4. Introduction to Cascading Style Sheets™ (CSS): Part 1

> ***Fashions fade, style is eternal.***
>
> **—Yves Saint Laurent**
>
> ***How liberating to work in the margins, outside a central perception.***
>
> **—Don DeLillo**

Objectives

In this chapter you'll:

- Control a website's appearance with style sheets.
- Use a style sheet to give all the pages of a website the same look and feel.
- Use the `class` attribute to apply styles.
- Specify the precise font, size, color and other properties of displayed text.
- Specify element backgrounds and colors.
- Understand the box model and how to control margins, borders and padding.
- Use style sheets to separate presentation from content.

Outline

4.1. Introduction

In Chapters 2–3, we introduced HTML5 for marking up information to be rendered in a browser. In this chapter and Chapter 5, we shift our focus to formatting and presenting information. To do this, we use a W3C technology called **Cascading Style Sheets** 3 (**CSS3**) that allows you to specify the *presentation* of elements on a web page (e.g., fonts, spacing, sizes, colors, positioning) *separately* from the document's *structure and content* (section headers, body text, links, etc.). This **separation of structure from presentation** simplifies maintaining and modifying web pages, especially on large-scale websites. In Chapter 5, we introduce many new features in CSS3.

HTML5 was designed to specify the content and structure of a document. Though HTML5 has some attributes that control presentation, *it's better not to mix presentation with content*. If a website's presentation is determined entirely by a style sheet, you can simply swap in a new style sheet to completely change the site's appearance.

The W3C provides a CSS3 code validator at jigsaw.w3.org/css-validator/. This tool can help you make sure that your code is correct and will work on CSS3-compliant browsers. We've run this validator on every CSS3/HTML5 document in this book. For more CSS3 information, check out our CSS3 Resource Center at www.deitel.com/css3.

4.2. Inline Styles

You can declare document styles inline in the HTML5 markup, in embedded style sheets or in separate CSS files. This section presents **inline styles** that declare an individual element's format using the HTML5 attribute `style`. Inline styles *override* any other styles applied using the techniques we discuss later in the chapter. Figure 4.1 applies inline styles to `p` elements to *alter* their font size and color.

SOFTWARE ENGINEERING OBSERVATION 4.1

Inline styles do not truly separate presentation from content. To apply similar styles to multiple elements, use embedded style sheets *or* external style sheets, *introduced later in this chapter.*

```
1 <!DOCTYPE html>
2
3 <!-- Fig. 4.1: inline.html -->
4 <!-- Using inline styles -->
5 <html>
6    <head>
```

```
7     <meta charset = "utf-8">
8     <title>Inline Styles</title>
9   </head>
10   <body>
11     <p>This text does not have any style applied to it.</p>
12
13     <!-- The style attribute allows you to declare -->
14     <!-- inline styles. Separate multiple -->
15     <!-- style properties with a semicolon. -->
16     <p style = "font-size: 20pt;">This text has the
17       <em>font-size</em> style applied to it, making it 20pt.
18     </p>
19
20     <p style = "font-size: 20pt; color: deepskyblue;">
21       This text has the <em>font-size</em> and
22       <em>color</em> styles applied to it, making it
23       20pt and deep sky blue.</p>
24   </body>
25 </html>
```

Fig. 4.1. Using inline styles.

The first inline style declaration appears in line 16. Attribute `style` specifies an element's style. Each **CSS property** (`font-size` in this case) is fol-

lowed by a colon and a value. In line 16, we declare this particular `p` element to use a 20-point font size.

Line 20 specifies the two properties, `font-size` and **`color`**, separated by a semicolon. In this line, we set the given paragraph's `color` to `deepskyblue`. Hexadecimal codes may be used in place of color names. Figure 4.2 contains the HTML standard color set. We provide a list of extended hexadecimal color codes and color names in Appendix B. You can also find a complete list of HTML standard and extended colors at www.w3.org/TR/css3-color/.

Color name	Value	Color name	Value
aqua	#00FFFF	navy	#000080
black	#000000	olive	#808000
blue	#0000FF	purple	#800080
fuchsia	#FF00FF	red	#FF0000
gray	#808080	silver	#C0C0C0
green	#008000	teal	#008080
lime	#00FF00	yellow	#FFFF00
maroon	#800000	white	#FFFFFF

Fig. 4.2. HTML standard colors and hexadecimal RGB values.

4.3. Embedded Style Sheets

A second technique for using style sheets is **embedded style sheets**, which enable you to *embed* a CSS3 document in an HTML5 document's `head` section. Figure 4.3 creates an embedded style sheet containing four styles.

```
1 <!DOCTYPE html>
2
3 <!-- Fig. 4.3: embedded.html -->
4 <!-- Embedded style sheet. -->
5 <html>
6    <head>
7       <meta charset = "utf-8">
```

```
8     <title>Embedded Style Sheet</title>
9
10    <!-- this begins the style sheet section -->
11    <style type = "text/css">
12       em      { font-weight:  bold;
13                color: black; }
14       h1      { font-family: tahoma, helvetica, sans-serif; }
15       p       { font-size: 12pt;
16                font-family: arial, sans-serif; }
17       .special { color: purple ; }
18    </style>
19  </head>
20  <body>
21    <!-- this attribute applies the .special style class -->
22    <h1 class = "special">Deitel & Associates, Inc.</h1>
23
24    <p>Deitel & Associates, Inc. is an authoring and
25       corporate training organization specializing in
26       programming languages, Internet and web technology,
27       iPhone and Android app development, and object
28       technology education.</p>
29
30    <h1>Clients</h1>
31    <p class = "special"> The company's clients include many
32       <em>Fortune 1000 companies</em>, government agencies,
33       branches of the military and business organizations.</p>
34  </body>
35 </html>
```

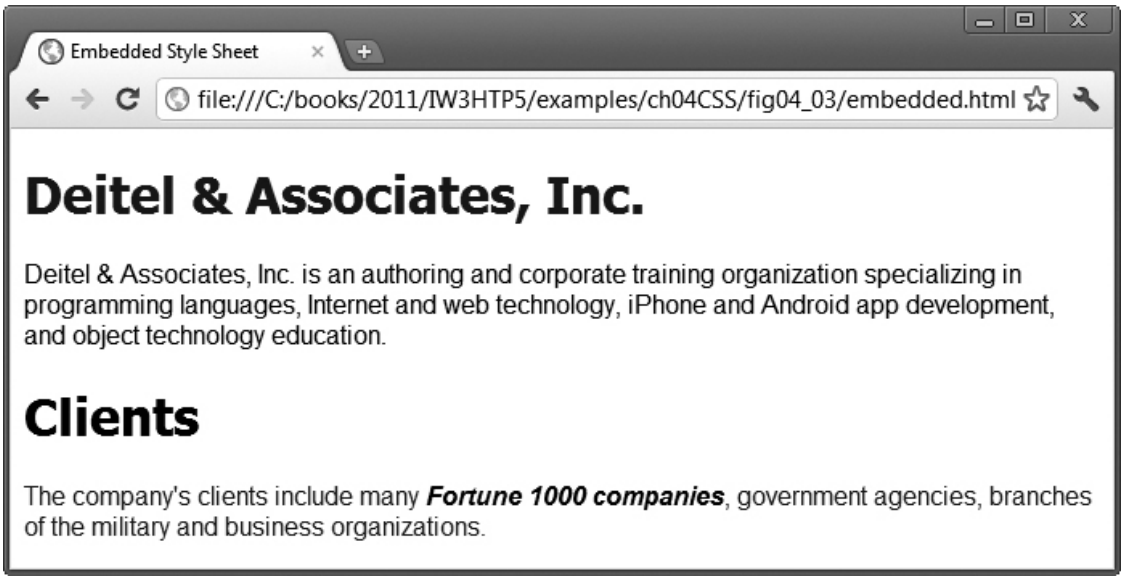

Fig. 4.3. Embedded style sheet.

The `style` Element and MIME Types

The `style` element (lines 11–18) defines the *embedded style sheet*. Styles placed in the `head` apply to matching elements wherever they appear in the `body`. The `style` element's `type` attribute specifies the **MIME (Multipurpose Internet Mail Extensions) type** that describes the `style` element's content. CSS documents use the MIME type `text/css`. As of HTML5, the default `type` for a `style` element is `"text/css"`, so this attribute is no longer needed—we kept it here because you'll see this used in legacy HTML code. Figure 4.4 lists common MIME types used in this book. For a complete list of MIME types, visit:

www.w3schools.com/media/media_mimeref.asp

MIME type	Description
`text/css`	CSS documents
`image/png`	PNG images
`text/javascript`	JavaScript markup
`text/plain`	Plain text
`image/jpeg`	JPEG image
`text/html`	HTML markup

Fig. 4.4. A few common MIME types.

The style sheet's body (lines 12–17) declares the **CSS rules** for the style sheet. To achieve the separation between the CSS3 code and the HTML5 that it styles, we'll use a **CSS selector** to specify the elements that will be styled according to a rule. Our first rule (line 12) begins with the selector `em`, which selects all **`em` elements** in the document. An **`em` element** indicates that its contents should be *emphasized*. Browsers usually render `em` elements in an *italic* font. Each rule's body is enclosed in *curly braces* (`{` and `}`). CSS rules in embedded style sheets use the same syntax as inline styles; the property name is followed by a colon (`:`) and the property value. Multiple properties are separated by semicolons (`;`). The **`font-weight`** property in line 12 specifies the "boldness" of text. Possible values are `bold`, `normal` (the default), `bolder` (bolder than `bold` text) and `lighter` (lighter than `normal` text). Boldness also can be specified with multiples of 100, from 100 to 900. Text specified as `normal` is equivalent to `400`, and `bold` text is equivalent to `700`. However, many systems do not have fonts that can scale with this level of precision, so using these numeric values might not display the desired effect.

In this example, all `em` elements will be displayed in a bold black font. We also apply styles to all `h1` and `p` elements (lines 14–16).

Style Classes

Line 17 declares a selector for a **style class** named `special`. Style-class declarations are preceded by a period (`.`). They define styles that can be applied to *any* element. In this example, class `special` sets `color` to `purple`. We'll show how to apply a style class momentarily. You can also declare `id` selectors. If an element in your page has an `id`, you can declare a selector of the form `#` *elementId* to specify that element's style.

`font-family` Property

The **`font-family`** property (line 14) specifies the name of the font to use. Not all users have the same fonts installed on their computers, so CSS allows you to specify a comma-separated list of fonts to use for a particular style. The browser attempts to use the fonts in the order in which they appear in the list. It's advisable to end a font list with a **generic font family** name in case the other fonts are not installed on the user's computer (Fig.

4.5). In this example, if the `tahoma` font is not found on the system, the browser will look for the `helvetica` font. If neither is found, the browser will display its default `sans-serif` font.

Generic font families	Examples
serif	times new roman, georgia
sans-serif	arial, verdana, futura
cursive	script
fantasy	critter
monospace	courier, fixedsys

Fig. 4.5. Generic font families.

`font-size` Property

Property **`font-size`** (line 15) specifies a 12-point font. Other possible measurements in addition to `pt` (point) are introduced in Section 4.4. Relative values— `xx-small`, `x-small`, `small`, `smaller`, `medium`, `large`, `larger`, `x-large` and `xx-large` —also can be used. Generally, *relative `font-size` values are preferred over points, because an author does not know the specific measurements of each client's display*. Relative values permit more flexible viewing of web pages. For example, users can change font sizes the browser displays for readability.

A user may view a web page on a handheld device with a small screen. Specifying a fixed font size (such as 18pt) prevents the browser from scaling fonts. A relative font size, such as `large` or `larger`, allows the browser to determine the *actual size* of the text displayed. Using relative sizes also makes pages more accessible to users with disabilities. Users with impaired vision, for example, may configure their browser to use a larger default font, upon which all relative sizes are based. Text that the author specifies to be `smaller` than the main text still displays in a smaller size font. Accessibility is an important consideration—in 1998, Congress passed the Section 508 Amendment to the Rehabilitation Act of 1973, mandating that websites of federal government agencies be accessible to disabled users. For more information, visit www.access-board.gov/508.htm.

Applying a Style Class

Line 22 uses the HTML5 attribute `class` in an `h1` element to apply a style class—in this case, the class named `special` (declared with the `.special` selector in the style sheet on line 17). When the browser renders the `h1` element, the text appears on screen with the properties of both an `h1` element (`tahoma` , helvetica or `sans-serif` font defined in line 14) and the `.special` style class applied (the color `purple` defined in line 17). The browser also still applies its own default style to the `h1` element—the header is displayed in a large font size. Similarly, all `em` elements will still be italicized by the browser, but they will also be bold as a result of lines 12–13.

The formatting rules for both the `p` element and the `.special` class are applied to the text in lines 31–33. In many cases, the styles applied to an element (the **parent** or **ancestor element**) also apply to the element's *nested elements* (**child** or **descendant elements**). The `em` element nested in the `p` element in line 32 **inherits** the style from the `p` element (namely, the 12-point font size in line 15) but retains its italic style. So styles defined for the paragraph and *not* defined for the `em` element are still applied to this `em` element that's nested in the `p` element. Multiple values of one property can be set or inherited on the same element, so the browser must reduce them to one value for that property per element before they're rendered. We discuss the rules for resolving these conflicts in the next section.

4.4. Conflicting Styles

Styles may be defined by a **user**, an **author** or a **user agent**. A user is a person viewing your web page, you're the author—the person who writes the document—and the user agent is the program used to render and display the document (e.g., a web browser).

- Styles **cascade** (and hence the term "Cascading Style Sheets"), or flow together, such that the ultimate appearance of elements on a page results from combining styles defined in several ways.

- Styles defined by the user take precedence over styles defined by the user agent.

- Styles defined by authors take precedence over styles defined by the user.

Most styles defined for parent elements are also **inherited** by child (nested) elements. This makes sense for most styles, such as font properties, but there are certain properties that you don't want to be inherited. For example, the `background-image` property allows you to set an image as the background of an element. If the `body` element is assigned a background image, we don't want the same image to be in the background of every element in the body of our page. Instead, the `background-image` property of all child elements retains its default value of `none`. In this section, we discuss the rules for *resolving conflicts* between *styles defined for elements* and styles inherited from parent and ancestor elements.

Figure 4.3 contains an example of inheritance in which a child `em` element inherits the `font-size` property from its parent `p` element. However, in Fig. 4.3, the child `em` element has a `color` property that *conflicts with* (i.e., has a different value than) the `color` property of its parent `p` element. Properties defined for child and descendant elements have a higher **specificity** than properties defined for parent and ancestor elements. Conflicts are resolved in favor of properties with a *higher* specificity, so the child's styles take precedence. Figure 4.6 illustrates examples of inheritance and specificity.

```
1  <!DOCTYPE html>
2
3  <!-- Fig. 4.6: advanced.html -->
4  <!-- Inheritance in style sheets. -->
5  <html>
6     <head>
7        <meta charset = "utf-8">
8        <title>More Styles</title>
9        <style type = "text/css">
10          body     { font-family: arial, helvetica, sans-serif; }
```

```
11        a.nodec { text-decoration: none; }
12        a:hover { text-decoration: underline; }
13        li em   { font-weight: bold; }
14        h1, em  { text-decoration: underline; }
15        ul      { margin-left: 20px; }
16        ul ul   { font-size: .8em; }
17     </style>
18  </head>
19  <body>
20     <h1>Shopping list for Monday:</h1>
21
22     <ul>
23        <li>Milk</li>
24        <li>Bread
25           <ul>
26              <li>white bread</li>
27              <li>Rye bread</li>
28              <li>Whole wheat bread</li>
29           </ul>
30        </li>
31        <li>Carrots</li>
32        <li>Yogurt</li>
33        <li>Pizza <em>with mushrooms</em></li>
34     </ul>
35
36     <p><em>Go to the</em>
37        <a class = "nodec" href = "http://www.deitel.com">
38           Grocery store</a>
39     </p>
40  </body>
41 </html>
```

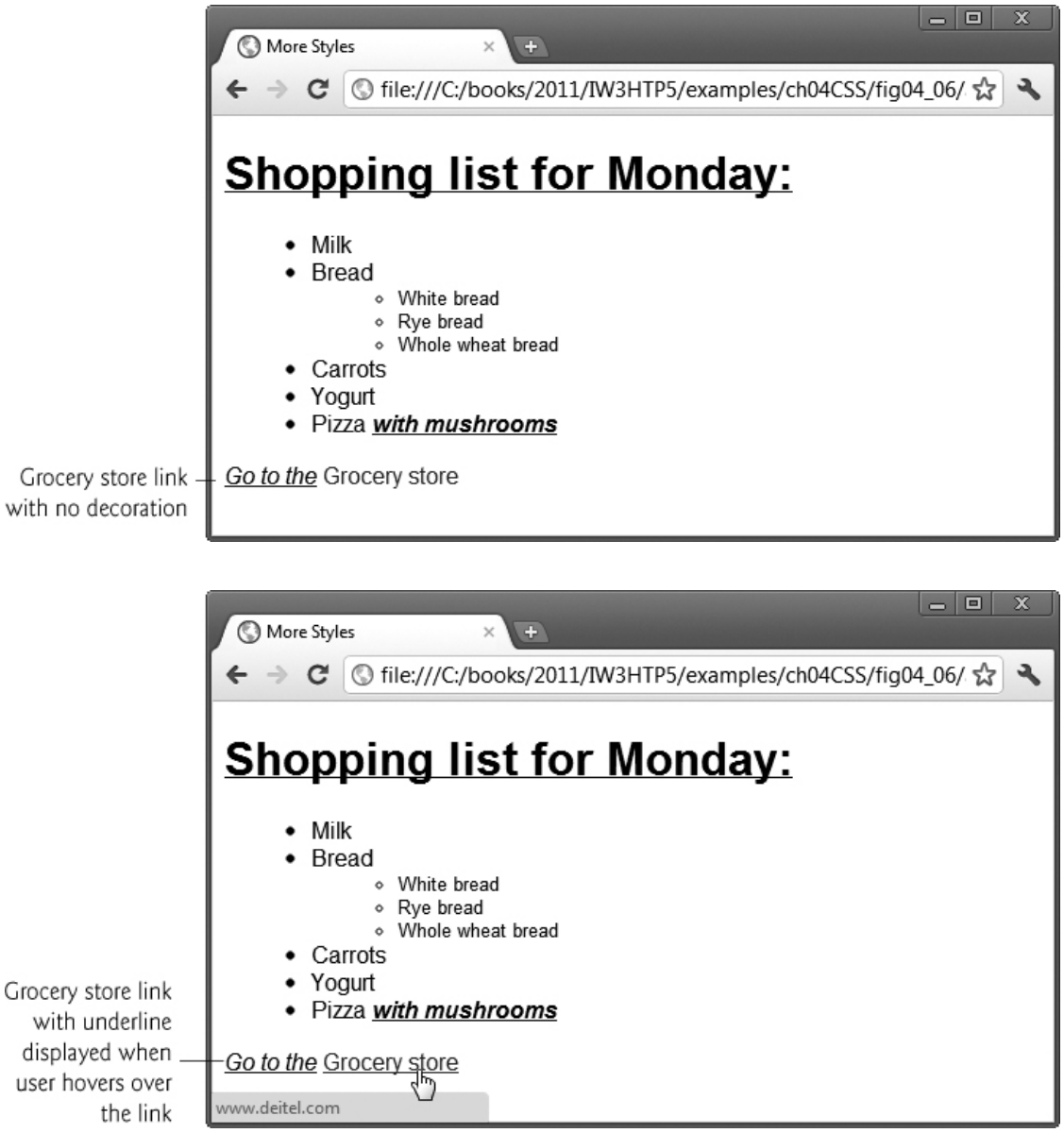

Fig. 4.6. Inheritance in style sheets.

Line 11 applies property `text-decoration` to all `a` elements whose `class` attribute is set to `nodec` (line 37). The `text-decoration` property applies **decorations** to text in an element. By default, browsers underline the text of an `a` (anchor) element. Here, we set the `text-decoration` property to `none` to indicate that the browser should *not* underline hyperlinks. Other possible values for `text-decoration` include `overline`, `line-through` and `underline`. The `.nodec` appended to `a` is a *more specific* class selector; this style in line 11 applies only to `a` (anchor) elements that specify the `nodec` in their `class` attribute.

PORTABILITY TIP 4.1

To ensure that your style sheets work in various web browsers, test them on many client web browsers, and use the W3C CSS Validator.

Line 12 specifies a style for `hover`, which is a **pseudo-class**. Pseudo-classes give you access to information that's not declared in the document, such as whether the mouse is hovering over an element or whether the user has previously clicked (visited) a particular hyperlink. The **`hover` pseudo-class** is activated dynamically when the user moves the mouse cursor over (that is, hovers over) an element. Pseudo-classes are separated by a *colon* (with no surrounding spaces) from the name of the element to which they're applied.

COMMON PROGRAMMING ERROR 4.1

Including a space before or after the colon separating a pseudo-class from the name of the element to which it's applied prevents the pseudo-class from being applied properly.

Line 13 causes all `em` elements that are children of `li` elements to be bold. In the screen output of Fig. 4.6, **Go to the** (contained in an `em` element in line 36) does not appear bold, because the `em` element is *not* nested in an `li` element. However, the `em` element containing **with mushrooms** (line 33) *is* nested in an `li` element, so it's formatted in bold. The syntax for applying rules to multiple elements is similar. In line 14, we separate the selectors with a *comma* to apply an *underline* style rule to all `h1` *and* all `em` elements.

Line 15 assigns a 20-pixel left margin to all `ul` elements. We'll discuss the `margin` properties in detail in Section 4.10. A pixel is a **relative-length**

measurement—it varies in size, based on screen resolution. Other relative lengths include `em` (which, as a measurement, means the font's uppercase *M* height—the most frequently used font measurement), `ex` (the font's *x*-height—usually set to a lowercase *x*'s height) and percentages (e.g., `font-size: 50%`). To set an element to display text at 150 percent of its default text size, you could use

```
font-size: 1.5em
```

or

```
font-size: 150%
```

Other units of measurement available in CSS are **absolute-length measurements**—i.e., units that do *not* vary in size based on the system. These units are `in` (inches), `cm` (centimeters), `mm` (millimeters), `pt` (points; 1 `pt` = 1/72 `in`) and `pc` (picas; 1 `pc` = 12 `pt`). Line 16 specifies that all nested unordered lists (`ul` elements that are descendants of `ul` elements) are to have font size `.8em`. [*Note:* When setting a style property that takes a measurement (e.g. `font-size`, `margin-left`), no units are necessary if the value is zero.]

GOOD PROGRAMMING PRACTICE 4.1

Whenever possible, use relative-length measurements. If you use absolute-length measurements, your document may not scale well on some client browsers (e.g., smartphones).

4.5. Linking External Style Sheets

Style sheets are a convenient way to create a document with a uniform theme. With **external style sheets** (i.e., *separate* documents that contain only CSS rules), you can provide a *uniform look and feel* to an entire website (or to a portion of one). You can also *reuse* the same external style sheet across multiple websites. Different pages on a site can all use the same style sheet. When changes to the styles are required, you need to

modify only a single CSS file to make style changes across *all* the pages that use those styles. This concept is sometimes known as **skinning**. While embedded style sheets separate content from presentation, both are still contained in a *single* file, preventing a web designer and a content author from conveniently working in parallel. External style sheets solve this problem by separating the content and style into separate files.

Figure 4.7 presents an external style sheet. Lines 1–2 are **CSS comments**. These may be placed in any type of CSS code (i.e., inline styles, embedded style sheets and external style sheets) and always start with `/*` and end with `*/`. Text between these delimiters is ignored by the browser. The rules in this external style sheet are the same as those in the embedded style sheet in Fig. 4.6, lines 10–16.

```
1  /* Fig. 4.7: styles.css */
2  /* External style sheet */
3  body     { font-family: arial, helvetica, sans-serif; }
4  a.nodec  { text-decoration: none; }
5  a:hover  { text-decoration: underline; }
6  li em    { font-weight: bold; }
7  h1, em   { text-decoration: underline; }
8  ul       { margin-left: 20px; }
9  ul ul    { font-size: .8em; }
```

Fig. 4.7. External style sheet.

Figure 4.8 contains an HTML5 document that references the external style sheet. Lines 9–10 show a `link` element that uses the `rel` attribute to specify a **relationship** between the current document and another document. Here, we declare the linked document to be a `stylesheet` for this document. The `type` attribute specifies the related document's MIME type as `text/css`. The `href` attribute provides the style sheet document's URL. Using just the file name `styles.css`, as we do here, indicates that `styles.css` is in the same directory as `external.html`. The rendering results are the same as in Fig. 4.6.

```
<!DOCTYPE html>

<!-- Fig. 4.8: external.html -->
<!-- Linking an external style sheet. -->
<html>
   <head>
      <meta charset = "utf-8">
      <title>Linking External Style Sheets</title>
      <link rel = "stylesheet" type = "text/css"
         href = "styles.css">
   </head>
   <body>
      <h1>Shopping list for <em>Monday</em>:</h1>

      <ul>
         <li>Milk</li>
         <li>Bread
            <ul>
               <li>white bread</li>
               <li>Rye bread</li>
               <li>Whole wheat bread</li>
            </ul>
         </li>
         <li>Carrots</li>
         <li>Yogurt</li>
         <li>Pizza <em>with mushrooms</em></li>
      </ul>

      <p><em>Go to the</em>
         <a class = "nodec"  href = "http://www.deitel.com">
            Grocery store</a>
      </p>
   </body>
</html>
```

Fig. 4.8. Linking an external style sheet.

4.6. Positioning Elements: Absolute Positioning, `z-index`

Before CSS, controlling element positioning in HTML documents was difficult—the browser determined positioning. CSS introduced the **`position`** property and a capability called **absolute positioning**, which gives

you greater control over how document elements are displayed. Figure 4.9 demonstrates absolute positioning.

```
<!DOCTYPE html>

<!-- Fig. 4.9: positioning.html -->
<!-- Absolute positioning of elements. -->
<html>
   <head>
      <meta charset = "utf-8">
      <title>Absolute Positioning</title>
      <style type = "text/css">
         .background_image { position: absolute;
                             top: 0px;
                             left: 0px;
                             z-index: 1; }
         .foreground_image { position: absolute;
                             top: 25px;
                             left: 100px;
                             z-index: 2; }
         .text             { position: absolute;
                             top: 25px;
                             left: 100px;
                             z-index: 3;
                             font-size: 20pt;
                             font-family: tahoma, geneva, sans-serif; }
      </style>
   </head>
   <body>
      <p><img src = "background_image.png" class = "background_image"
         alt = "First positioned image" /></p>

      <p><img src = "foreground_image.png" class = "foreground_image"
         alt =       "Second positioned image" /></p>

```

```
33      <p class = "text">Positioned Text</p>
34    </body>
35  </html>
```

Fig. 4.9. Absolute positioning of elements.

Normally, elements are positioned on the page in the order in which they appear in the HTML5 document. Lines 10–13 define a style called `background_image` for the first `img` element (`background_image.png`) on the page. Specifying an element's `position` as `absolute` removes the element from the normal flow of elements on the page, instead positioning it according to the distance from the `top`, `left`, `right` or `bottom` margins of its **containing block-level element**. This means that it's displayed on its own line and has a **virtual box** around it. Some examples of block-level elements include `section`, `div`, `p` and heading elements (`h1` through `h6`). Here, we position the element to be `0` pixels away

from both the `top` and `left` margins of its containing element. In line 27, this style is applied to the image, which is contained in a `p` element.

The **`z-index`** property allows you to *layer overlapping elements*. Elements that have *higher* `z-index` values are displayed in *front* of elements with *lower* `z-index` values. In this example, `.background_image` has the lowest `z-index` (`1`), so it displays in the background. The `.foreground_image` CSS rule (lines 14–17) gives the circle image (`foreground_image.png`, in lines 30–31) a `z-index` of `2`, so it displays in front of `background_image.png`. The `p` element in line 33 is given a `z-index` of `3` in line 21, so its content (`Positioned Text`) displays in front of the other two. If you do not specify a `z-index` or if elements have the same `z-index` value, the elements are placed from background to foreground in the order in which they're encountered in the document. The default `z-index` value is 0.

4.7. Positioning Elements: Relative Positioning, `span`

Absolute positioning is not the only way to specify page layout. Figure 4.10 demonstrates relative positioning, in which elements are positioned *relative to other elements*.

Setting the `position` property to `relative`, as in class `super` (lines 15–16), lays out the element on the page and *offsets* it by the specified `top`, `bottom`, `left` or `right` value. Unlike absolute positioning, relative positioning keeps elements in the general flow of elements on the page, so positioning is relative to other elements in the flow. Recall that `ex` (line 16) is the *x*-height of a font, a relative-length measurement typically equal to the height of a lowercase *x*. Class `super` (lines 15–16) lays out the text at the end of the sentence as superscript, and class `sub` (lines 17–18) lays out the text as subscript relative to the other text. Class `shiftleft` (lines 19–20) shifts the text at the end of the sentence left and class `shiftright` (lines 21–22) shifts the text right.

```
1 <!DOCTYPE html>
2
3 <!-- Fig. 4.10: positioning2.html -->
```

```
4  <!-- Relative positioning of elements. -->
5  <html>
6    <head>
7      <meta charset = "utf-8">
8      <title>Relative Positioning</title>
9      <style type = "text/css">
10       p       { font-size: 1.3em;
11                 font-family: verdana, arial, sans-serif; }
12       span     { color: red;
13                 font-size: .6em;
14                 height: 1em; }
15       .super    { position: relative;
16                 top: -1ex; }
17       .sub      { position: relative;
18                 bottom: -1ex; }
19       .shiftleft  { position: relative;
20                 left: -1ex; }
21       .shiftright { position: relative;
22                 right: -1ex; }
23     </style>
24   </head>
25   <body>
26     <p>The text at the end of this sentence
27       <span class = "super">is in superscript</span>.</p>
28
29     <p>The text at the end of this sentence
30       <span class = "sub">is in subscript</span>.</p>
31
32     <p>The text at the end of this sentence
33       <span class = "shiftleft">is shifted left</span>.</p>
34
35     <p>The text at the end of this sentence
36       <span class = "shiftright">is shifted right</span>.</p>
37   </body>
38 </html>
```

Fig. 4.10. Relative positioning of elements.

Inline and Block-Level Elements

We introduce the `span` element in line 27. Lines 12–14 define the CSS rule for all `span` elements in this example. The `span`'s `height` determines how much vertical space it will occupy. The `font-size` determines the size of the text inside the `span`.

Element `span` is a **grouping element**—by default, it does not apply any formatting to its contents. Its primary purpose is to apply CSS rules or `id` attributes to a section of text. Element `span` is an **inline-level element**—it does not change the flow of elements in the document. Examples of inline elements include `span`, `img`, `a`, `em` and `strong`. The `div` element is also a grouping element, but it's a block-level element. We'll discuss inline and block-level elements in more detail in Section 4.10.

4.8. Backgrounds

CSS provides control over the backgrounds of block-level elements. CSS can set a background color or add background images to HTML5 elements. Figure 4.11 adds a corporate logo to the bottom-right corner of the document. This logo stays *fixed* in the corner even when the user scrolls up or down the screen.

`background-image` Property

The **`background-image`** property (line 10) specifies the image URL for the image `logo.png` in the format `url(`*fileLocation*`)`. You can also set the **`background-color`** property (line 14) in case the image is not found (and to fill in areas the image does not cover).

```
1  <!DOCTYPE html>
2
3  <!-- Fig. 4.11: background.html -->
4  <!-- Adding background images and indentation -->
5  <html>
6     <head>
7        <meta charset = "utf-8">
8        <title>Background Images</title>
9        <style type = "text/css">
10          body  { background-image: url(logo.png);
11                  background-position: bottom right;
12                  background-repeat: no-repeat;
13                  background-attachment: fixed;
14                  background-color: lightgrey; }
15          p     { font-size: 18pt;
16                  color: Darkblue;
17                  text-indent: 1em;
18                  font-family: arial, sans-serif; }
19          .dark { font-weight: bold; }
20       </style>
21    </head>
22    <body>
23       <p>
24          This example uses the background-image,
25          background-position and background-attachment
26          styles to place the <span class = "dark">Deitel
27          & Associates, Inc.</span> logo in the
28          bottom-right corner of the page. Notice how the logo
29          stays in the proper position when you resize the
30          browser window. The background-color fills in where
```

```
31          there is no image.
32       </p>
33    </body>
34 </html>
```

Fig. 4.11. Adding background images and indentation.

`background-position` Property

The **`background-position`** property (line 11) places the image on the page. The keywords `top`, `bottom`, `center`, `left` and `right` are used individually or in combination for vertical and horizontal positioning. You can position an image using lengths by specifying the horizontal length followed by the vertical length. For example, to position the image as *horizontally centered* (positioned at 50 percent of the distance across the screen) and 30 pixels from the top, use

background-position: 50% 30px;

`background-repeat` Property

The **`background-repeat`** property (line 12) controls background image **tiling**, which places *multiple copies* of the image next to each other to fill

the background. Here, we set the tiling to `no-repeat` to display only one copy of the background image. Other values include `repeat` (the default) to tile the image *vertically and horizontally*, `repeat-x` to tile the image only *horizontally* or `repeat-y` to tile the image only *vertically*.

`background-attachment: fixed` Property

The next property setting, **`background-attachment: fixed`** (line 13), fixes the image in the position specified by **`background-position`**. Scrolling the browser window will *not* move the image from its position. The default value, `scroll`, moves the image as the user scrolls through the document.

`text-indent` property

Line 17 uses the **`text-indent`** property to indent the first line of text in the element by a specified amount, in this case `1em`. You might use this property to create a web page that reads more like a novel, in which the first line of every paragraph is indented.

`font-style` property

Another CSS property that formats text is the **`font-style`** property, which allows you to set text to `none`, `italic` or `oblique` (`oblique` is simply more slanted than italic—the browser will default to `italic` if the system or font does not support oblique text).

4.9. Element Dimensions

In addition to positioning elements, CSS rules can specify the actual *dimensions* of each page element. Figure 4.12 demonstrates how to set the dimensions of elements.

```
1 <!DOCTYPE html>
2
3 <!-- Fig. 4.12: width.html -->
4 <!-- Element dimensions and text alignment. -->
```

```
<html>
  <head>
    <meta charset = "utf-8">
    <title>Box Dimensions</title>
    <style type = "text/css">
       p { background-color: lightskyblue;
           margin-bottom: .5em;
           font-family: arial, helvetica, sans-serif; }
     </style>
  </head>
  <body>
    <p style = "width: 20%">Here is some
      text that goes in a box which is
      set to stretch across twenty percent
      of the width of the screen.</p>

    <p style = "width: 80%; text-align: center">
      Here is some CENTERED text that goes in a box
      which is set to stretch across eighty percent of
      the width of the screen.</section>

    <p style = "width: 20%; height: 150px; overflow: scroll">
      This box is only twenty percent of
      the width and has a fixed height.
      What do we do if it overflows? Set the
      overflow property to scroll!</p>
  </body>
</html>
```

Fig. 4.12. Element dimensions and text alignment.

Specifying the `width` and `height` of an Element

The inline style in line 16 illustrates how to set the **`width`** of an element on screen; here, we indicate that the `p` element should occupy 20 percent of the screen width. If not specified, the width will fit the size of the browser window. The height of an element can be set similarly, using the **`height`** property. The `width` and `height` values also can be specified as relative or absolute lengths. For example,

```
width: 10em
```

sets the element's width to 10 times the font size. This works only for block-level elements.

`text-align` Property

Most elements are left-aligned by default, but this alignment can be altered. Line 21 sets text in the element to be `center` aligned; other values for the **`text-align`** property include `left` and `right`.

`overflow` Property and Scroll Bars

In the third `p` element, we specify a percentage width and a pixel height. One problem with setting *both* dimensions of an element is that the content inside the element can exceed the set boundaries, in which case the element is simply made large enough for all the content to fit. However, in line 26, we set the **`overflow`** property to `scroll`, a setting that adds scroll bars if the text overflows the boundaries.

4.10. Box Model and Text Flow

All *block-level* HTML5 elements have a *virtual box* drawn around them, based on what is known as the **box model**. When the browser renders an element using the box model, the content is surrounded by **padding**, a **border** and a **margin** (Fig. 4.13).

Fig. 4.13. Box model for block-level elements.

CSS controls the border using three properties: **`border-width`**, **`border-color`** and **`border-style`**. We illustrate these properties in Fig. 4.14.

```
1  <!DOCTYPE html>
2
3  <!-- Fig. 4.14: borders.html -->
4  <!-- Borders of block-level elements. -->
5  <html>
6     <head>
7        <meta charset = "utf-8">
8        <title>Borders</title>
9        <style type = "text/css">
```

```
10          div    { text-align: center;
11                   width: 50%;
12                   position: relative;
13                   left: 25%;
14                   border-width: 6px; }
15          .thick  { border-width: thick; }
16          .medium { border-width: medium; }
17          .thin   { border-width: thin; }
18          .solid  { border-style: solid; }
19          .double { border-style: double; }
20          .groove { border-style: groove; }
21          .ridge  { border-style: ridge; }
22          .dotted { border-style: dotted; }
23          .inset  { border-style: inset; }
24          .outset { border-style: outset; }
25          .dashed { border-style: dashed; }
26          .red    { border-color: red; }
27          .blue   { border-color: blue; }
28       </style>
29    </head>
30    <body>
31       <div class = "solid">Solid border</div><hr>
32       <div class = "double">Double border</div><hr>
33       <div class = "groove">Groove border</div><hr>
34       <div class = "ridge">Ridge border</div><hr>
35       <div class = "dotted">Dotted border</div><hr>
36       <div class = "inset">Inset border</div><hr>
37       <div class = "thick dashed">Thick dashed border</div><hr>
38       <div class = "thin red solid">Thin red solid border</div><hr>
39       <div class = "medium blue outset">Medium blue outset border</div>
40    </body>
41 </html>
```

Fig. 4.14. Borders of block-level elements.

The `border-width` property may be set to any valid CSS length (e.g., `em`, `ex`, `px`) or to the predefined value of `thin`, `medium` or `thick`. The **`border-color` property** sets the color. [*Note:* This property has different meanings for different border styles—e.g., some display the border color in multiple shades.] The `border-style` options are `none`, `hidden`, `dotted`, `dashed`, `solid`, `double`, `groove`, `ridge`, `inset` and `outset`. Borders `groove` and `ridge` have opposite effects, as do `inset` and `outset`. When border-style is set to `none`, no border is rendered. Each border property may be set for an individual side of the box (e.g., `border-top-style` or `border-left-color`).

Floating Elements

We've seen with absolute positioning that it's possible to remove elements from the normal flow of text. Floating allows you to move an element to one side of the screen; other content in the document then *flows around*

the floated element. Figure 4.15 demonstrates how floating elements and the box model can be used to control the layout of an entire page.

Looking at the HTML5 code, we can see that the general structure of this document consists of a `header` and two main `section`s. Each section contains an `h1` subheading and a paragraph of text.

Block-level elements (such as `section`s) render with a *line break* before and after their content, so the `header` and two `section`s will render vertically one on top of another. In the absence of our styles, the `h1`s that represent our subheadings would also stack vertically on top of the text in the `p` tags. However, in line 24 we set the `float` property to `right` in the class `floated`, which is applied to the `h1` headings. This causes each `h1` to float to the right edge of its containing element, while the paragraph of text will flow around it.

```
1  <!DOCTYPE html>
2
3  <!-- Fig. 4.15: floating.html -->
4  <!-- Floating elements. -->
5  <html>
6     <head>
7        <meta charset = "utf-8">
8        <title>Flowing Text Around Floating Elements</title>
9        <style type = "text/css">
10          header   { background-color: skyblue;
11                     text-align: center;
12                     font-family: arial, helvetica, sans-serif;
13                     padding: .2em; }
14          p        { text-align: justify;
15                     font-family: verdana, geneva, sans-serif;
16                     margin: .5em; }
17          h1       { margin-top: 0px; }
18          .floated { background-color: lightgrey;
19                     font-size: 1.5em;
20                     font-family: arial, helvetica, sans-serif;
```

```
21                padding: .2em;
22                margin-left: .5em;
23                margin-bottom: .5em;
24                float: right;
25                text-align: right;
26                width: 50%; }
27          section  { border: 1px solid skyblue; }
28       </style>
29    </head>
30    <body>
31       <header><img src = "deitel.png" alt = "Deitel" /></header>
32       <section>
33          <h1 class = "floated">Corporate Training and Authoring</h1>
34          <p>Deitel & Associates, Inc. is an internationally
35             recognized corporate training and authoring organization
36             specializing in programming languages, Internet/web
37             technology, iPhone and Android app development and
38             object technology education. The company provides courses
39             on Java, C++, C#, Visual Basic, C, Internet and web
40             programming, Object Technology and iPhone and Android
41             app development.</p>
42       </section>
43       <section>
44          <h1 class = "floated">Programming Books and Videos</h1>
45          <p>Through its publishing
46             partnership with Pearson, Deitel & Associates,
47             Inc. publishes leading-edge programming textbooks,
48             professional books and interactive web-based and DVD
49             LiveLessons video courses.</p>
50       </section>
51    </body>
52 </html>
```

Fig. 4.15. Floating elements.

`margin` and `padding` Properties

Line 16 assigns a margin of `.5em` to all paragraph elements. The **`margin` property** sets the space between the outside of an element's border and all other content on the page. Line 21 assigns `.2em` of padding to the floated `h1` s. The **`padding` property** determines the distance between the content inside an element and the inside of the element's border. Margins for individual sides of an element can be specified (lines 17, 22 and 23) by using the properties **`margin-top`** , **`margin-right`** , **`margin-left`** and **`margin-bottom`** . Padding can be specified in the same way, using **`padding-top`** , **`padding-right`** , **`padding-left`** and **`padding-bottom.`** To see the effects of margins and padding, try putting the margin and padding properties inside comments and observing the difference.

In line 27, we assign a border to the `section` boxes using a shorthand declaration of the border properties, which allow you to define all three border properties in one line. The syntax for this shorthand is

border: *width style color*

Our border is one pixel thick, `solid`, and the same color as the `background-color` property of the `header` (line 10). This allows the border to blend with the `header` and makes the page appear as one box with a line dividing its sections.

4.11. Media Types and Media Queries

CSS **media types** allow you to decide what a page should look like, depending on the kind of media being used to display the page. The most common media type for a web page is the **`screen` media type**, which is a standard computer screen. Other media types in CSS include **`handheld`**, **`braille`**, **`speech`** and **`print`**. The `handheld` medium is designed for mobile Internet devices such as smartphones, while `braille` is for machines that can read or print web pages in braille. `speech` styles allow you to give a speech-synthesizing web browser more information about the content of a page. The `print` media type affects a web page's appearance when it's printed. For a complete list of CSS media types, see

http://www.w3.org/TR/REC-CSS2/media.html#media-types

Media types allow you to decide how a page should be presented on any one of these media without affecting the others. Figure 4.16 gives a simple classic example that applies one set of styles when the document is *viewed on all media (including screens) other than a printer*, and another when the document is *printed*. To see the difference, look at the screen captures below the paragraph or use the **Print Preview** feature in your browser if it has one.

```
1 <!DOCTYPE html>
2
3 <!-- Fig. 4.16: mediatypes.html -->
4 <!-- CSS media types. -->
5 <html>
6    <head>
7       <meta charset = "utf-8">
```

```
 8     <title>Media Types</title>
 9     <style type = "text/css">
10        @media all
11        {
12           body  { background-color: steelblue; }
13           h1    { font-family: verdana, helvetica, sans-serif;
14                   color: palegreen; }
15           p     { font-size: 12pt;
16                   color: white;
17                   font-family: arial, sans-serif; }
18        } /* End @media all declaration. */
19        @media print
20        {
21           body  { background-color: white; }
22           h1    { color: seagreen; }
23           p     { font-size: 14pt;
24                   color: steelblue;
25                   font-family: "times new roman", times, serif; }
26        } /* End @media print declaration. */
27     </style>
28  </head>
29  <body>
30     <h1>CSS Media Types Example</h1>
31
32     <p>
33        This example uses CSS media types to vary how the page
34        appears in print and how it appears on any other media.
35        This text will appear in one font on the screen and a
36        different font on paper or in a print preview. To see
37        the difference in Internet Explorer, go to the Print
38        menu and select Print Preview. In Firefox, select Print
39        Preview from the File menu.
40     </p>
41  </body>
42 </html>
```

Fig. 4.16. CSS media types.

In line 10, we begin a block of styles that applies to all media types, declared by `@media all` and enclosed in curly braces (`{` and `}`). In lines 10–18, we define some styles for *all* media types. Lines 19–26 set styles to be applied only when the page is printed.

The styles we applied for all media types look nice on a screen but would *not* look good on a printed page. A colored background would use a lot of ink, and a black-and-white printer may print a page that's hard to read because there isn't enough contrast between the colors.

LOOK-AND-FEEL OBSERVATION 4.1

Pages with dark background colors and light text use a lot of ink and may be difficult to read when printed, especially on a black-and white-printer. Use the `print` media type to avoid this.

LOOK-AND-FEEL OBSERVATION 4.2

In general, sans-serif fonts look better on a screen, while serif fonts look better on paper. The `print` media type allows your web page to display a sans-serif font on a screen and change to a serif font when it's printed.

To solve these problems, we apply specific styles for the `print` media type. We change the `body`'s `background-color`, the `color` of the `h1` tag, and the `font-size`, `color`, and `font-family` of the `p` tag to be more suited for printing *and* viewing on paper. Notice that most of these styles conflict with the declarations in the section for all media types. Since the `print` media type has *higher specificity* than the `all` media type, the `print` styles override the `all` media type's styles when the page is printed. The `h1`'s `font-family` property is *not* overridden in the `print` section, so it retains its old value when the page is printed.

Media Queries

Media queries (covered in detail in Section 5.17) allow you to format your content to specific output devices. Media queries include a media type and expressions that check the **media features** of the output device. Some of the common media features include:

- **`width`** —the width of the part of the screen on which the document is rendered, including any scrollbars
- **`height`** —the height of the part of the screen on which the document is rendered, including any scrollbars

- **`device-width`** —the width of the screen of the output device
- **`device-height`** —the height of the screen of the output device
- **`orientation`** —if the `height` is greater than the `width`, `orientation` is `portrait`, and if the `width` is greater than the `height`, `orientation` is `landscape`
- **`aspect-ratio`** —the ratio of `width` to `height`
- **`device-aspect-ratio`** —the ratio of `device-width` to `device-height`

For a complete list of media features and for more information on media queries, see

http://www.w3.org/TR/css3-mediaqueries/

4.12. Drop-Down Menus

Drop-down menus are a good way to provide navigation links without using a lot of screen space. In this section, we take a second look at the `:hover` pseudo-class and introduce the `display` property to create a simple drop-down menu using CSS3 and HTML5.

We've already seen the `:hover` pseudo-class used to change a link's style when the mouse hovers over it. We'll use this feature in a more advanced way to cause a menu to appear when the mouse hovers over a menu button. Another important property is **`display`**, which allows you to decide whether an element is rendered on the page or not. Possible values include `block`, `inline` and `none`. The `block` and `inline` values display the element as a block element or an inline element, while `none` stops the element from being rendered. The code for the drop-down menu is shown in Fig. 4.17.

```
1 <!DOCTYPE html>
2
3 <!-- Fig. 4.17: dropdown.html -->
```

```
<!-- CSS drop-down menu. -->
<html>
  <head>
    <meta charset = "utf-8">
    <title>
      Drop-Down Menu
    </title>
    <style type = "text/css">
      body          { font-family: arial, sans-serif }
      nav           { font-weight: bold;
                      color: white;
                      border: 2px solid royalblue;
                      text-align: center;
                      width: 10em;
                      background-color: royalblue; }
      nav ul        { display: none;
                      list-style: none;
                      margin: 0;
                      padding: 0; }
      nav:hover ul    { display: block }
      nav ul li       { border-top: 2px solid royalblue;
                      background-color: white;
                      width: 10em;
                      color: black; }
      nav ul li:hover { background-color: powderblue; }
      a             { text-decoration: none; }
    </style>
  </head>
  <body>
    <nav>Menu
      <ul>
        <li><a href = "#">Home</a></li>
        <li><a href = "#">News</a></li>
        <li><a href = "#">Articles</a></li>
        <li><a href = "#">Blog</a></li>
        <li><a href = "#">Contact</a></li>
```

```
40          </ul>
41       </nav>
42    </body>
43 </html>
```

Fig. 4.17. CSS drop-down menu.

Lines 33–41 create a `nav` element containing the the text `Menu` and an unordered list (`ul`) of five links that should appear in the drop-down menu— `Home` , `News` , `Articles` , `Blog` and `Contact` . Initially, `Menu` is the only text visible on the page. When the mouse cursor hovers over the `nav` element, the five links appear below the menu.

The drop-down menu functionality is located in the CSS3 code. Two lines define the drop-down functionality. Line 19 sets `display` to `none` for any unordered list (`ul`) that's nested in a `nav` . This instructs the browser not to render the `ul` 's contents. Line 23, which is similar to line 19, selects only `ul` elements nested in a `nav` element that currently has the mouse hovering over it. Setting `display` to `block` specifies that when the mouse is over the `nav` , the `ul` will be displayed as a block-level element.

The style in line 28 is applied only to a `li` element that's a child of a `ul` element in a `nav` element, and only when that `li` has the mouse cursor over it. This style changes the `background-color` of the currently highlighted menu option. The rest of the CSS simply adds style to the menu's components.

This drop-down menu is just one example of more advanced CSS formatting. Many additional resources are available online for CSS navigation menus and lists.

4.13. (Optional) User Style Sheets

Users can define their own **user style sheets** to format pages based on their preferences. For example, people with *visual impairments* may want to increase the page's text size. You need to *be careful not to inadvertently override user preferences with defined styles*. This section discusses possible conflicts between **author styles** and **user styles**. For the purpose of this section, we demonstrate the concepts in Internet Explorer 9.

Figure 4.18 contains an author style. The `font-size` is set to `9pt` for all `<p>` tags that have class `note` applied to them.

```
1 <!DOCTYPE html>
2
3 <!-- Fig. 4.18: user_absolute.html -->
4 <!-- pt measurement for text size. -->
5 <html>
6    <head>
7       <meta charset = "utf-8">
```

```
 8      <title>User Styles</title>
 9      <style type = "text/css">
10         .note { font-size: 9pt; }
11      </style>
12   </head>
13   <body>
14      <p>Thanks for visiting my website. I hope you enjoy it.
15      </p><p class = "note">Please Note: This site will be
16      moving soon. Please check periodically for updates.</p>
17   </body>
18 </html>
```

Fig. 4.18. `pt` measurement for text size.

User style sheets are *external style sheets*. Figure 4.19 shows a user style sheet that sets the `body`'s `font-size` to `20pt`, `color` to `yellow` and `background-color` to `navy`. The font-size value specified in the user style sheet conflicts with the one in line 10 of Fig. 4.18.

```
1  /* Fig. 4.19: userstyles.css */
2  /* A user style sheet */
3  body    { font-size: 20pt;
4            color: yellow;
5            background-color: navy; }
```

Fig. 4.19. A user style sheet.

Adding a User Style Sheet

User style sheets are *not* `link`ed to a document; rather, they're set in the browser's options. To add a user style sheet in IE9, select **Internet Options...**, located in the **Tools** menu. In the **Internet Options** dialog (Fig. 4.20) that appears, click **Accessibility...**, check the **Format documents using my style sheet** checkbox, and type the location of the user style sheet. IE9 applies the user style sheet to any document it loads. To add a user style sheet in Firefox, find your Firefox profile using the instructions at www.mozilla.org/support/firefox/profile#locate and place a style sheet called `userContent.css` in the `chrome` subdirectory. For information on adding a user style sheet in Chrome, see www.google.com/support/forum/p/Chrome/thread?tid=1fa0dd079dbdc2ff&hl=en.

Fig. 4.20. User style sheet in Internet Explorer 9.

The web page from Fig. 4.18 is displayed in Fig. 4.21, with the user style sheet from Fig. 4.19 applied.

Defining `font-size` in a User Style Sheet

In the preceding example, if the user defines `font-size` in a user style sheet, the author style has a higher precedence and *overrides* the user style. The `9pt` font specified in the author style sheet overrides the `20pt` font specified in the user style sheet. This small font may make pages difficult to read, especially for individuals with *visual impairments*. You can avoid this problem by using relative measurements (e.g., `em` or `ex`) instead of absolute measurements, such as `pt`. Figure 4.22 changes the `font-size` property to use a relative measurement (line 10) that does *not* override the user style set in Fig. 4.19. Instead, the font size displayed is relative to the one specified in the user style sheet. In this case, text enclosed in the `<p>` tag displays as `20pt`, and `<p>` tags that have the class `note` applied to them are displayed in `15pt` (`.75` times `20pt`).

Fig. 4.21. User style sheet applied with `pt` measurement.

```
1  <!DOCTYPE html>
2
3  <!-- Fig. 4.22: user_relative.html -->
4  <!-- em measurement for text size. -->
5  <html>
6     <head>
7        <meta charset = "utf-8">
8        <title>User Styles</title>
9        <style type = "text/css">
10          .note { font-size: .75em; }
11       </style>
12    </head>
```

```
13    <body>
14       <p>Thanks for visiting my website. I hope you enjoy it.
15       </p><p class = "note">Please Note: This site will be
16       moving soon. Please check periodically for updates.</p>
17    </body>
18 </html>
```

Fig. 4.22. `em` measurement for text size.

Figure 4.23 displays the web page from Fig. 4.22 in Internet Explorer with the user style sheet from Fig. 4.19 applied. Note that the second line of text displayed is larger than the same line of text in Fig. 4.21.

Fig. 4.23. User style sheet applied with `em` measurement.

4.14. Web Resources

http://www.deitel.com/css3

The Deitel CSS3 Resource Center contains links to some of the best CSS3 information on the web. There you'll find categorized links to tutorials, references, code examples, demos, videos, and more. Check out the

demos section for more advanced examples of layouts, menus and other web-page components.

Summary

Section 4.1 Introduction

- Cascading Style Sheets™ 3 (CSS3; p. 106) allows you to specify the presentation of elements on a web page (e.g., fonts, spacing, sizes, colors, positioning) separately from the structure and content of the document (section headers, body text, links, etc.).
- This separation of structure from presentation (p. 106) simplifies maintaining and modifying web pages, especially on large-scale websites.

Section 4.2 Inline Styles

- An inline style (p. 106) allows you to declare a style for an individual element by using the `style` attribute (p. 106) in the element's start tag.
- Each CSS property (such as `font-size`, p. 107) is followed by a colon and the value of the attribute. Multiple property declarations are separated by a semicolon.
- The `color` property (p. 107) sets text color. Hexadecimal codes or color names may be used.

Section 4.3 Embedded Style Sheets

- Embedded style sheets (p. 108) enable you to embed an entire CSS3 document in an HTML5 document's head section.
- Styles that are placed in a `style` element use selectors (p. 109) to apply style elements throughout the entire document body.
- An `em` element indicates that its contents should be emphasized. Browsers usually render `em` elements in an italic font.

- `style` element attribute `type` specifies the MIME type (the specific encoding format, p. 109) of the style sheet. Style sheets use `text/css`.

- Each rule body (p. 109) in a style sheet begins and ends with a curly brace (`{` and `}`).

- The `font-weight` property (p. 110) specifies the "boldness" of text. Possible values are `bold`, `normal` (the default), `bolder` (bolder than bold text) and `lighter` (lighter than `normal` text).

- Boldness also can be specified with multiples of 100, from 100 to 900. Text specified as `normal` is equivalent to `400`, and `bold` text is equivalent to `700`.

- Style-class declarations are preceded by a period and are applied to elements of the specific class. The `class` attribute (p. 111) applies a style class to an element.

- The CSS rules in a style sheet use the same format as inline styles.

- The `background-color` attribute specifies the background color of the element.

- The `font-family` property (p. 110) names a specific font that should be displayed. Generic font families allow authors to specify a type of font instead of a specific one, in case a browser does not support a specific font.

- The `font-size` property (p. 110) specifies the size used to render the font.

- You should end a font list with a generic font family (p. 110) name in case the other fonts are not installed on the user's computer.

- In many cases, the styles applied to an element (the parent or ancestor element, p. 111) also apply to the element's nested elements (child or descendant elements, p. 111).

Section 4.4 Conflicting Styles

- Styles may be defined by a user, an author or a user agent. A user (p. 111) is a person viewing your web page, you're the author (p. 111)—the person who writes the document—and the user agent (p. 111) is the program used to render and display the document (e.g., a web browser).

- Styles cascade (hence the term "Cascading Style Sheets," p. 111), or flow together, such that the ultimate appearance of elements on a page results from combining styles defined in several ways.

- Most styles are inherited from parent elements (p. 111). Styles defined for children (p. 111) have higher specificity (p. 112) and take precedence over the parent's styles.

- Pseudo-classes (p. 113) give the author access to information that's not declared in the document, such as whether the mouse is hovering over an element or whether the user has previously clicked (visited) a particular hyperlink. The `hover` pseudo-class (p. 114) is activated when the user moves the mouse cursor over an element.

- The `text-decoration` property (p. 113) applies decorations to text in an element, such as `underline`, `overline` and `line-through`.

- To apply rules to multiple elements, separate the elements with commas in the style sheet.

- To apply rules only to a certain type of element that's a child of another type, separate the element names with spaces.

- A pixel is a relative-length measurement (p. 114): It varies in size based on screen resolution. Other relative lengths are `em` (p. 114), `ex` (p. 114) and percentages.

- The other units of measurement available in CSS are absolute-length measurements (p. 114)—that is, units that do not vary in size. These units can be `in` (inches), `cm` (centimeters, p. 114), `mm` (millimeters, p. 114), `pt` (points; 1 `pt` = 1/72 `in`, p. 114) or `pc` (picas; 1 `pc` = 12 `pt`).

Section 4.5 Linking External Style Sheets

- With external style sheets (i.e., separate documents that contain only CSS rules; p. 114), you can provide a uniform look and feel to an entire web-site (or to a portion of one).

- When you need to change styles, you need to modify only a single CSS file to make style changes across all the pages that use those styles. This is sometimes known as skinning (p. 114).

- CSS comments (p. 115) may be placed in any type of CSS code (i.e., inline styles, embedded style sheets and external style sheets) and always start with `/*` and end with `*/`.

- `link`'s `rel` attribute (p. 115) specifies a relationship between two documents (p. 115). For style sheets, the `rel` attribute declares the linked document to be a `stylesheet` (p. 115) for the document. The `type` attribute specifies the MIME type of the related document as `text/css`. The `href` attribute provides the URL for the document containing the style sheet.

Section 4.6 Positioning Elements: Absolute Positioning, `z-index`

- The CSS `position` property (p. 116) allows absolute positioning (p. 117), which provides greater control over where on a page elements reside. Specifying an element's `position` as `absolute` removes it from the normal flow of elements on the page and positions it according to distance from the `top`, `left`, `right` or `bottom` margin of its parent element.

- The `z-index` property (p. 118) allows a developer to layer overlapping elements. Elements that have higher `z-index` values are displayed in front of elements with lower `z-index` values.

Section 4.7 Positioning Elements: Relative Positioning, `span`

- Unlike absolute positioning, relative positioning keeps elements in the general flow on the page and offsets them by the specified `top`, `left`, `right` or `bottom` value.

- Element `span` (p. 120) is a grouping element (p. 120)—it does not apply any inherent formatting to its contents. Its primary purpose is to apply

CSS rules or `id` attributes to a section of text.

- `span` is an inline-level element (p. 120)—it applies formatting to text without changing the flow of the document. Examples of inline elements include `span`, `img`, `a`, `em` and `strong`.

- The `div` element is also a grouping element, but it's a block-level element. This means it's displayed on its own line and has a virtual box around it. Examples of block-level elements (p. 120) include `div` (p. 120), `p` and heading elements (`h1` through `h6`).

Section 4.8 Backgrounds

- Property `background-image` specifies the URL of the image, in the format `url(`*fileLocation*`)`. The property `background-position` (p. 121) places the image on the page using the values top, bottom, center, left and right individually or in combination for vertical and horizontal positioning. You can also position by using lengths.

- The `background-repeat` property (p. 121) controls the tiling of the background image (p. 121). Setting the tiling to `no-repeat` displays one copy of the background image on screen. The `background-repeat` property can be set to `repeat` (the default) to tile the image vertically and horizontally, to `repeat-x` to tile the image only horizontally or to `repeat-y` to tile the image only vertically.

- The `background-attachment` (p. 121) setting `fixed` fixes the image in the position specified by `background-position`. Scrolling the browser window will not move the image from its set position. The default value, `scroll`, moves the image as the user scrolls the window.

- The `text-indent` property (p. 122) indents the first line of text in the element by the specified amount.

- The `font-style` property (p. 122) allows you to set text to `none`, `italic` or `oblique`.

Section 4.9 Element Dimensions

- An element's dimensions can be set with CSS by using properties `height` and `width` (p. 123).

- Text in an element can be centered using `text-align` (p. 123); other values for the `text-align` property are `left` and `right`.

- A problem with setting both vertical and horizontal dimensions of an element is that the content inside the element might sometimes exceed the set boundaries, in which case the element grows to fit the content. You can set the `overflow` property (p. 123) to scroll; this setting adds scroll bars if the text overflows the boundaries set for it.

Section 4.10 Box Model and Text Flow

- All block-level HTML5 elements have a virtual box drawn around them, based on what is known as the box model (p. 123).

- When the browser renders elements using the box model, the content of each element is surrounded by padding (p. 123), a border (p. 123) and a margin (p. 123).

- The `border-width` property (p. 124) may be set to any of the CSS lengths or to the predefined value of `thin`, `medium` or `thick`.

- The `border-style`s (p. 124) available are `none`, `hidden`, `dotted`, `dashed`, `solid`, `double`, `groove`, `ridge`, `inset` and `outset`.

- The `border-color` property (p. 124) sets the color used for the border.

- The `class` attribute allows more than one class to be assigned to an element by separating each class name from the next with a space.

- Browsers normally place text and elements on screen in the order in which they appear in the document. Elements can be removed from the normal flow of text. Floating allows you to move an element to one side of the screen; other content in the document will then flow around the floated element.

- CSS uses a box model to render elements on screen. The content of each element is surrounded by padding, a border and margins. The properties of this box are easily adjusted.

- The `margin` property (p. 127) determines the distance between the outside edge of the element's border and any adjacent element.

- Margins for individual sides of an element can be specified by using `margin-top`, `margin-right`, `margin-left` and `margin-bottom`.

- The `padding` property (p. 127) determines the distance between the content inside an element and the inside edge of the border. Padding also can be set for each side of the box by using `padding-top`, `padding-right`, `padding-left` and `padding-bottom`.

Section 4.11 Media Types and Media Queries

- CSS media types (p. 127) allow you to decide what a page should look like depending on the kind of media being used to display the page. The most commonly used for a web page is the `screen` media type (p. 127), which is a standard computer screen.

- A block of styles that applies to all media types is declared by `@media all` and enclosed in curly braces. To create a block of styles that apply to a single media type such as `print`, use `@media print` and enclose the style rules in curly braces.

- Other media types in CSS 2 include `handheld`, `braille`, `aural` and `print`. The `handheld` medium (p. 127) is designed for mobile Internet devices, while `braille` (p. 127) is for machines that can read or print web pages in braille. `aural` styles (p. 127) allow the programmer to give a speech-synthesizing web browser more information about the content of the web page. The `print` media type (p. 127) affects a web page's appearance when it's printed.

- Media queries (p. 130) allow you to format your content to specific output devices. Media queries include a media type and expressions that check the devices' media features (p. 130).

Section 4.12 Drop-Down Menus

- The `:hover` pseudo-class is used to apply styles to an element when the mouse cursor is over it.
- The `display` property (p. 130) allows you to decide whether an element is displayed as a `block` element or `inline` element or not rendered at all (`none`).

Section 4.13 (Optional) User Style Sheets

- Users can define their own user style sheets (p. 132) to format pages based on their preferences.
- Absolute font-size measurements override user style sheets, while relative font sizes will yield to a user-defined style.
- If the user defines font size in a user style sheet, the author style (p. 133) has a higher precedence and overrides the user style.

Self-Review Exercises

4.1 Assume that the size of the base font on a system is 12 points.

a. How big is a 36-point font in `em`s?

b. How big is a 9-point font in `em`s?

c. How big is a 24-point font in picas?

d. How big is a 12-point font in inches?

e. How big is a 1-inch font in picas?

4.2 Fill in the blanks in the following statements:

a. Using the _____ element allows you to use external style sheets in your pages.

b. To apply a CSS rule to more than one element at a time, separate the element names with a(n)_____.

c. Pixels are a(n)_____-length measurement unit.

d. The _____ pseudo-class is activated when the user moves the mouse cursor over the specified element.

e. Setting the _____ property to `scroll` provides a mechanism for creating scrollable content without compromising an element's dimensions.

f. _____ is a generic inline element that applies no inherent formatting, and _____is a generic block-level element that applies no inherent formatting.

g. Setting property `background-repeat` to _____ tiles the specified `background-image` vertically.

h. To begin a block of styles that applies only to the `print` media type, you use the declaration _____ `print` , followed by an opening curly brace (`{`).

i. The _____ property allows you to indent the first line of text in an element.

j. The three components of the box model are the _____, _____ and _____.

Answers to Self-Review Exercises

4.1

a. 3 `em` s.

b. 0.75 `em` s.

c. 2 picas.

d. 1/6 inch.

e. 6 picas.

4.2

a. `link`.

b. comma.

c. relative.

d. : `hover`.

e. `overflow`.

f. `span`, `div`.

g. `repeat-y`.

h. `@media`.

i. `text-indent`.

j. padding, border, margin.

Exercises

4.3 Write a CSS rule that makes all text 1.5 times larger than the base font of the system and colors the text red.

4.4 Write a CSS rule that places a background image halfway down the page, tiling it horizontally. The image should remain in place when the user scrolls up or down.

4.5 Write a CSS rule that gives all `h1` and `h2` elements a padding of 0.5 `em`s, a `dashed` border style and a margin of 0.5 `em`s.

4.6 Write a CSS rule that changes the color of all elements containing attribute `class = "greenMove"` to green and shifts them down 25 pixels and right 15 pixels.

4.7 Make a layout template that contains a `header` and two paragraphs. Use `float` to line up the two paragraphs as columns side by side. Give the `header` and two paragraphs a border and/or a background color so you can see where they are.

4.8 Add an *embedded style sheet* to the HTML5 document in Fig. 2.3. The style sheet should contain a rule that displays `h1` elements in blue. In addition, create a rule that displays all links in blue without underlining them. When the mouse hovers over a link, change the link's background color to yellow.

4.9 Make a navigation button using a `div` with a link inside it. Give it a border, background, and text color, and make them change when the user hovers the mouse over the button. Use an external style sheet. Make sure your style sheet validates at http://jigsaw.w3.org/css-validator/. Note that some warnings may be unavoidable, but your CSS should have no errors.