Open the `ex0308.html` file in your text or HTML editor. Delete the following block of code:

```html
<link rel="stylesheet" type="text/css" href="styles/ex0307.css" media="all" />
<style type="text/css">
body {color: #C60;}
#sidebar {color: rgb(0,128,128);}
```
Save and close ex0308.html, return to your browser, and reload the page.

Here you go. This page is now almost completely unstyled. The exception is the link styling, which is derived from the browser preferences.

At this point, I recommend you browse the Web and check out some of your favorite sites such as lynda.com, for example. Script elements, like JavaScript, are ordinarily set to Display None, but you
removed these settings, so you will see snippets of code appear, and so forth. Also, items such as forms
will not work—presentation-wise at least. For example, your cursor may not completely align with the
address bar.

10 Quit Firefox. Then, depending on your operating system, do the following:

**Mac users:** Reopen your resources directory (the res folder) by choosing Window > res. Open the
Recycle Bin icon on your desktop. Multiple select the three CSS files in the Recycle Bin, and drag these
to the open resource directory window.

**Windows users:** Go to the Recycle Bin to locate html.css, right-click, and select Restore. Alternatively,
you can copy and paste it to the following location: C:\Program Files\Mozilla Firefox\res.

11 Open ex0308.html from the 03_cascade folder you copied to your desktop. This should launch
Firefox again.

The Web page should appear as it did the first time you opened it, with minimal styling.

12 Navigate to lynda.com by entering www.lynda.com in your address bar.

Everything should be back to normal! No JavaScript, no missing form fields, and so on.

In this exercise, you learned how to modify your browser settings to remove the built-in styles. Most users
are unlikely to modify their settings in this way—it kind of ruins the visual experience. However, it’s useful
to know just how important the browser (or user agent, as I’ve referred to it elsewhere) preferences and
defaults are that affect your final document. It is a critical element in the cascade.
EXERCISE

Applying User Style Sheets

In this exercise, you'll learn how to associate a user style sheet with your browser so it will apply to every Web page you visit.

1. In a browser, open ex0309.html from the 03_cascade folder you copied to your desktop, as shown in the illustration here.

I took this screen shot using Safari on a Mac, and although I generally use Mozilla Firefox, Safari makes it really easy to apply a user style sheet. I've also included instructions in this exercise for Windows users working with Internet Explorer.

2. Depending on what operating system you're using, follow these steps:

Mac users: Open Safari, click the Safari menu item, select Preferences and then select the Advanced tab. From the Style Sheet pop-up menu, choose Other. In the window that opens, navigate to the 03_cascade folder you copied to your desktop. Navigate to the Styles folder, select ex0309.css, and click Choose. Close the Safari Preferences dialog box.
Windows users: Open Internet Explorer. Choose Tools > Internet Options, and select the General tab. Click the Accessibility button. In the Accessibility dialog box that opens, turn on the Format documents using my style sheet check box, and then click the Browse button. Navigate to the 03_cascade folder you copied to your desktop. Navigate to the Styles folder, select ex0309.css, and click Open. In the Accessibility dialog box, click OK, and then click OK again to close the Internet Options dialog box.

You’ll notice that the page just changed. That’s because you have applied the user style sheet. Let’s take a look at the contents of the style sheet.
Open `ex0309.css` in your text or HTML editor.

You’ll notice the style sheet has applied a background and text colors, as well as some text decoration. You have made all these declarations `!important`, as discussed previously, to make sure they override the author style guide.

In a browser, type `www.lynda.com` in the navigation bar.
Well, isn’t this interesting? You might just wander around the Internet for a bit to see how the user agent preferences interact with different sites.

To remove the user style sheet, do the following depending on your operating system:

Mac users: Click the Safari menu item, and select the Advanced tab. From the Style Sheet drop-down list, choose None Selected. Close the Safari Preferences dialog box, which will restore the page default settings. Close Safari.

Windows users: Choose Tools > Internet Options, and click the Accessibility button. Turn on the Format documents using my style sheet check box, and click OK. Click OK to close the Internet Options dialog box, which will restore the default settings. Close Internet Explorer.

Close Firefox and close ex0309.css in your text or Web editor.

This was a short exercise, but I hope you learned the importance of user style sheets and how they can affect your document presentation. They are yet another important element in the cascade.

Moreover, I hope you’ve enjoyed this chapter. I covered some key concepts, including selectors and the important elements of the cascade. In the next chapter, you’ll explore how to use CSS to modify page layout, including tables and columns. I find this to be one of the biggest incentives for using CSS—getting rid of that pesky table markup. See you there!