3 CSS BASICS BONUS ONLINE LESSON

Lesson overview

In this lesson, you will gain valuable skills working with cascading style sheets and learn the following:

- How to write CSS rules by hand
- How to write different types of selectors based on cascade, inheritance, and descendant theories
- How to format your HTML text and structural elements

This lesson will take about 2 hours and 30 minutes to complete. If you have not already done so, download the project files for this lesson from the Lesson & Update Files tab on your Account page at www.peachpit.com, store them on your computer in a convenient location, and define a new site in Dreamweaver based on this folder, as described in the "Getting Started" section of the book. Your Account page is also where you'll find any updates to the chapters or to the lesson files. Look on the Lesson & Update Files tab to access the most current content.

Previewing the completed file

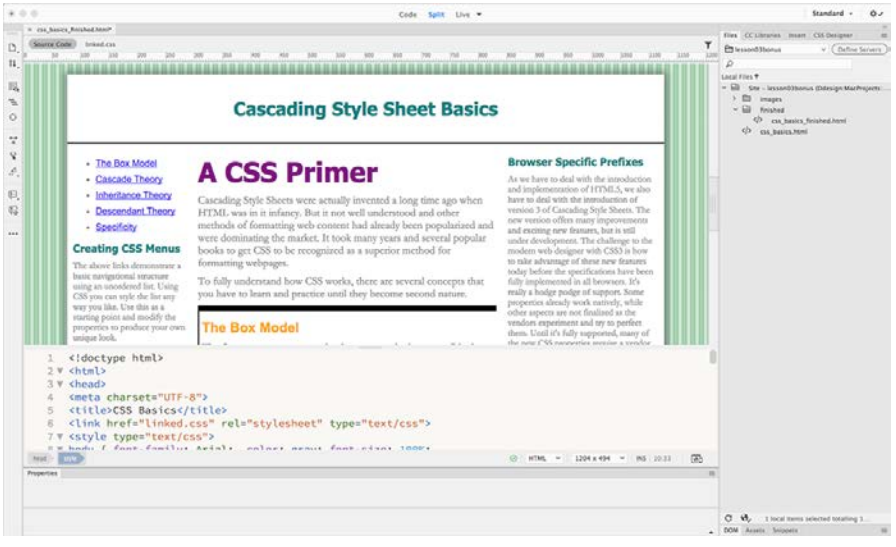
The best way to learn CSS is by creating your own style sheets. In this online bonus lesson, you'll learn how CSS works by creating a complete style sheet for a sample HTML file like the one you will view in this exercise.

1 Define a new site based on the lesson03bonus folder as described in the "Getting Started" at the beginning of the book.

2 Select the Standard workspace in the Workspace menu, if necessary.

3 Open the **css_basics_finished.html** file from the finished folder in lesson03bonus.

4 If necessary, switch to Split view using Code view and Live view. Observe the content and code in the two windows.

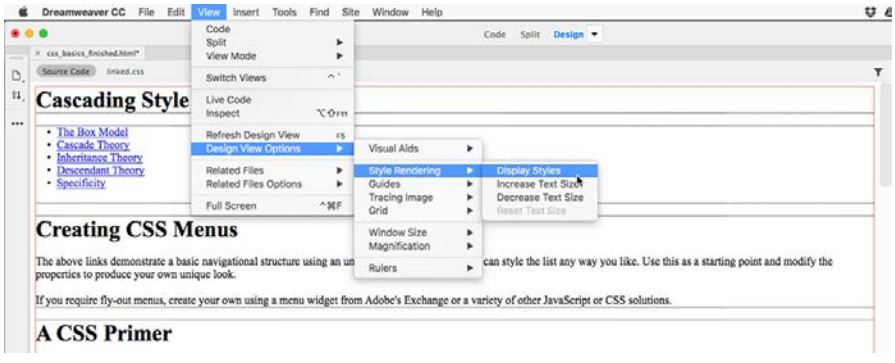


The file contains a complete HTML page with a variety of elements, including headings, paragraphs, lists, and links all fully formatted by CSS. Note the text styling, as well as the colors and borders assigned to each element. Typically, files will open in Live view or with Live view active in Split view. This is important because some styling may not display properly in Design view alone.

5 If necessary, switch to Live view only.

The entire webpage is displayed in one window. Live view displays CSS styling more accurately than Design view. For example, in Live view you should see a drop shadow around the main content section. As before, the best way to see the true power of CSS is by shutting it off.

6 Switch to Design view.
Choose View > Design View Options > Style Rendering > Display Styles to toggle off the CSS rendering in Dreamweaver.



The page is no longer formatted by CSS and displays only the default styling you would expect on standard HTML elements. The text content now stacks vertically in a single column with no colors, borders, backgrounds, or shadows.

7 Choose View > Design View Options > Style Rendering > Display Styles to toggle on the CSS rendering in Dreamweaver again.

The CSS styling is turned back on.

8 Select Live view.

Dreamweaver previews the page in Live view again.

9 Choose File > Close.
Close all sample files without saving them.

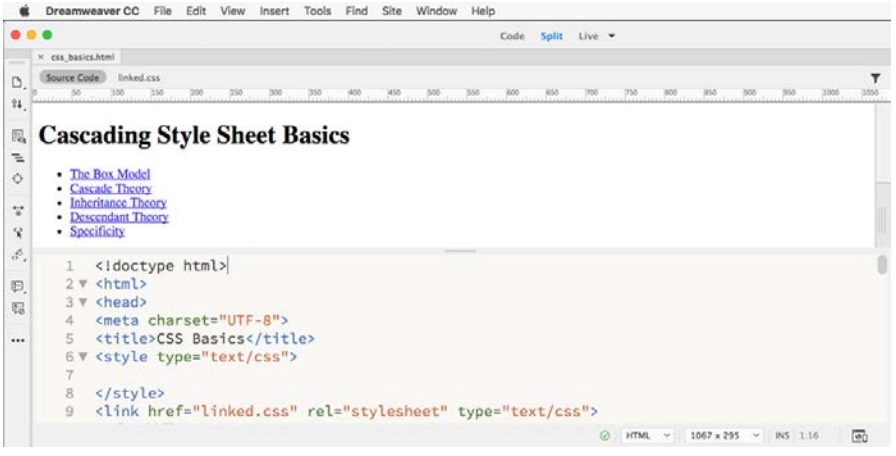
As you can see, CSS can style all aspects of a webpage with amazing variety and detail. First let's take a look at how CSS can format text.

Formatting Text

▶ **Tip:** Dreamweaver CC (2017 release) will open most files in Live view by default. In the following exercises, be aware that certain commands will function only in Design view. Be prepared to switch between Live and Design views as necessary.

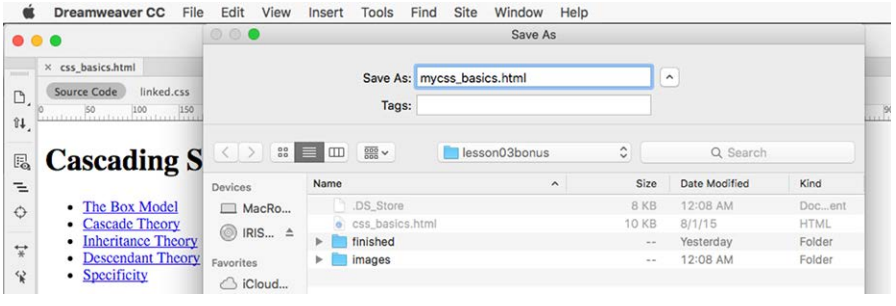
1 Open **css_basics.html** from the lesson03bonus folder.

2 Switch to Split view, if necessary.



The file contains a complete HTML page with various elements, headings, paragraphs, lists, and links that are currently formatted only by default HTML styling.

3 Save the file as **mycss_basics.html** in the site root folder.



Dreamweaver creates a copy of the file using the new name and displays two tabs at the top of the document window. Since the original file is still open, it could cause confusion during the following exercises. Let's close the original file.

4 Select the document tab for the **css_basics.html** file. Choose File > Close.

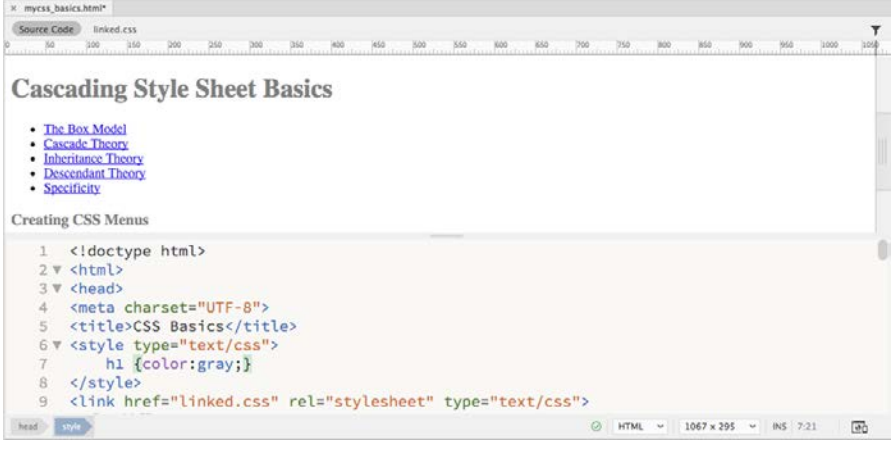
The **mycss_basics.html** file should be the only one open.

5 Observe the `<head>` section in the Code view window.

Note that the code contains a `<style>` section but no CSS rules.

6 Insert the cursor between the `<style>` and `</style>` tags.

7 Type `h1 { color:gray; }` and, if necessary, click in the Live view window to refresh the display.



8 Save the file.

The `h1` headings throughout the page now display in gray. The rest of the text still displays the default formatting. Congratulations! You wrote your first CSS rule.

Note: As you type the rule markup, Dreamweaver provides code hints as it did with the HTML code in bonus Lesson 2, "HTML Basics Bonus Online Lesson." Feel free to use these hints to speed up your typing, or simply ignore them and continue typing.

Tip: Live view should automatically refresh the display when you edit content in the Code window. But if you don't see the display change as described, you may need to refresh the display manually.

Cascade Theory in action

Once you start writing CSS rules, you will encounter various types of conflicts. Putting your knowledge of CSS theory to the test will help you better understand how to use cascading style sheets and how to troubleshoot the conflicts.

In this exercise, you will learn how Cascade Theory functions first hand.

1 Open **mycss_basics.html** in Split view, if necessary.

Note: CSS does not require line breaks between rules, but they do make the code easier to read. Feel free to add them between rules or properties at any time.

2 In Code view, insert the cursor at the end of the rule created in step 7 of the previous exercise.
Press Enter/Return to create a new line.

3 Type `h1 { color:red; }` and click in the Live view window, to refresh the display.

The `h1` headings now display in red. The styling of the new rule supersedes the formatting applied by the first rule. It's important to understand that the two rules are identical except that they apply different colors: red or gray. Both rules want to format the same elements, but only one will be honored.

It's clear the second rule won, but why? In this case, the second rule is the *last* one declared, making it the *closest* one to the actual content. Whether intentional or not, a style applied by one rule may be overridden by declarations in one or more subsequent rules.

4 Click the line number containing the rule `h1 { color: gray; }` (around line 7).

The entire line is selected.

5 Choose Edit > Cut.

6 Insert the cursor at the end of the rule `h1 { color: red; }` and press Enter/Return to insert a new line.

Tip: When code is selected in Code view you can also drag the content to a new location in the window.

7 Choose Edit > Paste or press Ctrl+V/Cmd+V.

```
6 ▼ <style type="text/css">
7 ▼     h1 {color:gray;}
8       h1 {color:red;}
9   </style>
10  <link href="linked.css" rel="stylesheet"
```

```
6 ▼ <style type="text/css">
7       h1 {color:red;}
8 ▼     h1 {color:gray;}
9
10  </style>
```

The rule applying the gray color now appears last. You have switched the order of the rules.

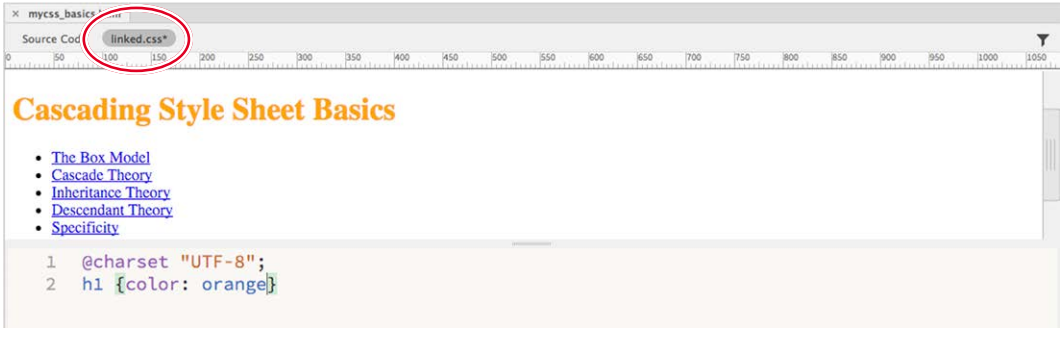
8 If necessary, click in the Live view window to refresh the preview display.

The h1 headings display in gray again. *Cascade* applies to styles whether the rules are embedded in the webpage or located in a separate external, linked style sheet.

9 Select **linked.css** in the Related Files interface.

When you select the name of the referenced file, the contents of that file appear in the Code view window. If the file is stored on a local hard drive, Dreamweaver allows you to edit the contents without actually opening the file.

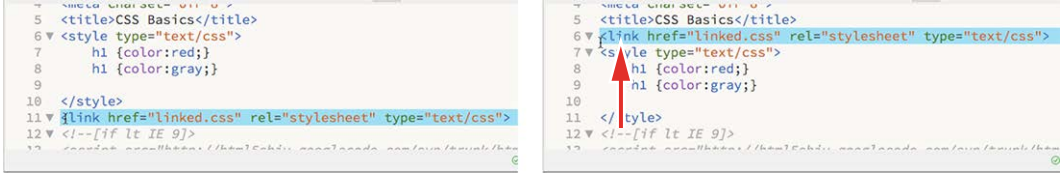
10 Insert the cursor in line 2. Type `h1 { color:orange; }` and press Ctrl+S/Cmd+S to save the file.
Click in the Live view window to refresh the display, if necessary.



11 Select Source Code in the Related Files interface. Locate the <link> reference for **linked.css** in the <head> section (around line 11).

The <link> element appears *after* the closing </style> tag. Based strictly on cascade, this means any rule that appears in the linked file will supersede duplicate rules in the embedded sheet.

12 Click the line number for the external CSS <link> reference to select the entire reference. Cut and paste, or drag, the entire <link> reference above the <style> element.



13 Click in the Live view window to refresh the display, if necessary.

The headings revert to gray.

14 Choose File > Save All.

As you can clearly see, the order and *proximity* of the rules within the markup are powerful factors in how CSS is applied.

Inheritance Theory in action

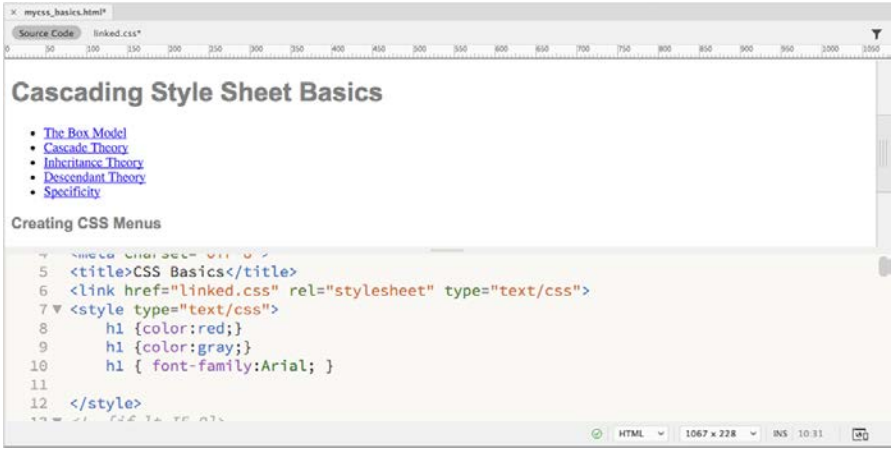
Inheritance Theory speaks to the lineage or parental origins of an element. In other words, if your parent has blue eyes and blond hair, the odds of you having blue eyes and blond hair also are very high.

In this exercise, you will learn how Inheritance Theory functions first hand.

1 If necessary, open **mycss_basics.html** in Split view.

2 Insert the cursor after the rule `h1 { color: gray; }` in the embedded style sheet. Press Enter/Return to insert a new line.

3 Type `h1 { font-family:Arial; }` and click in the Live view window to refresh the display.



The `h1` elements appear in Arial and gray. The other content remains formatted by default styling.

Now there are four CSS rules that format `<h1>` elements. Can you tell, by looking at the Live view window, which of the rules is formatting the `<h1>` text now? If you said *two* of them, you're the winner.

At first glance, you may think that the rules formatting `<h1>` elements are separate from each other. And technically, that's true. But if you look closer, you'll see that the new rule doesn't contradict the others. It's not resetting the `color` attribute as you did in the previous exercises; it's declaring a new, additional attribute. In other words, since both rules do something different, both will be honored, or inherited, by the `h1` element. All `<h1>` elements will be formatted as gray *and* Arial.

Far from being a mistake or an unintended consequence, the ability to build rich and elaborate formatting using multiple rules is one of the most powerful and complex aspects of cascading style sheets.

4 Insert the cursor after the last `h1` rule.
Insert a new line in the code.

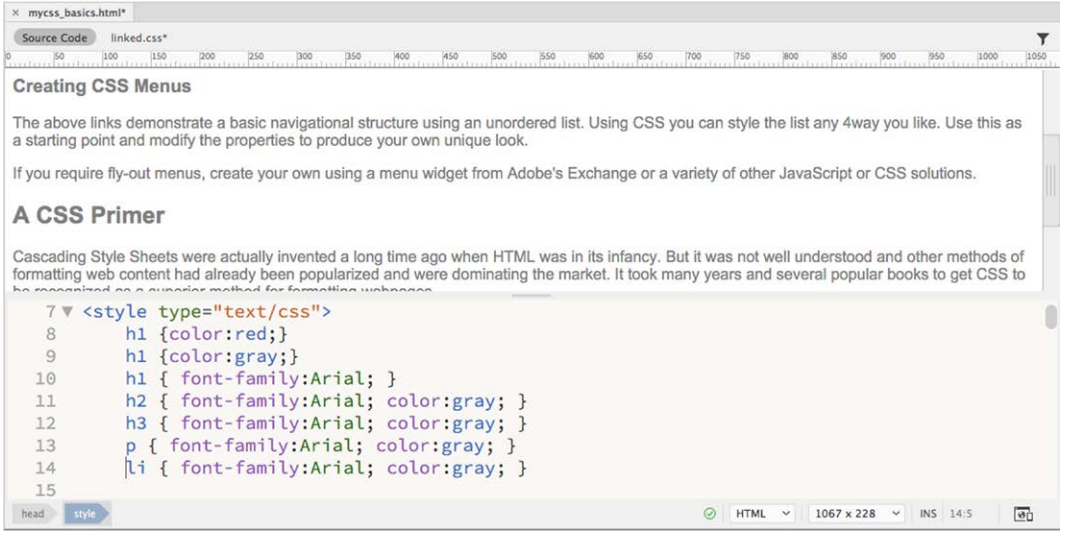
5 Type `h2 { font-family:Arial; color:gray; }` and click in the Live view window to refresh the display.

The `h2` elements appear in Arial and gray; they originally displayed in a serif font in black.

6 After the `h2` rule, type the following code:

```
h3 { font-family:Arial; color:gray; }
p { font-family:Arial; color:gray; }
li { font-family:Arial; color:gray; }
```

7 If necessary, refresh the Live view window display by clicking in it or clicking the Refresh button in the Properties panel.



All the elements now display the same styling, but you used six rules to format the entire page.

Although CSS styling is far more efficient than the obsolete HTML-based method, inheritance can help you optimize your styling chores even more. For example, all the rules include the statement `{ font-family:Arial; color:gray; }`. Redundant code like this should be avoided whenever possible. It adds to the amount of code in each webpage as well as adds to the time it takes to download and process it. By using inheritance, sometimes you can create the same effect with a single rule. One way to make your styling more efficient is to apply formatting to a *parent* element instead of to the elements themselves.

Note: There is no requirement to create rules in any specific order or hierarchy. You may order them any way you please, as long as you keep in mind how the application of styling is governed by cascade and inheritance.

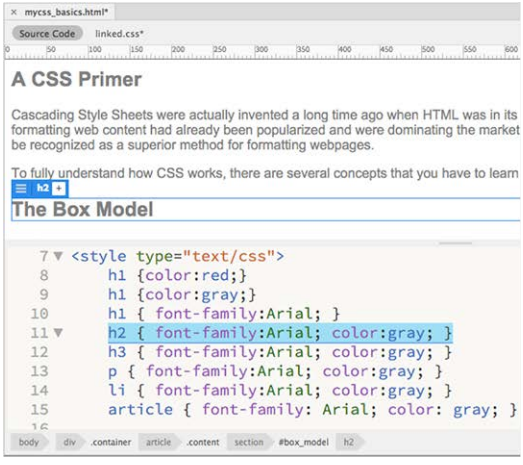
8 Create a new line in the `<style>` section.
Type the following code:

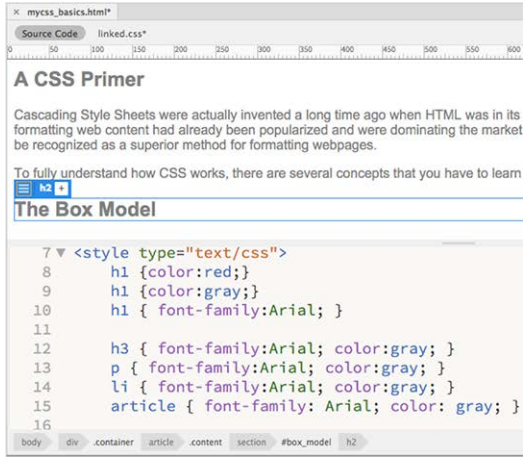
```
article { font-family: Arial; color: gray; }
```

If you look through the code, you will see that the <article> tag contains much but not all of the webpage content. Let's see what happens if you delete some of your CSS rules.

Tip: Rules typically contain multiple property declarations.

9 Select and delete the rule:
~~h2 { font-family: Arial; color: gray; }~~





10 Refresh the Live view window display.

The h2 elements appearing within the <article> element remain formatted as gray Arial. The other h2 element down the page and outside the <article> now appears in HTML default styling.

11 Select and delete all h1 rules. Don't forget the one in the **linked.css** file. Refresh the Live view display.

The h1 elements contained within the <article> element continue to be styled. Those outside the <article> element have reverted to the HTML defaults. Since the new rule targets only the <article> element, only the elements contained within it are styled.

12 Select and delete the rules formatting h3, p, and li.
Refresh the Live view display.

As in step 11, any content contained in the <article> tag remains formatted, while content elsewhere has reverted.

This is the way inheritance works. You could simply re-create the rules to format the other content, but there's a simpler alternative. Instead of adding additional CSS rules, can you figure out a way to use inheritance to format all the content on the page the same way? A hint: Look carefully at the entire structure of the webpage.

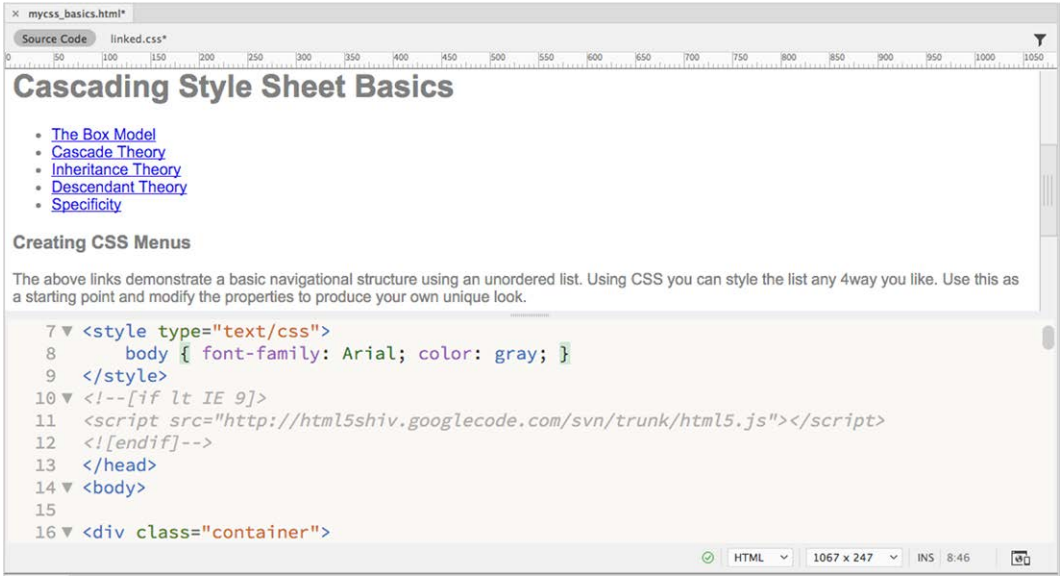
Did you select the <body> element? If so, you win again. The <body> element contains all the visible content on the webpage and therefore is the parent of all of it.

13 Change the rule selector from article to **body** and delete any blank lines.

▶ **Tip:** In this particular page, you could also use the element to achieve the same result. But since is a frequently used element, it might pose unpredictable conflicts in the future. Since webpages have only one <body> element, it is definitely the preferred target.

14 Choose File > Save All.

Refresh the Live view display, if necessary.



Once again, all the text displays in gray Arial. By using inheritance, all the content is formatted using one rule instead of six. You'll find that the <body> element is a popular target for setting various default styles.

Note: Dreamweaver frequently edits externally linked files. Use Save All whenever changes on your page may affect multiple files in your site.

Descendant Theory in action

Although Inheritance affects elements automatically, the randomness of how it functions is often frustrating. Styling an element that is nested within another is not always predictable or reliable. Descendant Theory instead targets the child element directly and specifically. Although inheritance can affect any element—headings, paragraphs, and list elements—Descendant Theory can target styling to a specific child element if desired.

In this exercise, you will learn how Descendant Theory functions firsthand.

1 If necessary, open **mycss_basics.html** in Split view and observe the structure of the HTML content.

The page contains headings and paragraphs in various HTML5 structural elements, such as `article`, `section`, and `aside`. The rule `body {font-family:Arial; color: gray; }` applies a default font and color to the entire page. In this exercise, you will learn how to create descendant CSS rules to target styling to specific elements in context.

Note: You will examine the role of specificity later in this lesson.

2 In Code view, insert the cursor at the end of the body rule.
Press Enter/Return to insert a new line.

3 Type `p { font-family:Garamond; }` and refresh the Live view display.

● **Note:** Step 3 assumes you have Garamond installed on your computer. If it is not, select another serif style font, such as Times.

All p elements on the page display in Garamond. The rest of the page continues to be formatted in Arial.

By creating a selector using the p tag, the font formatting has been overridden for all p elements no matter where they appear. You may be thinking that since the p rule appears *after* the body rule, this type of styling simply relates to the cascade order. Let's try an experiment to see whether that's true.

● **Note:** Dragging code can be a difficult skill to master. Feel free to simply cut and paste the code.

4 Click the line number for the p rule.
Drag the p rule above the body rule.
Refresh the Live view display.

The p rule now appears above the body rule, but the styling of the paragraphs did not change.

If the styling of p elements were determined simply by cascade, you would expect the headings to revert to gray Arial. Yet here, the styling is unaffected by changing the order of the rules. Instead, by using a more specific tag name in the selector, the new p rule becomes more powerful than the generic body rule. By properly combining two or more tags in the selector, you can craft the CSS styling on the page in even more sophisticated ways.

5 Create a new line after the p rule.
On the new line, type the following:
`article p { font-size:120%; color:darkblue; }`

By adding a p tag immediately after `article` in the selector, you are telling the browser to format only p elements that are children, or *descendants*, of `article` elements. Remember, a *child* element is one contained or nested within another element.

6 If necessary, refresh the Live view display.

All paragraphs appearing within the `<article>` element now display in dark blue, 120 percent larger than the other paragraph text on the page.

7 Choose File > Save All.

Although it may be hard to understand at this moment, the styling in other rules—both body and p—is still being inherited by the newly formatted paragraphs. But wherever two or more rules conflict, a descendant selector will win over any less specific styling.

Using class and id selectors

In **mycss_basics.html**, all h1 elements are formatted identically regardless of where they appear in the layout. In the following exercise, you'll use classes and ids to differentiate the styling among the headings.

1 Insert a new line after the rule `article p`.
Type `h1 { font-family:Tahoma; color:teal; }` and refresh the display.

All `h1` headings now display in the color teal and the font Tahoma.

Although it's tagged identically to the other `h1` headings, "A CSS Primer" is the main heading in the `<article>` element. To make it stand out from the other headings, you can use the class attribute assigned to its parent to target it for special formatting.

2 Create the following new rule:
`.content h1 { color:red; font-size:300%; }`

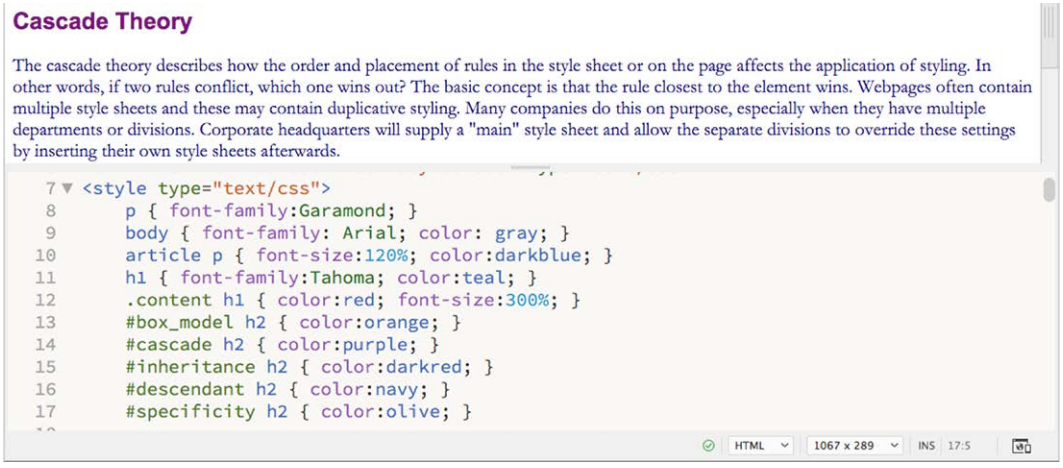
The heading "A CSS Primer" displays in red, 300 percent larger than the body text.

In CSS syntax, the period (`.`) refers to a *class* attribute, and the hash (`#`) means *id*. By adding `.content` to the selector, you have targeted the styling only to `h1` elements in `<article class="content">`.

In the same way, you can assign custom styling to the subheadings (`h2`). You can use the id attributes of each `<section>`.

3 Create the following rules:

```
#box_model h2 { color:orange; }
#cascade h2 { color:purple; }
#inheritance h2 { color:darkred; }
#descendant h2 { color:navy; }
#specificity h2 { color:olive; }
```



4 Choose File > Save All.
If necessary, refresh the display.

All the h2 headings now display unique colors. What's important to understand here is that all the formatting you see has been applied without adding a single attribute to any of the headings. They are being formatted based solely on their position within the structure of the code.

Understanding descendant styling

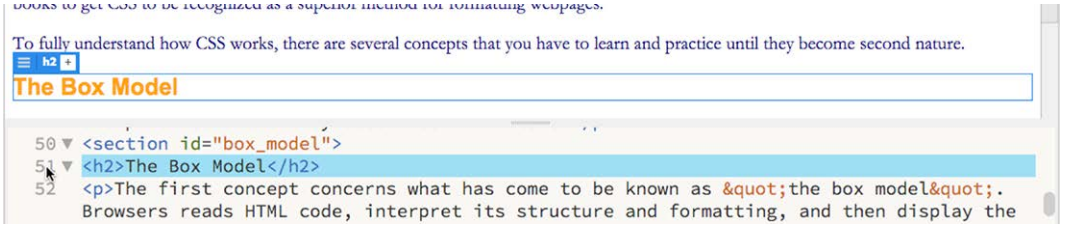
CSS formatting can be confusing for designers coming from the print world. Print designers are accustomed to applying styles directly to text and objects, one at a time. In some cases, styles can be based on one another, but this relationship is intentional. In print-based styling, it's impossible for one paragraph or character style to affect another *un*intentionally. On the other hand, in CSS, the chance of one element's formatting overlapping or influencing another's happens all the time.

It may be helpful to think of it as if the elements were formatting themselves. When you use CSS properly, the formatting is not intrinsic to the element but to the entire page and to the way the code is structured.

The ability to separate the content from its presentation is an important concept in modern web design. It allows you great freedom in moving content from page to page and structure to structure without worrying about the effects of residual or latent formatting. Since the formatting doesn't reside with the element itself, the content is free to adapt instantly to its new surroundings.

In this exercise, you'll move one of the uniquely styled elements to a different location in the page to see how its position dictates how it is styled.

1 In Code view, click the line number for the "The Box Model" heading (around line 51).

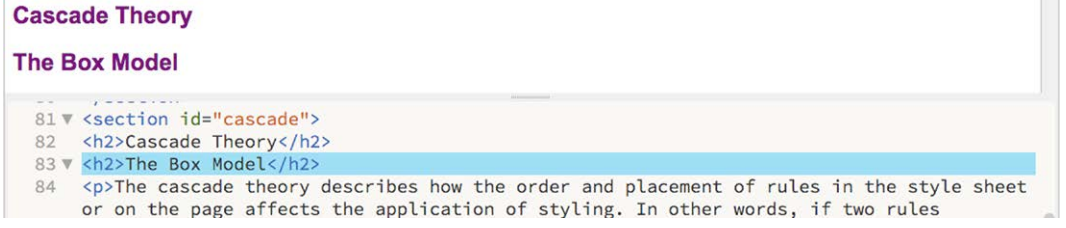


This should highlight the entire element. Note that the heading is orange.

2 Choose Edit > Copy or press Ctrl+C/Cmd+C.

3 Insert the cursor at the end of the h2 element "Cascade Theory" (around line 82) and create a new line.

4 Click to select the line number for the new blank line. Choose Edit > Paste or press Ctrl+V/Cmd+V to replace the blank line.



5 Refresh the display, if necessary.

The heading "The Box Model" appears following and formatted identically to the heading "Cascade Theory."

6 Insert the cursor at the end of the "Inheritance Theory" heading and create a new line.

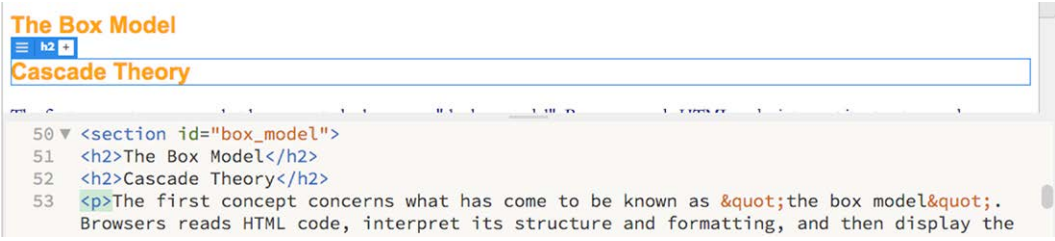
7 Press Ctrl+V/Cmd+V to paste the heading again.

The pasted heading is dark red and styled identically as the heading "Inheritance Theory."

As you can see, the formatting of the heading in the original instance does not travel with the text pasted in a new location. That's the point of separating content from presentation—you can insert the content anywhere, and it will adopt the formatting native to that position. It even works in reverse.

8 In Code view, select and copy the "Inheritance Theory" heading.

9 Insert the cursor after the original "The Box Model" heading and paste the text on a new line.



The heading appears and adopts the same styling as "The Box Model."

Once again, the pasted text matches the formatting applied to the other `h2` element within the `<article>`, ignoring its original styling altogether. Now that you've seen how Descendant Theory works, there's no need for the extra headings.

10 Choose File > Revert. Click Yes to revert the file to the previously saved version.

Specificity Theory in action

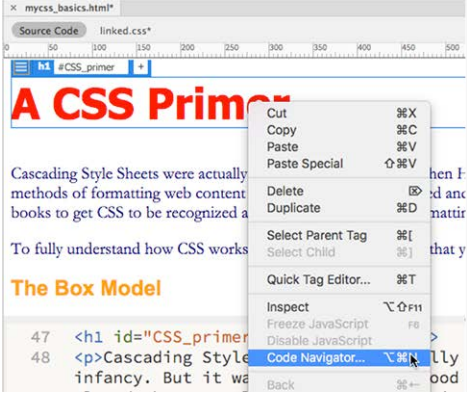
As explained in Lesson 3, "CSS Basics," specificity is the means by which browsers and other web-compatible applications determine how CSS styling is applied, especially when two or more rules conflict. You could compare rules manually and calculate which rule or rules should win, but Dreamweaver provides a couple of built-in tools that can do that work for you: Code Navigator and CSS Designer.

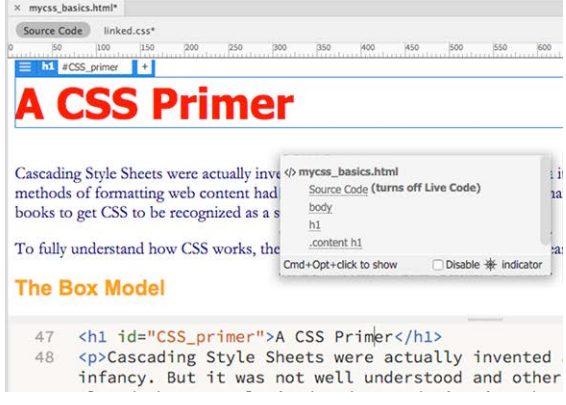
Using Code Navigator to identify specificity

Code Navigator puts a CSS specificity checker a mouse-click away.

1 In Live view, insert the cursor in the heading "A CSS Primer."

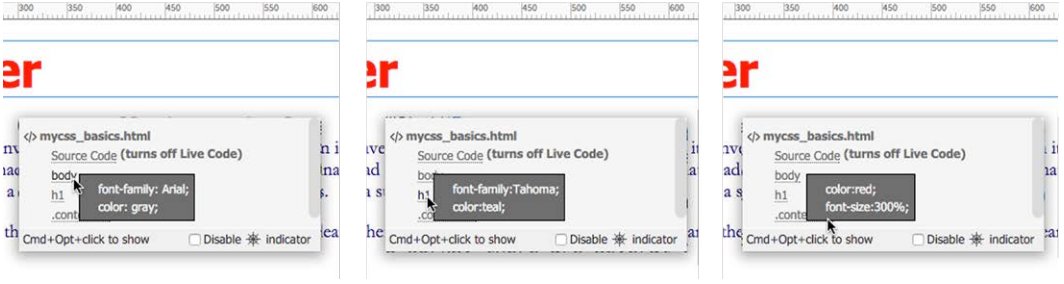
2 Right-click the heading and select Code Navigator from the context menu.





A pop-up window appears listing all the rules formatting the heading. Notice that the display shows three rules (`body`, `h1`, and `.content h1`) in descending order. The order is important because Code Navigator shows the rules with the most specific one at the bottom. In this case, that's `.content h1`. But that's not all it does; it can also show you what styling is being applied by each rule.

3 Position the cursor over each rule in the pop-up window, but do not click them.



When the cursor hovers over the rule, another window opens showing the properties and values assigned to that rule. Note how all three rules apply a color specification, but `.content h1` wins. Code Navigator also enables you to edit the CSS rules.

4 Click the rule `.content h1`.

The Code view window focuses on the CSS rule `.content h1` enabling you to edit the CSS if so desired. If the rule appeared in an external CSS file, the Code view window would load the contents of that file.

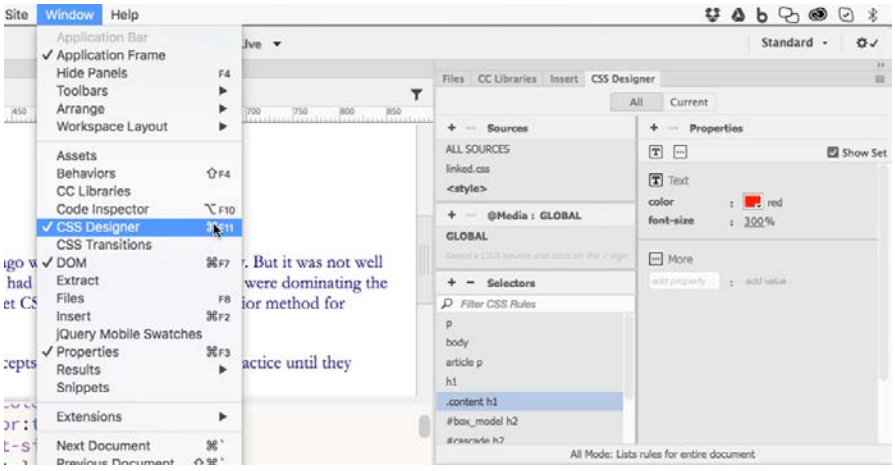
Another tool that provides similar features gives you even more power to troubleshoot and edit the CSS styling.

Using CSS Designer to identify specificity

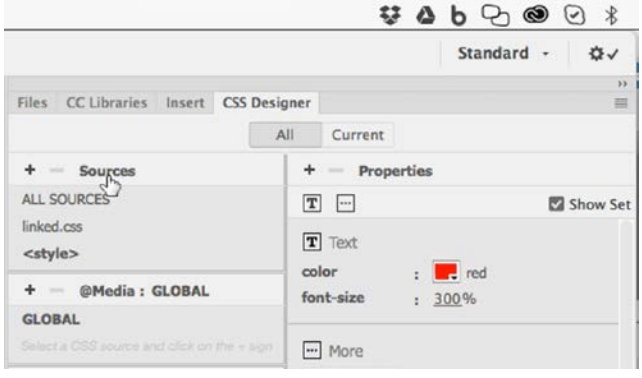
The CSS Designer is the main tool in Dreamweaver for creating, editing, and troubleshooting CSS styling. You will use it extensively in this lesson and the book to write CSS rules and format content in the book.

1 If necessary, choose Window > CSS Designer to display the panel.

The CSS Designer should be a permanent fixture of the Standard workspace, but if you see it appear as a floating panel, feel free to dock it to the right side of the interface.



2 Click the Sources tab to view the style sheets defined in the page.

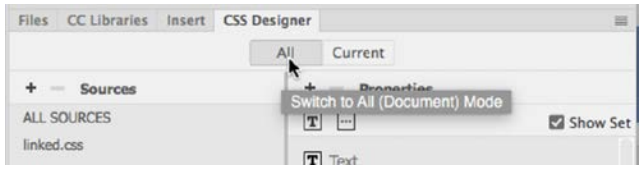


When the All button is selected, the Source pane shows two style sheets defined in the page: `linked.css` and `<style>`. The first is obviously an external CSS file, while the notation `<style>` indicates that the second style sheet is embedded within the webpage. CSS Designer can work with both types, as well as inline CSS styles.

3 Click All Sources.

This option shows all rules in all style sheets.

4 Click the All button at the top of the CSS Designer.



CSS Designer displays all CSS rules defined in the webpage and its associated style sheets. The Selectors pane lists the names of the rules. In All mode, rules are listed in the order they are defined in the style sheets. The first in the pane should be the `p` rule.

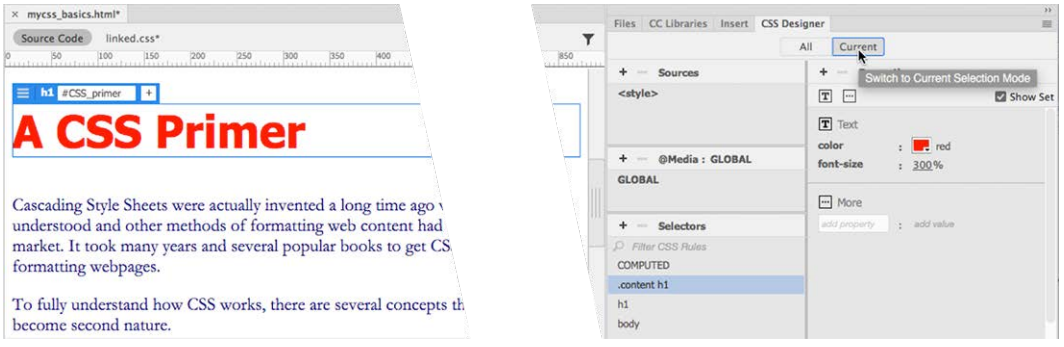
5 Click the `p` rule.

The Properties pane displays the settings assigned to the rule, in this case `font-family: Garamond`. Note how all the `<p>` elements are highlighted in blue in the document window. This is a nice little side benefit of the CSS Designer, which allows you to graphically see what elements are being styled by a particular rule. The CSS Designer also works the other way.

6 Click the heading "A CSS Primer" in the Live view window.

The Element HUD appears around the heading.

7 Click the Current button in the CSS Designer.

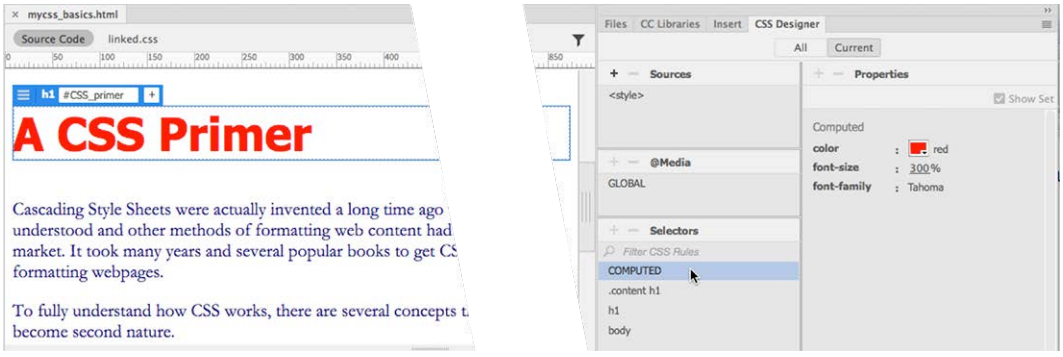


The CSS Designer now displays the list of three rules supplying styling to the selected element. The most specific rule appears at the top. Although `.content h1` is the most specific, this does not mean the other rules are not contributing to the current styling in some fashion.

8 Click each rule listed in the Selectors pane and observe the properties defined in it.

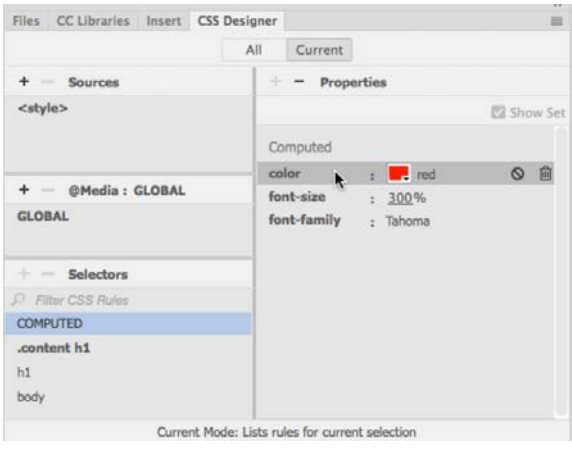
All three apply a font color, two apply a font family, and one sets the font size. In the past, manually checking the specifications was the only way to find out whether more than one rule was styling an element. Today, the CSS Designer can do that tedious work with a single click of the mouse.

9 Click COMPUTED in the Selectors pane.



The CSS Designer compares all the competing rules and specifications and calculates their specificity. It then displays the properties actually applied. You can even use the display to identify the source of the styling.

10 Click the `color: Red` property.



Note how the `<style>` notation in Sources and the `.content h1` selector in Selectors are highlighted in bold. The bold tells you that the property is contained in the `.content h1` rule in the embedded style sheet. Keep an eye on the bold notations to help identify the source of formatting as you work through the other exercises.

Formatting objects

After the last few exercises, you may be thinking that styling text with CSS is perplexing and difficult to understand. But hold onto your hat. The next concept you'll explore is even more complex and sometimes even controversial: object formatting. Consider object formatting as specifications directed at modifying an element's size, background, borders, margins, padding, and positioning. Since CSS can redefine any HTML element, object formatting can basically be applied to any tag, although it's most commonly directed at HTML container elements, such as `<div>`, `<header>`, `<article>`, and `<section>`, among others.

By default, all elements start at the top of the browser window and appear consecutively one after the other from left to right, top to bottom. Block elements generate their own line or paragraph breaks, inline elements appear at the point of insertion, and hidden elements take up no space on the screen at all. CSS can control all these default constraints and lets you size, style, and position elements almost any way you want them.

Size is the most basic specification and least problematic for an HTML element. CSS can control both the width and the height of an element with varying degrees of success. All specifications can be expressed in absolute terms (pixels, inches, points, centimeters, and so on) or in relative terms (percentages, ems, or exes).

Width

As you should already be aware, all HTML block elements take up the entire width of the screen by default, but there are a variety of reasons why you'd want to set the width of an element to something less than that. For example, studies have shown that text is easier to read, and more understandable, if the length of a line of type is between 35 and 50 characters. This means that on a normal computer screen a line of type stretching across 1,000 or more pixels could include easily 120 characters or more. That's why most websites today break up their layouts and display text in two or more columns at a time.

CSS makes it simple to set the width of an element. In this exercise, we'll experiment by applying different types of measurements to the various content elements on the page.

1 Open **mycss_basics.html** from the lesson03bonus folder.

2 View the page in Split view and observe the CSS code and HTML structure.

The file contains headings, paragraphs, and lists in several HTML5 semantic elements. The text is partially formatted by several CSS rules, but the structural elements display only default styling.

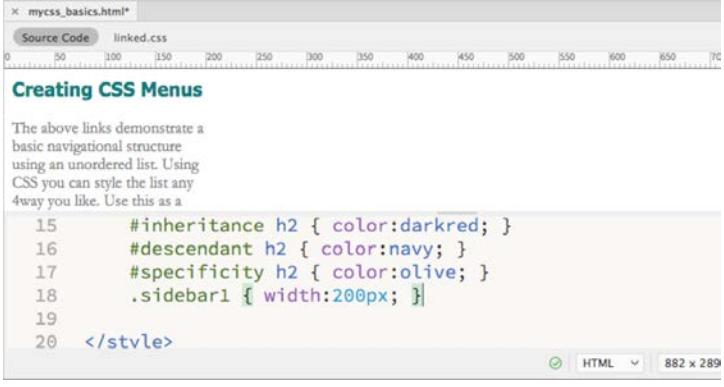
Fixed widths

The container elements, such as `<div>`, `<header>`, `<article>`, `<section>`, and `<aside>`, each currently occupy 100 percent of the width of the browser window or their parent element by default. CSS allows you to control the width by applying absolute (fixed) or relative (flexible) measurements.

1 If necessary, open **mycss_basics.html** in Split view.
In Code view, insert a new line after the last rule in the `<style>` section.

2 Type `.sidebar1 { width:200px; }` to create a new rule to style `sidebar1`.

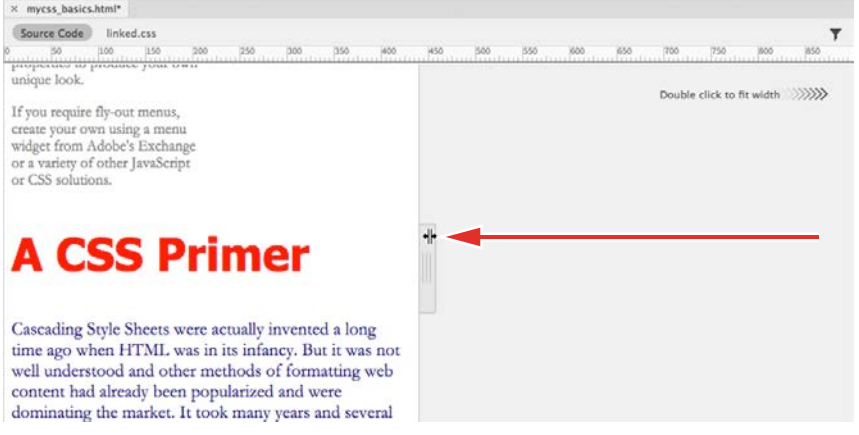
3 Save the file and refresh the Live view display.



The `sidebar1` element now occupies only 200 pixels in width; the other elements in the layout are unchanged.

By using pixels, you have set the width of this element, and its children, by an absolute, or *fixed*, measurement. This means `sidebar1` will maintain its width regardless of changes to the browser window or screen orientation.

4 Select the Scrubber, drag it to the left and right, and observe how the different elements react.



As expected, the element `sidebar1` displays at 200 pixels in width no matter what size the screen assumes. The other containers remain at full screen width.

5 Drag the Scrubber fully to the right side of the document window.

Fixed widths are still popular all over the Internet, but in some cases, such as when designing for mobile devices, you'll want elements to change or adapt to the screen size or user interaction. CSS provides three methods for setting widths using relative, or flexible, measurements such as em, ex, and percentage (%).

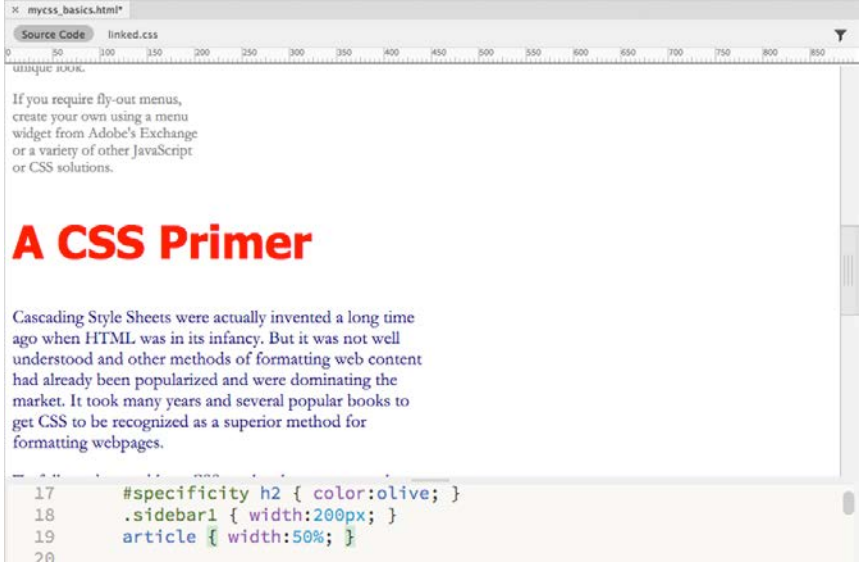
Relative widths

Relative measurements set by percentage (%) are the easiest to define and understand. The width is set in relation to the size of the screen: 100% is the entire width of the screen, 50% is half, and so on. If the screen or browser window changes, so does the width of the element. Percentage-based designs are popular because they can adapt instantly to different displays and devices. But they are also problematic because changing the width of a page layout dramatically can also play havoc with your content and its layout.

1 If necessary, open **mycss_basics.html** in Split view.
Add a new rule: `article { width:50%; }`

▶ **Tip:** To test element widths, it may be easier to display Split view windows vertically. You can access this preference in the View menu.

2 Save the file and refresh the Live view display.



The `<article>` element displays at 50 percent of the screen width. Widths set in percentage adapt automatically to any changes to the screen size.

3 Drag the Scrubber to the left and right, and observe how the `<article>` element reacts.

While sidebar 1 remains at a fixed width, the article scales larger and smaller, continuing to occupy 50 percent of the width no matter what size the screen becomes.

Observe how the text wraps within the element as it changes in size. Note how it stops scaling when the frame shrinks to the size of the largest word and how the box model diagram juts out of the element below certain widths. Do you think these issues would affect the page's readability or usability?

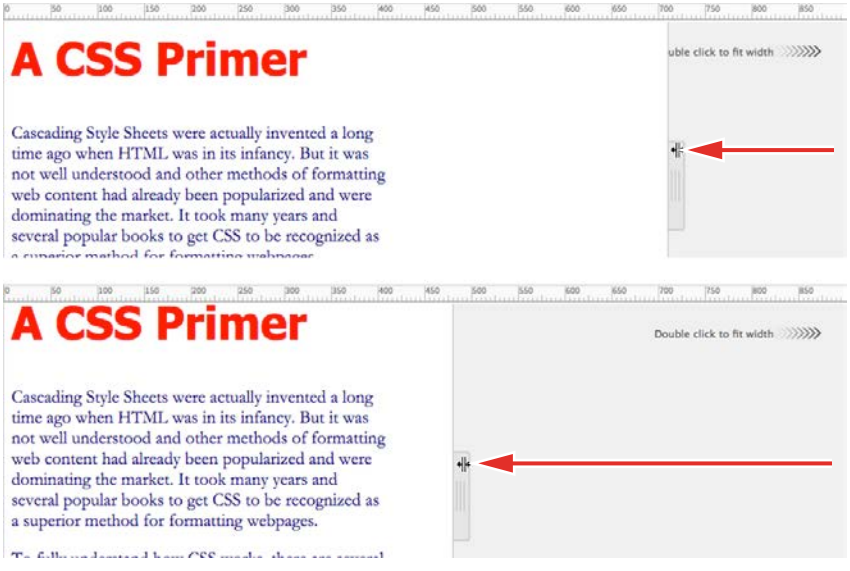
Many designers forgo the use of percentage-based settings for these reasons. Although they like that the containers scale to fit the browser window, they'd prefer that it would stop scaling before it affects the content detrimentally. This is one of the reasons for which the properties `min-width` and `max-width` were created.

Note: Hyphenation is still under development and not fully supported. The specification has been defined in CSS3 and is waiting for full support by all browsers.

4 Add the highlighted notation in the rule `article { width:50%; min-width:400px; }` and save the file.

The `min-width` property prevents the element from scaling smaller than 400 pixels. Note that the `min-width` specification unit is defined in pixels. This is important, because when combining the `width` setting with `min-width` or `max-width`, you must use differing measurement units or only one of the specifications will be applied. To see the effect of the `min-width` specification, you need to use the entire screen.

5 Switch to Live view and refresh the display, if necessary.
Drag the Scrubber left and right.
Observe how the `<article>` element reacts.



The <article> displays at 50 percent of the screen width as you scale it smaller. When the screen becomes narrower than 800 pixels, the <article> stops scaling and remains at a fixed width of 400 pixels. To limit scaling at the upper end, you can also add the max-width property.

6 Switch to Split view.
Insert the highlighted notation as shown here:
`article { width:50%; min-width:400px; max-width:700px; }`
Save the file.

7 Refresh the Live view display and drag the Scrubber left and right.

The <article> displays at 50 percent of the screen only between the widths of 800 pixels to 1400 pixels, where it stops scaling at the dimensions specified.

It's all relative, or not

Ems and exes are kind of a hybrid cross between fixed and relative systems. The em is a measurement that is more familiar to print designers. It's based on the size of the typeface and font being used. In other words, use a large font and the em gets bigger; use a small font and the em gets smaller. It even changes based on whether the font is a condensed or expanded face.

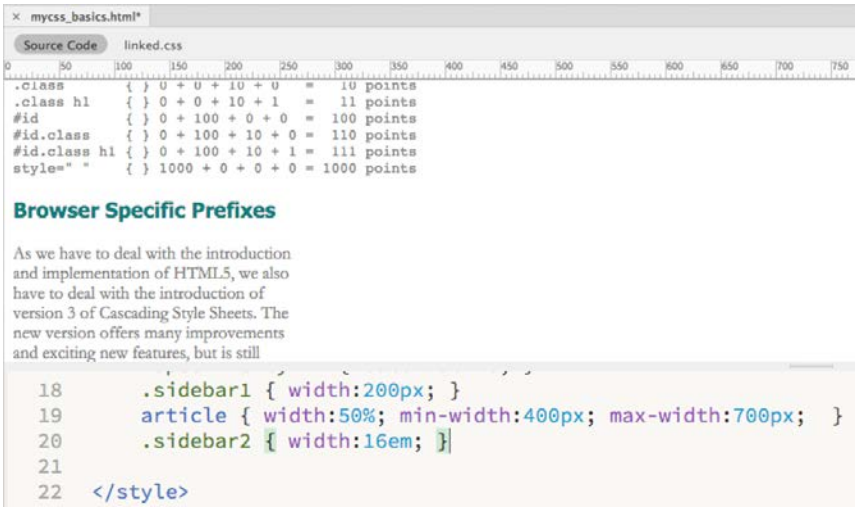
This type of measurement is typically used to build text-based components, such as navigation menus where you want the structure to adapt to user actions that may increase or decrease the font size on a site but where you don't want the text to reflow.

1 If necessary, open **mycss_basics.html** in Split view.
Add the rule `.sidebar2 { width:16em; }` and save the page.

2 Refresh the Live view display, if necessary.

Although ems are considered a relative measure, they behave differently than widths set in percentages. Unfortunately, em measurements don't display properly in Design view; to see the exact relationship, you'll need to use Live view.

3 Switch to Live view, if necessary.
Scroll down to view `sidebar2`.

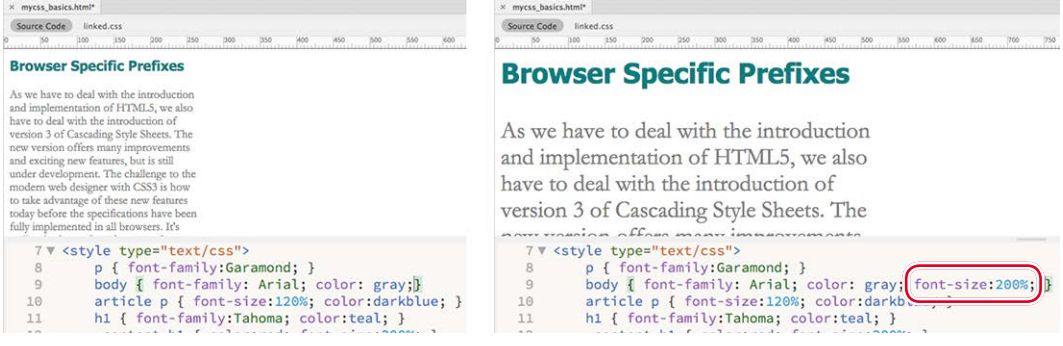


The "Browser Specific Prefixes" heading should exactly fit the width of the element without breaking to two lines or leaving any extra space.

4 Refresh the display if necessary and drag the Scrubber left and right.

The element seems to react like an element with a fixed measurement; it doesn't change size as you make the screen bigger and smaller. That's because the "relative" nature of ems isn't based on screen size but on the font size.

5 Switch to Split view.
Add the highlighted code as shown here:
`body { font-family:Arial; color:gray; font-size:200%; }`
Refresh the display.



All the text on the page scales 200 percent. The text in `sidebar1` and the `<article>` element have to wrap to fit within the container. On the other hand, `sidebar2` actually scales larger to accommodate the bigger text. Note how using ems also preserves the line endings in that element.

There's one small caveat when you use ems: The measurement is based on the base font size of the nearest parent element, which means it can change any time the font size changes within the element's HTML structure. And you always need to remember that the relative size of the em is influenced by inheritance.

6 Change the `font-size` property for `body` to **100%** and save the file.

```
7 ▼ <style type="text/css">
8       p { font-family:Garamond; }
9       body { font-family: Arial; color: gray; font-size:100%; }
10      article p { font-size:120%; color:darkblue; }
```

The text in the page returns to its previous size.

By assigning various widths to the containers, you're setting up the basic structure for creating a multicolumn layout. The next step would be to start repositioning these containers on the page. But CSS positioning can be tricky; lots of factors can affect the display and interaction of these elements. Before you move the containers to their final positions, it may help you to understand these techniques better if you first apply some borders and background effects to make them easier to see.

Borders and backgrounds

Each element can feature four individually formatted borders (top, bottom, left, and right). These are handy for creating boxes around paragraphs, headings, or containers, but there's no requirement to use all four borders on every element. For example, you can place them at the top or bottom (or both) of paragraphs in place of `<hr />` (horizontal rule) elements or to create custom bullet effects.

Borders

It's easy to create different border effects using CSS.

1 If necessary, open **mycss_basics.html** in Split view and observe the CSS and HTML code.

You can assign borders to text or containers.

2 Add the following rule and properties to the `<style>` section:

```
article section {
border-top:solid 10px #000;
border-left:solid 2px #ccc; }
```

Note: Remember that you can add the CSS rules with or without the line breaks and indents.

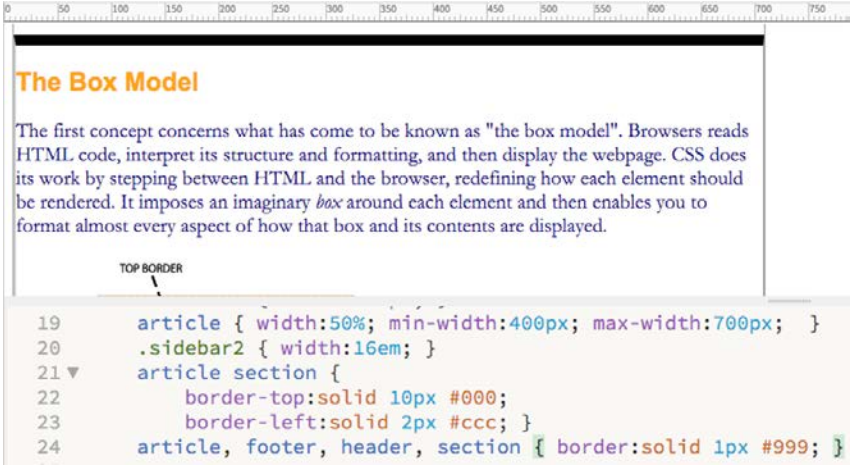
3 Refresh the display, if necessary.

A custom border appears at the top and left sides of each `section` in the `<article>` element. The border gives a visible indication of the width and height of the HTML section. At the moment, the borders sit uncomfortably close to the text, but don't worry—we'll address this issue later. Let's apply borders to the other main containers now.

4 Add the following rule and properties:

```
article, footer, header, section { border:solid 1px #999; }
```

5 Refresh the display, if necessary.



A 1-pixel gray border appears on the `article`, `footer`, `header`, and every `section` element on the page, including the ones already formatted in step 2.

6 Save the file.

Note: By combining these tags in one selector and using CSS shorthand, this new rule saves at least 11 lines of code.

Note: The rule created in step 2 is more specific than the one created in step 4. So, only the right and bottom borders will be affected in the nested elements.

Tip: When an element inherits properties from another rule that are undesirable, you may need to add a new rule (or new property in an existing rule) specifically to turn off that styling.

As described earlier, it's not unusual for an element to be formatted by two or more rules. Even though the `<section>` elements nested in the `<article>` were styled with borders on the top and left by the first rule, they are now inheriting the 1-pixel border from the second rule, for the right and bottom sides.

It's also important to emphasize that there is no extraneous markup within the actual content; all the effects are generated by CSS code alone. That means you can quickly adjust, turn on and off effects, and move the content easily without having to worry about graphical elements or extra code cluttering it up. You keep your code sleek and efficient.

Now that you can see the outer boundaries of each container, keep a wary eye on each to see how they react to the CSS styling created in the upcoming exercises.

Backgrounds

By default, all element backgrounds are transparent, but CSS lets you format them with colors, images, or both. If you use both, the image will appear above, or in front of, the color. This behavior allows you to use an image with a transparent or translucent background to create layered graphical effects. If you use an opaque image and it fills the visible space or is set to repeat like wallpaper, it may obscure the color entirely.

1 Open **mycss_basics.html** from the lesson03bonus folder in Split view. Observe the CSS and HTML code.

Backgrounds can be assigned to any visible block or inline element. If you want the background to appear behind the entire webpage, assign it to the `body` element.

2 In the `body` rule, add **`background-color:#ccc;`** and refresh the display.

The background of the document window is now filled with light gray.

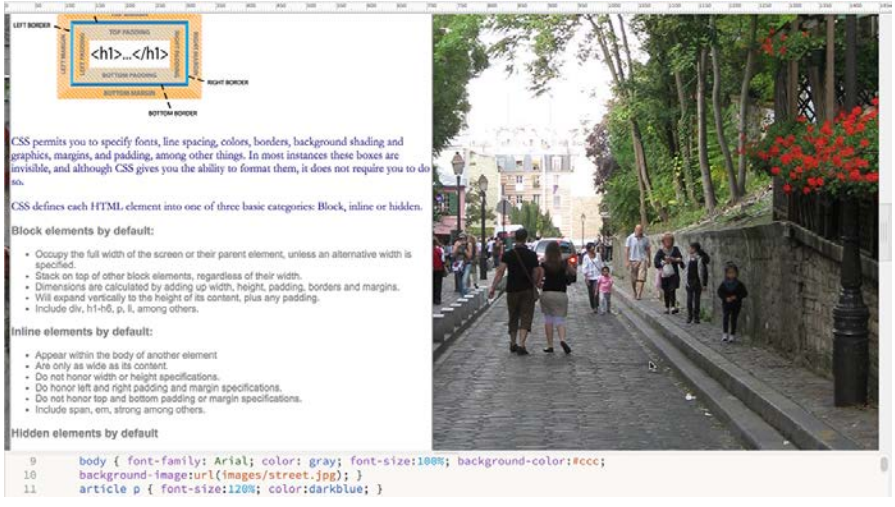
Some background colors may make content hard to read. Studies show that white is still the best color on which to read text. Let's fill the main text containers with a white background.

3 In the `article, footer, header, section` rule, add the property **`background-color:#fff;`** and refresh the display.

Each of the targeted containers now displays a white background.

Websites have been using graphical backgrounds for many years. In the beginning, images or icons were used to create wallpaper effects. Since the connection speeds of the Internet were much slower in those days, these images typically were very small. Today, large images are becoming a popular way to apply a custom look to websites everywhere. Let's experiment with both types.

4 In the `body` rule, add the property **`background-image:url(images/street.jpg);`** and refresh the display.



A photograph of a street scene appears in the background of the page, the exact composition of which is determined by the width and height of your document window.

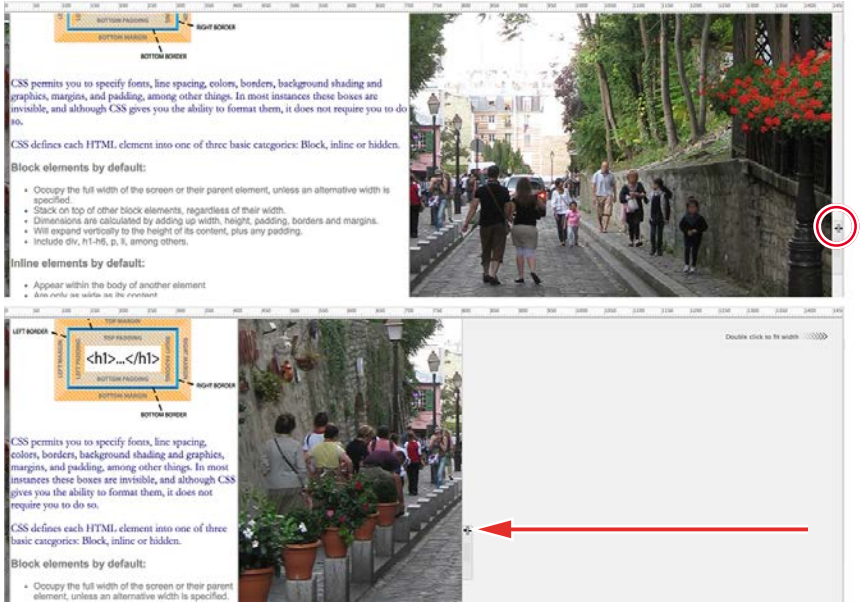
Note: Background and background-image can both be declared at the same time. The background-image will appear above or in front of the background settings, but depending on the image, both settings may be visible at the same time.

By default, background images display at 100 percent of their original size and attach at the upper-left corner of the screen. If you check the dimensions of the image, you'll see that it's 1900 pixels by 2500 pixels and nearly 4 MB in size. That's big enough to fit almost any type of screen, but some aspects of a background image can't be seen properly in Design view.

5 If necessary, activate Live view and click the button to fill the entire document window with the webpage display.

Live view renders web content for a browser-like environment.

6 Drag the Scrubber left and right and observe how the background image reacts to the changing window size.



The background image does not respond to the changing window. As the window narrows, the right side of the image is hidden and off-center. It would look better if the image scaled along with the window.

7 Switch to Code view. Add the property **`background-size:100% auto;`** to the `body` rule and refresh the display.

Now the background image scales automatically to fit the width of the browser screen. But if you scroll down the page, you will notice that the image repeats vertically once you get to the bottom of the image. CSS allows you to control many aspects of the background image to make it look and respond better on a variety of devices. By adding another property, you can make the image stay fixed in one location.

Note: Background-size, like many CSS properties, can be specified in fixed or relative measurements. The specification can be expressed in one or two values. When you use two values, the first applies the width, the second the height.

8 Add the following properties to the body rule and switch to Live view:

```
background-position: center center;
background-attachment:fixed;
```

9 Drag the Scrubber left and right and scroll the screen down to view the page content.

The background image no longer scrolls along with the content and remains centered, regardless of how the window changes.

In the past, using such a large image on a webpage would have been avoided. But as more people access the Internet with high-speed connections, this type of large background image is becoming more popular. Although the user has to download an image that may be several megabytes, they have to do it only once, and the image will be cached on the visitor's hard drive in most cases for those who regularly visit the site or visit multiple pages.

Yet for many designers, an image of this size will never be acceptable. They know how large images can cause undesirable delays as webpages and resources download. Instead, these designers resort to a method that is still very popular: creating a background pattern or wallpaper by using a simple, smaller graphic. Such graphics can be only a few kilobytes yet can produce beautiful, sophisticated effects.

10 Switch to Code view.

Change the body rule as highlighted:

```
background-color:#acd8b6;
background-image:url(images/stripes.png);
```

This new graphic is 15 pixels by 100 pixels and only 2 KB. Even on a slow connection, this graphic will download almost instantly. However, its small size requires a few changes to the styling.

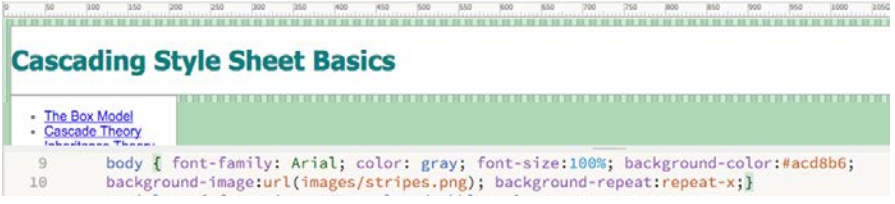
11 Delete the following properties from the body rule:
~~background-size:100% auto;~~
~~background-position:center center;~~
~~background-attachment:fixed;~~

```
7 ▼ <style type="text/css">
8       p { font-family:Garamond; }
9       body { font-family: Arial; color: gray; font-size:100%; background-color:#acd8b6;
10      background-image:url(images/stripes.png); }
```

By default, background images are intended to repeat vertically and horizontally. But before you allow this to happen, make the following changes to get a better idea of what's going to happen.

12 Add the following property to the body rule:
background-repeat:repeat-x;

13 Switch to Split view.



You can see the graphic repeating horizontally along the x-axis at the top of the page. To make the graphic repeat vertically, you can make a simple change.

14 Edit the highlighted property in the body rule:
background-repeat:repeat-**y;**

The graphic now repeats vertically along the left edge of the screen. Background graphics repeat horizontally and vertically by default. If you delete the repeat property altogether, the wallpaper effect would be seamless.

15 Delete the ~~background-repeat:repeat-y;~~ property from the body rule and refresh the display.

Without the `repeat-y` property, the background repeats vertically *and* horizontally by default; the graphic is now repeating across the entire page.

16 Save the file.

Combined with the right color or gradient, these types of backgrounds are both attractive and efficient in the use of resources. The choice is yours. No matter what type you choose, be sure to fully test any background treatments. In some applications, CSS background specifications are not fully supported or are supported inconsistently.

Positioning

As you have already learned, block elements generate their own line or paragraph breaks; inline elements appear at the point of insertion. CSS can break all these default constraints and let you place elements almost anywhere you want them to be.

As with other object formatting, positioning can be specified in relative terms (such as left, right, center, and so on) or by absolute coordinates measured in pixels, inches, centimeters, or other standard measurement systems. Using CSS, you can even layer one element above or below another to create amazing graphical effects. By using positioning commands carefully, you can create a variety of page layouts, including popular multicolumn designs.

1 If necessary, open **mycss_basics.html** in Split view.
Observe the CSS and HTML code.

The file contains headings, paragraph text, and various HTML5 container elements partially formatted by CSS. The rules, created in previous exercises, set specific sizes on the main container's `sidebar1`, `sidebar2`, and `article` elements so they no longer take up the entire width of the screen. In the same way—using only CSS—you can control the positioning of all these elements on the page. There are several ways to do this, but the *float* method is by far the most popular. The options for the `float` property are `left`, `right`, and `none` and can have a dramatic effect on the positioning of the targeted elements. If no property is actually set by CSS, the default styling is `none`.

2 Create the following rule:

```
.sidebar1, article, .sidebar2 { float:right; }
```

3 Save the file. If necessary, switch to Live view and refresh the display.
Maximize the program window to the full size of the computer display.

Note: The `float` property can also be applied individually to the existing rules for these elements. By combining selectors it is easier to update the property by simply editing one value.

Depending on the width of your screen, the `aside` and `article` elements now display horizontally, side by side, from right to left in the document window. By using `float:right`, the elements display from right to left on the screen. Notice how `sidebar1` appears on the far right, followed by `article` and then by `sidebar2`. If you change the float value, you can produce the opposite effect.

4 Change the property `float:right` to **`float:left`** and refresh the display.

All three elements reverse direction, now starting on the left.

As you can see, the `float` property takes an element out of the normal HTML flow. By setting widths smaller than the default 100 percent on the sidebars and the article, `float` allows these block elements to behave in a totally different manner and *share* the space with each other. And `float` is also a dynamic property in that it reacts to the width of the document window.

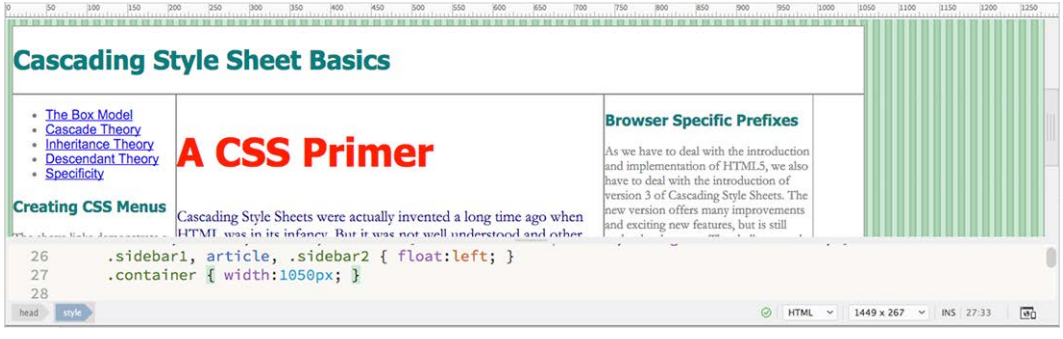
5 Drag the Scrubber left and right to change the width of the display.

As the window narrows, there's no longer enough space to accommodate the widths of the individual elements. They are forced to move down into the open space below. When the window gets wider, the elements move up to share the space again. This type of behavior allows websites to display rows of items that adapt automatically to any type of screen, no matter how big or small. As you make the window larger and smaller, you may notice that the footer element slips up into the row with the other elements if your screen is wide enough.

You may be saying, "But the footer isn't floated!" and you'd be right. This is one of the consequences of using `float`: The first subsequent *non*floated element will share the space with any floated ones if it doesn't have a specific width or other property that prevents it. The first nonfloated block element will honor all the width, margin, and padding settings of any floated element and then occupy 100 percent of the space left over. At times you may take advantage of this behavior to create some multicolumn layouts. However, for this layout, you want the footer to stay at the bottom.

You can force an element to move, or position, itself differently by simply setting a specific width to the parent container, or to the children themselves, that will preclude them from sharing the available space. At the moment, the combined width of the three floated elements is less than the width of the whole screen, which allows the footer to sneak in if the screen is wide enough. To prevent this from happening, you need to limit the amount of space the floated elements can use.

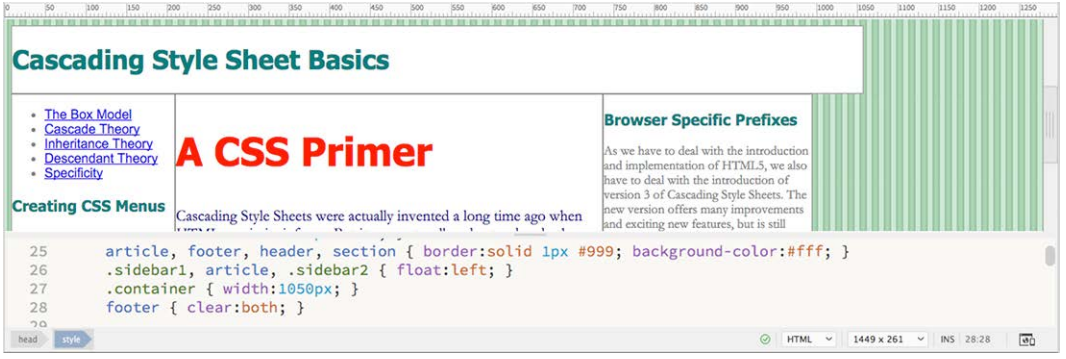
6 Add a new rule `.container { width:1050px; }` and refresh the display. If necessary, switch to Live view.



The `div` element displays at a width of 1050 pixels.

Now there's not enough room for the elements to float all the way across the screen, but the footer is still trying to sneak into the layout at the bottom of `sidebar2`. There is a CSS property specifically designed to keep this from happening.

7 Create a new rule `footer { clear:both; }` and refresh the display.



The footer moves down to the bottom of the page again.

Although you fixed this situation with a single CSS property, it's not always that easy. Unfortunately, as powerful as CSS positioning seems to be, it is the one aspect of CSS that is most prone to misinterpretation by the browsers in use today. Commands and formatting that work fine in one browser can be translated differently or totally ignored—with tragic results—in another. In fact, specifications that work fine on one page of your website can even fail on another page containing a different mix of code elements.

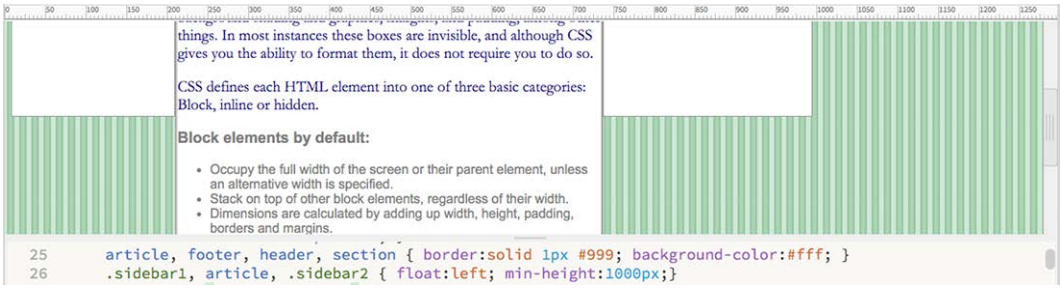
Note: For this current layout, the footer could suffice with `clear:left`. But the footer is an element that should clear all potential content elements on the page; therefore, I chose to use `clear:both`.

Height

Sidebar 1, sidebar 2, and the `article` element contain different amounts of content and display at different heights. You could set a fixed height for all three that would work on this page, but what dimension would work on the other pages of the site?

Height is not specified as frequently on the web as width is. That's mainly because the height of an element or component is usually determined by the content contained within it, combined with any assigned margins and padding. Setting a fixed height can often result in undesirable effects, such as truncating, or clipping, text or pictures. If you must set a height, the safest way is to use the `min-height` property.

1 In the `.sidebar1, article, .sidebar2` rule, add the **`min-height:1000px;`** property and refresh the display.



Sidebars 1 and 2 now display at a minimum height of 1,000 pixels but will grow as needed to match the length of their content.

On a different page, setting a common element height might work, but with the amount of content in this article, a common element height isn't really a solution to the problem at hand. The main issue is the graphical background on the page. It makes it pretty obvious that sidebars 1 and 2 are shorter than the `article` element.

One answer would be to ditch the page background graphic altogether and apply a background color that matches the one used in the elements. Or you could simply apply a matching background color to the `<div>` containing the layout itself.

2 In the rule `.container` add the following property: **`background-color:#fff;`**

The background color for all the elements is now identical.

3 Save the file.

For all intents and purposes, the heights of sidebar 1, sidebar 2, and the `article` element are irrelevant. If not for the gray borders applied to each, you'd have no idea how tall the elements are at all. Problem solved.

Margins and padding

Margins control the space outside the boundaries, or borders, of an element; padding controls the space inside an element, between its content and its border. It doesn't matter whether the borders are visible; the effective use of such spacing is vital in the overall design of your webpage.

Margins

Margins are used to separate one block element from another.

1 If necessary, open **mycss_basics.html**.

Margins and padding don't always render properly in Design view.

2 If necessary, switch to the full Live view display.
Observe the page layout and styling.

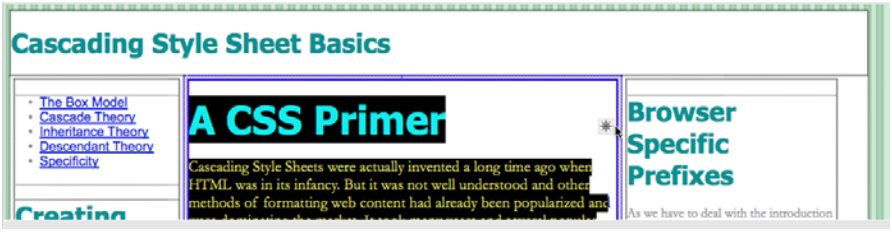
The page displays a header, three columns, and a footer. The column elements are touching each other, and the text within each column is touching the edges of each container.

3 In the rule `.sidebar1, article, .sidebar2` add the property `margin:5px;` and refresh the display.

Note: In most cases, horizontal margins between two adjacent objects combine to increase the total spacing. On the other hand, only the larger of the two settings is honored for vertical margins between two adjacent elements.

The new property adds 5 pixels of spacing outside the borders of each targeted element. Although Live view gives you a more accurate browser-like display, Design view still has a few tricks up its sleeve.

4 Switch to Design view.
Click an edge of the `<article>` element.



Design view highlights the element and displays a hashed pattern to provide a visual representation of the margin specifications. In HTML 4, the align attribute was used to align elements left, right, or center. This attribute was deprecated in HTML5, and CSS has no specific method for centering block elements. Until something better comes along, you can use margins to center content on the screen.

5 In the `.container` rule add the following property:
`margin: 20px auto;`

6 Refresh the display.

The element centers in the window. In this CSS shorthand notation, the `auto` value applies equal amounts of spacing to the left and right sides of the article.

You've added spacing between the elements. Now, let's add some spacing inside the elements too.

Padding

The text inside the layout is touching the borders within the containers. Padding puts spacing between the content and an element's border.

1 In the rule `.sidebar1, article, .sidebar2` add the property
`padding: 5px;`

2 In the rule `footer` add the property **`padding:10px;`**

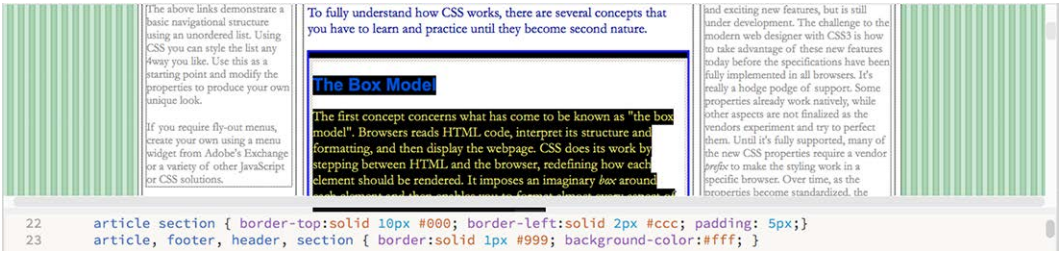
3 Refresh the display, if necessary.

The text inside the targeted elements is now spaced away from all four element borders.

Did you notice that the subsections in the article element didn't inherit the padding themselves? The text is still touching the border of its element. This may be confusing, because earlier we discussed how styling is inherited from a parent element. Although it works for many properties, inheritance isn't a guarantee, for several reasons.

Inheriting text formatting from a parent element makes a lot of sense if you think about it. You would want the text to have the same font, size, and color. But is the same logic true for *structural* properties, such as width, height, padding, and margins? For example, if you set a width of 300 pixels on the container `div` element, would you want all the child elements, such as sidebar 1 and 2 and the `article`, to inherit the same width too? For this reason and others, padding and margins are a few of the properties that are not inherited via CSS.

4 In the rule `article section` add the property **`padding: 5px;`**

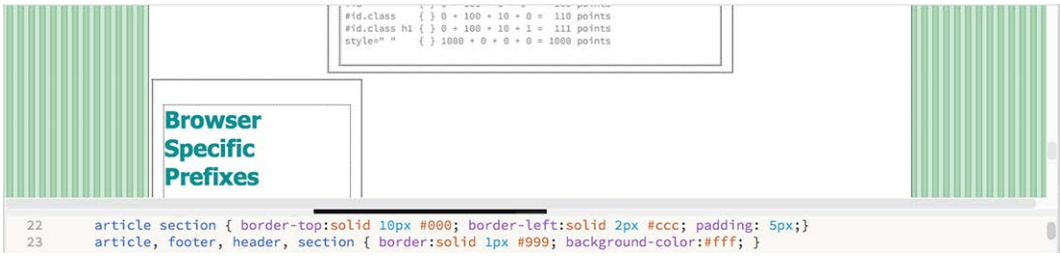


Tip: In Design view you may need to click in the document window to force Dreamweaver to refresh the display.

5 Click the edge of the box model `<section>` to select it.

Using Design view you can see 5 pixels of padding appear within it. Did you notice how the element grew slightly larger when you applied padding? Margins, padding, and even borders increase the width and height of an element. Add too much, and you might break your carefully constructed layout.

6 In the rule `.sidebar1, article, .sidebar2` change the padding value to `15px;` and refresh the display.



Increasing the padding has broken the layout. Sidebar 2 no longer fits beside the article element. It will move down the page until it can find enough space. This type of conflict happens frequently in web design. The constant interplay between the elements and the CSS can produce undesirable results like this. Luckily, in this case, the solution is as simple as the cause.

7 In the rule `.sidebar1, article, .sidebar2` change the padding value back to `5px;` and refresh the display.

Reducing the padding value has fixed the layout and allows sidebar 2 to display side by side with the other elements again.

Normalization

Now you know that margins and padding—among other things—affect the overall size of an element. You have to factor these specifications into the design of your page components. But apart from the properties applied directly by the style sheet, don't forget that some elements feature default margin specifications too. In fact, you can see these very settings in the extra space appearing above and below the headings and paragraphs on the current page.

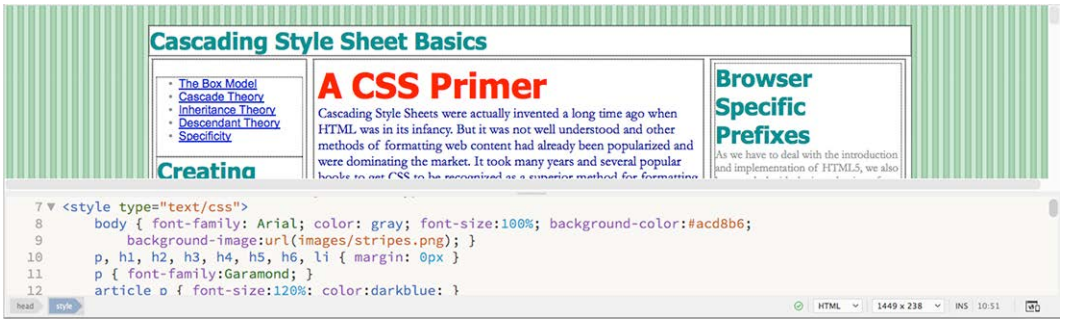
Many designers abhor these default specifications, especially because they vary so much among browsers. They start off most projects by purposely removing, or resetting, these settings using a technique called *normalization*. In other words, they declare a list of common elements and reset their default specifications to more desirable, consistent settings.

1 In the CSS section of the page code, move the `body` rule to the top of the style sheet.

Since the styles in the `body` rule are inherited by all elements on the page, most designers place it high in the style sheet, if not in the first position. Next should come rules designed to normalize basic elements.

2 After the body rule add the following rule:

```
p, h1, h2, h3, h4, h5, h6, li { margin: 0px }
```



3 Save the file.

As you learned in Lesson 3, "CSS Basics," the comma (,) means "and" in CSS syntax, indicating that you want to format all the tags listed. This rule resets the default margin settings for all the listed elements. It's important that this rule be placed as high in the style sheet as possible, typically after the body rule, if one exists. That way, you can still add margins to specific instances of any of the targeted elements later in the style sheet without worrying about conflicts with this rule.

4 Refresh the display.

The text elements now display without the default spacing.

Using zero margins may be a bit extreme for your tastes, but you get the picture. As you become more comfortable with CSS and webpage design, you can develop your own default specifications and implement them using CSS.

The page has come a long way from the beginning of this lesson. Let's put some final tweaks on the design to match the original finished page.

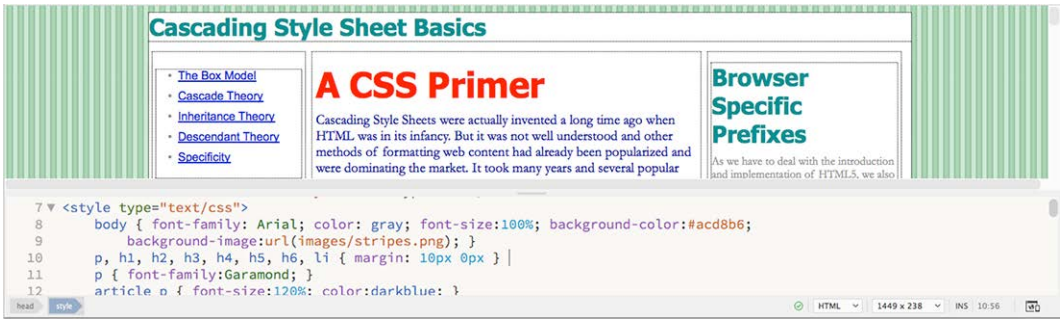
Final touches

You're nearly finished; the page needs only a few last touches to make it match the design you saw at the beginning of the lesson.

1 In the rule `article, footer, header, section` delete the property ~~`border:solid 1px #999;`~~

In Design view, the elements will display a light gray-colored border, but the black border displayed earlier is now gone.

2 In the rule `p, h1, h2, h3, h4, h5, h6, li` add the highlighted value in the property: `margin: 10px 0px;`



Note: Unless otherwise specified, you can add rules anywhere in the style sheet.

3 Create the following rule:

```
header { padding:30px;
border-bottom:2px solid #000;
text-align:center;  }
```

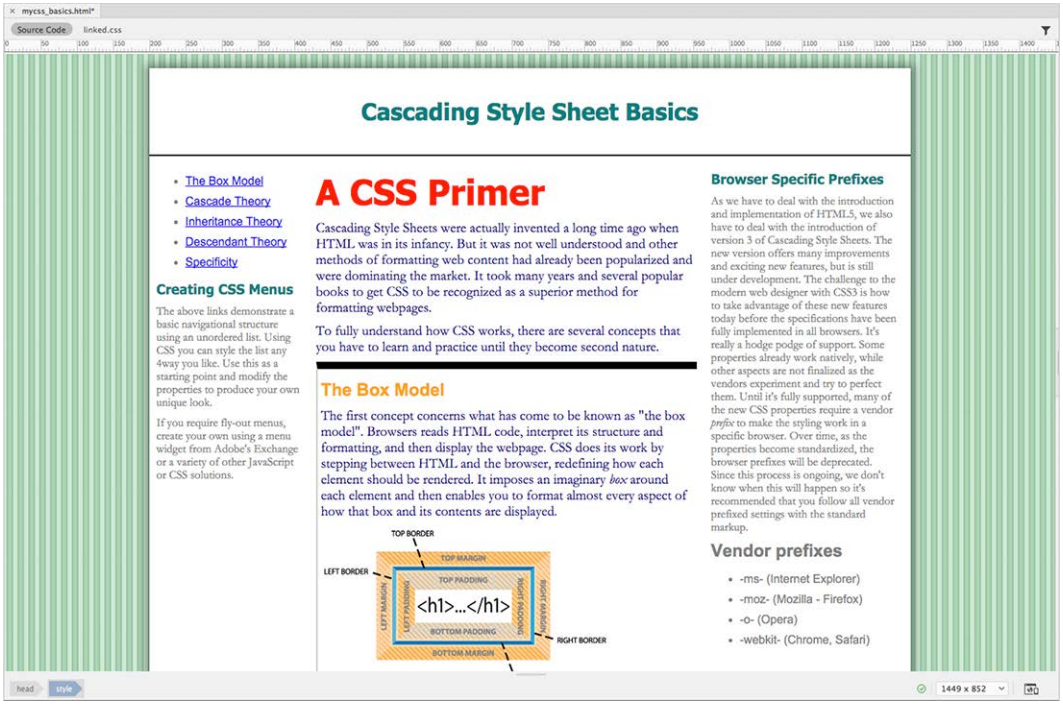
The final style you need to add is a new CSS3 feature.

4 Add the following properties to the `.container` rule:

```
-webkit-box-shadow: 0px 0px 20px 5px rgba(0,0,0,0.40);
box-shadow: 0px 0px 20px 5px rgba(0,0,0,0.40);
```

Advanced CSS properties cannot be seen in Design view.

5 Save the file. Switch to Live view and refresh the display.



The sample page is complete. Congratulations! You successfully styled an entire page with CSS.