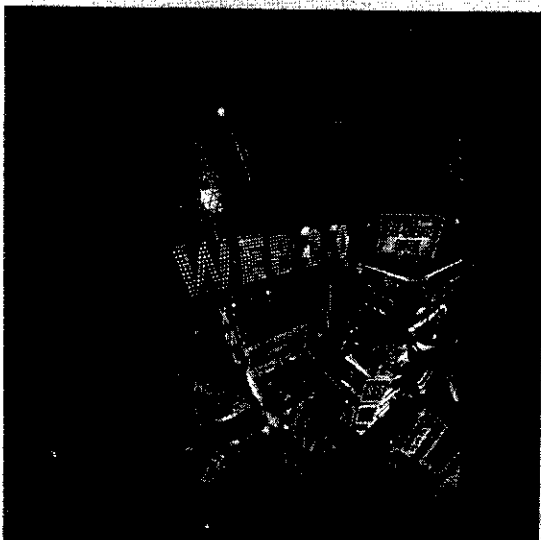
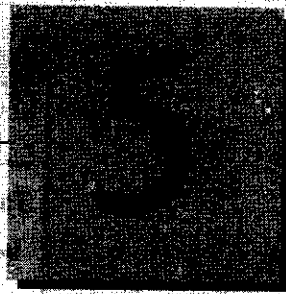


Fig. 4.16 | XHTML table for Exercise 4.12.

4.13 A local university has asked you to create an XHTML document that allows prospective students to provide feedback about their campus visit. Your XHTML document should contain a form with text boxes for a name, address and e-mail. Provide checkboxes that allow prospective students to indicate what they liked most about the campus. The checkboxes should include: students, location, campus, atmosphere, dorm rooms and sports. Also, provide radio buttons that ask the prospective students how they became interested in the university. Options should include: friends, television, Internet and other. In addition, provide a text area for additional comments, a submit button and a reset button.

4.14 Create an XHTML document titled "How to Get Good Grades." Use `meta` tags to include a series of keywords that describe your document.



Fashions fade, style is eternal.

—Yves Saint Laurent

A style does not go out of style as long as it adapts itself to its period. When there is an incompatibility between the style and a certain state of mind, it is never the style that triumphs.

—Coco Chanel

How liberating to work in the margins, outside a central perception.

—Don DeLillo

I've gradually risen from lower-class background to lower-class foreground.

—Marvin Cohen

Cascading Style Sheets™ (CSS)

OBJECTIVES

In this chapter you will learn:

- To control the appearance of a website by creating style sheets.
- To use a style sheet to give all the pages of a website the same look and feel.
- To use the `class` attribute to apply styles.
- To specify the precise font, size, color and other properties of displayed text.
- To specify element backgrounds and colors.
- To understand the box model and how to control margins, borders and padding.
- To use style sheets to separate presentation from content.

Outline

- 5.1 Introduction
- 5.2 Inline Styles
- 5.3 Embedded Style Sheets
- 5.4 Conflicting Styles
- 5.5 Linking External Style Sheets
- 5.6 Positioning Elements
- 5.7 Backgrounds
- 5.8 Element Dimensions
- 5.9 Box Model and Text Flow
- 5.10 Media Types
- 5.11 Building a CSS Drop-Down Menu
- 5.12 User Style Sheets
- 5.13 CSS 3
- 5.14 Web Resources

Summary | Terminology | Self-Review Exercise | Exercises

5.1 Introduction

In Chapter 4, we introduced the Extensible HyperText Markup Language (XHTML) for marking up information to be rendered in a browser. In this chapter, we shift our focus to formatting and presenting information. To do this, we use a W3C technology called **Cascading Style Sheets™ (CSS)** that allows document authors to specify the presentation of elements on a web page (e.g., fonts, spacing, colors) separately from the structure of the document (section headers, body text, links, etc.). This **separation of structure from presentation** simplifies maintaining and modifying a web page.

XHTML was designed to specify the content and structure of a document. Though it has some attributes that control presentation, it is better not to mix presentation with content. If a website's presentation is determined entirely by a style sheet, a web designer can simply swap in a new style sheet to completely change the appearance of the site. CSS provides a way to apply style outside of XHTML, allowing the XHTML to dictate the content while the CSS dictates how it's presented.

As with XHTML, the W3C provides a CSS code validator located at jigsaw.w3.org/css-validator/. It is a good idea to validate all CSS code with this tool to make sure that your code is correct and works on as many browsers as possible.

CSS is a large topic. As such, we can introduce only the basic knowledge of CSS that you'll need to understand the examples and exercises in the rest of the book. For more CSS references and resources, check out our CSS Resource Center at www.deitel.com/css21.

The W3C's CSS specification is currently in its second major version, with a third in development. The current versions of most major browsers support much of the functionality in CSS 2. This allows programmers to make full use of its features. In this chapter, we introduce CSS, demonstrate some of the features introduced in CSS 2 and discuss some of the upcoming CSS 3 features. As you read this book, open each XHTML document in your web browser so you can view and interact with it in a web browser, as it was originally intended.

Remember that the examples in this book have been tested in Internet Explorer 7 and Firefox 2. The latest versions of many other browsers (e.g., Safari, Opera, Konqueror) should render this chapter's examples properly, but we have not tested them. Some examples in this chapter *will not work* in older browsers, such as Internet Explorer 6 and earlier. Make sure you have either Internet Explorer 7 (Windows only) or Firefox 2 (available for all major platforms) installed before running the examples in this chapter.

5.2 Inline Styles

You can declare document styles in several ways. This section presents **inline styles** that declare an individual element's format using the XHTML attribute **style**. Inline styles override any other styles applied using the techniques we discuss later in the chapter. Figure 5.1 applies inline styles to p elements to alter their font size and color. Figure 5.1 applies inline styles to p elements to alter their font size and color.



Good Programming Practice 5.1

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.

The first inline style declaration appears in line 17. Attribute **style** specifies an element's style. Each CSS property (**font-size** in this case) is followed by a colon and a value. In line 17, we declare this particular p element to use 20-point font size.

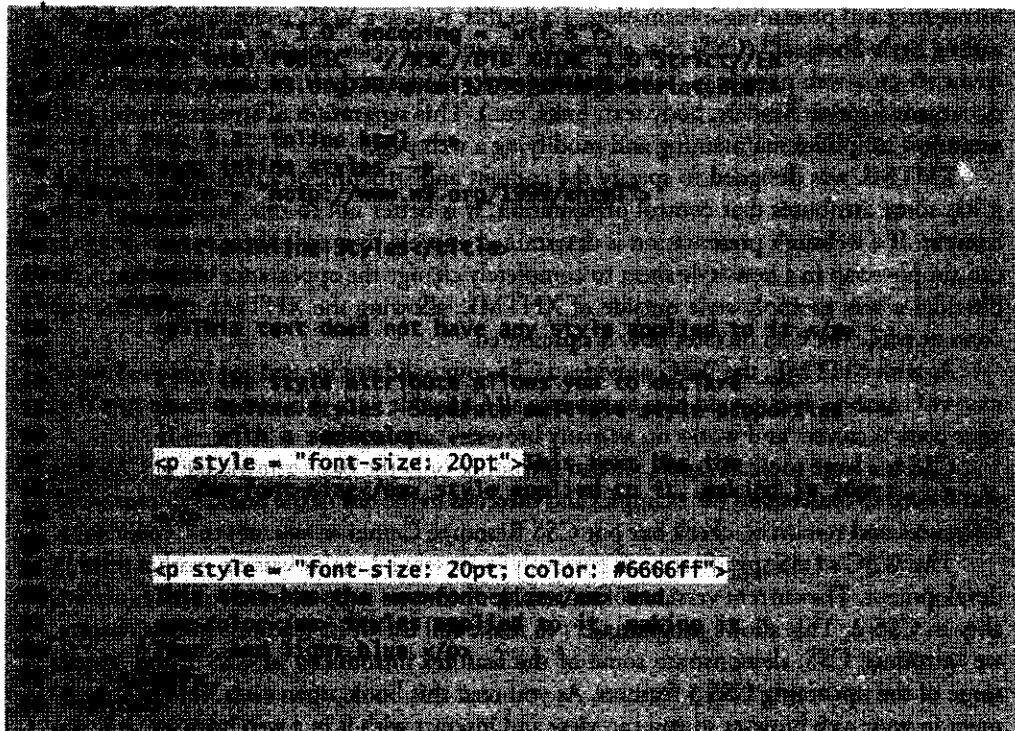


Fig. 5.1 | Using inline styles. (Part 1 of 2.)

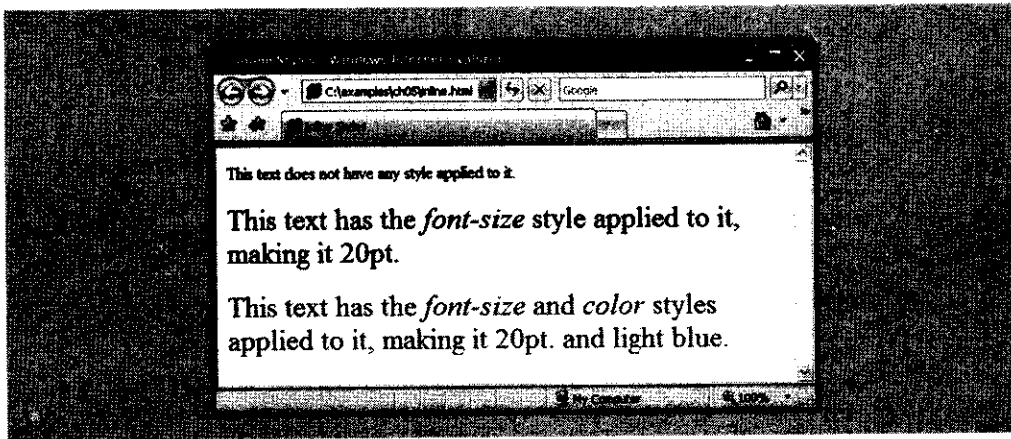


Fig. 5.1 | Using inline styles. (Part 2 of 2.)

Line 21 specifies the two properties, `font-size` and `color`, separated by a semicolon. In this line, we set the given paragraph's `color` to light blue, using the hexadecimal code `#6666ff`. Color names may be used in place of hexadecimal codes. We provide a list of hexadecimal color codes and color names in Appendix B, XHTML Colors.

5.3 Embedded Style Sheets

A second technique for using style sheets is **embedded style sheets**. Embedded style sheets enable a you to embed an entire CSS document in an XHTML document's head section. To achieve this separation between the CSS code and the XHTML that it styles, we will use **CSS selectors**. Figure 5.2 creates an embedded style sheet containing four styles.

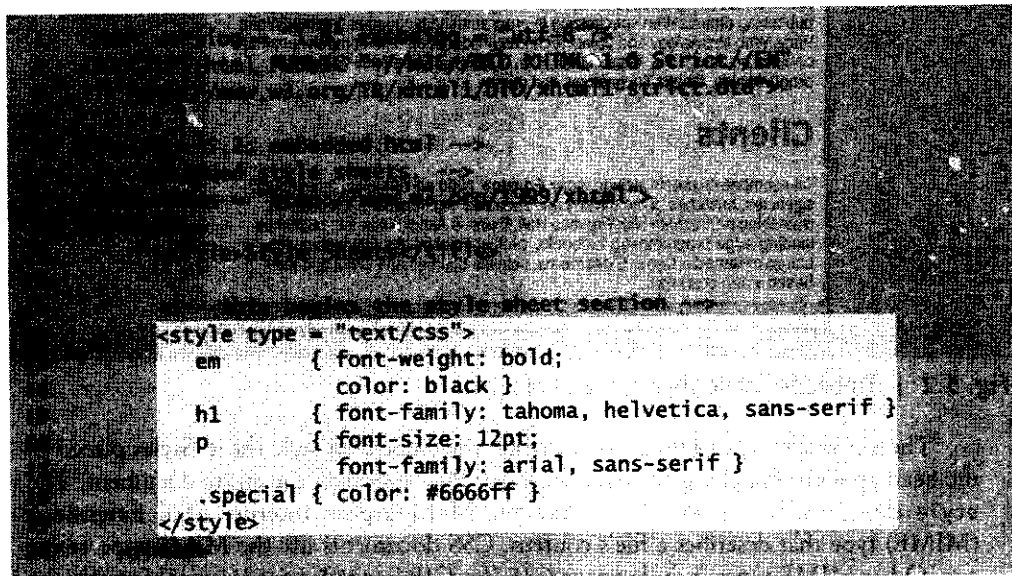


Fig. 5.2 | Embedded style sheets. (Part 1 of 2.)

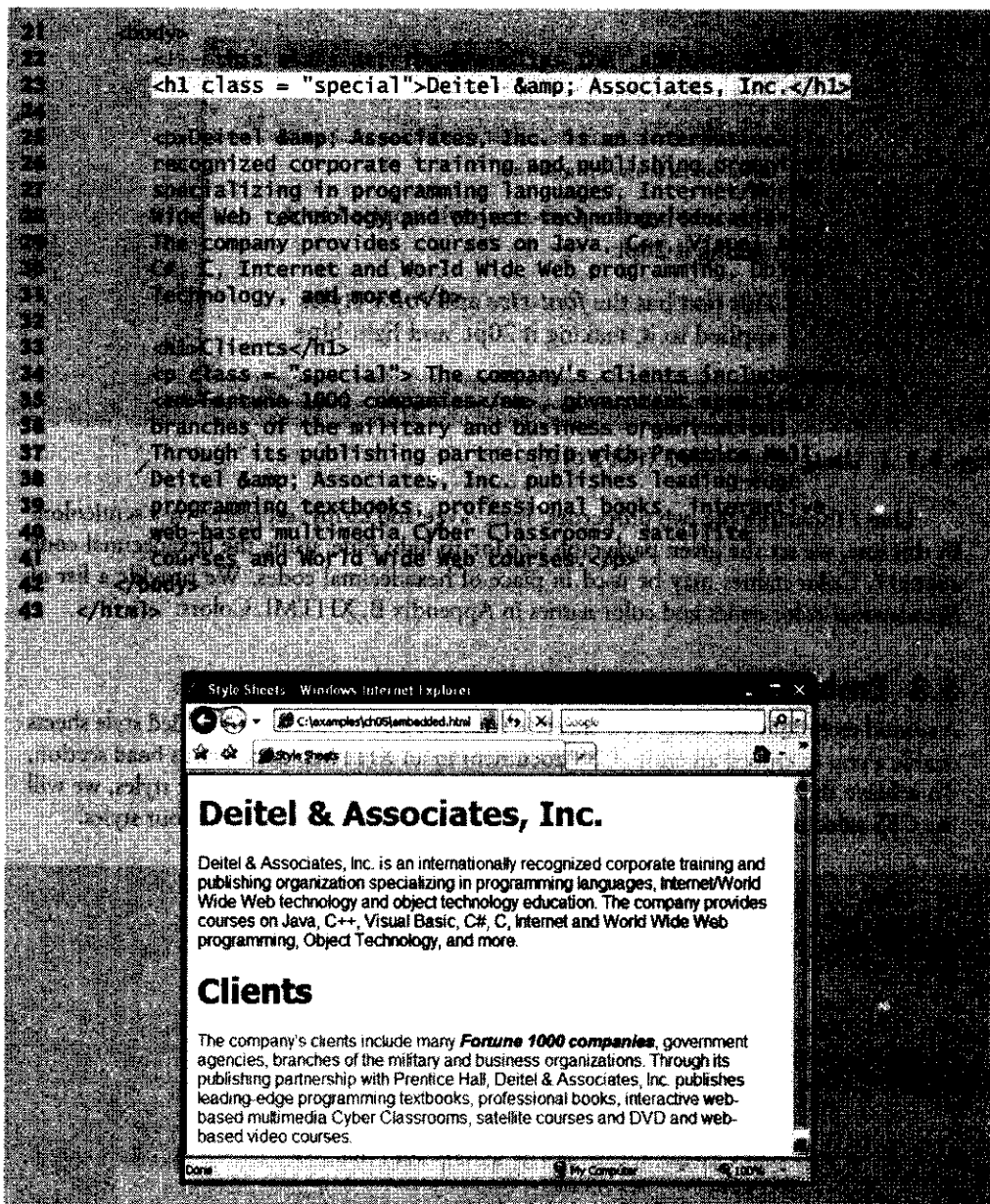


Fig. 5.2 | Embedded style sheets. (Part 2 of 2.)

The `style` element (lines 12–19) defines the embedded style sheet. Styles placed in the head apply to matching elements wherever they appear in the entire document. The `style` element's `type` attribute specifies the **Multipurpose Internet Mail Extensions (MIME)** type that describes a file's content. CSS documents use the MIME type `text/css`. Other MIME types include `image/gif` (for GIF images), `text/javascript` (for the JavaScript scripting language, which we discuss in Chapters 6–11), and more.

The body of the style sheet (lines 13–18) declares the CSS rules for the style sheet. A CSS selector determines which elements will be styled according to a rule. Our first rule begins with the selector `em` (line 13) to select all `em` elements in the document. The **font-weight** property in line 13 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 (e.g., 100, 200, ..., 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 the values from 100 to 900 might not display the desired effect.

In this example, all `em` elements will be displayed in a bold font. We also apply styles to all `h1` (line 15) and `p` (lines 16–17) elements. The body of each rule is enclosed in curly braces (`{` and `}`).

Line 18 uses a new kind of selector to declare a style class named `special`. Style classes define styles that can be applied to any element. In this example, we declare class `special`, which sets `color` to `blue`. We can apply this style to any element type, whereas the other rules in this style sheet apply only to specific element types defined in the style sheet (i.e., `em`, `h1` or `p`). Style-class declarations are preceded by a period. We will discuss how to apply a style class momentarily.

CSS rules in embedded style sheets use the same syntax as inline styles; the property name is followed by a colon (`:`) and the value of the property. Multiple properties are separated by semicolons (`;`). In the rule for `em` elements, the `color` property specifies the color of the text, and property `font-weight` makes the font bold.

The **font-family** property (line 15) 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 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. 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. Other generic font families include `serif` (e.g., `times new roman`, `Georgia`), `cursive` (e.g., `script`), `fantasy` (e.g., `critter`) and `monospace` (e.g., `courier`, `fixedsys`).

The **font-size** property (line 16) specifies a 12-point font. Other possible measurements in addition to `pt` (point) are introduced later in the chapter. Relative values—`xx-small`, `x-small`, `small`, `smaller`, `medium`, `large`, `larger`, `x-large` and `xx-large`—also can be used. Generally, relative values for `font-size` are preferred over point sizes because an author does not know the specific measurements of the display for each client. Relative `font-size` values permit more flexible viewing of web pages.

For example, a user may wish to view a web page on a handheld device with a small screen. Specifying an 18-point font size in a style sheet will prevent such a user from seeing more than one or two characters at a time. However, if a relative font size is specified, such as `large` or `larger`, the actual size is determined by the browser that displays the font. 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, yet it is clearly visible to each user. Accessibility is an important consideration—in 1998, congress passed the Section 508

Amendment to the Rehabilitation Act of 1973, mandating that websites of government agencies are required to be accessible to disabled users.

Line 23 uses the XHTML attribute `class` in an `h1` element to apply a style class—in this case class `special` (declared with the `.special` selector in the style sheet on line 18). When the browser renders the `h1` element, note that the text appears on screen with the properties of both an `h1` element (`arial` or `sans-serif` font defined in line 17) and the `.special` style class applied (the color `#6666ff` defined in line 18). Also notice that the browser still applies its own default style to the `h1` element—the header is still 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 our style rule.

The formatting for the `p` element and the `.special` class is applied to the text in lines 34–41. 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 35 **inherits** the style from the `p` element (namely, the 12-point font size in line 16) but retains its italic style. In other words, styles defined for the paragraph and not defined for the `em` element is applied to the `em` element. Because multiple values of one property can be set or inherited on the same element, they must be reduced to one style per element before being rendered. We discuss the rules for resolving these conflicts in the next section.

5.4 Conflicting Styles

Styles may be defined by a **user**, an **author** or a **user agent** (e.g., a web browser). A user is a person viewing your web page, you are the author—the person who writes the document—and the user agent is the program used to render and display the document. Styles “cascade,” 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, and 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. While it makes sense to inherit most styles, such as font properties, there are certain properties that we don’t want to be inherited. Consider for example the `background-image` property, which allows the programmer 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 5.2 presented an example of **inheritance** in which a child `em` element inherited the `font-size` property from its parent `p` element. However, in Fig. 5.2, the child `em` element had a `color` property that conflicted with (i.e., had a different value than) the `color` property of its parent `p` element. Properties defined for child and descendant elements have a greater **specificity** than properties defined for parent and ancestor elements. Conflicts are resolved in favor of properties with a higher specificity. In other words, the styles explicitly defined for a child element are more specific than the styles defined for the child’s parent element; therefore, the child’s styles take precedence. Figure 5.3 illustrates examples of inheritance and specificity.

Line 12 applies property text-decoration to all a elements whose class attribute is set to nodec. 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, underline and blink. [Note: blink is not supported by Internet Explorer.] The .nodec appended to a is a more specific class selector; this style will apply only to a (anchor) elements that specify nodec in their class attribute.

```

1 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
2 http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
3
4 <!-- Fig. 5.3: advanced.html -->
5 <!-- Inheritance in style sheets. -->
6 <html xmlns = "http://www.w3.org/1999/xhtml">
7 <head>
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 h1 { margin-left: 20px }
16 h2 { font-size: 0.8em }
17 </style>
18 </head>
19 <body>
20 <h1>Shopping List for Monday</h1>
21
22 <ul style="list-style-type: none;">
23 <li><input type="checkbox"/> Milk</li>
24 <li><input type="checkbox"/> Eggs</li>
25 <li><input type="checkbox"/> White bread</li>
26 <li><input type="checkbox"/> Rye bread</li>
27 <li><input type="checkbox"/> Whole wheat bread</li>
28 </ul>
29
30 <h2>Common Programming Error 5.1</h2>
31
32 <ul style="list-style-type: none;">
33 <li><input type="checkbox"/> Don't use <em> with <strong></li>
34 </ul>
35
36 <p>For more information on this error, see the book "Common Programming Errors in C++" by Ericnie C. Niebler, published by No Starch Press, Inc. (http://www.nosp.org).</p>
37
38 <p><a href = "http://www.daitel.com" class = "nodec">http://www.daitel.com</a></p>
39 <p>Happy coding!</p>
40
41 </body>
42 </html>

```

Fig. 5.3 | Inheritance in style sheets. (Part 1 of 2.)

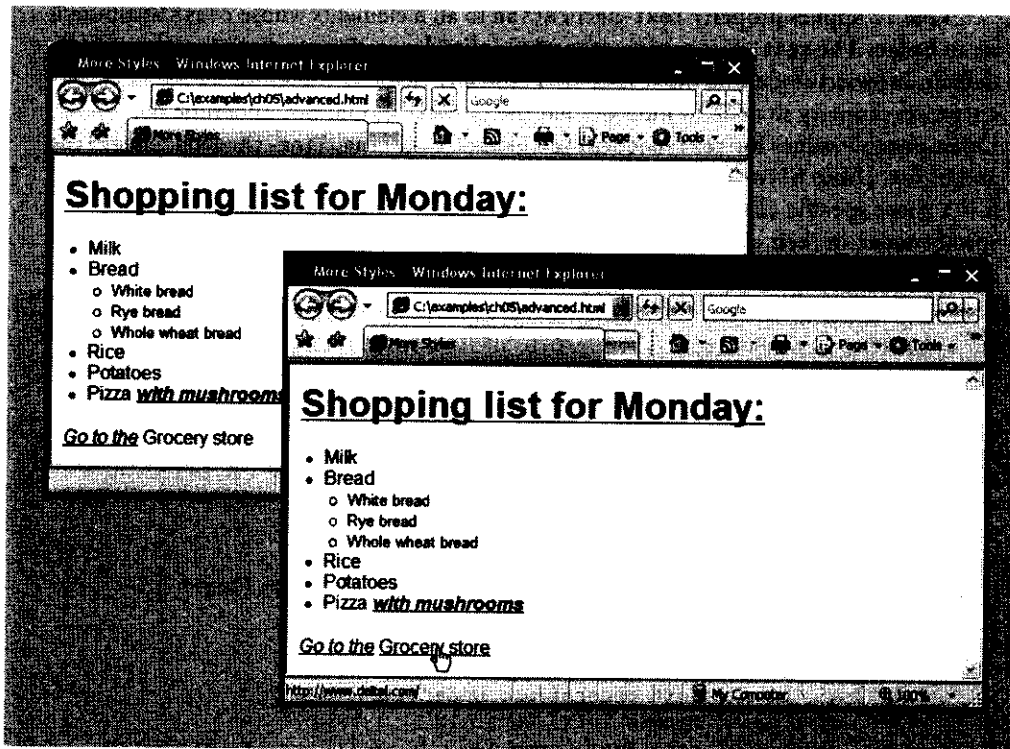


Fig. 5.3 | Inheritance in style sheets. (Part 2 of 2.)



Portability Tip 5.1

To ensure that your style sheets work in various web browsers, test them on all the client web browsers that will render documents using your styles, as well as using the W3C CSS Validator.

Line 13 specifies a style for hover, which is a **pseudoclass**. Pseudoclasses give the author access to content not specifically declared in the document. The **hover** pseudoclass is activated dynamically when the user moves the mouse cursor over an element. Note that pseudoclasses are separated by a colon (with no surrounding spaces) from the name of the element to which they are applied.



Common Programming Error 5.1

Including a space before or after the colon separating a pseudoclass from the name of the element to which it is applied is an error that prevents the pseudoclass from being applied properly.

Line 14 causes all **em** elements that are children of **i** elements to be bold. In the screen output of Fig. 5.3, note that **Go to the** (contained in an **em** element in line 37) does not appear bold, because the **em** element is not in an **i** element. However, the **em** element containing **with mushrooms** (line 34) is nested in an **i** element; therefore, it is formatted in bold. The syntax for applying rules to multiple elements is similar. In line 15, we separate the selectors with a comma to apply an underline style rule to all **h1** and all **em** elements.

Line 16 assigns a left margin of 20 pixels to all **u1** elements. We will discuss the **margin** properties in detail in Section 5.9. A pixel is a **relative-length measurement**—it varies in

size, based on screen resolution. Other relative lengths include **em** (the *M*-height of the font, which is usually set to the height of an uppercase *M*), **ex** (the *x*-height of the font, which is usually set to the height of a lowercase *x*) and percentages (e.g., `font-size: 50%`). To set an element to display text at 150 percent of its default text size, the author could use the syntax

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

Alternatively, you could use

```
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 17 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 5.2

Whenever possible, use relative-length measurements. If you use absolute-length measurements, your document may not be readable on some client browsers (e.g., wireless phones).

5.5 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. Different pages on a site can all use the same style sheet. When changes to the styles are required, the author needs to modify only a single CSS file to make style changes across the entire website. Note that 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 working in parallel. External style sheets solve this problem by separating the content and style into separate files.



Software Engineering Observation 5.1

Always use an external style sheet when developing a website with multiple pages. External style sheets separate content from presentation, allowing for more consistent look-and-feel, more efficient development, and better performance.

Figure 5.4 presents an external style sheet. Lines 1–2 are **CSS comments**. Like XHTML comments, CSS comments describe the content of a CSS document. Comments 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.

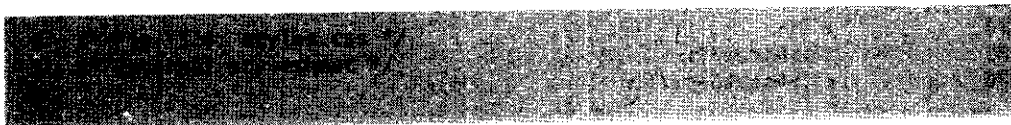


Fig. 5.4 | External style sheet. (Part 1 of 2.)

```

1 body { font-family: arial, helvetica, sans-serif; }
2 a:link { text-decoration: none; }
3 a:visited { text-decoration: underline; }
4 h1 { font-weight: bold; }
5 h1:link { text-decoration: underline; }
6 h2 { margin-left: 20px; }
7 h3 { font-size: 80%; }

```

Fig. 5.4 | External style sheet. (Part 2 of 2.)

Figure 5.5 contains an XHTML document that references the external style sheet in Fig. 5.4. Lines 10–11 (Fig. 5.5) show a `link` element that uses the `rel` attribute to specify a relationship between the current document and another document. In this case, we declare the linked document to be a `stylesheet` for this 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. In this case, `styles.css` is in the same directory as `external.html`.

```

1 <?xml version = "1.0" encoding = "utf-8" ?>
2 <DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
3 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd"
4 [
5 <!-- The 1.0 minimal DTD -->
6 <!-- Linking an external style sheet -->
7 <link href = "http://www.w3.org/1999/xhtml"
8 type = "text/css" />
9
10 <link rel = "stylesheet" type = "text/css"
11 href = "styles.css" />
12
13 </head>
14 <body>
15 <h1>Shopping list for groceries</h1>
16
17 <ul>
18 <li><a href = "#">apples</a></li>
19 <li><a href = "#">bananas</a></li>
20 <li><a href = "#">white bread</a></li>
21 <li><a href = "#">rye bread</a></li>
22 <li><a href = "#">whole wheat bread</a></li>
23 </ul>
24
25 </body>
26 </html>

```

Fig. 5.5 | Linking an external style sheet. (Part 1 of 2.)

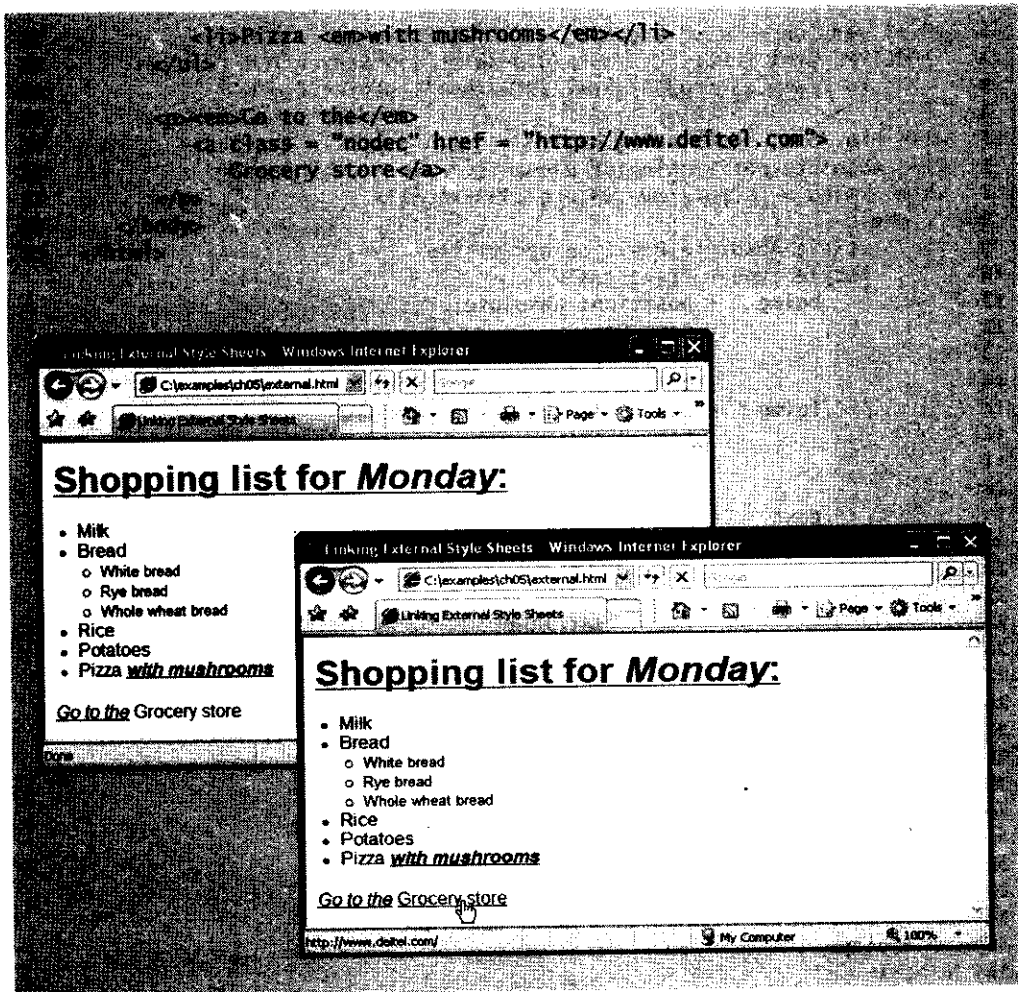


Fig. 5.5 | Linking an external style sheet. (Part 2 of 2.)



Software Engineering Observation 5.2

External style sheets are reusable. Creating them once and reusing them reduces programming effort.



Performance Tip 5.1

Reusing external style sheets reduces load time and bandwidth usage on a server, since the style sheet can be downloaded once, stored by the web browser, and applied to all pages on a website.

5.6 Positioning Elements

Before CSS, controlling the positioning of elements in an XHTML document was difficult—the browser determined positioning. CSS introduced the **position** property and a capability called **absolute positioning**, which gives authors greater control over how document elements are displayed. Figure 5.6 demonstrates absolute positioning.

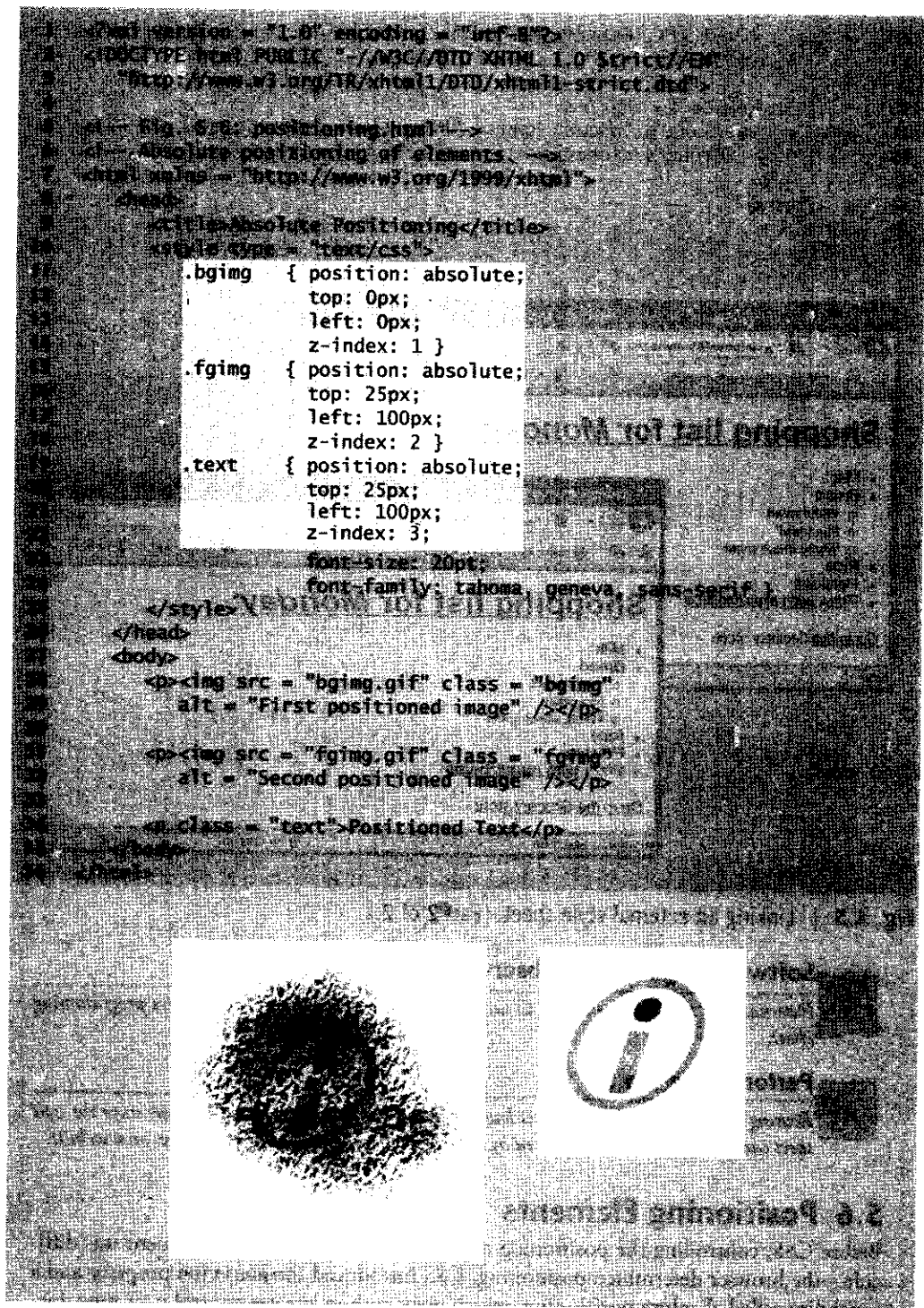


Fig. 5.6 | Absolute positioning of elements. (Part I of 2.)

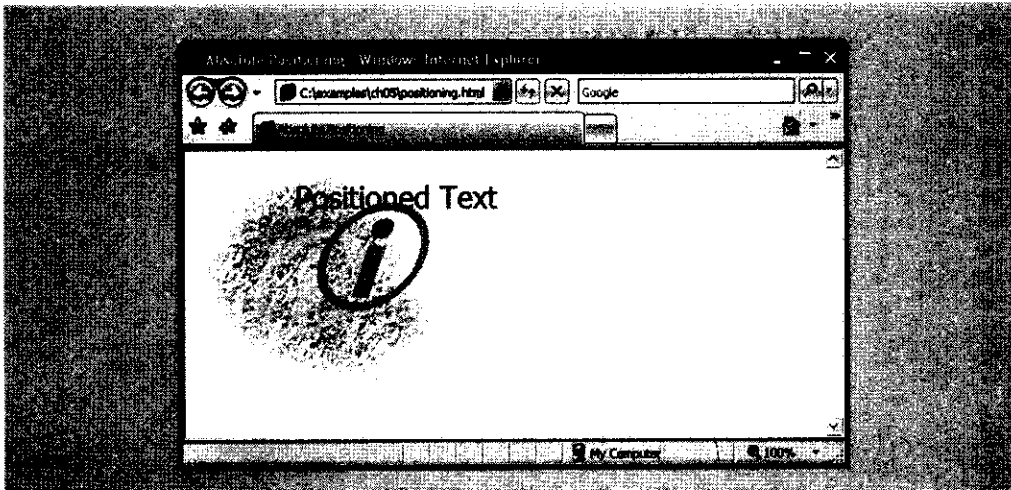


Fig. 5.6 | Absolute positioning of elements. (Part 2 of 2.)

Normally, elements are positioned on the page in the order that they appear in the XHTML document. Lines 11–14 define a style called `bgimg` for the first `img` element (`i.gif`) 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 (i.e., an element such as `body` or `p`). Here, we position the element to be 0 pixels away from both the top and left margins of its containing element. In line 28, this style is applied to the image, which is contained in a `p` element.

The `z-index` property allows you to layer overlapping elements properly. Elements that have higher `z-index` values are displayed in front of elements with lower `z-index` values. In this example, `i.gif` has the lowest `z-index` (1), so it displays in the background. The `.fgimg` CSS rule in lines 15–18 gives the circle image (`circle.gif`, in lines 31–32) a `z-index` of 2, so it displays in front of `i.gif`. The `p` element in line 34 (`Positioned Text`) is given a `z-index` of 3 in line 22, so it 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 they are encountered in the document.

Absolute positioning is not the only way to specify page layout. Figure 5.7 demonstrates **relative positioning**, in which elements are positioned relative to other elements.

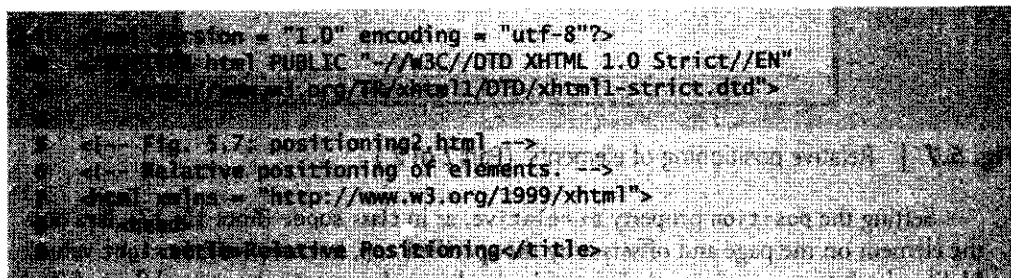


Fig. 5.7 | Relative positioning of elements. (Part 1 of 2.)

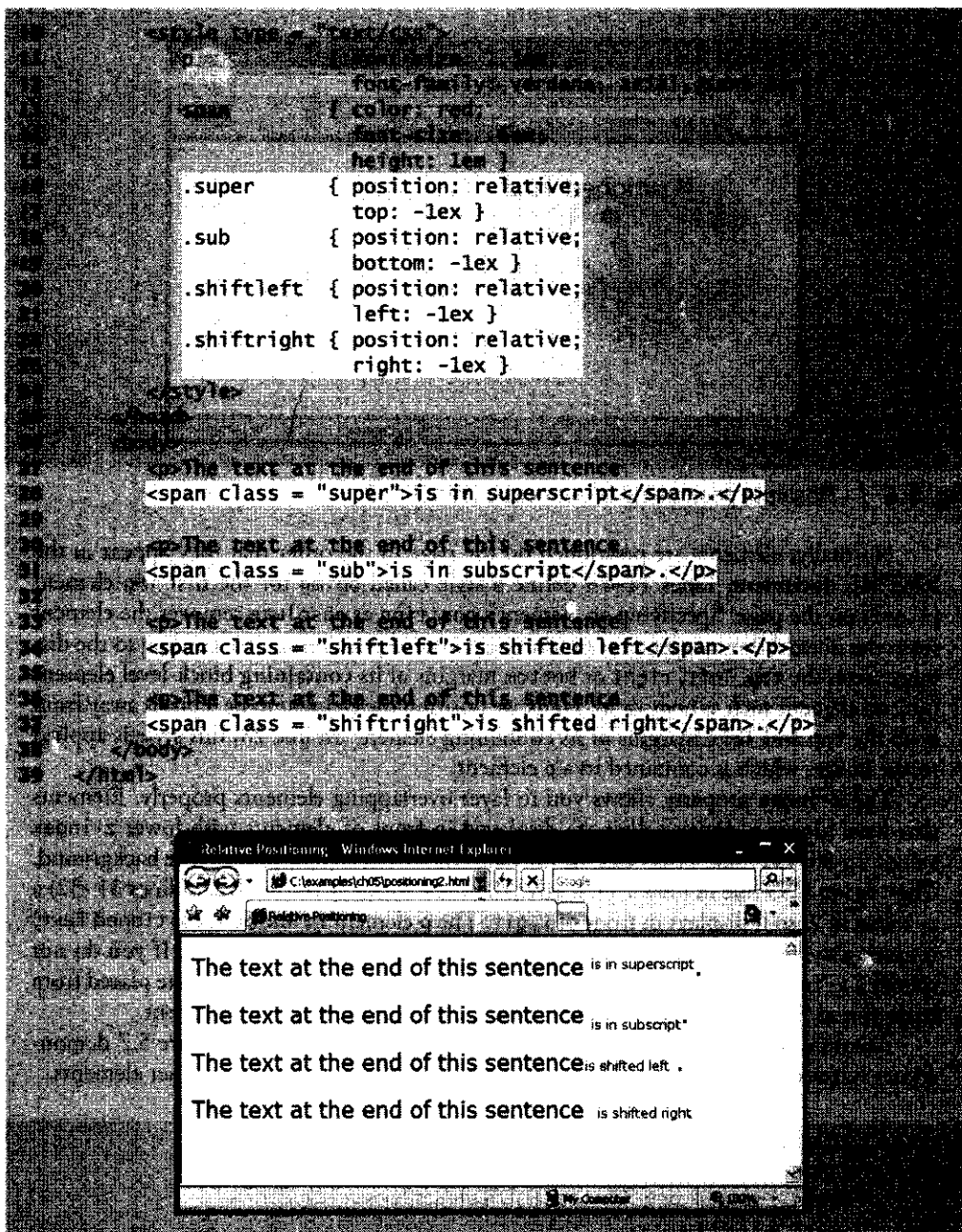


Fig. 5.7 | Relative positioning of elements. (Part 2 of 2.)

Setting the position property to relative, as in class `super` (lines 16–17), 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 17) is the *x*-height of a font, a relative-length measurement typically equal to the height of a lowercase *x*.



Common Programming Error 5.2

Because relative positioning keeps elements in the flow of text in your documents, be careful to avoid unintentionally overlapping text.

Inline and Block-Level Elements

We introduce the `span` element in line 28. Lines 13–15 define the CSS rule for all `span` elements. The height of the `span` determines how much vertical space the `span` will occupy. The font-size determines the size of the text inside the `span`.

Element `span` is a **grouping element**—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. Element `span` is an **inline-level element**—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 is a **block-level element**. This means it is displayed on its own line and has a virtual box around it. Examples of block-level elements include `div`, `p` and heading elements (`h1` through `h6`). We'll discuss inline and block-level elements in more detail in Section 5.9.

5.7 Backgrounds

CSS provides control over the background of block-level elements. CSS can set a background color or add background images to XHTML elements. Figure 5.8 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.

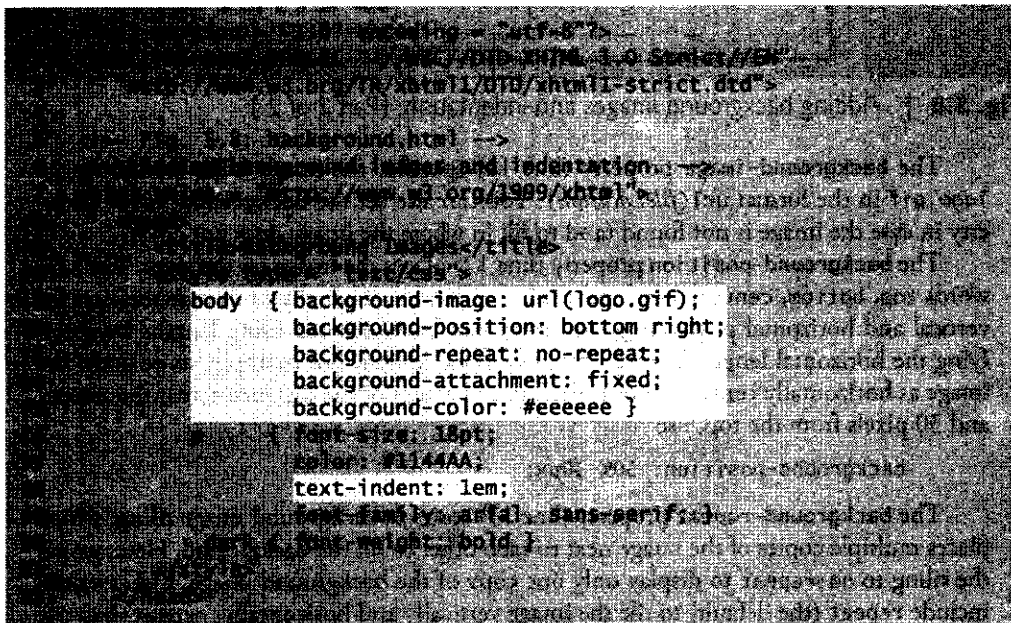


Fig. 5.8 | Adding background images and indentation. (Part I of 2.)

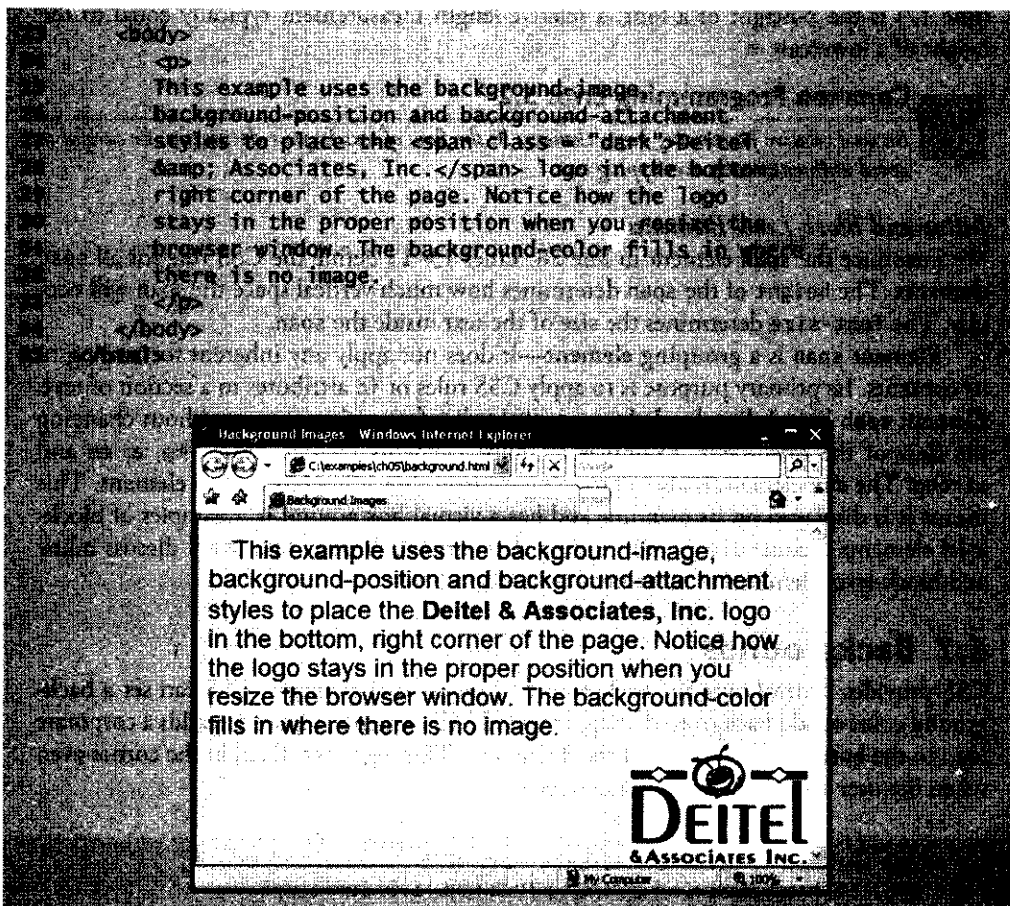


Fig. 5.8 | Adding background images and indentation. (Part 2 of 2.)

The **background-image** property (line 11) specifies the image URL for the image `logo.gif` in the format `url(fileLocation)`. You can also set the **background-color** property in case the image is not found (and to fill in where the image does not cover).

The **background-position** property (line 12) 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. An image can be positioned 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;
```

The **background-repeat** property (line 13) 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.

The final property setting, **background-attachment: fixed** (line 14), 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.

Line 18 uses the **text-indent** property to indent the first line of text in the element by a specified amount, in this case `1em`. An author 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.

Another CSS property that formats text is the **font-style** property, which allows the developer 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).

5.8 Element Dimensions

In addition to positioning elements, CSS rules can specify the actual dimensions of each page element. Figure 5.9 demonstrates how to set the dimensions of elements.

```

1 <?xml version = "1.0" encoding = "utf-8"?>
2 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
3   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
4 <html xmlns = "http://www.w3.org/1999/xhtml">
5   <head>
6     <title>Element Dimensions</title>
7     <style type = "text/css">
8       div { background-color: #aacdff;
9         margin-bottom: 5em;
10        font-family: arial, helvetica, sans-serif }
11    </style>
12  </head>
13  <body>
14    <div style = "width: 20%">Here is some
15    text that goes in a box which is
16    set to stretch across twenty percent
17    of the width of the screen.</div>
18
19    <div style = "width: 80%; text-align: center">
20    Here is some CENTERED text that goes in a box
21    which is set to stretch across eighty percent of
22    the width of the screen.</div>
23
24    <div style = "width: 20%; height: 150px; overflow: scroll">
25    This box is only twenty percent of
26    the width and has a fixed height.
27    What do we do if it overflows? Set the
28    overflow property to scroll!</div>
29  </body>
30 </html>

```

Fig. 5.9 | Element dimensions and text alignment. (Part 1 of 2.)

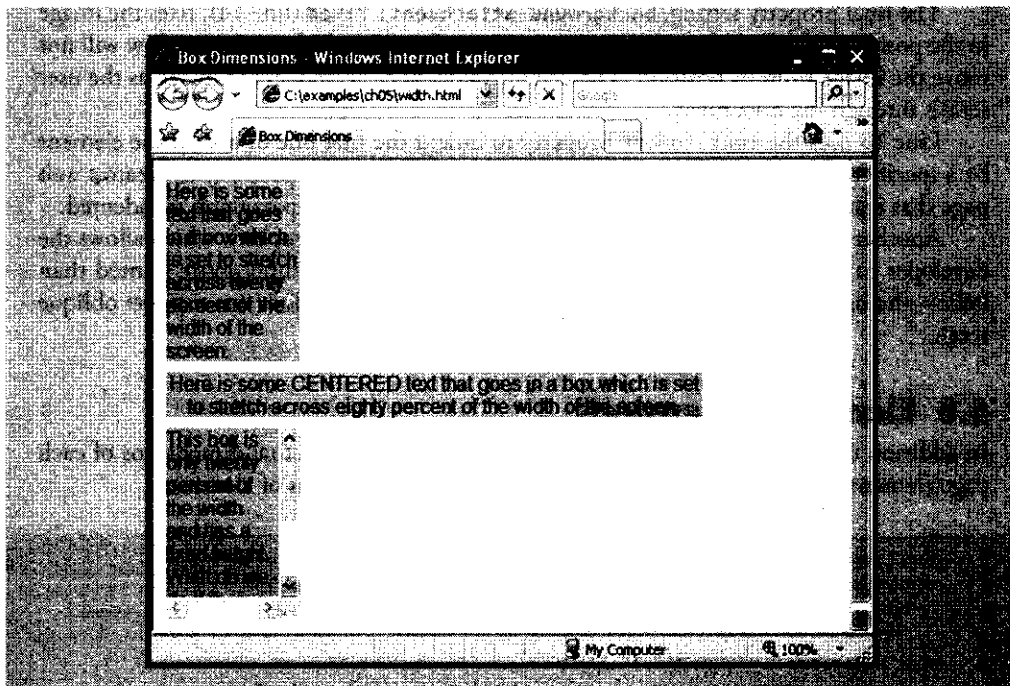


Fig. 5.9 | Element dimensions and text alignment. (Part 2 of 2.)

The inline style in line 17 illustrates how to set the **width** of an element on screen; here, we indicate that the `div` element should occupy 20 percent of the screen width. The height of an element can be set similarly, using the **height** property. The **height** and **width** 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. Most elements are left aligned by default; however, this alignment can be altered to position the element elsewhere. Line 22 sets text in the element to be center aligned; other values for the **text-align** property include **left** and **right**.

In the third `div`, we specify a percentage height and a pixel width. 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 27, we set the **overflow** property to `scroll`, a setting that adds scroll bars if the text overflows the boundaries.

5.9 Box Model and Text Flow

All block-level XHTML elements have a virtual box drawn around them based on what is known as the **box model**. When the browser renders elements using the box model, the content of each element is surrounded by **padding**, a **border** and a **margin** (Fig. 5.10).

CSS controls the border using three properties: **border-width**, **border-color** and **border-style**. We illustrate these three properties in Fig. 5.11.

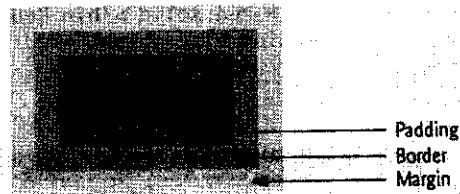


Fig. 5.10 | Box model for block-level elements.

Property `border-width` may be set to any valid CSS length (e.g., `em`, `ex`, `px`, etc.) 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 style borders.] 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.

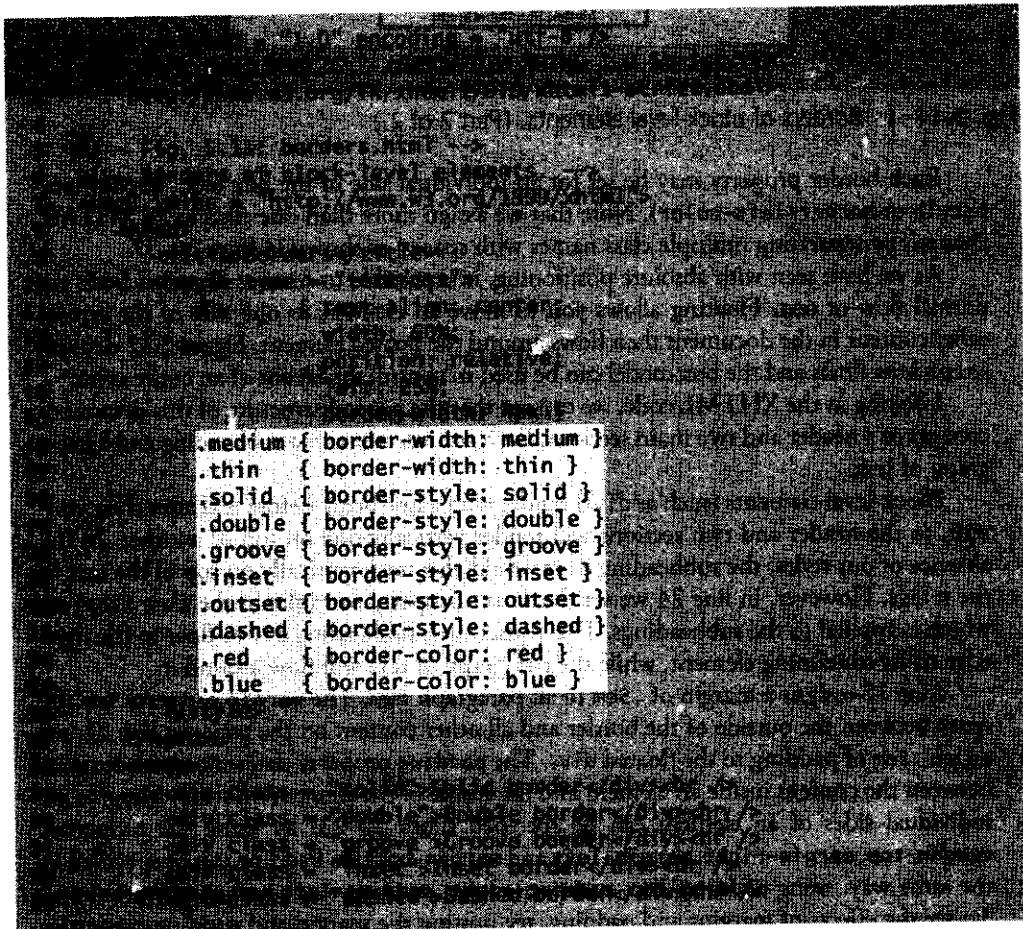


Fig. 5.11 | Borders of block-level elements. (Part 1 of 2.)

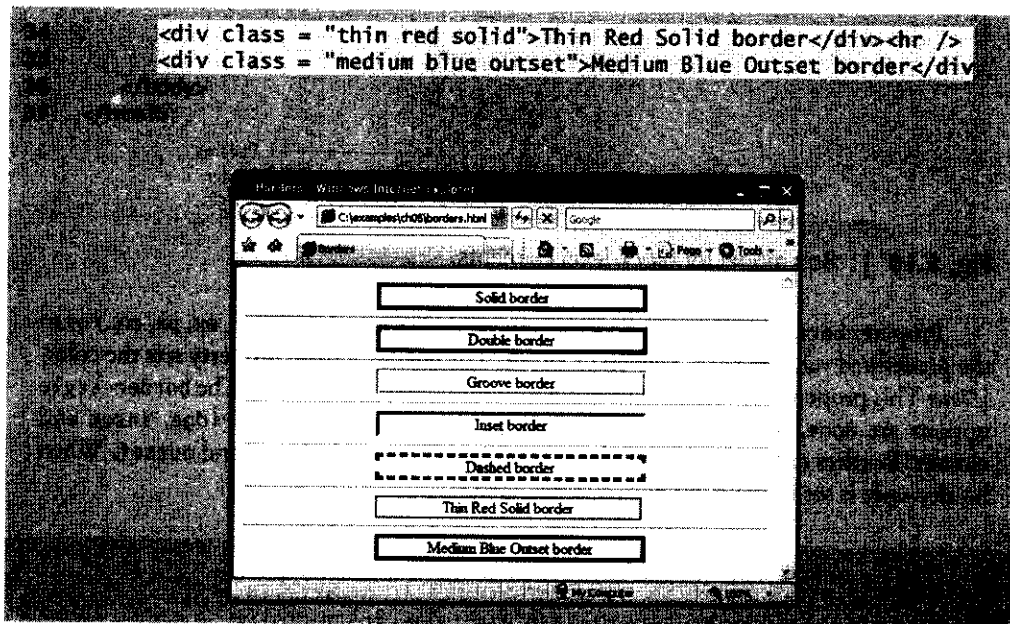


Fig. 5.11 | Borders of block-level elements. (Part 2 of 2.)

Each border property may be set for an individual side of the box (e.g., `border-top-style` or `border-left-color`). Note that we assign more than one class to an XHTML element by separating multiple class names with spaces, as shown in lines 36–37.

As we have seen with absolute positioning, it is 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 5.12 demonstrates how floats and the box model can be used to control the layout of an entire page.

Looking at the XHTML code, we can see that the general structure of this document consists of a header and two main sections. Each section contains a subheading and a paragraph of text.

Block-level elements (such as `div`s) render with a line break before and after their content, so the header and two sections will render vertically one on top of another. In the absence of our styles, the subheading `div`s 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 `float`d, which is applied to the subheadings. This causes each subheading `div` to float to the right edge of its containing element, while the paragraph of text will flow around it.

Line 17 assigns a margin of `.5em` to all paragraph tags. The **margin property** sets the space between the outside of the border and all other content on the page. In line 21, we assign `.2em` of padding to the floated `div`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 22–23) 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.

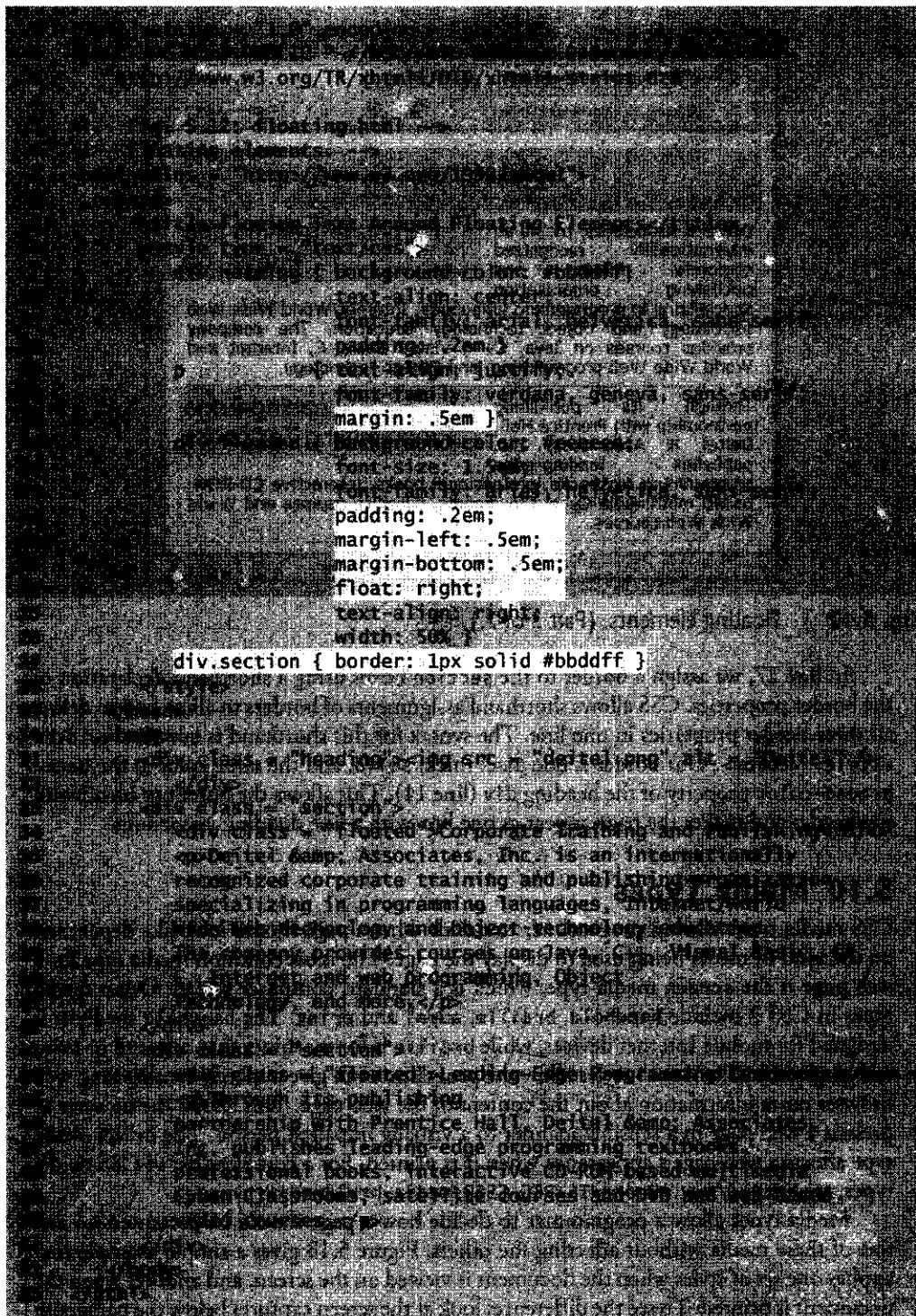


Fig. 5.12 | Floating elements. (Part I of 2.)



Fig. 5.12 | Floating elements. (Part 2 of 2.)

In line 27, we assign a border to the section boxes using a shorthand declaration of the border properties. CSS allows shorthand assignments of borders to 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 heading `div` (line 11). This allows the border to blend with the header and makes the page appear as one box with a line dividing its sections.

5.10 Media Types

CSS media types allow a programmer 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 2 include **handheld**, **braille**, **aural** and **print**. The **handheld** medium is designed for mobile Internet devices, while **braille** is for machines that can read or print web pages in braille. **aural** styles allow the programmer to give a speech-synthesizing web browser more information about the content of the web page. This allows the browser to present a web page in a sensible manner to a visually impaired person. The **print** media type affects a web page's appearance when it is printed. For a complete list of CSS media types, see <http://www.w3.org/TR/REC-CSS2/media.html#media-types>.

Media types allow a programmer to decide how a page should be presented on any one of these media without affecting the others. Figure 5.13 gives a simple example that applies one set of styles when the document is viewed on the screen, 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 Internet Explorer or Firefox.

In line 11, we begin a block of styles that applies to all media types, declared by `@media all` and enclosed in curly braces (`{` and `}`). In lines 13–18, we define some styles for all media types. Lines 20–27 set styles to be applied only when the page is printed, beginning with the declaration `@media print` and enclosed in curly braces.

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

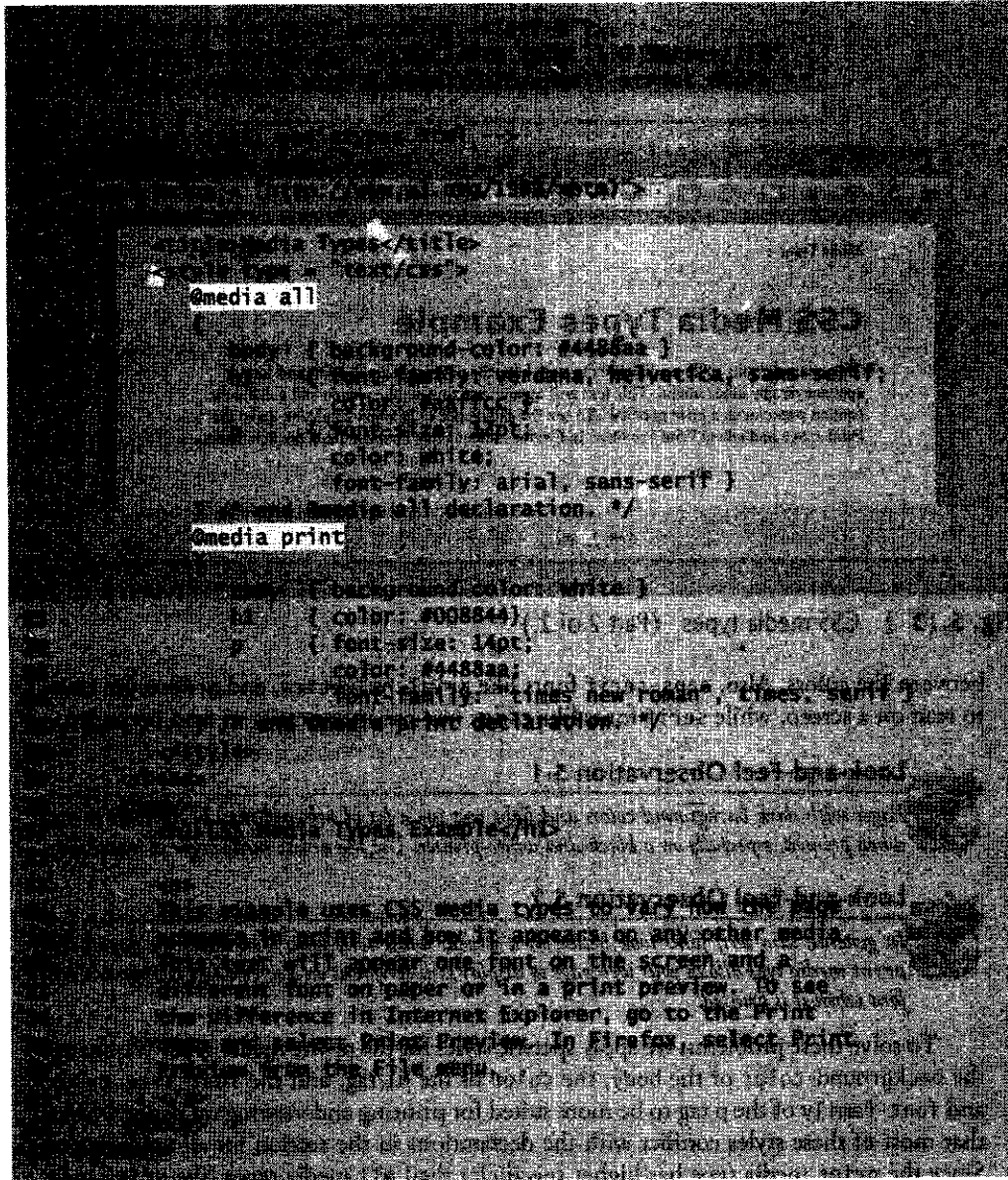


Fig. 5.13 | CSS media types. (Part 1 of 2.)

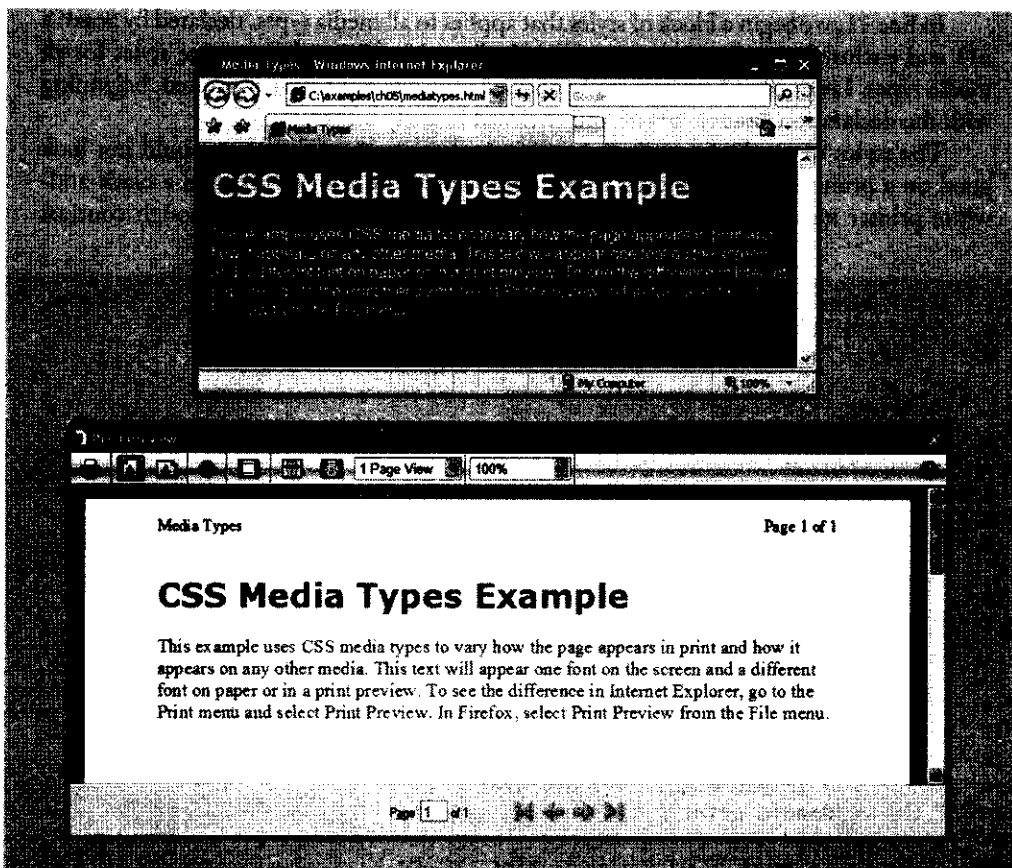


Fig. 5.13 | CSS media types. (Part 2 of 2.)

between the colors. Also, sans-serif fonts like arial, helvetica, and geneva are easier to read on a screen, while serif fonts like times new roman are easier to read on paper.



Look-and-Feel Observation 5.1

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 5.2

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 sans-serif font on a screen and change to a serif font when it is printed.

To solve these problems, we apply specific styles for the print media type. We change the background-color of the body, 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 all media types, the print styles override the styles for all media types when the page is printed. Since the font-family

property of the `h1` tag is not overridden in the `print` section, it retains its old value even when the page is printed.

5.11 Building a CSS Drop-Down Menu

Drop-down menus are a good way to provide navigation links on a website without using a lot of screen space. In this section, we take a second look at the `:hover` pseudoclass and introduce the `display` property to create a drop-down menu using CSS and XHTML.

We've already seen the `:hover` pseudoclass used to change a link's style when the mouse hovers over it. We will use this feature in a more advanced way to cause a menu to appear when the mouse hovers over a menu button. The other important property we need is the `display` property. This allows a programmer 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. 5.14.

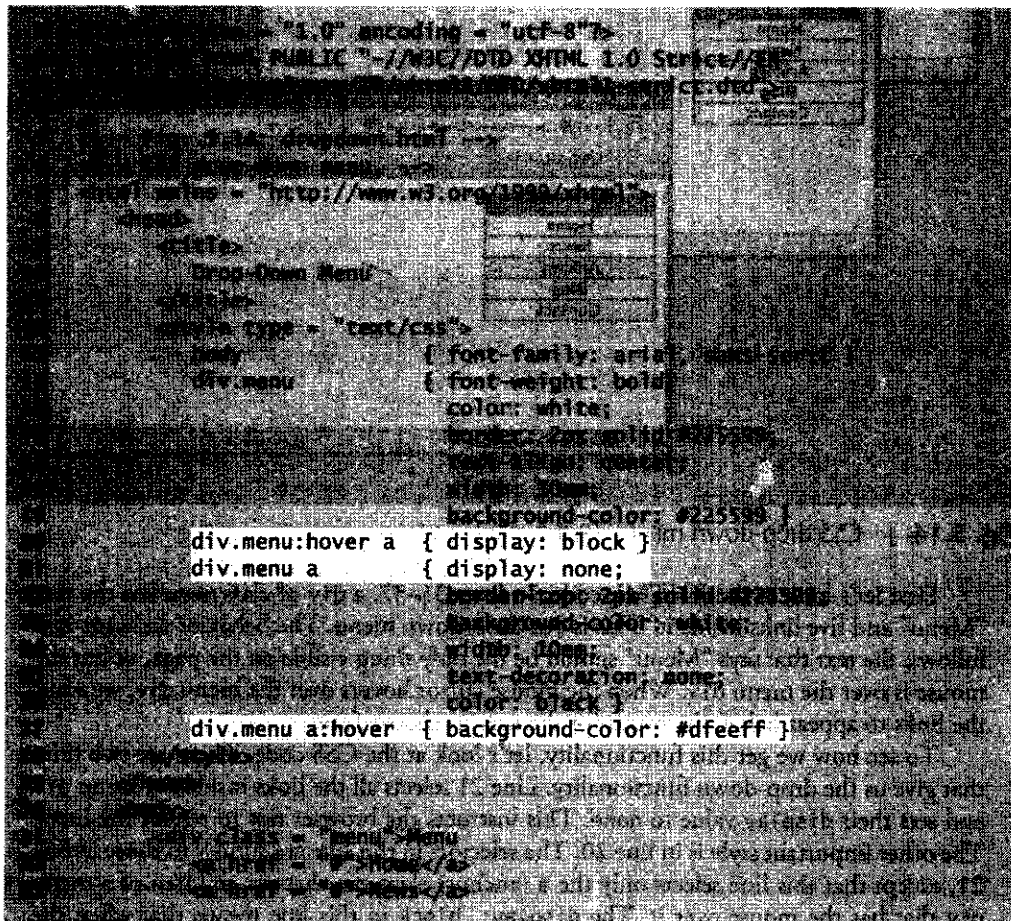


Fig. 5.14 | CSS drop-down menu. (Part 1 of 2.)

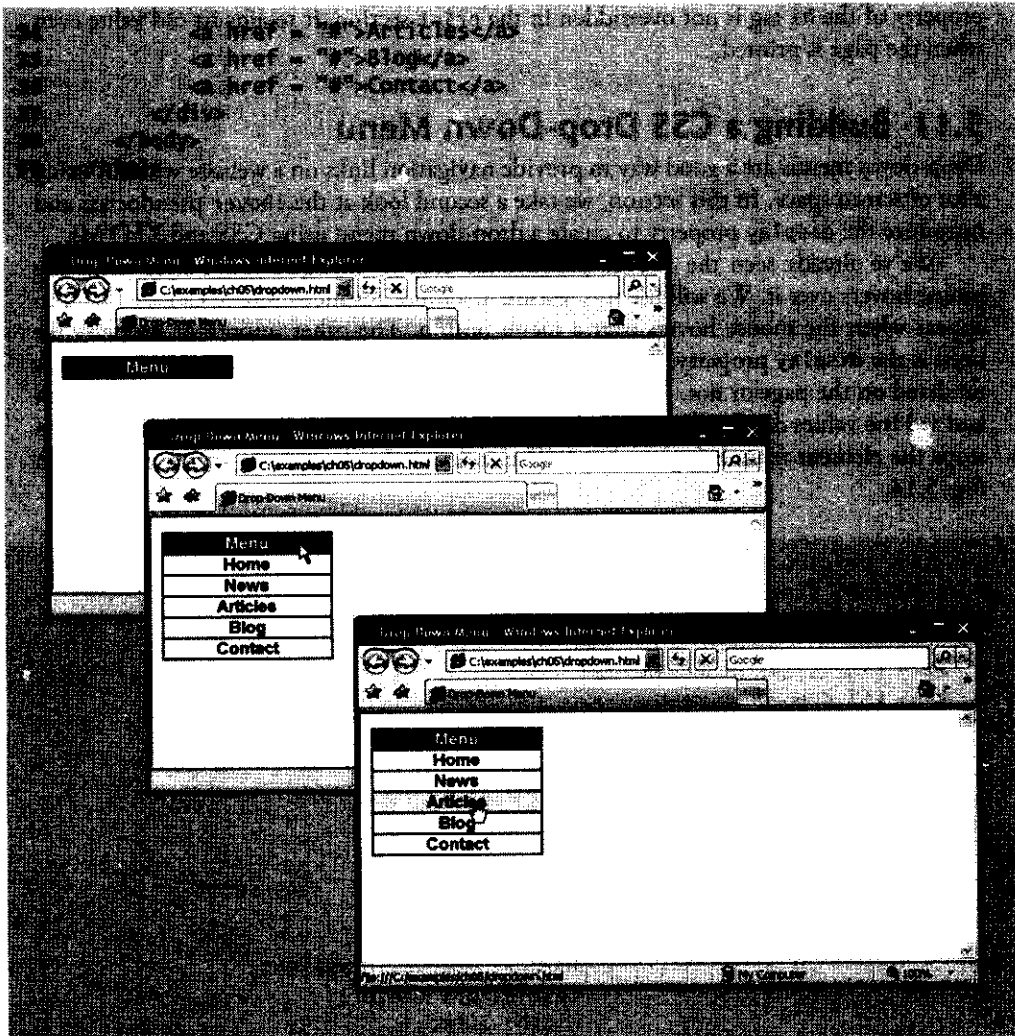


Fig. 5.14 | CSS drop-down menu. (Part 2 of 2.)

First let's look at the XHTML code. In lines 31–37, a `div` of class `menu` has the text "Menu" and five links inside it. This is our drop-down menu. The behavior we want is as follows: the text that says "Menu" should be the only thing visible on the page, unless the mouse is over the menu `div`. When the mouse cursor hovers over the menu `div`, we want the links to appear below the menu for the user to choose from.

To see how we get this functionality, let's look at the CSS code. There are two lines that give us the drop-down functionality. Line 21 selects all the links inside the menu `div` and sets their `display` value to `none`. This instructs the browser not to render the links. The other important style is in line 20. The selectors in this line are similar to those in line 21, except that this line selects only the `a` (anchor) elements that are children of a menu `div` that has the mouse over it. The `display: block` in this line means that when the mouse is over the menu `div`, the links inside it will be displayed as block-level elements.

The selectors in line 27 are also similar to lines 20 and 21. This time, however, the style is applied only to any a element that is a child of the menu div when that child 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 aesthetic style to the components of our menu. Look at the screen captures or run the code example to see the menu in action.

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. Specifically, check out List-o-Matic, an automatic CSS list generator located at www.accessify.com/tools-and-wizards/developer-tools/list-o-matic/ and Dynamic Drive's library of vertical and horizontal CSS menus at www.dynamicdrive.com/style/.

5.12 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. Web page authors need to be careful not to inadvertently override user preferences with defined styles. This section discusses possible conflicts between author styles and user styles.

Figure 5.15 contains an author style. The font-size is set to 9pt for all <p> tags that have class note applied to them.

User style sheets are external style sheets. Figure 5.16 shows a user style sheet that sets the body's font-size to 20pt, color to yellow and background-color to #000080.

User style sheets are not linked to a document; rather, they are set in the browser's options. To add a user style sheet in IE7, select **Internet Options...**, located in the **Tools** menu. In the **Internet Options** dialog (Fig. 5.17) that appears, click **Accessibility...**, check the **Format documents using my style sheet** checkbox, and type the location of the user style sheet. Internet Explorer 7 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

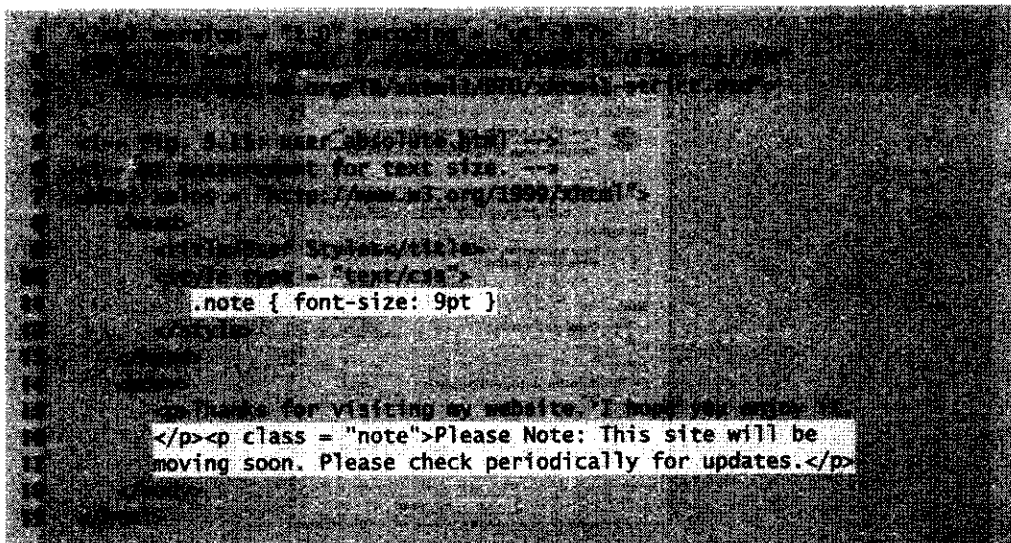


Fig. 5.15 | pt measurement for text size. (Part 1 of 2.)

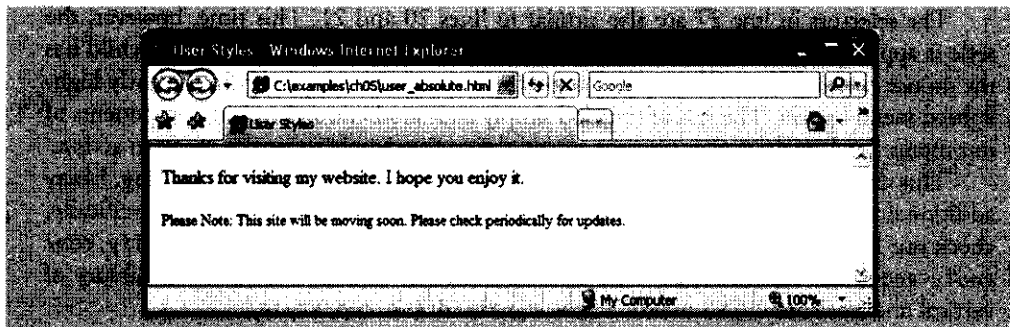


Fig. 5.15 | pt measurement for text size. (Part 2 of 2.)

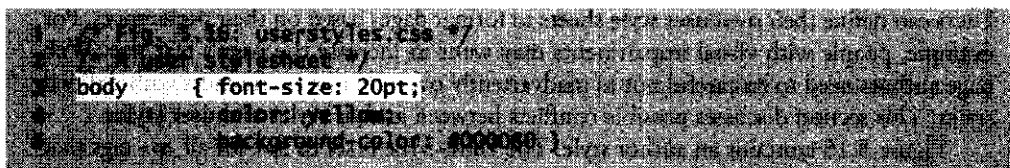


Fig. 5.16 | User style sheet.

www.mozilla.org/support/firefox/profile#locate and place a style sheet called `userContent.css` in the `chrome` subdirectory.

The web page from Fig. 5.15 is displayed in Fig. 5.18, with the user style sheet from Fig. 5.16 applied.

In this example, if users define their own 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

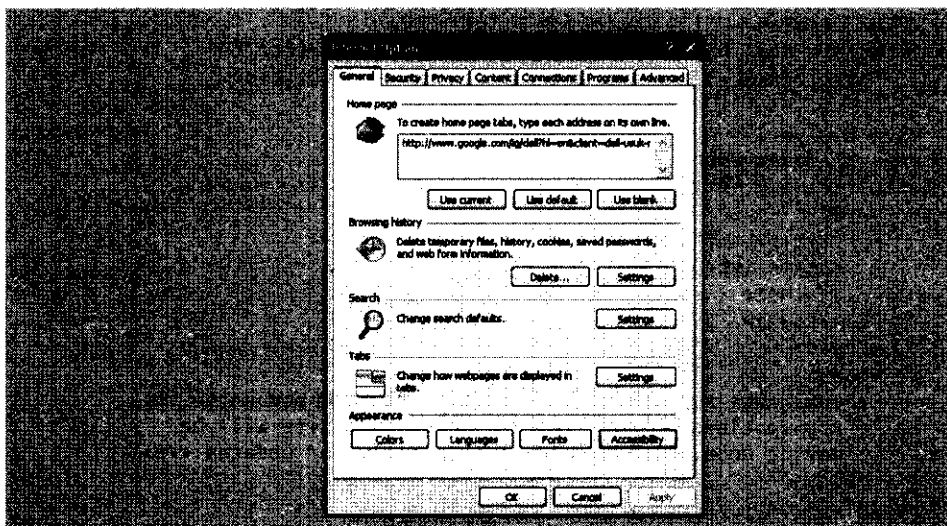


Fig. 5.17 | User style sheet in Internet Explorer 7. (Part 1 of 2.)

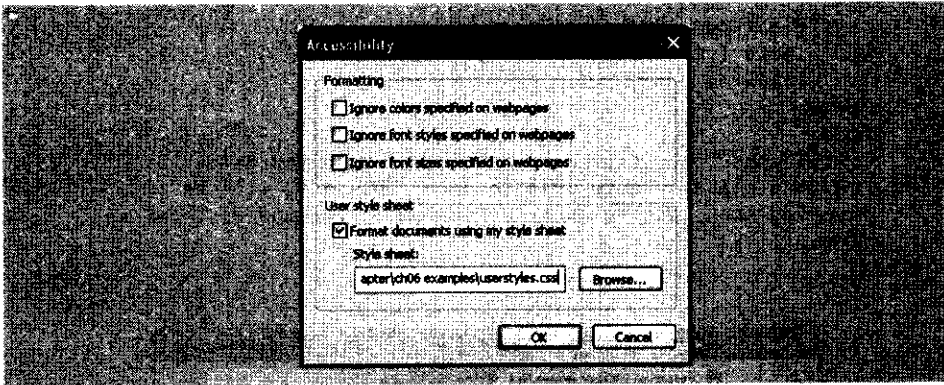


Fig. 5.17 | User style sheet in Internet Explorer 7. (Part 2 of 2.)

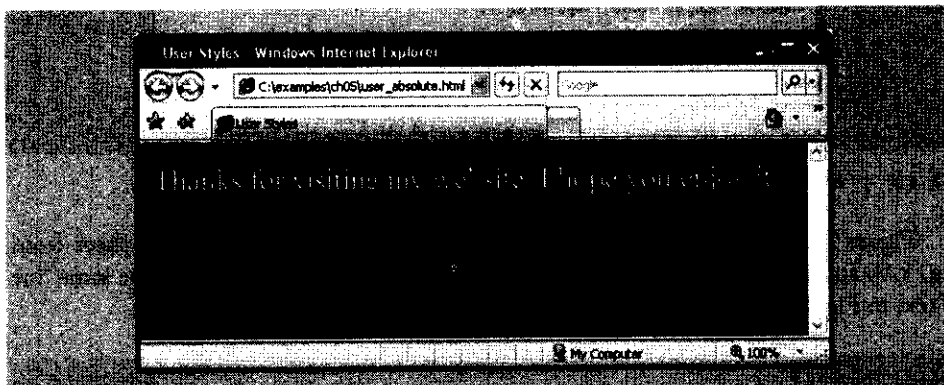


Fig. 5.18 | User style sheet applied with pt measurement.

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 5.19 changes the font-size property to use a relative measurement (line 11) that does not override the user style set in Fig. 5.16. 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 class note applied to them are displayed in 15pt (.75 times 20pt).

```

1 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
2   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
3
4 <!-- Fig. 5.19: user_relative.html -->
5 <!-- em measurement for text size. -->
6 <html xmlns="http://www.w3.org/1999/xhtml"
7   <!--
8   <title>user_styles</title>

```

Fig. 5.19 | em measurement for text size. (Part 1 of 2.)

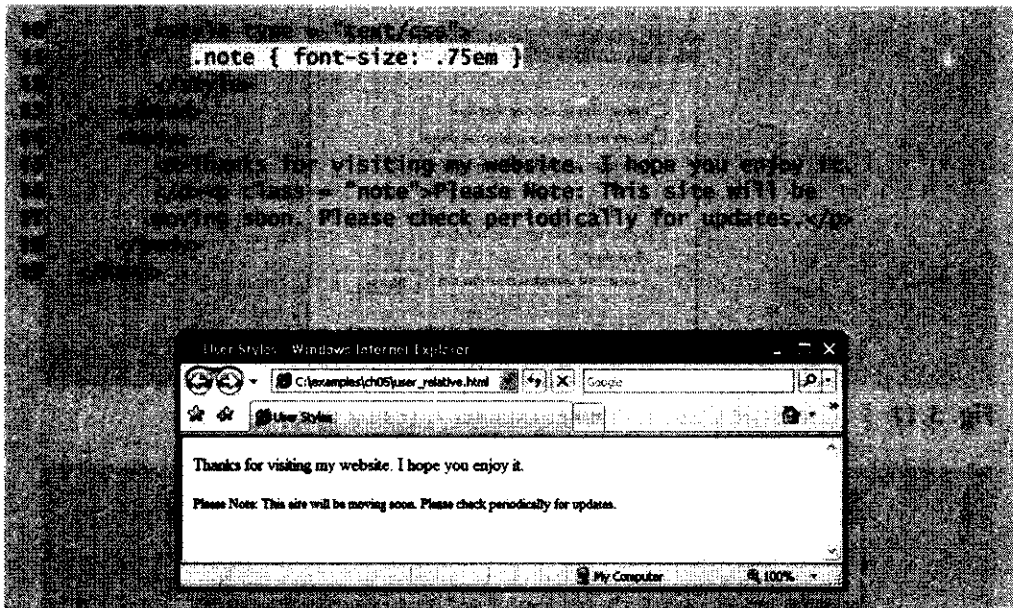


Fig. 5.19 | em measurement for text size. (Part 2 of 2.)

Figure 5.20 displays the web page from Fig. 5.19 with the user style sheet from Fig. 5.16 applied. Note that the second line of text displayed is larger than the same line of text in Fig. 5.18.

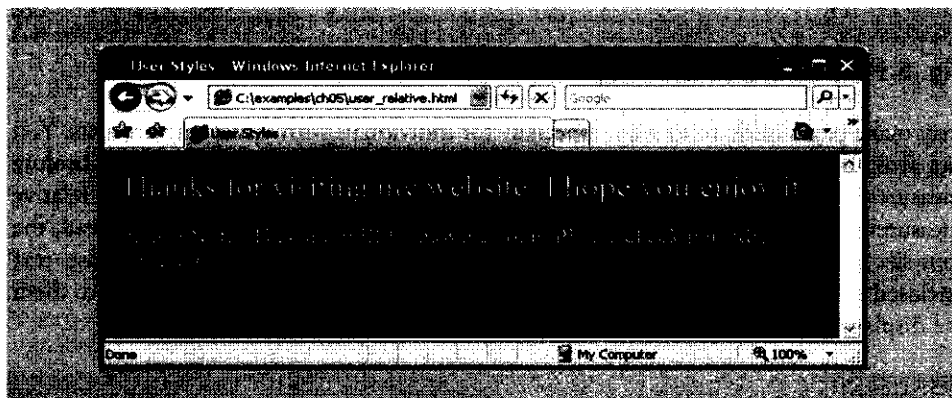


Fig. 5.20 | User style sheet applied with em measurement.

5.13 CSS 3

The W3C is currently developing CSS 3 and some browsers are beginning to implement some of the new features that will be in the CSS 3 specification. We discuss a few of the upcoming features that will most likely be included in CSS 3.

CSS 3 will allow for more advanced control of borders. In addition to the border-style, border-color, and border-width properties, you will be able to set multiple

border colors, use images for borders, add shadows to boxes, and create borders with rounded corners.

Background images will be more versatile in CSS 3, allowing the programmer to set the size of a background image, specify an offset to determine where in the element the image should be positioned, and use multiple background images in one element. There will also be properties to set shadow effects on text and more options for text overflow when the text is too long to fit in its containing element.

Additional features will include resizable boxes, enhanced selectors, multicolumn layouts, and more developed speech (aural) styles. The Web Resources section points you to the Deitel CSS Resource Center, where you can find links to the latest information on the development and features of CSS 3.

5.14 Web Resources

<http://www.deitel.com/css21>

The Deitel CSS Resource Center contains links to some of the best CSS 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 5.2 Inline Styles

- The inline style allows you to declare a style for an individual element by using the `style` attribute in the element's start tag.
- Each CSS property is followed by a colon and the value of the attribute. Multiple property declarations are separated by a semicolon.
- The `color` property sets text color. Color names and hexadecimal codes may be used as the value.

Section 5.3 Embedded Style Sheets

- Styles that are placed in a `style` element use selectors to apply style elements throughout the entire document.
- `style` element attribute `type` specifies the MIME type (the specific encoding format) of the style sheet. Style sheets use `text/css`.
- Each rule body in a style sheet begins and ends with a curly brace (`{` and `}`).
- The `font-weight` property 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 (e.g., 100, 200, ..., 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 applies a style class to an element.
- The CSS rules in a style sheet use the same format as inline styles: The property is followed by a colon (`:`) and the value of that property. Multiple properties are separated by semicolons (`;`).
- The `background-color` attribute specifies the background color of the element.

- The `font-family` attribute names a specific font that should be displayed. Generic font families allow authors to specify a type of font instead of a specific font, in case a browser does not support a specific font. The `font-size` property specifies the size used to render the font.

Section 5.4 *Conflicting Styles*

- Most styles are inherited from parent elements. Styles defined for children have higher specificity and take precedence over the parent's styles.
- Pseudoclasses give the author access to content not specifically declared in the document. The `hover` pseudoclass is activated when the user moves the mouse cursor over an element.
- The `text-decoration` property applies decorations to text in an element, such as `underline`, `overline`, `line-through` and `blink`.
- To apply rules to multiple elements, separate the elements with commas in the style sheet.
- To apply rules to only a certain type of element that is a child of another type, separate the element names with spaces.
- A pixel is a relative-length measurement: It varies in size based on screen resolution. Other relative lengths are `em`, `ex` and percentages.
- The other units of measurement available in CSS are absolute-length measurements—that is, units that do not vary in size. These units can be `in` (inches), `cm` (centimeters), `mm` (millimeters), `pt` (points; 1 pt = 1/72 in) or `pc` (picas; 1 pc = 12 pt).

Section 5.5 *Linking External Style Sheets*

- External linking of style sheets can create a uniform look for a website; separate pages can all use the same styles. Modifying a single style-sheet file makes changes to styles across an entire website.
- Link's `rel` attribute specifies a relationship between two documents. For style sheets, the `rel` attribute declares the linked document to be a `stylesheet` 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 5.6 *Positioning Elements*

- The CSS `position` property allows absolute positioning, 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 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.
- 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` is a grouping element—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—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 is a block-level element. This means it is displayed on its own line and has a virtual box around it. Examples of block-level elements include `div`, `p` and heading elements (`h1` through `h6`).

Section 5.7 Backgrounds

- Property `background-image` specifies the URL of the image, in the format `url(fileLocation)`. The property `background-position` 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 controls the tiling of the background image. 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 property setting `background-attachment: 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 indents the first line of text in the element by the specified amount.
- The `font-style` property allows you to set text to `none`, `italic` or `oblique` (`oblique` will default to `italic` if the system does not have a separate font file for oblique text, which is normally the case).

Section 5.8 Element Dimensions

- The dimensions of elements on a page can be set with CSS by using properties `height` and `width`.
- Text in an element can be centered using `text-align`; other values for the `text-align` property are `left` and `right`.
- One 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 must be made large enough for all the content to fit. However, a developer can set the `overflow` property to `scroll`; this setting adds scroll bars if the text overflows the boundaries set for it.

Section 5.9 Box Model and Text Flow

- The `border-width` property may be set to any of the CSS lengths or to the predefined value of `thin`, `medium` or `thick`.
- The `border-styles` available are `none`, `hidden`, `dotted`, `dashed`, `solid`, `double`, `groove`, `ridge`, `inset` and `outset`.
- The `border-color` property sets the color used for the border.
- The `class` attribute allows more than one class to be assigned to an XHTML 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 XHTML file. 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 determines the distance between the element's edge and any outside text.
- Margins for individual sides of an element can be specified by using `margin-top`, `margin-right`, `margin-left` and `margin-bottom`.
- The `padding` property determines the distance between the content inside an element and the edge of the element. Padding also can be set for each side of the box by using `padding-top`, `padding-right`, `padding-left` and `padding-bottom`.

Section 5.10 Media Types

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

Section 5.11 Building a CSS Drop-Down Menu

- The :hover pseudoclass is used to apply styles to an element when the mouse cursor is over it.
- The display property allows a programmer to decide if an element is displayed as a block element, inline element, or is not rendered at all (none).

Section 5.12 User Style Sheets

- Users can define their own user style sheets 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.

Section 5.13 CSS 3

- While CSS 2 is the current W3C Recommendation, CSS 3 is in development, and some browsers are beginning to implement some of the new features that will be in the CSS 3 specification.
- CSS 3 will introduce new features related to borders, backgrounds, text effects, layout, and more.

Terminology

| | |
|--------------------------------|---------------------------|
| absolute positioning | class attribute |
| absolute-length measurement | cm (centimeter) |
| ancestor element | colon (:) |
| arial font | color property |
| aural media type | CSS 2 |
| author style | CSS 3 |
| background-attachment property | CSS comment |
| background-color property | CSS property |
| background-image property | CSS rule |
| background-position property | CSS selector |
| background-repeat property | curly generic font family |
| blink text decoration | dashed border style |
| block-level element | descendant element |
| border | display property |
| border-color property | div element |
| border-style property | dotted border style |
| border-width property | double border style |
| box model | em (M-height of font) |
| braille media type | embedded style sheet |
| Cascading Style Sheets (CSS) | ex (x-height of font) |
| | external style sheets |

float property
 floated element
 font-family property
 font-size property
 font-style property
 font-weight property
 generic font family
 groove border style
 grouping element
 handheld media type
 height property
 hidden border style
 in (inch)
 inheritance
 inline-level element
 inline style
 inset border style
 large relative font size
 larger relative font size
 left property value
 line-through text decoration
 link element
 linking to an external style sheet
 margin property
 margin-bottom property
 margin-left property
 margin-right property
 margin-top property
 media types
 medium relative border width
 medium relative font size
 MIME (Multipurpose Internet Mail
 Extensions) type
 mm (millimeter)
 monospace font
 none border style
 outset border style
 overflow property
 overline text decoration
 padding property
 parent element
 pc (pica)
 position property
 print media type
 pseudoclass
 pt (point)
 rel attribute (link)
 relative positioning
 relative-length measurement
 repeat property value
 ridge border style
 right property value
 sans-serif generic font family
 screen media type
 scroll property value
 selector
 separation of structure from content
 serif generic font family
 small relative font size
 smaller relative font size
 solid border style
 span element
 specificity
 style
 style attribute
 style class
 style in header section of document
 text flow
 text/css MIME type
 text-align property
 text-decoration property
 text-indent property
 thick border width
 thin border width
 user agent
 user style sheet
 width property
 x-large relative font size
 x-small relative font size
 xx-large relative font size
 xx-small relative font size
 z-index property

Self-Review Exercises

- 5.1 Assume that the size of the base font on a system is 12 points.
- How big is a 36-point font in ems?
 - How big is a 9-point font in ems?
 - How big is a 24-point font in picas?
 - How big is a 12-point font in inches?
 - How big is a 1-inch font in picas?

5.2 Fill in the blanks in the following statements:

- a) Using the _____ element allows authors to use external style sheets in their 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 _____ pseudoclass is activated when the user moves the mouse cursor over the specified element.
- e) Setting the overflow property to _____ provides a mechanism for containing inner content without compromising specified box 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 to only 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 _____.


Exercises

5.3 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.

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

5.5 Add an embedded style sheet to the XHTML document in Fig. 4.5. 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.

5.6 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.



Comment is free, but facts are sacred.

—C. P. Scott

The creditor hath a better memory than the debtor.

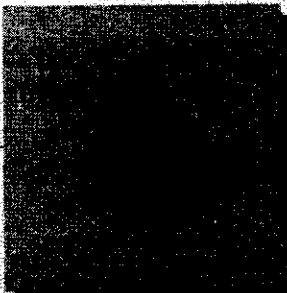
—James Howell

When faced with a decision, I always ask, "What would be the most fun?"

—Peggy Walker

Equality, in a social sense, may be divided into that of condition and that of rights.

—James Fenimore Cooper



JavaScript: Introduction to Scripting

OBJECTIVES

In this chapter you will learn:

- To write simple JavaScript programs.
- To use input and output statements.
- Basic memory concepts.
- To use arithmetic operators.
- The precedence of arithmetic operators.
- To write decision-making statements.
- To use relational and equality operators.

- 6.1 Introduction
- 6.2 Simple Program: Displaying a Line of Text in a Web Page
- 6.3 Modifying Our First Program
- 6.4 Obtaining User Input with `prompt` Dialogs
 - 6.4.1 Dynamic Welcome Page
 - 6.4.2 Adding Integers
- 6.5 Memory Concepts
- 6.6 Arithmetic
- 6.7 Decision Making: Equality and Relational Operators
- 6.8 Web Resources

Summary | Terminology | Self-Review Exercises | Exercises

6.1 Introduction

In the first five chapters, we introduced the Internet and Web, web browsers, Web 2.0, XHTML and Cascading Style Sheets (CSS). In this chapter, we begin our introduction to the JavaScript¹ scripting language, which facilitates a disciplined approach to designing computer programs that enhance the functionality and appearance of web pages.²

In Chapters 6–11, we present a detailed discussion of JavaScript—the *de facto* standard client-side scripting language for web-based applications due to its highly portable nature. Our treatment of JavaScript serves two purposes—it introduces client-side scripting (used in Chapters 6–13), which makes web pages more dynamic and interactive, and it provides the programming foundation for the more complex server-side scripting presented later in the book.

We now introduce JavaScript programming and present examples that illustrate several important features of JavaScript. Each example is carefully analyzed one line at a time. In Chapters 7–8, we present a detailed treatment of program development and program control in JavaScript.

Before you can run code examples with JavaScript on your computer, you may need to change your browser's security settings. By default, Internet Explorer 7 prevents scripts on your local computer from running, displaying a yellow warning bar at the top of the window instead. To allow scripts to run in files on your computer, select **Internet Options** from the **Tools** menu. Click the **Advanced** tab and scroll down to the **Security** section of

1. Many people confuse the scripting language JavaScript with the programming language Java (from Sun Microsystems, Inc.). Java is a full-fledged object-oriented programming language. It can be used to develop applications that execute on a range of devices—from the smallest devices (such as cell phones and PDAs) to supercomputers. Java is popular for developing large-scale distributed enterprise applications and web applications. JavaScript is a browser-based scripting language developed by Netscape and implemented in all major browsers.
2. JavaScript was originally created by Netscape. Both Netscape and Microsoft have been instrumental in the standardization of JavaScript by ECMA International as ECMAScript. Detailed information about the current ECMAScript standard can be found at www.ecma-international.org/publications/standards/ECMA-262.htm.

the **Settings** list. Check the box labeled **Allow active content to run in files on My Computer** (Fig. 6.1). Click **OK** and restart Internet Explorer. XHTML documents on your own computer that contain JavaScript code will now run properly. Firefox has JavaScript enabled by default.

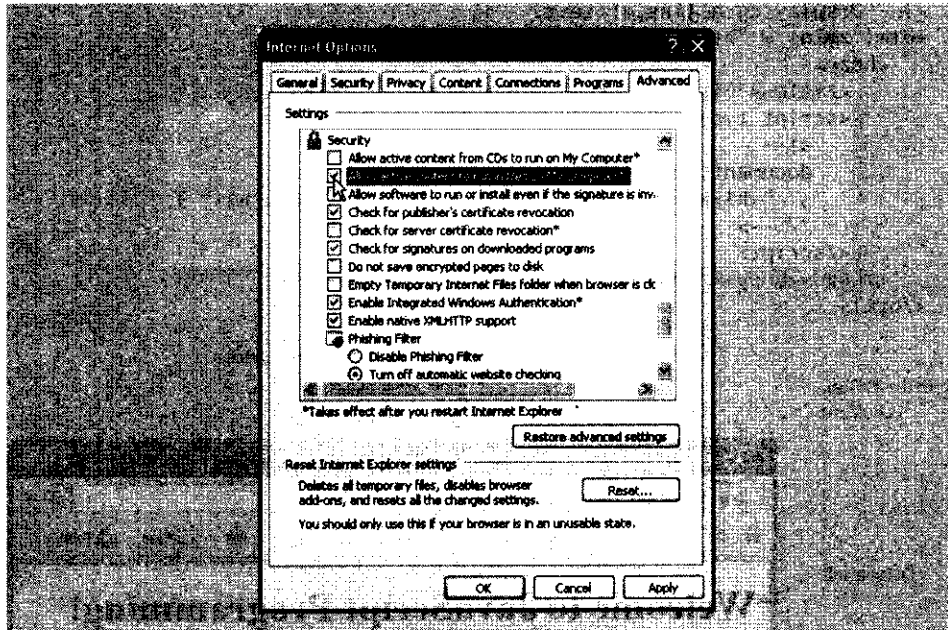


Fig. 6.1 | Enabling JavaScript in Internet Explorer 7

6.2 Simple Program: Displaying a Line of Text in a Web Page

JavaScript uses notations that may appear strange to nonprogrammers. We begin by considering a simple **script** (or **program**) that displays the text "Welcome to JavaScript Programming!" in the body of an XHTML document. All major web browsers contain **JavaScript interpreters**, which process the commands written in JavaScript. The JavaScript code and its output in Internet Explorer are shown in Fig. 6.2.

This program illustrates several important JavaScript features. We consider each line of the XHTML document and script in detail. As in the preceding chapters, we have given each XHTML document line numbers for the reader's convenience; the line numbers are not part of the XHTML document or of the JavaScript programs. Lines 12–13 do the "real work" of the script, namely, displaying the phrase `Welcome to JavaScript Programming!` in the web page.

Line 8 indicates the beginning of the `<head>` section of the XHTML document. For the moment, the JavaScript code we write will appear in the `<head>` section. The browser interprets the contents of the `<head>` section first, so the JavaScript programs we write there execute before the `<body>` of the XHTML document displays. In later chapters on JavaScript and in the chapters on dynamic HTML, we illustrate **inline scripting**, in which JavaScript code is written in the `<body>` of an XHTML document.

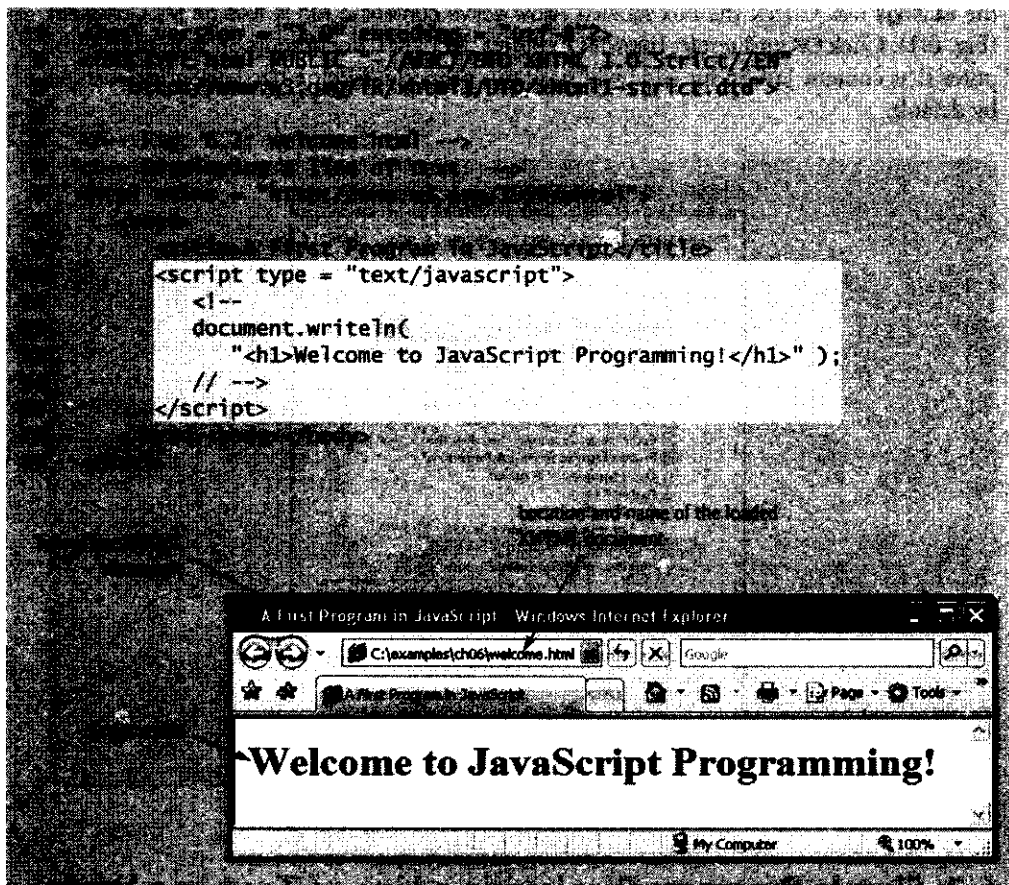


Fig. 6.2 | Displaying a line of text.

Line 10 uses the `<script>` tag to indicate to the browser that the text which follows is part of a script. The `type` attribute specifies the type of file as well as the **scripting language** used in the script—in this case, a text file written in javascript. Both Internet Explorer and Firefox use JavaScript as the default scripting language.

Line 11 contains the XHTML opening comment tag `<!--`. Some older web browsers do not support scripting. In such browsers, the actual text of a script often will display in the web page. To prevent this from happening, many script programmers enclose the script code in an XHTML comment, so that browsers that do not support scripts will simply ignore the script. The syntax used is as follows:

```

<script type = "text/javascript">
  <!--
    script code here
  // -->
</script>

```

When a browser that does not support scripts encounters the preceding code, it ignores the `<script>` and `</script>` tags and the script code in the XHTML comment. Browsers

that do support scripting will interpret the JavaScript code as expected. [Note: Some browsers require the **JavaScript single-line comment** `//` (see Section 6.4 for an explanation) before the ending XHTML comment delimiter (`-->`) to interpret the script properly. The opening HTML comment tag (`<!--`) also serves as a single line comment delimiter in JavaScript, therefore it does not need to be commented.]



Portability Tip 6.1

Some browsers do not support the `<script>...</script>` tags. If your document is to be rendered with such browsers, enclose the script code between these tags in an XHTML comment, so that the script text does not get displayed as part of the web page. The closing comment tag of the XHTML comment (`-->`) is preceded by a JavaScript comment (`//`) to prevent the browser from trying to interpret the XHTML comment as a JavaScript statement.

Lines 12–13 instruct the browser’s JavaScript interpreter to perform an **action**, namely, to display in the web page the **string** of characters contained between the **double quotation** (`"`) marks. A string is sometimes called a **character string**, a **message** or a **string literal**. We refer to characters between double quotation marks as strings. Individual white-space characters between words in a string are not ignored by the browser. However, if consecutive spaces appear in a string, browsers condense them to a single space. Also, in most cases, browsers ignore leading white-space characters (i.e., white space at the beginning of a string).



Software Engineering Observation 6.1

Strings in JavaScript can be enclosed in either double quotation marks (`"`) or single quotation marks (`'`).

Lines 12–13 use the browser’s **document object**, which represents the XHTML document the browser is currently displaying. The document object allows you to specify text to display in the XHTML document. The browser contains a complete set of objects that allow script programmers to access and manipulate every element of an XHTML document. In the next several chapters, we overview some of these objects as we discuss the Document Object Model (DOM).

An object resides in the computer’s memory and contains information used by the script. The term **object** normally implies that **attributes** (**data**) and **behaviors** (**methods**) are associated with the object. The object’s methods use the attributes to perform useful actions for the **client of the object** (i.e., the script that calls the methods). A method may require additional information (**arguments**) to perform its action; this information is enclosed in parentheses after the name of the method in the script. In lines 12–13, we call the document object’s **writeIn** method to write a line of XHTML markup in the XHTML document. The parentheses following the method name `writeIn` contain the one argument that method `writeIn` requires (in this case, the string of XHTML that the browser is to display). Method `writeIn` instructs the browser to display the argument string. If the string contains XHTML elements, the browser interprets these elements and renders them on the screen. In this example, the browser displays the phrase `Welcome to JavaScript Programming!` as an h1-level XHTML heading, because the phrase is enclosed in an h1 element.

The code elements in lines 12–13, including `document.writeIn`, its argument in the parentheses (the string) and the semicolon (`;`), together are called a **statement**. Every

statement ends with a semicolon (also known as the **statement terminator**), although this practice is not required by JavaScript. Line 15 indicates the end of the script.



Good Programming Practice 6.1

Always include a semicolon at the end of a statement to terminate the statement. This notation clarifies where one statement ends and the next statement begins.



Common Programming Error 6.1

Forgetting the ending `</script>` tag for a script may prevent the browser from interpreting the script properly and may prevent the XHTML document from loading properly.

The `</head>` tag in line 16 indicates the end of the `<head>` section. Also in line 16, the tags `<body>` and `</body>` specify that this XHTML document has an empty body. Line 17 indicates the end of this XHTML document.

We are now ready to view our XHTML document in a web browser—open it in Internet Explorer or Firefox. If the script contains no syntax errors, it should produce the output shown in Fig. 6.2.



Common Programming Error 6.2

*JavaScript is case sensitive. Not using the proper uppercase and lowercase letters is a syntax error. A syntax error occurs when the script interpreter cannot recognize a statement. The interpreter normally issues an error message to help you locate and fix the incorrect statement. Syntax errors are violations of the rules of the programming language. The interpreter notifies you of a syntax error when it attempts to execute the statement containing the error. The JavaScript interpreter in Internet Explorer reports all syntax errors by indicating in a separate popup window that a “runtime error” has occurred (i.e., a problem occurred while the interpreter was running the script). [Note: To enable this feature in IE7, select **Internet Options...** from the **Tools** menu. In the **Internet Options** dialog that appears, select the **Advanced** tab and click the checkbox labelled **Display a notification about every script error** under the **Browsing** category. Firefox has an error console that reports JavaScript errors and warnings. It is accessible by choosing **Error Console** from the **Tools** menu.]*



Error-Prevention Tip 6.1

When the interpreter reports a syntax error, sometimes the error is not on the line number indicated by the error message. First, check the line for which the error was reported. If that line does not contain errors, check the preceding several lines in the script.

6.3 Modifying Our First Program

This section continues our introduction to JavaScript programming with two examples that modify the example in Fig. 6.2.

Displaying a Line of Colored Text

A script can display `Welcome to JavaScript Programming!` several ways. Figure 6.3 uses two JavaScript statements to produce one line of text in the XHTML document. This example also displays the text in a different color, using the CSS `color` property.

Most of this XHTML document is identical to Fig. 6.2, so we concentrate only on lines 12–14 of Fig. 6.3, which display one line of text in the XHTML document. The first statement uses document method `write` to display a string. Unlike `writeln`, `write` does

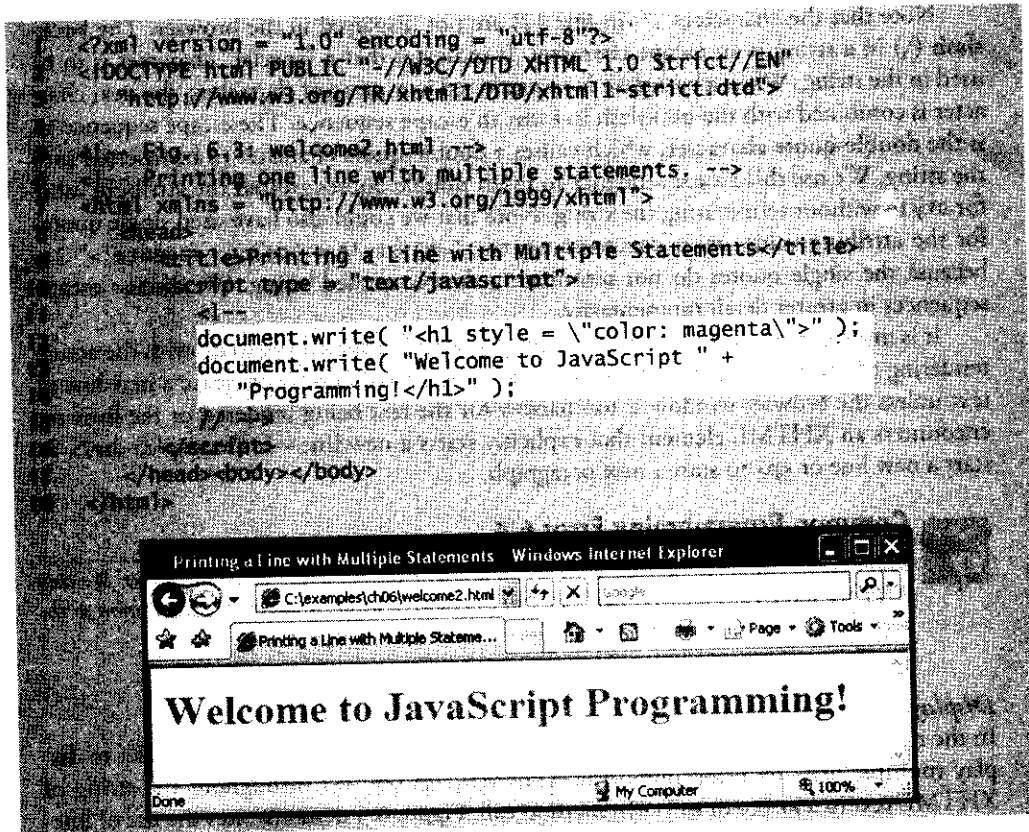


Fig. 6.3 | Printing one line with separate statements.

not position the output cursor in the XHTML document at the beginning of the next line after writing its argument. [Note: The output cursor keeps track of where the next character appears in the XHTML document, not where the next character appears in the web page as rendered by the browser.] The next character written in the XHTML document appears immediately after the last character written with `write`. Thus, when lines 13–14 execute, the first character written, “w,” appears immediately after the last character displayed with `write` (the `>` character inside the right double quote in line 12). Each `write` or `writeln` statement resumes writing characters where the last `write` or `writeln` statement stopped writing characters. So, after a `writeln` statement, the next output appears on the beginning of the next line. In effect, the two statements in lines 12–14 result in one line of XHTML text. Remember that statements in JavaScript are separated by semicolons (;). Therefore, lines 13–14 represent only one complete statement. JavaScript allows large statements to be split over many lines. However, you cannot split a statement in the middle of a string. The `+` operator (called the “concatenation operator” when used in this manner) in line 13 joins two strings together and is explained in more detail later in this chapter.



Common Programming Error 6.3

Splitting a statement in the middle of a string is a syntax error.

Note that the characters `\` (in line 12) are not displayed in the browser. The **backslash** (`\`) in a string is an **escape character**. It indicates that a “special” character is to be used in the string. When a backslash is encountered in a string of characters, the next character is combined with the backslash to form an **escape sequence**. The escape sequence `\` is the **double-quote character**, which causes a double-quote character to be inserted into the string. We use this escape sequence to insert double quotes around the attribute value for `style` without terminating the string. Note that we could also have used single quotes for the attribute value, as in `document.write("<h1 style = 'color: magenta'>");`, because the single quotes do not terminate a double-quoted string. We discuss escape sequences in greater detail momentarily.

It is important to note that the preceding discussion has nothing to do with the actual rendering of the XHTML text. Remember that the browser does not create a new line of text unless the browser window is too narrow for the text being rendered or the browser encounters an XHTML element that explicitly starts a new line—for example, `
` to start a new line or `<p>` to start a new paragraph.



Common Programming Error 6.4

Many people confuse the writing of XHTML text with the rendering of XHTML text. Writing XHTML text creates the XHTML that will be rendered by the browser for presentation to the user.

Displaying Multiple Lines of Text

In the next example, we demonstrate that a single statement can cause the browser to display multiple lines by using line-break XHTML tags (`
`) throughout the string of XHTML text in a `write` or `writeln` method call. Figure 6.4 demonstrates the use of line-break XHTML tags. Lines 12–13 produce three separate lines of text when the browser renders the XHTML document.

```

1 <?xml version = "1.0" encoding = "utf-8"?>
2 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
3   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
4
5 Fig. 6.4: welcome3.html -->
6 Printing on multiple lines with a single statement. -->
7 <html xmlns = "http://www.w3.org/1999/xhtml">
8   <head>
9     <title>Printing Multiple Lines</title>
10    <script type = "text/javascript">
11
12      document.writeln( "<h1>Welcome to<br />JavaScript" +
13        "<br />Programming!</h1>" );
14
15    </script>
16  </head><body></body>
17 </html>

```

Fig. 6.4 | Printing on multiple lines with a single statement. (Part 1 of 2.)

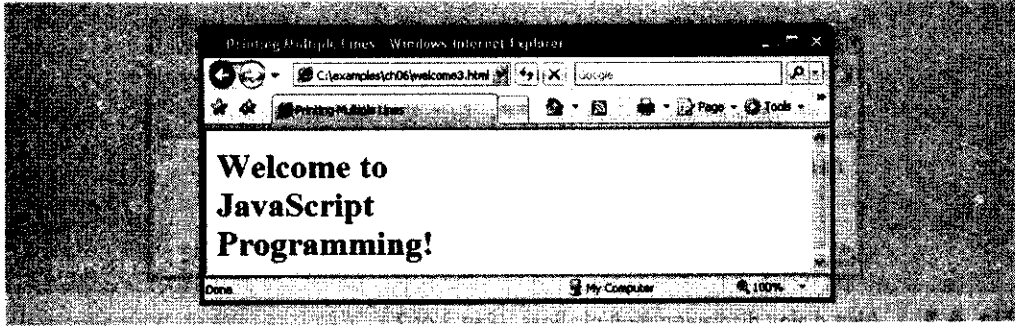


Fig. 6.4 | Printing on multiple lines with a single statement. (Part 2 of 2.)

Displaying Text in an Alert Dialog

The first several programs in this chapter display text in the XHTML document. Sometimes it is useful to display information in windows called **dialogs** (or **dialog boxes**) that “pop up” on the screen to grab the user’s attention. Dialogs typically display important messages to users browsing the web page. JavaScript allows you easily to display a dialog box containing a message. The program in Fig. 6.5 displays `Welcome to JavaScript Programming!` as three lines in a predefined dialog called an **alert dialog**.

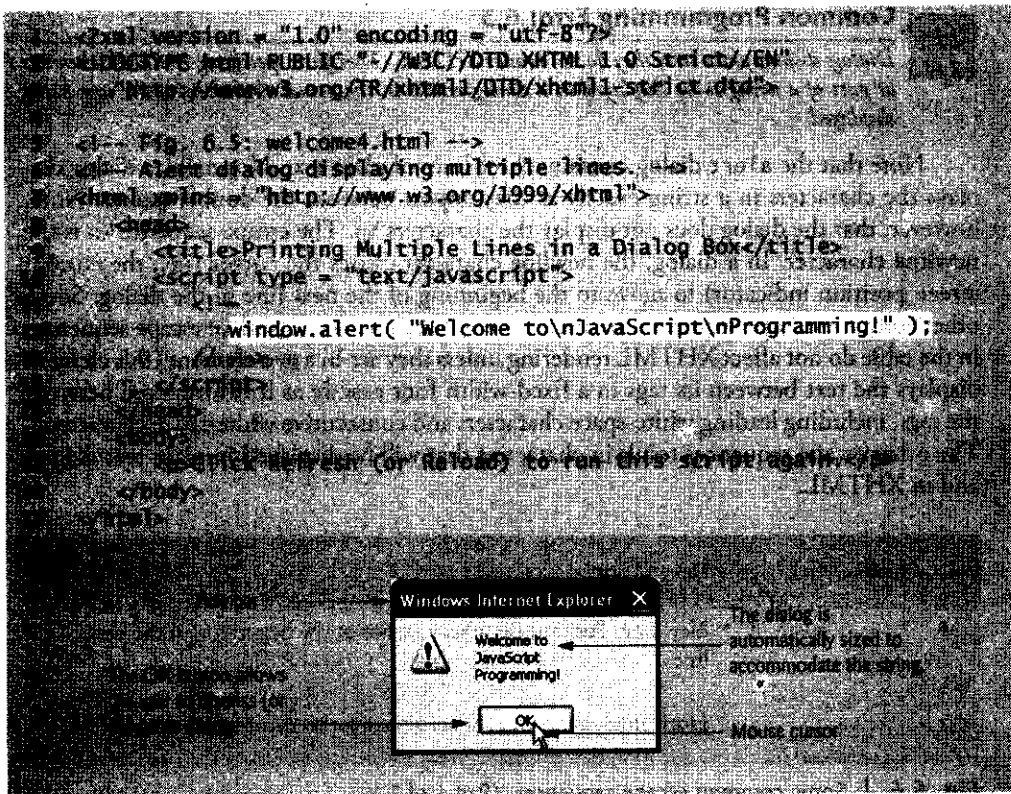


Fig. 6.5 | Alert dialog displaying multiple lines. (Part 1 of 2.)

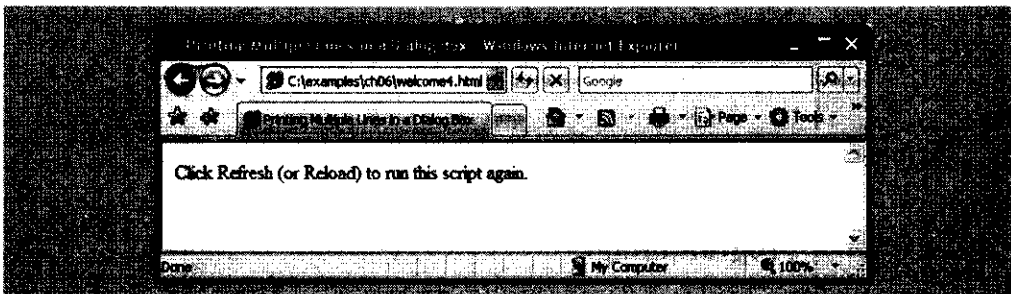


Fig. 6.5 | Alert dialog displaying multiple lines. (Part 2 of 2.)

Line 12 in the script uses the browser's **window** object to display an alert dialog. The argument to the window object's **alert** method is the string to display. Executing the preceding statement displays the dialog shown in the first window of Fig. 6.5. The title bar of the dialog contains the string **Windows Internet Explorer** to indicate that the browser is presenting a message to the user. The dialog provides an **OK** button that allows the user to **dismiss** (i.e., close) the dialog by clicking the button. To dismiss the dialog, position the **mouse cursor** (also called the **mouse pointer**) over the **OK** button and click the mouse. Firefox's alert dialog looks similar, but the title bar contains the text **[JavaScript Application]**.



Common Programming Error 6.5

Dialogs display plain text; they do not render XHTML. Therefore, specifying XHTML elements as part of a string to be displayed in a dialog results in the actual characters of the tags being displayed.

Note that the **alert** dialog contains three lines of plain text. Normally, a dialog displays the characters in a string exactly as they appear between the double quotes. Note, however, that the dialog does not display the characters `\n`. The escape sequence `\n` is the **newline character**. In a dialog, the newline character causes the **cursor** (i.e., the current screen position indicator) to move to the beginning of the next line in the dialog. Some other common escape sequences are listed in Fig. 6.6. The `\n`, `\t` and `\r` escape sequences in the table do not affect XHTML rendering unless they are in a **pre** element (this element displays the text between its tags in a fixed-width font exactly as it is formatted between the tags, including leading white-space characters and consecutive white-space characters). The other escape sequences result in characters that will be displayed in plain text dialogs and in XHTML.

| | |
|-----------------|---|
| <code>\n</code> | New line. Position the screen cursor at the beginning of the next line. |
| <code>\t</code> | Horizontal tab. Move the screen cursor to the next tab stop. |

Fig. 6.6 | Some common escape sequences. (Part 1 of 2.)

| | |
|----|---|
| \r | Carriage return. Position the screen cursor to the beginning of the current line; do not advance to the next line. Any characters output after the carriage return overwrite the characters previously output on that line. |
| \\ | Backslash. Used to represent a backslash character in a string. |
| \" | Double quote. Used to represent a double-quote character in a string contained in double quotes. For example, <pre>window.alert("\"" in quotes");</pre> displays "in quotes" in an alert dialog. |
| ' | Single quote. Used to represent a single-quote character in a string. For example, <pre>window.alert(\'in quotes\');</pre> displays 'in quotes' in an alert dialog. |

Fig. 6.6 | Some common escape sequences. (Part 2 of 2.)



Common Programming Error 6.6

XHTML elements in an alert dialog's message are not interpreted as XHTML. This means that using `
`, for example, to create a line break in an alert box is an error. The string `
` will simply be included in your message.

6.4 Obtaining User Input with prompt Dialogs

Scripting gives you the ability to generate part or all of a web page's content at the time it is shown to the user. A script can adapt the content based on input from the user or other variables, such as the time of day or the type of browser used by the client. Such web pages are said to be **dynamic**, as opposed to static, since their content has the ability to change. The next two subsections use scripts to demonstrate dynamic web pages.

6.4.1 Dynamic Welcome Page

Our next script builds on prior scripts to create a dynamic welcome page that obtains the user's name, then displays it on the page. The script uses another predefined dialog box from the `window` object—a **prompt** dialog—which allows the user to input a value that the script can use. The program asks the user to input a name, then displays the name in the XHTML document. Figure 6.7 presents the script and sample output. [Note: In later JavaScript chapters, we obtain input via GUI components in XHTML forms, as introduced in Chapter 4.]

Line 12 is a **declaration** that contains the JavaScript keyword `var`. Keywords are words that have special meaning in JavaScript. The keyword `var` at the beginning of the statement indicates that the word `name` is a **variable**. A variable is a location in the computer's memory where a value can be stored for use by a program. All variables have a name, type and value, and should be declared with a `var` statement before they are used in

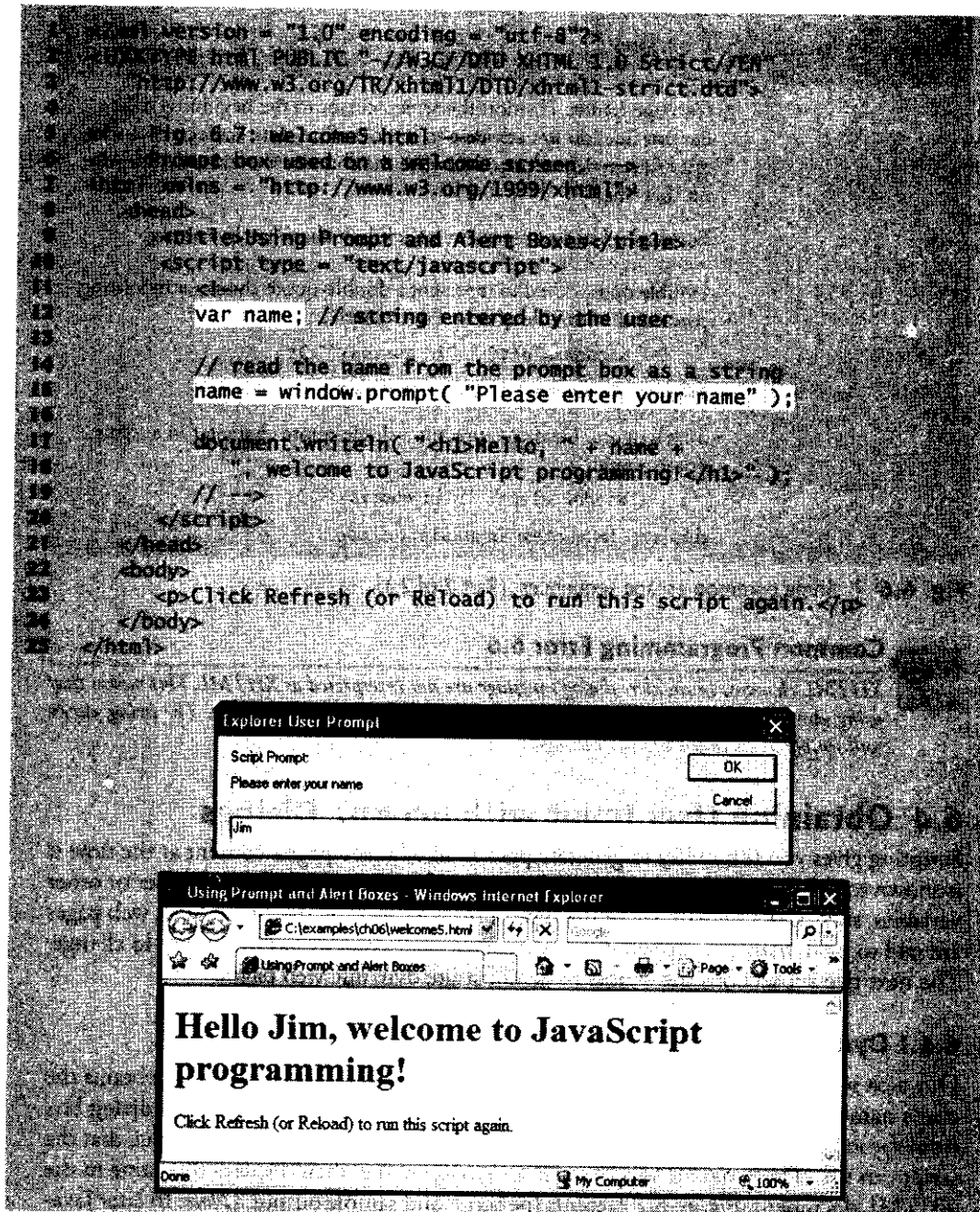


Fig. 6.7 | Prompt box used on a welcome screen.

a program. Although using `var` to declare variables is not required, we will see in Chapter 9, JavaScript: Functions, that `var` sometimes ensures proper behavior of a script.

The name of a variable can be any valid identifier. An identifier is a series of characters consisting of letters, digits, underscores (`_`) and dollar signs (`$`) that does not begin with a digit and is not a reserved JavaScript keyword. [Note: A complete list of keywords can be

found in Fig. 7.2.) Identifiers may not contain spaces. Some valid identifiers are `Welcome`, `$value`, `_value`, `m_inputField1` and `button7`. The name `7button` is not a valid identifier, because it begins with a digit, and the name `input field` is not valid, because it contains a space. Remember that JavaScript is **case sensitive**—uppercase and lowercase letters are considered to be different characters, so `name`, `Name` and `NAME` are different identifiers.



Good Programming Practice 6.2

Choosing meaningful variable names helps a script to be “self-documenting” (i.e., easy to understand by simply reading the script, rather than having to read manuals or extended comments).



Good Programming Practice 6.3

By convention, variable-name identifiers begin with a lowercase first letter. Each subsequent word should begin with a capital first letter. For example, identifier `itemPrice` has a capital `P` in its second word, `Price`.



Common Programming Error 6.7

Splitting a statement in the middle of an identifier is a syntax error.

Declarations end with a semicolon (;) and can be split over several lines with each variable in the declaration separated by a comma—known as a **comma-separated list** of variable names. Several variables may be declared either in one declaration or in multiple declarations.

Programmers often indicate the purpose of each variable in the program by placing a JavaScript comment at the end of each line in the declaration. In line 12, a **single-line comment** that begins with the characters `//` states the purpose of the variable in the script. This form of comment is called a single-line comment because it terminates at the end of the line in which it appears. A `//` comment can begin at any position in a line of JavaScript code and continues until the end of the line. Comments do not cause the browser to perform any action when the script is interpreted; rather, comments are ignored by the JavaScript interpreter.



Good Programming Practice 6.4

Some programmers prefer to declare each variable on a separate line. This format allows for easy insertion of a descriptive comment next to each declaration. This is a widely followed professional coding standard.

Another comment notation facilitates the writing of **multiline comments**. For example,

```
/* This is a multiline
   comment. It can be
   split over many lines. */
```

is a multiline comment spread over several lines. Such comments begin with the delimiter `/*` and end with the delimiter `*/`. All text between the delimiters of the comment is ignored by the interpreter.



Common Programming Error 6.8

Forgetting one of the delimiters of a multiline comment is a syntax error.



Common Programming Error 6.9

Nesting multiline comments (i.e., placing a multiline comment between the delimiters of another multiline comment) is a syntax error.

JavaScript adopted comments delimited with `/*` and `*/` from the C programming language and single-line comments delimited with `//` from the C++ programming language. JavaScript programmers generally prefer C++-style single-line comments over C-style comments. Throughout this book, we use C++-style single-line comments.

Line 14 is a comment indicating the purpose of the statement in the next line. Line 15 calls the window object's `prompt` method, which displays the dialog in Fig. 6.8. The dialog allows the user to enter a string representing the user's name.

The argument to `prompt` specifies a message telling the user what to type in the text field. This message is called a **prompt** because it directs the user to take a specific action. An optional second argument, separated from the first by a comma, may specify the default string displayed in the text field; our code does not supply a second argument. In this case, Internet Explorer displays the default value `undefined`, while Firefox and most other browsers leave the text field empty. The user types characters in the text field, then clicks the **OK** button to submit the string to the program. We normally receive input from a user through a GUI component such as the `prompt` dialog, as in this program, or through an XHTML form GUI component, as we will see in later chapters.

The user can type anything in the text field of the `prompt` dialog. For this program, whatever the user enters is considered the name. If the user clicks the **Cancel** button, no string value is sent to the program. Instead, the `prompt` dialog submits the value `null`, a JavaScript keyword signifying that a variable has no value. Note that `null` is not a string literal, but rather a predefined term indicating the absence of value. Writing a `null` value to the document, however, displays the word `null` in the web page.

The statement in line 15 **assigns** the value returned by the window object's `prompt` method (a string containing the characters typed by the user—or the default value or `null` if the **Cancel** button is clicked) to variable name by using the **assignment operator**, `=`. The statement is read as, "name gets the value returned by `window.prompt("Please enter your name")`." The `=` operator is called a **binary operator** because it has two **operands**—name and the result of the expression `window.prompt("Please enter your name")`. This entire statement is called an **assignment statement** because it assigns a value to a variable. The expression to the right of the assignment operator is always evaluated first.

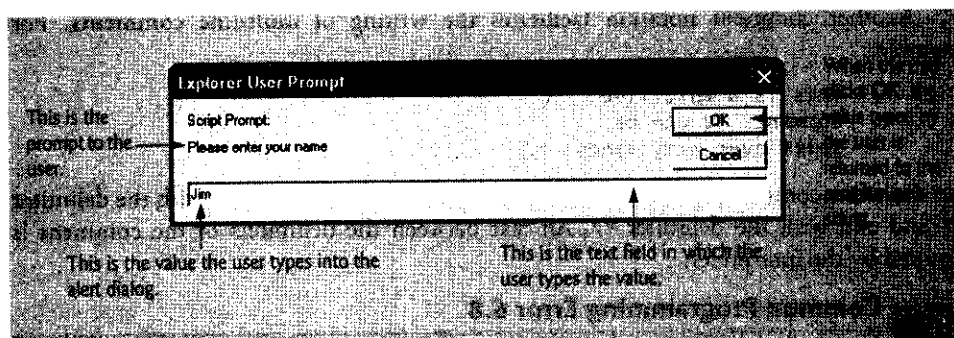


Fig. 6.8 | Prompt dialog displayed by the window object's `prompt` method.



Good Programming Practice 6.5

Place spaces on either side of a binary operator. This format makes the operator stand out and makes the program more readable.

Lines 17–18 use `document.writeln` to display the new welcome message. The expression inside the parentheses uses the operator `+` to “add” a string (the literal “`<h1>Hello,` ”), the variable name (the string that the user entered in line 15) and another string (the literal “`, welcome to JavaScript programming!</h1>`”). JavaScript has a version of the `+` operator for **string concatenation** that enables a string and a value of another data type (including another string) to be combined. The result of this operation is a new (and normally longer) string. If we assume that `name` contains the string literal “Jim”, the expression evaluates as follows: JavaScript determines that the two operands of the first `+` operator (the string “`<h1>Hello,` ” and the value of variable `name`) are both strings, then concatenates the two into one string. Next, JavaScript determines that the two operands of the second `+` operator (the result of the first concatenation operation, the string “`<h1>Hello, Jim`”, and the string “`, welcome to JavaScript programming!</h1>`”) are both strings and concatenates the two. This results in the string “`<h1>Hello, Jim, welcome to JavaScript programming!</h1>`”. The browser renders this string as part of the XHTML document. Note that the space between `Hello,` and `Jim` is part of the string “`<h1>Hello,` ”.

As we’ll illustrate later, the `+` operator used for string concatenation can convert other variable types to strings if necessary. Because string concatenation occurs between two strings, JavaScript must convert other variable types to strings before it can proceed with the operation. For example, if a variable `age` has an integer value equal to 21, then the expression “`my age is` ” + `age` evaluates to the string “`my age is 21`”. JavaScript converts the value of `age` to a string and concatenates it with the existing string literal “`my age is` ”.

After the browser interprets the `<head>` section of the XHTML document (which contains the JavaScript), it then interprets the `<body>` of the XHTML document (lines 22–24) and renders the XHTML. Notice that the XHTML page is not rendered until the prompt is dismissed because the prompt pauses execution in the head, before the body is processed. If you click your browser’s **Refresh** (Internet Explorer) or **Reload** (Firefox) button after entering a name, the browser will reload the XHTML document, so that you can execute the script again and change the name. [*Note:* In some cases, it may be necessary to hold down the *Shift* key while clicking the **Refresh** or **Reload** button, to ensure that the XHTML document reloads properly. Browsers often save a recent copy of a page in memory, and holding the *Shift* key forces the browser to download the most recent version of a page.]

6.4.2 Adding Integers

Our next script illustrates another use of prompt dialogs to obtain input from the user. Figure 6.9 inputs two **integers** (whole numbers, such as 7, –11, 0 and 31914) typed by a user at the keyboard, computes the sum of the values and displays the result.

Lines 12–16 declare the variables `firstNumber`, `secondNumber`, `number1`, `number2` and `sum`. Single-line comments state the purpose of each of these variables. Line 19 employs a prompt dialog to allow the user to enter a string representing the first of the two

```

1 <?xml version = "1.0" encoding = "utf-8"?>
2 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN
3   http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
4
5 <html xmlns = "http://www.w3.org/1999/xhtml">
6   <head>
7     <title>An Addition Program</title>
8     <script type = "text/javascript">
9       var firstNumber; // first string entered by user
10      var secondNumber; // second string entered by user
11      var number1; // first number to add
12      var number2; // second number to add
13      var sum; // sum of number1 and number2
14
15      // read in first number from user as a string
16      firstNumber = window.prompt( "Enter first integer" );
17
18      // read in second number from user as a string
19      secondNumber = window.prompt( "Enter second integer" );
20
21      // convert numbers from strings to integers
22      number1 = parseInt( firstNumber );
23      number2 = parseInt( secondNumber );
24
25      sum = number1 + number2; // add the numbers
26
27      // display the results
28      document.write( "The sum is " + sum + "<br>" );
29    </script>
30  </head>
31  <body>
32    <p>Click Refresh (or Reload) to run the script again.</p>
33  </body>
34 </html>

```

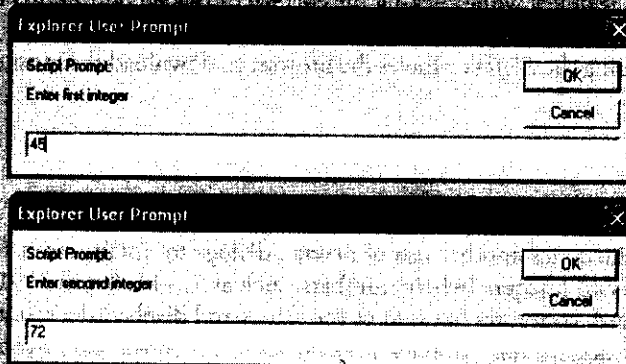


Fig. 6.9 | Addition script. (Part 1 of 2.)

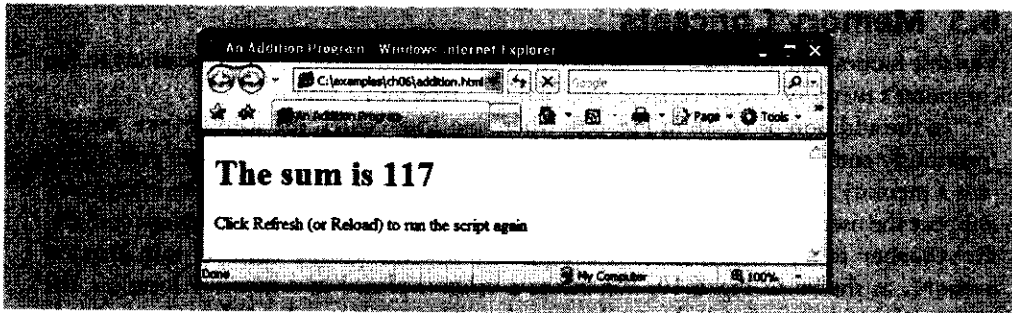


Fig. 6.9 | Addition script. (Part 2 of 2.)

integers that will be added. The script assigns the first value entered by the user to the variable `firstNumber`. Line 22 displays a prompt dialog to obtain the second number to add and assign this value to the variable `secondNumber`.

As in the preceding example, the user can type anything in the prompt dialog. For this program, if the user either types a noninteger value or clicks the **Cancel** button, a logic error will occur, and the sum of the two values will appear in the XHTML document as **NaN** (meaning **not a number**). A logic error is caused by syntactically correct code that produces an undesired result. In Chapter 11, *JavaScript: Objects*, we discuss the `Number` object and its methods that can determine whether a value is not a number.

Recall that a prompt dialog returns to the program as a string the value typed by the user. Lines 25–26 convert the two strings input by the user to integer values that can be used in a calculation. Function `parseInt` converts its string argument to an integer. Line 25 assigns to the variable `number1` the integer that function `parseInt` returns. Line 26 assigns an integer value to variable `number2` in a similar manner. Any subsequent references to `number1` and `number2` in the program use these integer values. [Note: We refer to `parseInt` as a **function** rather than a **method** because we do not precede the function call with an object name (such as `document` or `window`) and a dot (`.`). The term **method** means that the function belongs to a particular object. For example, method `writeln` belongs to the `document` object and method `prompt` belongs to the `window` object.]

Line 28 calculates the sum of the variables `number1` and `number2` using the **addition operator**, `+`, and assigns the result to variable `sum` by using the **assignment operator**, `=`. Notice that the `+` operator can perform both addition and string concatenation. In this case, the `+` operator performs addition, because both operands contain integers. After line 28 performs this calculation, line 31 uses `document.writeln` to display the result of the addition on the web page. Lines 33 and 34 close the script and head elements, respectively. Lines 35–37 render the body of XHTML document. Use your browser's **Refresh** or **Reload** button to reload the XHTML document and run the script again.



Common Programming Error 6.10

Confusing the `+` operator used for string concatenation with the `+` operator used for addition often leads to undesired results. For example, if integer variable `y` has the value 5, the expression `"y + 2 = " + y + 2` results in `"y + 2 = 52"`, not `"y + 2 = 7"`, because first the value of `y` (i.e., 5) is concatenated with the string `"y + 2 = "`, then the value 2 is concatenated with the new, larger string `"y + 2 = 5"`. The expression `"y + 2 = " + (y + 2)` produces the string `"y + 2 = 7"` because the parentheses ensure that `y + 2` is executed mathematically before it is converted to a string.

6.5 Memory Concepts

Variable names such as `number1`, `number2` and `sum` actually correspond to **locations** in the computer's memory. Every variable has a **name**, a **type** and a **value**.

In the addition program in Fig. 6.9, when line 25 executes, the string `first Number` (previously entered by the user in a prompt dialog) is converted to an integer and placed into a memory location to which the name `number1` has been assigned by the interpreter. Suppose the user entered the string 45 as the value for `firstNumber`. The program converts `firstNumber` to an integer, and the computer places the integer value 45 into location `number1`, as shown in Fig. 6.10. Whenever a value is placed in a memory location, the value replaces the previous value in that location. The previous value is lost.

Suppose that the user enters 72 as the second integer. When line 26 executes, the program converts `secondNumber` to an integer and places that integer value, 72, into location `number2`; then the memory appears as shown in Fig. 6.11.

Once the program has obtained values for `number1` and `number2`, it adds the values and places the sum into variable `sum`. The statement

```
sum = number1 + number2;
```

performs the addition and also replaces `sum`'s previous value. After `sum` is calculated, the memory appears as shown in Fig. 6.12. Note that the values of `number1` and `number2` appear exactly as they did before they were used in the calculation of `sum`. These values were used, but not destroyed, when the computer performed the calculation—when a value is read from a memory location, the process is nondestructive.



Fig. 6.10 | Memory location showing the name and value of variable `number1`.

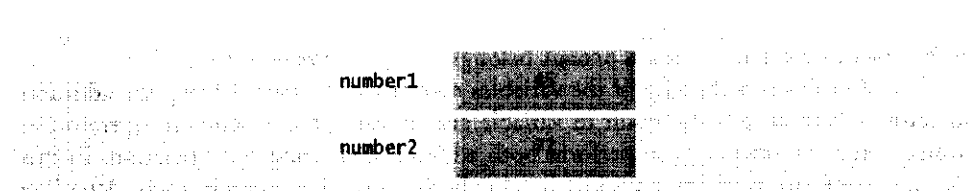


Fig. 6.11 | Memory locations after inputting values for variables `number1` and `number2`.

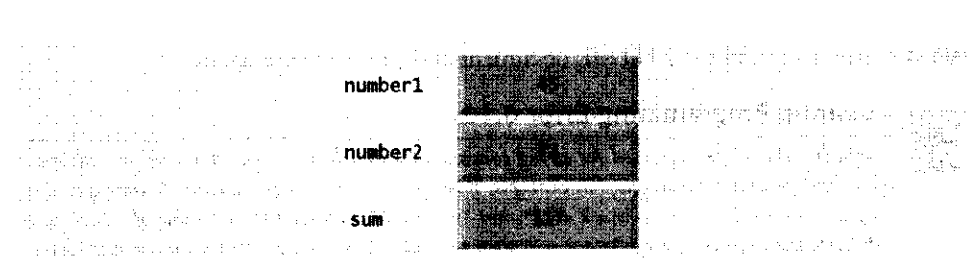


Fig. 6.12 | Memory locations after calculating the sum of `number1` and `number2`.

Data Types in JavaScript

Unlike its predecessor languages C, C++ and Java, JavaScript does not require variables to have a declared type before they can be used in a program. A variable in JavaScript can contain a value of any data type, and in many situations JavaScript automatically converts between values of different types for you. For this reason, JavaScript is referred to as a **loosely typed language**. When a variable is declared in JavaScript, but is not given a value, the variable has an **undefined** value. Attempting to use the value of such a variable is normally a logic error.

When variables are declared, they are not assigned values unless specified by the programmer. Assigning the value `null` to a variable indicates that it does not contain a value.

6.6 Arithmetic

Many scripts perform arithmetic calculations. Figure 6.13 summarizes the **arithmetic operators**. Note the use of various special symbols not used in algebra. The asterisk (*) indicates multiplication; the percent sign (%) is the **remainder operator**, which will be discussed shortly. The arithmetic operators in Fig. 6.13 are binary operators, because each operates on two operands. For example, the expression `sum + value` contains the binary operator `+` and the two operands `sum` and `value`.

JavaScript provides the remainder operator, `%`, which yields the remainder after division. [Note: The `%` operator is known as the modulus operator in some programming languages.] The expression `x % y` yields the remainder after `x` is divided by `y`. Thus, `17 % 5` yields 2 (i.e., 17 divided by 5 is 3, with a remainder of 2), and `7.4 % 3.1` yields 1.2. In later chapters, we consider applications of the remainder operator, such as determining whether one number is a multiple of another. There is no arithmetic operator for exponentiation in JavaScript. (Chapter 8, JavaScript: Control Statements II, shows how to perform exponentiation in JavaScript using the `Math` object's `pow` method.)

Arithmetic expressions in JavaScript must be written in **straight-line form** to facilitate entering programs into the computer. Thus, expressions such as “a divided by b” must be written as `a / b`, so that all constants, variables and operators appear in a straight line. The following algebraic notation is generally not acceptable to computers:

$$\frac{a}{b}$$

| | | | |
|----------------|---|--|--------------------|
| Addition | + | $f + 7$ | <code>f + 7</code> |
| Subtraction | - | $p - c$ | <code>p - c</code> |
| Multiplication | * | bm | <code>b * m</code> |
| Division | / | x / y or $\frac{x}{y}$ or $x \div y$ | <code>x / y</code> |
| Remainder | % | $r \text{ mod } s$ | <code>r % s</code> |

Fig. 6.13 | Arithmetic operators.

Parentheses are used to group expressions in the same manner as in algebraic expressions. For example, to multiply a times the quantity $b + c$ we write:

$$a * (b + c)$$

JavaScript applies the operators in arithmetic expressions in a precise sequence determined by the following **rules of operator precedence**, which are generally the same as those followed in algebra:

1. Multiplication, division and remainder operations are applied first. If an expression contains several multiplication, division and remainder operations, operators are applied from left to right. Multiplication, division and remainder operations are said to have the same level of precedence.
2. Addition and subtraction operations are applied next. If an expression contains several addition and subtraction operations, operators are applied from left to right. Addition and subtraction operations have the same level of precedence.

The rules of operator precedence enable JavaScript to apply operators in the correct order. When we say that operators are applied from left to right, we are referring to the **associativity** of the operators—the order in which operators of equal priority are evaluated. We will see that some operators associate from right to left. Figure 6.14 summarizes the rules of operator precedence. The table in Fig. 6.14 will be expanded as additional JavaScript operators are introduced. A complete precedence chart is included in Appendix C.

Now, in light of the rules of operator precedence, let us consider several algebraic expressions. Each example lists an algebraic expression and the equivalent JavaScript expression.

The following is an example of an arithmetic mean (average) of five terms:

$$\text{Algebra: } m = \frac{a + b + c + d + e}{5}$$

$$\text{JavaScript: } m = (a + b + c + d + e) / 5;$$

The parentheses are required to group the addition operators, because division has higher precedence than addition. The entire quantity $(a + b + c + d + e)$ is to be divided by 5. If the parentheses are erroneously omitted, we obtain $a + b + c + d + e / 5$, which evaluates as

$$a + b + c + d + \frac{e}{5}$$

and would not lead to the correct answer.

| | | |
|-----------|---|---|
| *, / or % | Multiplication Division Remainder | Evaluated first. If there are several such operations, they are evaluated from left to right. |
| + or - | Addition Subtraction | Evaluated last. If there are several such operations, they are evaluated from left to right. |

Fig. 6.14 | Precedence of arithmetic operators.

The following is an example of the equation of a straight line:

Algebra: $y = mx + b$

JavaScript: `y = m * x + b;`

No parentheses are required. The multiplication operator is applied first, because multiplication has a higher precedence than addition. The assignment occurs last, because it has a lower precedence than multiplication and addition.

The following example contains remainder (%), multiplication, division, addition and subtraction operations:

Algebra: $z = pr \% q + w/x - y$

Java: `z = p * r % q + w / x - y;`



The circled numbers under the statement indicate the order in which JavaScript applies the operators. The multiplication, remainder and division operations are evaluated first in left-to-right order (i.e., they associate from left to right), because they have higher precedence than addition and subtraction. The addition and subtraction operations are evaluated next. These operations are also applied from left to right.

To develop a better understanding of the rules of operator precedence, consider the evaluation of a second-degree polynomial ($y = ax^2 + bx + c$):

$y = a * x * x + b * x + c;$



The circled numbers indicate the order in which JavaScript applies the operators.

Suppose that a , b , c and x are initialized as follows: $a = 2$, $b = 3$, $c = 7$ and $x = 5$. Figure 6.15 illustrates the order in which the operators are applied in the preceding second-degree polynomial.

As in algebra, it is acceptable to use unnecessary parentheses in an expression to make the expression clearer. These are also called **redundant parentheses**. For example, the preceding second-degree polynomial might be parenthesized as follows:

$y = (a * x * x) + (b * x) + c;$



Good Programming Practice 6.6

Using parentheses for complex arithmetic expressions, even when the parentheses are not necessary, can make the arithmetic expressions easier to read.

6.7 Decision Making: Equality and Relational Operators

This section introduces a version of JavaScript's `if` statement that allows a program to make a decision based on the truth or falsity of a **condition**. If the condition is met (i.e., the condition is **true**), the statement in the body of the `if` statement is executed. If the condition is not met (i.e., the condition is **false**), the statement in the body of the `if` statement

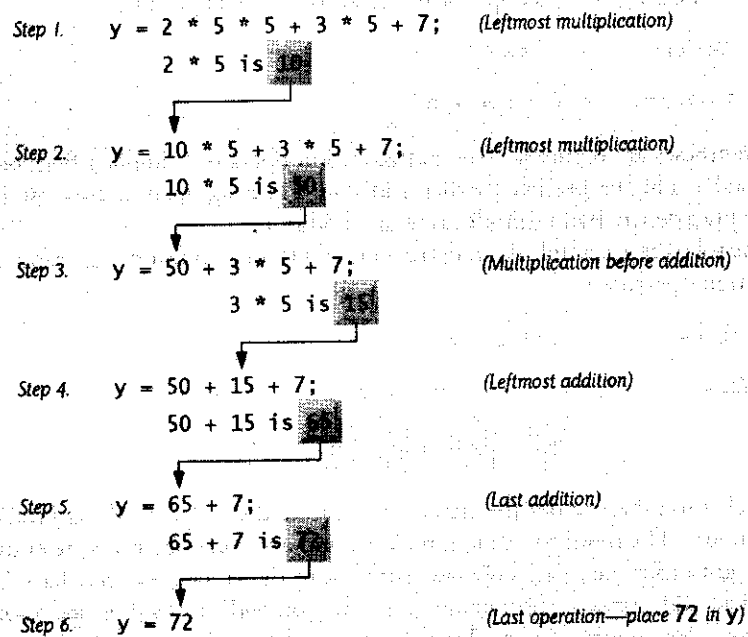


Fig. 6.15 | Order in which a second-degree polynomial is evaluated.

is not executed. We will see an example shortly. [Note: Other versions of the if statement are introduced in Chapter 7, JavaScript: Control Statements I.]

Conditions in if statements can be formed by using the equality operators and relational operators summarized in Fig. 6.16. The relational operators all have the same level

| Equality operators | | | |
|----------------------|----|--------|---------------------------------|
| = | == | x == y | x is equal to y |
| ≠ | != | x != y | x is not equal to y |
| Relational operators | | | |
| > | > | x > y | x is greater than y |
| < | < | x < y | x is less than y |
| ≥ | >= | x >= y | x is greater than or equal to y |
| ≤ | <= | x <= y | x is less than or equal to y |

Fig. 6.16 | Equality and relational operators.

of precedence and associate from left to right. The equality operators both have the same level of precedence, which is lower than the precedence of the relational operators. The equality operators also associate from left to right.



Common Programming Error 6.11

It is a syntax error if the operators `==`, `!=`, `>=` and `<=` contain spaces between their symbols, as in `= =`, `! =`, `> =` and `< =`, respectively.



Common Programming Error 6.12

Reversing the operators `!=`, `>=` and `<=`, as in `=!`, `=>` and `=<`, respectively, is a syntax error.



Common Programming Error 6.13

Confusing the equality operator, `==`, with the assignment operator, `=`, is a logic error. The equality operator should be read as “is equal to,” and the assignment operator should be read as “gets” or “gets the value of.” Some people prefer to read the equality operator as “double equals” or “equals equals.”

The script in Fig. 6.17 uses four `if` statements to display a time-sensitive greeting on a welcome page. The script obtains the local time from the user’s computer and converts it from 24-hour clock format (0–23) to a 12-hour clock format (0–11). Using this value, the script displays an appropriate greeting for the current time of day. The script and sample output are shown in Fig. 6.17.

Lines 12–14 declare the variables used in the script. Remember that variables may be declared in one declaration or in multiple declarations. If more than one variable is declared in a single declaration (as in this example), the names are separated by commas (,). This list of names is referred to as a comma-separated list. Once again, note the comment at the end of each line, indicating the purpose of each variable in the program. Also note that some of the variables are assigned a value in the declaration—JavaScript allows you to assign a value to a variable when the variable is declared.

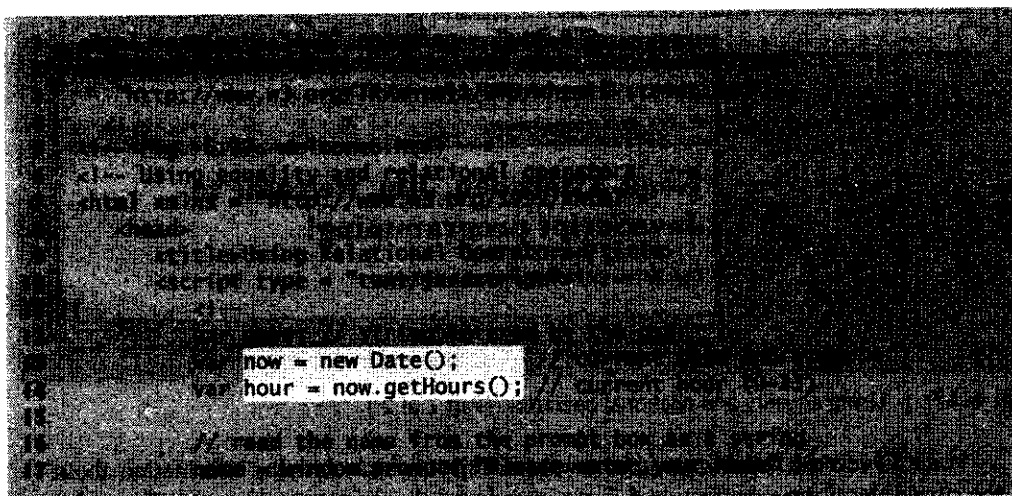


Fig. 6.17 | Using equality and relational operators. (Part 1 of 2.)

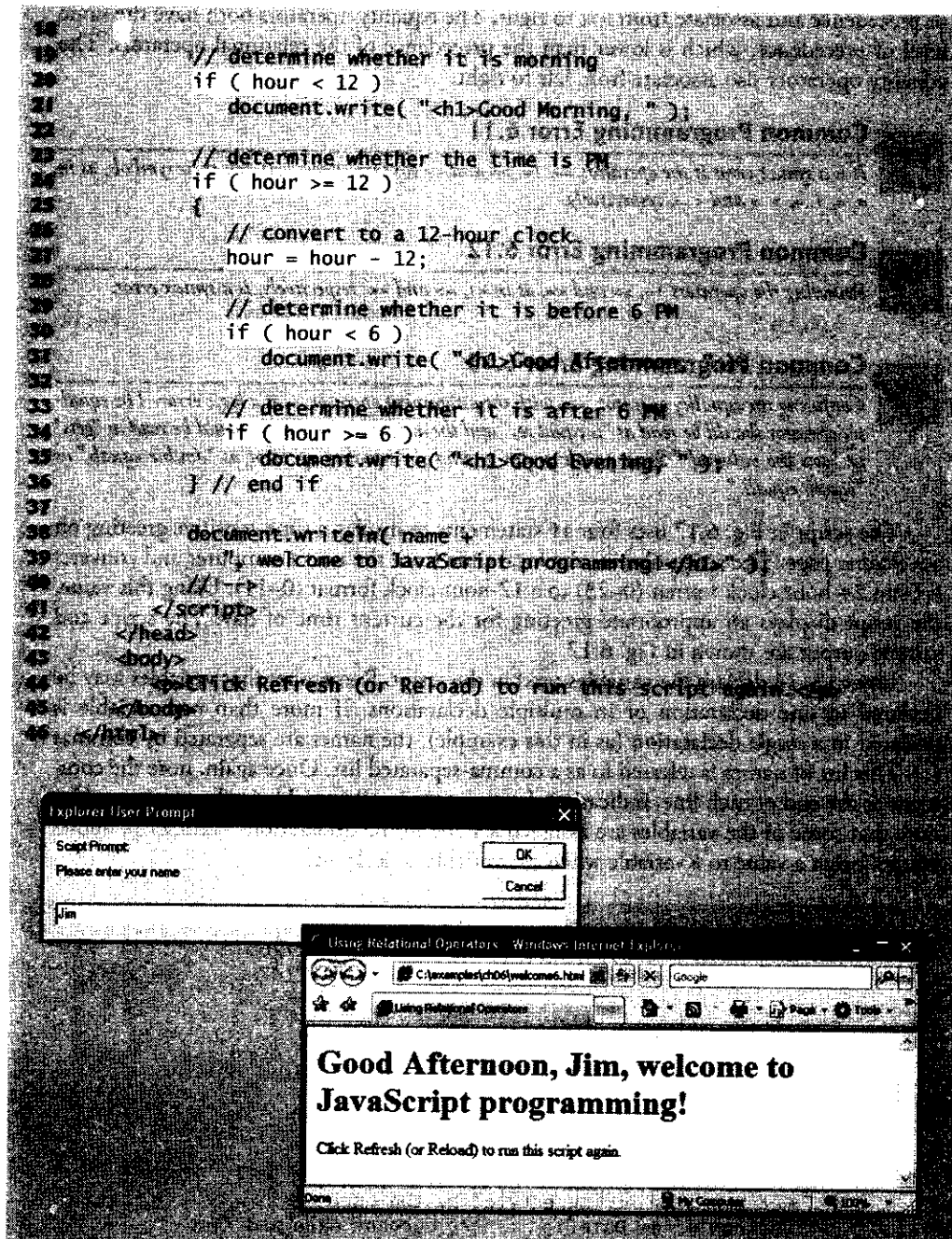


Fig. 6.17 | Using equality and relational operators. (Part 2 of 2.)

Line 13 sets the variable `now` to a new **Date** object, which contains information about the current local time. In Section 6.2, we introduced the **document** object, an object that encapsulates data pertaining to the current web page. Programmers may choose to use

other objects to perform specific tasks or obtain particular pieces of information. Here, we use JavaScript's built-in `Date` object to acquire the current local time. We create a new instance of an object by using the `new` operator followed by the type of the object, `Date`, and a pair of parentheses. Some objects require that arguments be placed in the parentheses to specify details about the object to be created. In this case, we leave the parentheses empty to create a default `Date` object containing information about the current date and time. After line 13 executes, the variable `now` refers to the new `Date` object. [*Note:* We did not need to use the `new` operator when we used the `document` and `window` objects because these objects always are created by the browser.] Line 14 sets the variable `hour` to an integer equal to the current hour (in a 24-hour clock format) returned by the `Date` object's `getHours` method. Chapter 11 presents a more detailed discussion of the `Date` object's attributes and methods, and of objects in general. As in the preceding example, the script uses `window.prompt` to allow the user to enter a name to display as part of the greeting (line 17).

To display the correct time-sensitive greeting, the script must determine whether the user is visiting the page during the morning, afternoon or evening. The first `if` statement (lines 20–21) compares the value of variable `hour` with 12. If `hour` is less than 12, then the user is visiting the page during the morning, and the statement at line 21 outputs the string "Good morning". If this condition is not met, line 21 is not executed. Line 24 determines whether `hour` is greater than or equal to 12. If `hour` is greater than or equal to 12, then the user is visiting the page in either the afternoon or the evening. Lines 25–36 execute to determine the appropriate greeting. If `hour` is less than 12, then the JavaScript interpreter does not execute these lines and continues to line 38.

The brace `{` in line 25 begins a block of statements (lines 27–35) that are all executed together if `hour` is greater than or equal to 12—to execute multiple statements inside an `if` construct, enclose them in curly braces. Line 27 subtracts 12 from `hour`, converting the current hour from a 24-hour clock format (0–23) to a 12-hour clock format (0–11). The `if` statement (line 30) determines whether `hour` is now less than 6. If it is, then the time is between noon and 6 PM, and line 31 outputs the beginning of an XHTML `h1` element ("`<h1>Good Afternoon,` "). If `hour` is greater than or equal to 6, the time is between 6 PM and midnight, and the script outputs the greeting "Good Evening" (lines 34–35). The brace `}` in line 36 ends the block of statements associated with the `if` statement in line 24. Note that `if` statements can be *nested*, i.e., one `if` statement can be placed inside another `if` statement. The `if` statements that determine whether the user is visiting the page in the afternoon or the evening (lines 30–31 and lines 34–35) execute only if the script has already established that `hour` is greater than or equal to 12 (line 24). If the script has already determined the current time of day to be morning, these additional comparisons are not performed. (Chapter 7, JavaScript: Control Statements I, presents a more in-depth discussion of blocks and nested `if` statements.) Finally, lines 38–39 output the rest of the XHTML `h1` element (the remaining part of the greeting), which does not depend on the time of day.



Good Programming Practice 6.7

Include comments after the closing curly brace of control statements (such as `if` statements) to indicate where the statements end, as in line 36 of Fig. 6.17.

Note the indentation of the `if` statements throughout the program. Such indentation enhances program readability.

**Good Programming Practice 6.8**

Indent the statement in the body of an if statement to make the body of the statement stand out and to enhance program readability.

**Good Programming Practice 6.9**

Place only one statement per line in a program. This enhances program readability.

**Common Programming Error 6.14**

Forgetting the left and/or right parentheses for the condition in an if statement is a syntax error. The parentheses are required.

Note that there is no semicolon (;) at the end of the first line of each if statement. Including such a semicolon would result in a logic error at execution time. For example,

```
if ( hour < 12 ) ;
    document.write( "<h1>Good Morning, " );
```

would actually be interpreted by JavaScript erroneously as

```
if ( hour < 12 )
;
document.write( "<h1>Good Morning, " );
```

where the semicolon on the line by itself—called the **empty statement**—is the statement to execute if the condition in the if statement is true. When the empty statement executes, no task is performed in the program. The program then continues with the next statement, which executes regardless of whether the condition is true or false. In this example, "<h1>Good Morning, " would be printed regardless of the time of day.

**Common Programming Error 6.15**

Placing a semicolon immediately after the right parenthesis of the condition in an if statement is normally a logic error. The semicolon would cause the body of the if statement to be empty, so the if statement itself would perform no action, regardless of whether its condition was true. Worse yet, the intended body statement of the if statement would now become a statement in sequence after the if statement and would always be executed.

**Common Programming Error 6.16**

Leaving out a condition in a series of if statements is normally a logic error. For instance, checking if hour is greater than 12 or less than 12, but not if hour is equal to 12, would mean that the script takes no action when hour is equal to 12. Always be sure to handle every possible condition.

Note the use of spacing in lines 38–39 of Fig. 6.17. Remember that white-space characters, such as tabs, newlines and spaces, are normally ignored by the browser. So, statements may be split over several lines and may be spaced according to the programmer's preferences without affecting the meaning of a program. However, it is incorrect to split identifiers and string literals. Ideally, statements should be kept small, but it is not always possible to do so.

**Good Programming Practice 6.10**

A lengthy statement may be spread over several lines. If a single statement must be split across lines, choose breaking points that make sense, such as after a comma in a comma-separated list or after an operator in a lengthy expression. If a statement is split across two or more lines, indent all subsequent lines.

The chart in Fig. 6.18 shows the precedence of the operators introduced in this chapter. The operators are shown from top to bottom in decreasing order of precedence. Note that all of these operators, with the exception of the assignment operator, =, associate from left to right. Addition is left associative, so an expression like $x + y + z$ is evaluated as if it had been written as $(x + y) + z$. The assignment operator, =, associates from right to left, so an expression like $x = y = 0$ is evaluated as if it had been written as $x = (y = 0)$, which first assigns the value 0 to variable y , then assigns the result of that assignment, 0, to x .

**Good Programming Practice 6.11**

Refer to the operator precedence chart when writing expressions containing many operators. Confirm that the operations are performed in the order in which you expect them to be performed. If you are uncertain about the order of evaluation in a complex expression, use parentheses to force the order, exactly as you would do in algebraic expressions. Be sure to observe that some operators, such as assignment (=), associate from right to left rather than from left to right.

| | | |
|-----------|---------------|----------------|
| * / % | left to right | multiplicative |
| + - | left to right | additive |
| < <= > >= | left to right | relational |
| == != | left to right | equality |
| = | right to left | assignment |

Fig. 6.18 | Precedence and associativity of the operators discussed so far.

6.8 Web Resources

www.deitel.com/javascript

The Deitel JavaScript Resource Center contains links to some of the best JavaScript resources on the web. There you'll find categorized links to JavaScript tools, code generators, forums, books, libraries, frameworks and more. Also check out the tutorials for all skill levels, from introductory to advanced.

Summary

Section 6.1 Introduction

- The JavaScript language facilitates a disciplined approach to the design of computer programs that enhance web pages.

Section 6.2 Simple Program: Displaying a Line of Text in a Web Page

- The spacing displayed by a browser in a web page is determined by the XHTML elements used to format the page.

- Often, JavaScripts appear in the <head> section of the XHTML document.
- The browser interprets the contents of the <head> section first.
- The <script> tag indicates to the browser that the text that follows is part of a script. Attribute type specifies the scripting language used in the script—such as text/javascript.
- A string of characters can be contained between double (") or single (') quotation marks.
- A string is sometimes called a character string, a message or a string literal.
- The browser's document object represents the XHTML document currently being displayed in the browser. The document object allows a script programmer to specify XHTML text to be displayed in the XHTML document.
- The browser contains a complete set of objects that allow script programmers to access and manipulate every element of an XHTML document.
- An object resides in the computer's memory and contains information used by the script. The term object normally implies that attributes (data) and behaviors (methods) are associated with the object. The object's methods use the attributes' data to perform useful actions for the client of the object—the script that calls the methods.
- The document object's writeIn method writes a line of XHTML text in the XHTML document.
- The parentheses following the name of a method contain the arguments that the method requires to perform its task (or its action).
- Using writeIn to write a line of XHTML text into a document does not guarantee that a corresponding line of text will appear in the XHTML document. The text displayed is dependent on the contents of the string written, which is subsequently rendered by the browser. The browser will interpret the XHTML elements as it normally does to render the final text in the document.
- Every statement should end with a semicolon (also known as the statement terminator), although none is required by JavaScript.
- JavaScript is case sensitive. Not using the proper uppercase and lowercase letters is a syntax error.

Section 6.3 Modifying Our First Program

- Sometimes it is useful to display information in windows called dialogs that "pop up" on the screen to grab the user's attention. Dialogs are typically used to display important messages to the user browsing the web page. The browser's window object uses method alert to display an alert dialog. Method alert requires as its argument the string to be displayed.
- When a backslash is encountered in a string of characters, the next character is combined with the backslash to form an escape sequence. The escape sequence \n is the newline character. It causes the cursor in the XHTML document to move to the beginning of the next line.

Section 6.4 Obtaining User Input with prompt Dialogs

- Keywords are words with special meaning in JavaScript.
- The keyword var is used to declare the names of variables. A variable is a location in the computer's memory where a value can be stored for use by a program. All variables have a name, type and value, and should be declared with a var statement before they are used in a program.
- A variable name can be any valid identifier consisting of letters, digits, underscores (_) and dollar signs (\$) that does not begin with a digit and is not a reserved JavaScript keyword.
- Declarations end with a semicolon (;) and can be split over several lines, with each variable in the declaration separated by a comma (forming a comma-separated list of variable names). Several variables may be declared in one declaration or in multiple declarations.

- Programmers often indicate the purpose of a variable in the program by placing a JavaScript comment at the end of the variable's declaration. A single-line comment begins with the characters `//` and terminates at the end of the line. Comments do not cause the browser to perform any action when the script is interpreted; rather, comments are ignored by the JavaScript interpreter.
- Multiline comments begin with delimiter `/*` and end with delimiter `*/`. All text between the delimiters of the comment is ignored by the interpreter.
- The `window` object's `prompt` method displays a dialog into which the user can type a value. The first argument is a message (called a prompt) that directs the user to take a specific action. The optional second argument is the default string to display in the text field.
- A variable is assigned a value with an assignment statement, using the assignment operator, `=`. The `=` operator is called a binary operator, because it has two operands.
- The `null` keyword signifies that a variable has no value. Note that `null` is not a string literal, but rather a predefined term indicating the absence of value. Writing a `null` value to the document, however, displays the word "null".
- Function `parseInt` converts its string argument to an integer.
- JavaScript has a version of the `+` operator for string concatenation that enables a string and a value of another data type (including another string) to be concatenated.

Section 6.5 Memory Concepts

- Variable names correspond to locations in the computer's memory. Every variable has a name, a type and a value.
- When a value is placed in a memory location, the value replaces the previous value in that location. When a value is read out of a memory location, the process is nondestructive.
- JavaScript does not require variables to have a type before they can be used in a program. A variable in JavaScript can contain a value of any data type, and in many situations, JavaScript automatically converts between values of different types for you. For this reason, JavaScript is referred to as a loosely typed language.
- When a variable is declared in JavaScript, but is not given a value, it has an undefined value. Attempting to use the value of such a variable is normally a logic error.
- When variables are declared, they are not assigned default values, unless specified otherwise by the programmer. To indicate that a variable does not contain a value, you can assign the value `null` to it.

Section 6.6 Arithmetic

- The basic arithmetic operators (`+`, `-`, `*`, `/`, and `%`) are binary operators, because they each operate on two operands.
- Parentheses can be used to group expressions as in algebra.
- Operators in arithmetic expressions are applied in a precise sequence determined by the rules of operator precedence.
- When we say that operators are applied from left to right, we are referring to the associativity of the operators. Some operators associate from right to left.

Section 6.7 Decision Making: Equality and Relational Operators

- JavaScript's `if` statement allows a program to make a decision based on the truth or falsity of a condition. If the condition is met (i.e., the condition is true), the statement in the body of the `if` statement is executed. If the condition is not met (i.e., the condition is false), the statement in the body of the `if` statement is not executed.
- Conditions in `if` statements can be formed by using the equality operators and relational operators.

Terminology

\ " double-quote escape sequence
 \ \ backslash escape sequence
 \ ' single quote escape sequence
 \ n newline escape sequence
 \ r carriage return escape sequence
 \ t tab escape sequence
 action
 addition operator (+)
 alert dialog
 alert method of the window object
 argument to a method
 arithmetic expression in straight-line form
 arithmetic operator
 assignment
 assignment operator (=)
 assignment statement
 associativity of operators
 attribute
 backslash (\) escape character
 behavior
 binary operator
 case sensitive
 character string
 client of an object
 comma-separated list
 comment
 condition
 cursor
 data
 data type
 Date object
 decision making
 declaration
 dialog
 division operator (/)
 document object
 double quotation (") marks
 ECMAScript standard
 empty statement
 equality operators
 error message
 escape sequence
 false
 function
 identifier
 if statement
 inline scripting
 integer
 interpreter
 JavaScript
 JavaScript interpreter
 keyword
 level of precedence
 literal
 location in the computer's memory
 logic error
 loosely typed language
 meaningful variable name
 method
 mouse cursor
 multiline comment (/* and */)

multiplication operator (*)
 name of a variable
 NaN (not a number)
 nested if statements
 new operator
 newline character (\n)
 null
 object
 operand
 operator associativity
 operator precedence
 parentheses
 parseInt function
 perform an action
 pre element
 program
 prompt
 prompt dialog
 prompt method of the window object
 redundant parentheses
 relational operator
 remainder after division
 remainder operator (%)
 rules of operator precedence
 runtime error
 script
 script element
 scripting language
 semicolon (;) statement terminator
 single quotation (') mark
 single-line comment (//)
 statement
 straight-line form
 string
 string concatenation
 string concatenation operator (+)
 string literal

| | |
|------------------------------------|---------------------------------------|
| string of characters | value of a variable |
| subtraction operator (-) | var keyword |
| syntax error | variable |
| text field | violation of the language rules |
| title bar of a dialog | white-space character |
| true | whole number |
| type attribute of the <script> tag | window object |
| type of a variable | write method of the document object |
| undefined | writeLn method of the document object |

Self-Review Exercises

6.1 Fill in the blanks in each of the following statements:

- _____ begins a single-line comment.
- Every statement should end with a(n) _____.
- The _____ statement is used to make decisions.
- _____, _____, and _____ are known as white space.
- The _____ object displays alert dialogs and prompt dialogs.
- _____ are words that are reserved for use by JavaScript.
- Methods _____ and _____ of the _____ object write XHTML text into an XHTML document.

6.2 State whether each of the following is true or false. If false, explain why.

- Comments cause the computer to print the text after the // on the screen when the program is executed.
- JavaScript considers the variables number and NUMBER to be identical.
- The remainder operator (%) can be used only with numeric operands.
- The arithmetic operators *, /, %, + and - all have the same level of precedence.
- Method parseInt converts an integer to a string.

6.3 Write JavaScript statements to accomplish each of the following tasks:

- Declare variables c, thisIsAVariable, and q76354 as 1 number.
- Display a dialog asking the user to enter an integer. Show a default value of 0 in the text field.
- Convert a string to an integer, and store the converted value in variable age. Assume that the string is stored in stringValue.
- If the variable number is not equal to 7, display "The variable number is not equal to 7" in a message dialog.
- Output a line of XHTML text that will display the message "This is a JavaScript program" on one line in the XHTML document.
- Output a line of XHTML text that will display the message "This is a JavaScript program" on two lines in the XHTML document. Use only one statement.

6.4 Identify and correct the errors in each of the following statements:

- if (c < 7);
window.alert("c is less than 7");
- if (c >= 7);
window.alert("c is equal to or greater than 7");

6.5 Write a statement (or comment) to accomplish each of the following tasks:

- State that a program will calculate the product of three integers [Hint: Use text that helps to document a program.]
- Declare the variables x, y, z and result.

- c) Declare the variables `xVal`, `yVal` and `zVal`.
- d) Prompt the user to enter the first value, read the value from the user and store it in the variable `xVal`.
- e) Prompt the user to enter the second value, read the value from the user and store it in the variable `yVal`.
- f) Prompt the user to enter the third value, read the value from the user and store it in the variable `zVal`.
- g) Convert `xVal` to an integer, and store the result in the variable `x`.
- h) Convert `yVal` to an integer, and store the result in the variable `y`.
- i) Convert `zVal` to an integer, and store the result in the variable `z`.
- j) Compute the product of the three integers contained in variables `x`, `y` and `z`, and assign the result to the variable `result`.
- k) Write a line of XHTML text containing the string "The product is:" followed by the value of the variable `result`.

6.6 Using the statements you wrote in Exercise 6.5, write a complete program that calculates and prints the product of three integers.

Exercises

- 6.7** Write JavaScript statements that accomplish each of the following tasks:
- a) Display the message "Enter two numbers" using the `window` object.
 - b) Assign the product of variables `b` and `c` to variable `a`.
 - c) State that `a` program performs a sample payroll calculation.
- 6.8** State whether each of the following is *true* or *false*. If *false*, explain why.
- a) JavaScript operators are evaluated from left to right.
 - b) The following are all valid variable names: `under_bar_`, `m928134`, `t5`, `j7`, `her_sales`, `his_account_total`, `a`, `b5`, `c`, `z`, `z2`.
 - c) A valid JavaScript arithmetic expression with no parentheses is evaluated from left to right.
 - d) The following are all invalid variable names: `3g`, `87`, `67h2`, `h22`, `2h`.
- 6.9** Fill in the blanks in each of the following statements:
- a) What arithmetic operations have the same precedence as multiplication? _____
 - b) When parentheses are nested, which set of parentheses is evaluated first in an arithmetic expression? _____
 - c) A location in the computer's memory that may contain different values at various times throughout the execution of a program is called a _____.
- 6.10** What displays in the message dialog when each of the given JavaScript statements is performed? Assume that `x = 2` and `y = 3`.
- a) `window.alert("x = " + x);`
 - b) `window.alert("The value of x + x is " + (x + x));`
 - c) `window.alert("x =");`
 - d) `window.alert((x + y) + " = " + (y + x));`
- 6.11** Given $y = ax^3 + 7$, which of the following are correct JavaScript statements for this equation?
- a) `y = a * x * x * x + 7;`
 - b) `y = a * x * x * (x + 7);`
 - c) `y = (a * x) * x * (x + 7);`

- d) $y = (a * x) * x * x + 7;$
 e) $y = a * (x * x * x) + 7;$
 f) $y = a * x * (x * x + 7);$

6.12 Write a script that displays the numbers 1 to 4 on the same line, with each pair of adjacent numbers separated by one space. Write the program using the following methods:

- a) Using one document.write statement.
 b) Using four document.write statements.

6.13 Write a script that gets from the user the radius of a circle and outputs XHTML text that displays the circle's diameter, circumference and area. Use the constant value 3.14159 for π . Use the GUI techniques shown in Fig. 6.9. [Note: You may also use the predefined constant Math.PI for the value of π . This constant is more precise than the value 3.14159. The Math object is defined by JavaScript and provides many common mathematical capabilities.] Use the following formulas (r is the radius): $diameter = 2r$, $circumference = 2\pi r$, $area = \pi r^2$.

6.14 Write a script that reads five integers and determines and outputs XHTML text that displays the largest and smallest integers in the group. Use only the programming techniques you learned in this chapter.

6.15 Write a script that reads in two integers and determines and outputs XHTML text that displays whether the first is a multiple of the second. [Hint: Use the remainder operator.]

6.16 Write a script that outputs XHTML text that displays in the XHTML document a check-board pattern, as follows:

```

* * * * *
* * * * *
* * * * *
* * * * *
* * * * *
* * * * *
* * * * *
* * * * *

```

6.17 Write a script that calculates the squares and cubes of the numbers from 0 to 10 and outputs XHTML text that displays the resulting values in an XHTML table format, as follows:

| number | square | cube |
|--------|--------|------|
| 0 | 0 | 0 |
| 1 | 1 | 1 |
| 2 | 4 | 8 |
| 3 | 9 | 27 |
| 4 | 16 | 64 |
| 5 | 25 | 125 |
| 6 | 36 | 216 |
| 7 | 49 | 343 |
| 8 | 64 | 512 |
| 9 | 81 | 729 |
| 10 | 100 | 1000 |

[Note: This program does not require any input from the user.]

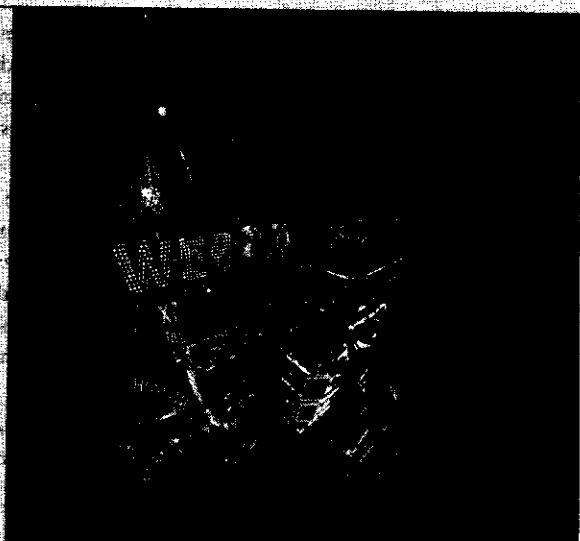


JavaScript: Control Statements I

OBJECTIVES

In this chapter you will learn:

- Basic problem-solving techniques.
- To develop algorithms through the process of top-down, stepwise refinement.
- To use the `if` and `if...else` selection statements to choose among alternative actions.
- To use the `while` repetition statement to execute statements in a script repeatedly.
- Counter-controlled repetition and sentinel-controlled repetition.
- To use the increment, decrement and assignment operators.



Let's all move one place on.

—Lewis Carroll

The wheel is come full circle.

—William Shakespeare

*How many apples fell on
Newton's head before he took
the hint!*

—Robert Frost