Chapter 6
JavaScript: Introduction to Scripting

6.1 Introduction

- JavaScript
  - Scripting language which is used to enhance the functionality and appearance of web pages.
- Before you can run code examples with JavaScript on your computer, you may need to change your browser's security settings.
  - IE9 *prevents* scripts on the local computer from running by default
  - Firefox, Chrome, Opera, Safari (including on the iPhone) and the Android browser have JavaScript enabled by default.

6.2 Your First Script: Displaying a Line of Text with JavaScript in a Web Page

- We begin with a simple script that displays the text `"Welcome to JavaScript Programming!"` in the HTML5 document.
- All major web browsers contain JavaScript **interpreters**, which process the commands written in JavaScript.
- The JavaScript code and its result are shown in Fig. 6.1.

```
<!DOCTYPE html>

<!-- Fig. 6.1: welcome.html -->
<!-- Displaying a line of text. -->
<html>
   <head>
      <meta charset = "utf-8">
      <title>A First Program in JavaScript</title>
      <script type = "text/javascript">

         document.writeln(
            "<h1>Welcome to JavaScript Programming!</h1>" );

      </script>
   </head><body></body>
</html>
```

Fig. 6.1 | Displaying a line of text. (Part 1 of 2.)

Fig. 6.1 | Displaying a line of text. (Part 2 of 2.)

6.2 Your First Script: Displaying a Line of Text with JavaScript in a Web Page (cont.)

- Spacing displayed by a browser in a web page is determined by the HTML5 elements used to format the page
- Often, JavaScripts appear in the <head> section of the HTML5 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`

6.2 Your First Script: Displaying a Line of Text with JavaScript in a Web Page (cont.)

The `script` Element and Commenting Your Scripts

- The `<script>` tag indicates to the browser that the text which follows is part of a script.
- The type attribute specifies the MIME type of the script as well as the **scripting language** used in the script—in this case, a `text` file written in `javascript`.
- In HTML5, the default MIME type for a `<script>` is `"text/html"`, so you can omit the `type` attribute from your `<script>` tags.
- You'll see it in legacy HTML documents with embedded JavaScripts.

6.2 Your First Script: Displaying a Line of Text with JavaScript in a Web Page (cont.)

- A string of characters can be contained between double quotation (") marks (also called a **string literal**)

Software Engineering Observation 6.1

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

6.2 Your First Script: Displaying a Line of Text with JavaScript in a Web Page (cont.)

- Browser's document object represents the HTML5 document currently being displayed in the browser
  - Allows a you to specify HTML5 text to be displayed in the HTML5 document
- Browser contains a complete set of objects that allow script programmers to access and manipulate every element of an HTML5 document
- 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
  - An object's methods use the attributes' data to perform useful actions for the client of the object—the script that calls the methods

6.2 Your First Script: Displaying a Line of Text with JavaScript in a Web Page (cont.)

- The parentheses following the name of a method contain the arguments that the method requires to perform its task (or its action)
- 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

Good Programming Practice 6.1

Terminate every statement with a semicolon. 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 HTML5 document from loading properly.

6.2 Your First Script: Displaying a Line of Text with JavaScript in a Web Page (cont.)

- The `document` object's `writeln` method
  - Writes a line of HTML5 text in the HTML5 document
  - Does not guarantee that a corresponding line of text will appear in the HTML5 document.
  - Text displayed is dependent on the contents of the string written, which is subsequently rendered by the browser.
  - Browser will interpret the HTML5 elements as it normally does to render the final text in the document

6.2 Your First Script: Displaying a Line of Text with JavaScript in a Web Page (cont.)

A Note About Embedding JavaScript Code into HTML5 Documents

- JavaScript code is typically placed in a separate file, then included in the HTML5 document that uses the script.
- This makes the code more reusable, because it can be included into any HTML5 document—as is the case with the many JavaScript libraries used in professional web development today.
- We'll begin separating both CSS3 and JavaScript into separate files starting in Chapter 10.

6.3 Modifying Your First Script

- A script can display `Welcome to JavaScript Programming!` in many ways.
- Figure 6.2 displays the text in magenta, using the CSS `color` property.
- Method `write` displays a string like `writeln`, but does not position the output cursor in the HTML5 document at the beginning of the next line after writing its argument

```
1  <!DOCTYPE html>
2
3  <!-- Fig. 6.2: welcome2.html -->
4  <!-- Printing one line with multiple statements. -->
5  <html>
6     <head>
7        <meta charset = "utf-8">
8        <title>Printing a Line with Multiple Statements</title>
9        <script type = "text/javascript">
10          <!--
11          document.write( "<h1 style = 'color: magenta'>" );
12          document.write( "Welcome to JavaScript " +
13             "Programming!</h1>" );
14          // -->
15       </script>
16    </head><body></body>
17 </html>
```

Fig. 6.2 | Printing one line with separate statements. (Part 1 of 2.)

Fig. 6.2 | Printing one line with separate statements. (Part 2 of 2.)

6.3 Modifying Your First Script (Cont.)

- The + operator (called the “concatenation operator” when used in this manner) joins two strings together

6.3 Modifying Your First Script (Cont.)

Displaying Text in an Alert Dialog

- Dialogs
  - Useful to display information in windows that "pop up" on the screen to grab the user's attention
  - Typically used to display important messages to the user browsing the web page
  - Browser's `window` object uses method `alert` to display an alert dialog
  - Method `alert` requires as its argument the string to be displayed

```
1  <!DOCTYPE html>
2
3  <!-- Fig. 6.3: welcome3.html -->
4  <!-- Alert dialog displaying multiple lines. -->
5  <html>
6     <head>
7        <meta charset = "utf-8">
8        <title>Printing Multiple Lines in a Dialog Box</title>
9        <script type = "text/javascript">
10          <!--
11          window.alert( "Welcome to\nJavaScript\nProgramming!" );
12          // -->
13       </script>
14    </head>
15    <body>
16       <p>Click Refresh (or Reload) to run this script again.</p>
17    </body>
18 </html>
```

Fig. 6.3 | Alert dialog displaying multiple lines. (Part 1 of 2.)

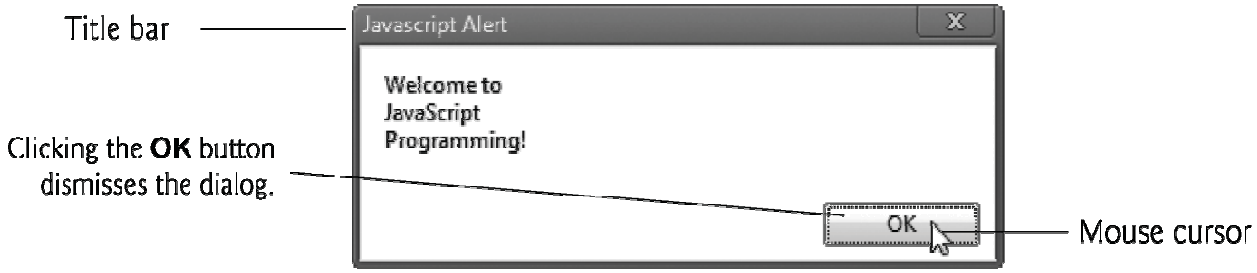


Fig. 6.3 | Alert dialog displaying multiple lines. (Part 2 of 2.)

6.3 Modifying Your First Script (Cont.)

Escape Sequences

- 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 HTML5 document to move to the beginning of the next line.

Escape sequence	Description
`\n`	*New line*—position the screen cursor at the beginning of the next line.
`\t`	*Horizontal tab*—move the screen cursor to the next tab stop.
`\\`	*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, `window.alert( "\"in double quotes\"" );` displays `"in double quotes"` in an `alert` dialog.
`\'`	*Single quote*—used to represent a single-quote character in a string. For example, `window.alert( '\'in single quotes\'' );` displays `'in single quotes'` in an `alert` dialog.

Fig. 6.4 | Some common escape sequences.

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
  - Such web pages are said to be dynamic, as opposed to static, since their content has the ability to change

6.4.1 Dynamic Welcome Page

- The next script creates 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 enter a value that the script can use.
- Figure 6.5 presents the script and sample output.

```
<!DOCTYPE html>

<!-- Fig. 6.5: welcome4.html -->
<!-- Prompt box used on a welcome screen -->
<html>
   <head>
      <meta charset = "utf-8">
      <title>Using Prompt and Alert Boxes</title>
      <script type = "text/javascript">
         <!--
         var name; // string entered by the user

         // read the name from the prompt box as a string
         name = window.prompt( "Please enter your name" );

         document.writeln( "<h1>Hello " + name +
            ", welcome to JavaScript programming!</h1>" );
         // -->
      </script>
   </head><body></body>
</html>
```

Fig. 6.5 | Prompt box used on a welcome screen. (Part 1 of 2.)

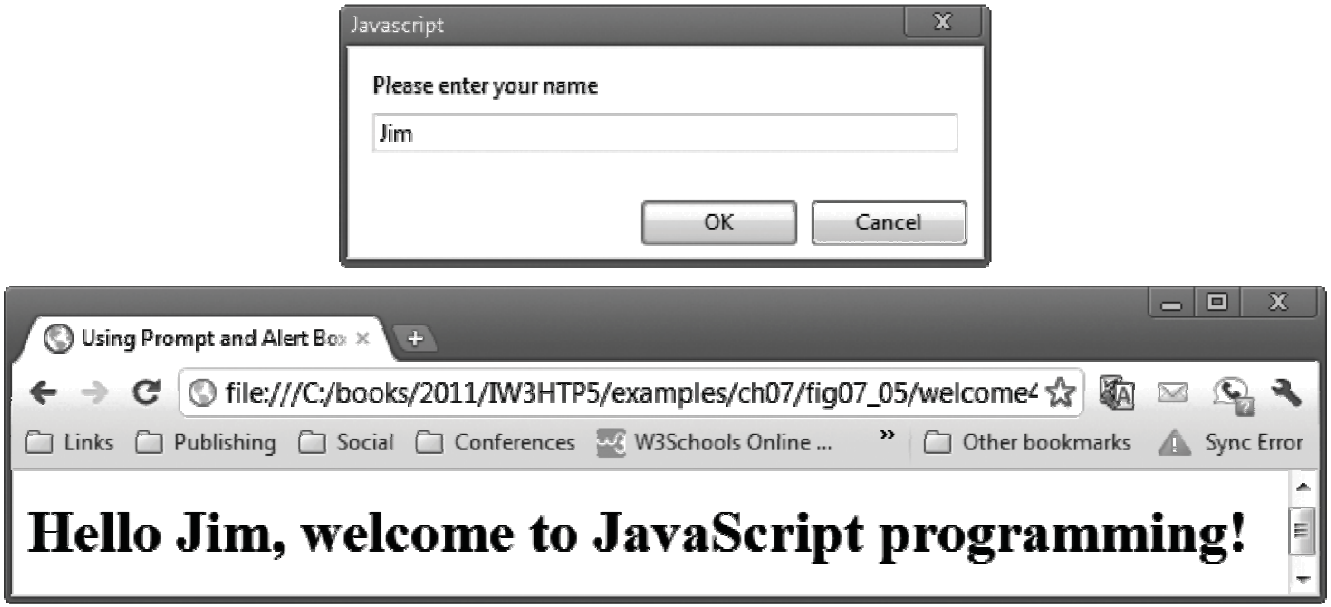


Fig. 6.5 | Prompt box used on a welcome screen. (Part 2 of 2.)

6.4.1 Dynamic Welcome Page (cont.)

- **Keywords** are words with special meaning in JavaScript
- Keyword `var`
  - 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 script
  - All variables have a name, type and value, and should be declared with a `var` statement before they are used in a script
- 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.

6.4.1 Dynamic Welcome Page (cont.)

- 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.
- Comments
  - 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.

6.4.1 Dynamic Welcome Page (cont.)

- 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.
- Script can then use the value that the user inputs.

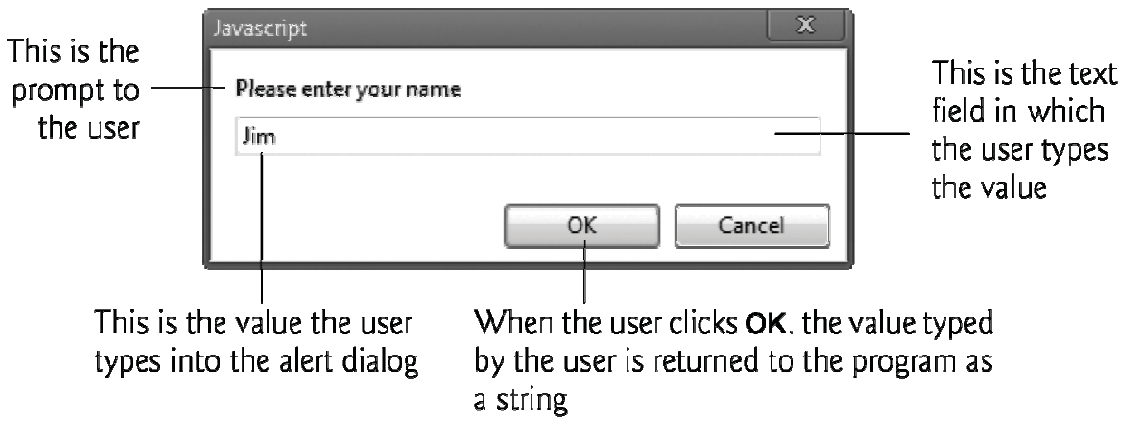


Fig. 6.6 | Prompt dialog displayed by the `window` object's `prompt` method.

6.4.1 Dynamic Welcome Page (cont.)

- 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.

6.4.1 Dynamic Welcome Page (cont.)

- `null` keyword
  - Signifies that a variable has no value
  - `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

6.4.2 Adding Integers

- Our next script illustrates another use of prompt dialogs to obtain input from the user.
- Figure 6.7 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.

```
<!DOCTYPE html>

<!-- Fig. 6.7: addition.html -->
<!-- Addition script. -->
<html>
   <head>
      <meta charset = "utf-8">
      <title>An Addition Program</title>
      <script type = "text/javascript">
         <!--
         var firstNumber; // first string entered by user
         var secondNumber; // second string entered by user
         var number1; // first number to add
         var number2; // second number to add
         var sum; // sum of number1 and number2

         // read in first number from user as a string
         firstNumber = window.prompt( "Enter first integer" );

         // read in second number from user as a string
         secondNumber = window.prompt( "Enter second integer" );

```

Fig. 6.7 | Addition script. (Part 1 of 3.)

```
23            // convert numbers from strings to integers
24            number1 = parseInt( firstNumber );
25            number2 = parseInt( secondNumber );
26
27            sum = number1 + number2; // add the numbers
28
29            // display the results
30            document.writeln( "<h1>The sum is " + sum + "</h1>" );
31            // -->
32         </script>
33      </head><body></body>
34   </html>
```

Fig. 6.7 | Addition script. (Part 2 of 3.)

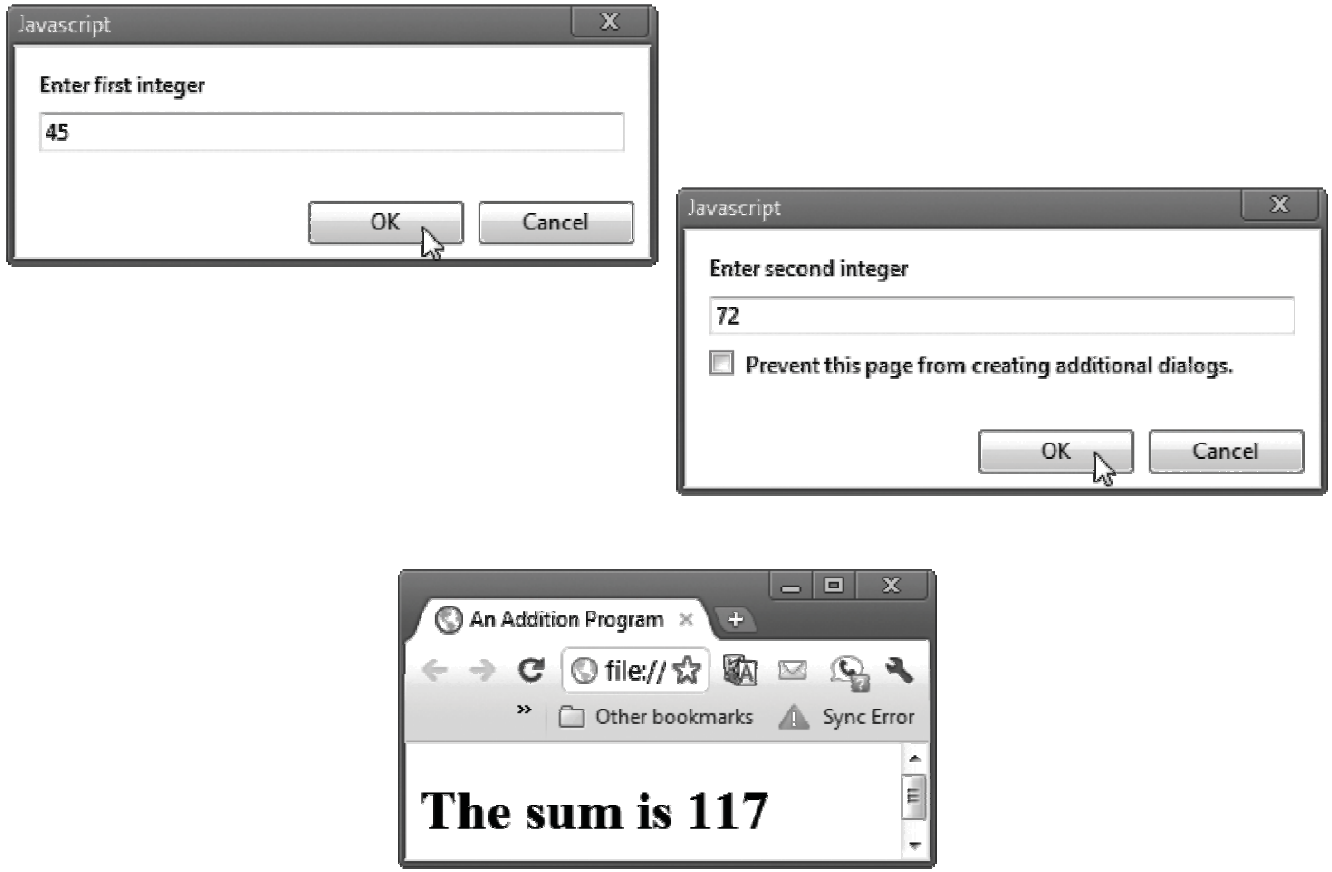


Fig. 6.7 | Addition script. (Part 3 of 3.)

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.

Fig. 6.8 | Memory location showing the name and value of variable `number1`.

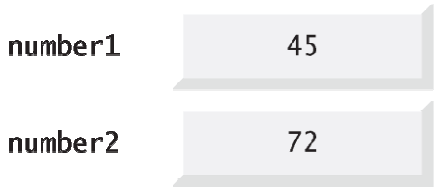


Fig. 6.9 | Memory locations after inputting values for variables `number1` and `number2`.

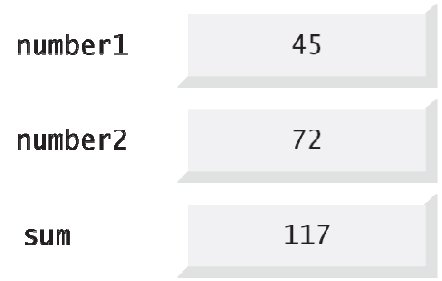


Fig. 6.10 | Memory locations after calculating the sum of number1 and number2.

6.5 Memory Concepts (Cont.)

- JavaScript does not require variables to have a type before they can be used in a script
- 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
- 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.

6.6 Arithmetic

- The basic arithmetic operators (+, -, *, /, and %) are binary operators, because they each operate on two operands
- JavaScript provides the remainder operator, %, which yields the remainder after division
- Arithmetic expressions in JavaScript must be written in straight-line form to facilitate entering programs into the computer

JavaScript operation	Arithmetic operator	Algebraic expression	JavaScript expression
Addition	`+`	$f + 7$	`f + 7`
Subtraction	`-`	$p - c$	`p - c`
Multiplication	`*`	bm	`b * m`
Division	`/`	x/y or $\frac{x}{y}$ or $x \div y$	`x / y`
Remainder	`%`	$r \bmod s$	`r % s`

Fig. 6.11 | Arithmetic operators.

6.6 Arithmetic (Cont.)

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

Operator(s)	Operation(s)	Order of evaluation (precedence)
*, / or %	Multiplication Division Remainder	Evaluated first. If there are several such operations, they're evaluated from left to right.
+ or -	Addition Subtraction	Evaluated last. If there are several such operations, they're evaluated from left to right.

Fig. 6.12 | Precedence of arithmetic operators.

6.7 Decision Making: Equality and Relational Operators

- `if` statement allows a script 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

6.7 Decision Making: Equality and Relational Operators (Cont.)

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

Standard algebraic equality operator or relational operator	JavaScript equality or relational operator	Sample JavaScript condition	Meaning of JavaScript condition
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. 6.13 | Equality and relational operators.

6.7 Decision Making: Equality and Relational Operators (Cont.)

- The script in Fig. 6.14 uses four `if` statements to display a time-sensitive greeting on a welcome page.

```
<!DOCTYPE html>

<!-- Fig. 6.14: welcome5.html -->
<!-- Using equality and relational operators. -->
<html>
   <head>
      <meta charset = "utf-8">
      <title>Using Relational Operators</title>
      <script type = "text/javascript">
         <!--
         var name; // string entered by the user
         var now = new Date();      // current date and time
         var hour = now.getHours(); // current hour (0-23)

         // read the name from the prompt box as a string
         name = window.prompt( "Please enter your name" );

         // determine whether it's morning
         if ( hour < 12 )
            document.write( "<h1>Good Morning, " );

```

Fig. 6.14 | Using equality and relational operators. (Part 1 of 3.)

```
22            // determine whether the time is PM
23            if ( hour >= 12 )
24            {
25               // convert to a 12-hour clock
26               hour = hour - 12;
27
28               // determine whether it is before 6 PM
29               if ( hour < 6 )
30                  document.write( "<h1>Good Afternoon, " );
31
32               // determine whether it is after 6 PM
33               if ( hour >= 6 )
34                  document.write( "<h1>Good Evening, " );
35            } // end if
36
37            document.writeln( name +
38               ", welcome to JavaScript programming!</h1>" );
39            // -->
40         </script>
41      </head><body></body>
42   </html>
```

Fig. 6.14 | Using equality and relational operators. (Part 2 of 3.)

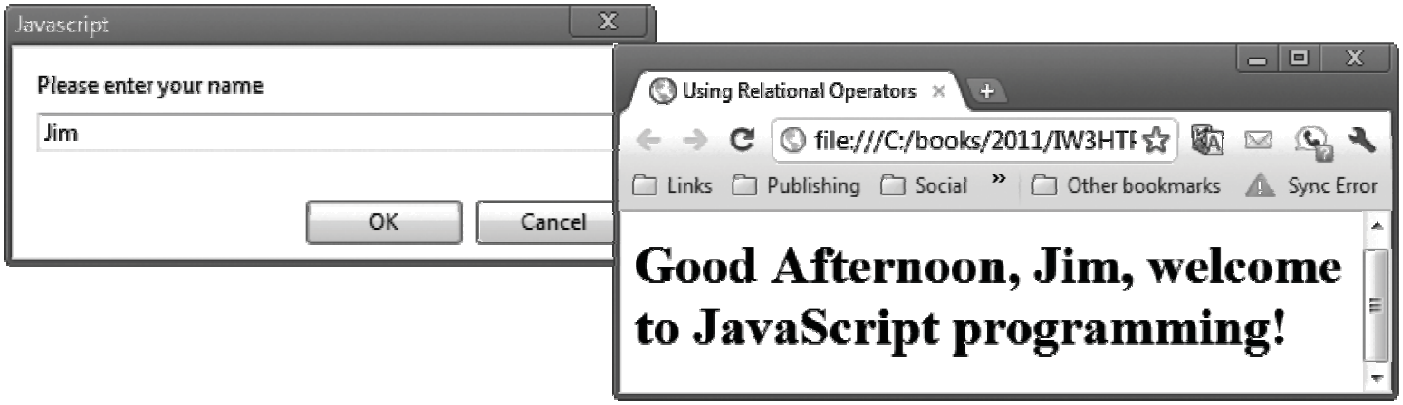


Fig. 6.14 | Using equality and relational operators. (Part 3 of 3.)

6.7 Decision Making: Equality and Relational Operators (Cont.)

- `Date` object
  - Used acquire the current local time
  - 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

Chapter 7
JavaScript: Control Statements, Part 1

Internet & World Wide Web
How to Program, 5/e

7.4 Control Statements (Cont.)

- JavaScript provides three selection structures.
  - The `if` statement either performs (selects) an action if a condition is true or skips the action if the condition is false.
    - Called a single-selection statement because it selects or ignores a single action or group of actions.
  - The `if…else` statement performs an action if a condition is true and performs a different action if the condition is false.
    - Double-selection statement because it selects between two different actions or group of actions.
  - The `switch` statement performs one of many different actions, depending on the value of an expression.
    - Multiple-selection statement because it selects among many different actions or groups of actions.

7.4 Control Statements (Cont.)

- JavaScript provides four repetition statements, namely, `while`, `do...while`, `for` and `for...in`.
- In addition to keywords, JavaScript has other words that are reserved for use by the language, such as the values `null`, `true` and `false`, and words that are reserved for possible future use.

JavaScript reserved keywords				
`break`	`case`	`catch`	`continue`	`default`
`delete`	`do`	`else`	`false`	`finally`
`for`	`function`	`if`	`in`	`instanceof`
`new`	`null`	`return`	`switch`	`this`
`throw`	`true`	`try`	`typeof`	`var`
`void`	`while`	`with`		
Keywords that are reserved but not used by JavaScript				
`class`	`const`	`enum`	`export`	`extends`
`implements`	`import`	`interface`	`let`	`package`
`private`	`protected`	`public`	`static`	`super`
`yield`				

Fig. 7.2 | JavaScript reserved keywords.

7.4 Control Statements (Cont.)

- Single-entry/single-exit control statements make it easy to build scripts.
- Control statements are attached to one another by connecting the exit point of one control statement to the entry point of the next.
  - Control-statement stacking.
- There is only one other way control statements may be connected
  - Control-statement nesting

7.5 `if` Selection Statement

- The JavaScript interpreter ignores *white-space characters*
  - blanks, tabs and newlines used for indentation and vertical spacing
- A decision can be made on any expression that evaluates to a value of JavaScript's boolean type (i.e., any expression that evaluates to `true` or `false`).
- The indentation convention you choose should be carefully applied throughout your scripts
  - It is difficult to read scripts that do not use uniform spacing conventions

7.6 `if...else` Selection Statement (Cont.)

- Conditional operator (`?:`)
  - Closely related to the `if…else` statement
  - JavaScript's only ternary operator—it takes three operands
  - The operands together with the `?:` operator form a conditional expression
  - The first operand is a boolean expression
  - The second is the value for the conditional expression if the boolean expression evaluates to `true`
  - Third is the value for the conditional expression if the boolean expression evaluates to `false`

7.6 `if...else` Selection Statement (Cont.)

- Nested `if…else` statements
  - Test for multiple cases by placing `if…else` statements inside other `if…else` statements
- The JavaScript interpreter always associates an `else` with the previous `if`, unless told to do otherwise by the placement of braces (`{}`)
- The `if` selection statement expects only one statement in its body
  - To include several statements, enclose the statements in braces (`{` and `}`)
  - A set of statements contained within a pair of braces is called a block

7.6 `if...else` Selection Statement (Cont.)

- A **logic error** has its effect at execution time.
- A **fatal logic error** causes a script to fail and terminate prematurely.
- A **nonfatal logic error** allows a script to continue executing, but the script produces incorrect results.

Software Engineering Observation 7.5

Just as a block can be placed anywhere a single statement can be placed, it's also possible to have no statement at all (the empty statement) in such places. We represent the empty statement by placing a semicolon (;) where a statement would normally be.

7.7 while Repetition Statement

- `while`
  - Allows you to specify that an action is to be repeated while some condition remains true
  - The body of a loop may be a single statement or a block
  - Eventually, the condition becomes false and repetition terminates

```
<!DOCTYPE html>

<!-- Fig. 7.7: average.html -->
<!-- Counter-controlled repetition to calculate a class average. -->
<html>
   <head>
      <meta charset = "utf-8">
      <title>Class Average Program</title>
      <script>

         var total; // sum of grades
         var gradeCounter; // number of grades entered
         var grade; // grade typed by user (as a string)
         var gradeValue; // grade value (converted to integer)
         var average; // average of all grades

         // initialization phase
         total = 0; // clear total
         gradeCounter = 1; // prepare to loop

```

Fig. 7.7 | Counter-controlled repetition to calculate a class average. (Part 1 of 4.)

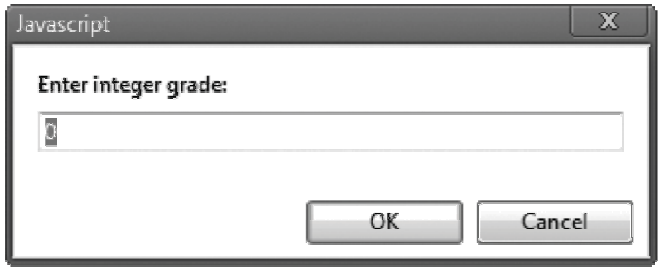
```
21          // processing phase
22          while ( gradeCounter <= 10 ) // loop 10 times
23          {
24
25             // prompt for input and read grade from user
26             grade = window.prompt( "Enter integer grade:", "0" );
27
28             // convert grade from a string to an integer
29             gradeValue = parseInt( grade );
30
31             // add gradeValue to total
32             total = total + gradeValue;
33
34             // add 1 to gradeCounter
35             gradeCounter = gradeCounter + 1;
36          } // end while
37
```

Fig. 7.7 | Counter-controlled repetition to calculate a class average. (Part 2 of 4.)

```
38            // termination phase
39            average = total / 10;   // calculate the average
40
41            // display average of exam grades
42            document.writeln(
43               "<h1>Class average is " + average + "</h1>" );
44
45         </script>
46      </head><body></body>
47   </html>
```

Fig. 7.7 | Counter-controlled repetition to calculate a class average. (Part 3 of 4.)

a) This dialog is displayed 10 times. User input is 100, 88, 93, 55, 68, 77, 83, 95, 73 and 62. User enters each grade and presses **OK**.



b) The class average is displayed in a web page

Fig. 7.7 | Counter-controlled repetition to calculate a class average. (Part 4 of 4.)

7.8 Formulating Algorithms: Counter-Controlled Repetition

- JavaScript represents all numbers as floating-point numbers in memory
- Floating-point numbers often develop through division
- The computer allocates only a fixed amount of space to hold such a value, so the stored floating-point value can only be an approximation

7.9 Formulating Algorithms: Sentinel-Controlled Repetition

- Sentinel-controlled repetition
  - Special value called a sentinel value (also called a signal value, a dummy value or a flag value) indicates the end of data entry
  - Often is called indefinite repetition, because the number of repetitions is not known in advance
- Choose a sentinel value that cannot be confused with an acceptable input value

```
1  Initialize total to zero
2  Initialize gradeCounter to zero
3
4  Input the first grade (possibly the sentinel)
5
6  While the user has not as yet entered the sentinel
7      Add this grade into the running total
8      Add one to the grade counter
9      Input the next grade (possibly the sentinel)
10
11 If the counter is not equal to zero
12     Set the average to the total divided by the counter
13     Print the average
14 Else
15     Print "No grades were entered"
```

Fig. 7.8 | Sentinel-controlled repetition to solve the class-average problem.

7.9 Formulating Algorithms: Sentinel-Controlled Repetition (Cont.)

- Control statements may be stacked on top of one another in sequence

```
<!DOCTYPE html>

<!-- Fig. 7.9: average2.html -->
<!-- Sentinel-controlled repetition to calculate a class average. -->
<html>
   <head>
      <meta charset = "utf-8">
      <title>Class Average Program: Sentinel-controlled Repetition</title>
      <script>

         var total; // sum of grades
         var gradeCounter; // number of grades entered
         var grade; // grade typed by user (as a string)
         var gradeValue; // grade value (converted to integer)
         var average; // average of all grades

         // initialization phase
         total = 0; // clear total
         gradeCounter = 0; // prepare to loop

```

Fig. 7.9 | Sentinel-controlled repetition to calculate a class average. (Part 1 of 4.)

```
21          // processing phase
22          // prompt for input and read grade from user
23          grade = window.prompt(
24               "Enter Integer Grade, -1 to Quit:", "0" );
25
26          // convert grade from a string to an integer
27          gradeValue = parseInt( grade );
28
29          while ( gradeValue != -1 )
30          {
31             // add gradeValue to total
32             total = total + gradeValue;
33
34             // add 1 to gradeCounter
35             gradeCounter = gradeCounter + 1;
36
37             // prompt for input and read grade from user
38             grade = window.prompt(
39                  "Enter Integer Grade, -1 to Quit:", "0" );
40
41             // convert grade from a string to an integer
42             gradeValue = parseInt( grade );
43          } // end while
```

Fig. 7.9 | Sentinel-controlled repetition to calculate a class average. (Part 2 of 4.)

```
44
45          // termination phase
46          if ( gradeCounter != 0 )
47          {
48             average = total / gradeCounter;
49
50             // display average of exam grades
51             document.writeln(
52                "<h1>Class average is " + average + "</h1>" );
53          } // end if
54          else
55             document.writeln( "<p>No grades were entered</p>" );
56
57       </script>
58    </head><body></body>
59 </html>
```

Fig. 7.9 | Sentinel-controlled repetition to calculate a class average. (Part 3 of 4.)

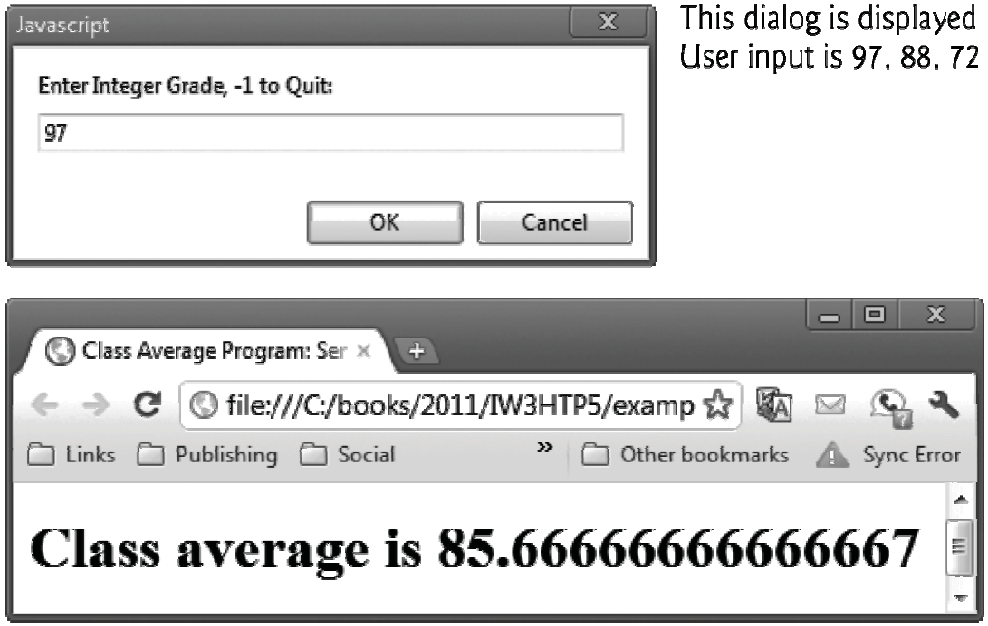


Fig. 7.9 | Sentinel-controlled repetition to calculate a class average. (Part 4 of 4.)

7.10 Formulating Algorithms: Nested Control Statements

- Control structures may be nested inside of one another

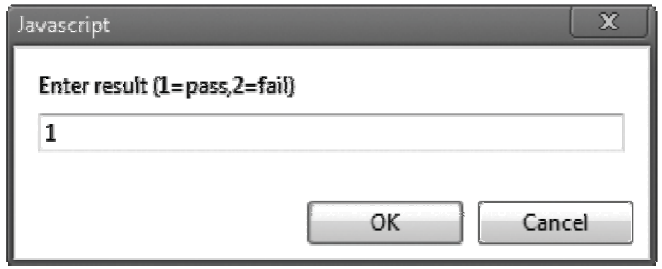
```
<!DOCTYPE html>

<!-- Fig. 7.11: analysis.html -->
<!-- Examination-results calculation. -->
<html>
   <head>
      <meta charset = "utf-8">
      <title>Analysis of Examination Results</title>
      <script>

         // initializing variables in declarations
         var passes = 0; // number of passes
         var failures = 0; // number of failures
         var student = 1; // student counter
         var result; // an exam result

```

Fig. 7.11 | Examination-results calculation. (Part 1 of 4.)

```
17          // process 10 students; counter-controlled loop
18          while ( student <= 10 )
19          {
20             result = window.prompt( "Enter result (1=pass,2=fail)", "0" );
21
22             if ( result == "1" )
23                passes = passes + 1;
24             else
25                failures = failures + 1;
26
27             student = student + 1;
28          } // end while
29
30          // termination phase
31          document.writeln( "<h1>Examination Results</h1>" );
32          document.writeln( "<p>Passed: " + passes +
33             "; Failed: " + failures + "</p>" );
34
35          if ( passes > 8 )
36             document.writeln( "<p>Bonus to instructor!</p>" );
37
38       </script>
39    </head><body></body>
40 </html>
```

Fig. 7.11 | Examination-results calculation. (Part 2 of 4.)

a) This dialog is displayed 10 times. User input is 1, 2, 1, 1, 1, 1, 1, 1, 1 and 1.



b) Nine students passed and one failed, therefore `"Bonus to instructor!"` is printed.

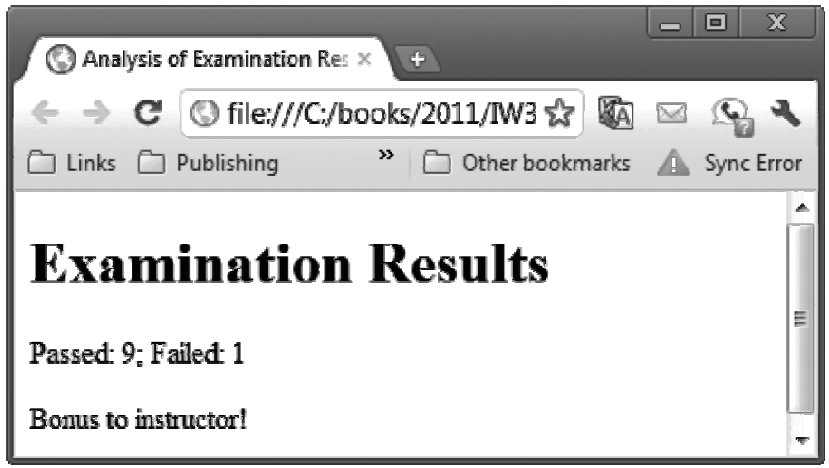
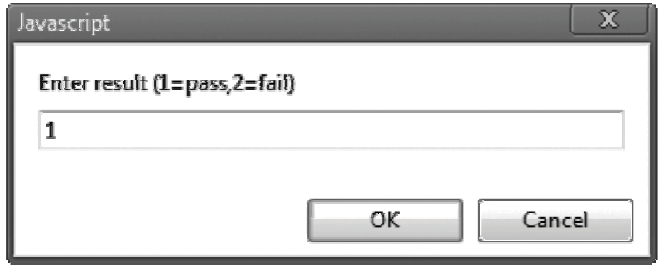


Fig. 7.11 | Examination-results calculation. (Part 3 of 4.)

c) This dialog is displayed 10 times. User input is 1, 2, 1, 2, 2, 1, 2, 2, 1 and 1.



d) Five students passed and five failed, so no bonus is paid to the instructor.

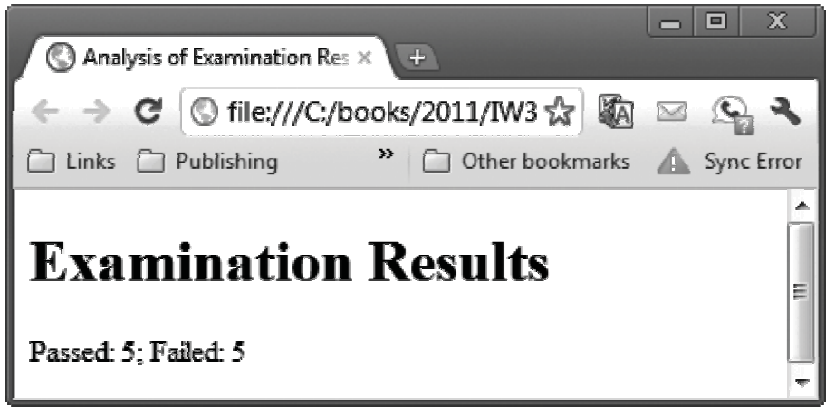


Fig. 7.11 | Examination-results calculation. (Part 4 of 4.)

7.11 Assignment Operators

- JavaScript provides the arithmetic assignment operators +=, -=, *=, /= and %=, which abbreviate certain common types of expressions.

Assignment operator	Initial value of variable	Sample expression	Explanation	Assigns
`+=`	`c = 3`	`c += 7`	`c = c + 7`	10 to c
`-=`	`d = 5`	`d -= 4`	`d = d - 4`	1 to d
`*=`	`e = 4`	`e *= 5`	`e = e * 5`	20 to e
`/=`	`f = 6`	`f /= 3`	`f = f / 3`	2 to f
`%=`	`g = 12`	`g %= 9`	`g = g % 9`	3 to g

Fig. 7.12 | Arithmetic assignment operators.

7.12 Increment and Decrement Operators

- The increment operator, ++, and the decrement operator, --, increment or decrement a variable by 1, respectively.
- If the operator is prefixed to the variable, the variable is incremented or decremented by 1, then used in its expression.
- If the operator is postfixed to the variable, the variable is used in its expression, then incremented or decremented by 1.

Operator	Example	Called	Explanation
`++`	`++a`	preincrement	Increment `a` by 1, then use the new value of `a` in the expression in which `a` resides.
`++`	`a++`	postincrement	Use the current value of `a` in the expression in which `a` resides, then increment `a` by 1.
`--`	`--b`	predecrement	Decrement `b` by 1, then use the new value of `b` in the expression in which `b` resides.
`--`	`b--`	postdecrement	Use the current value of `b` in the expression in which `b` resides, then decrement `b` by 1.

Fig. 7.13 | Increment and decrement operators.

```
<!DOCTYPE html>

<!-- Fig. 7.14: increment.html -->
<!-- Preincrementing and Postincrementing. -->
<html>
   <head>
      <meta charset = "utf-8">
      <title>Preincrementing and Postincrementing</title>
      <script>

         var c;

         c = 5;
         document.writeln( "<h3>Postincrementing</h3>" );
         document.writeln( "<p>" + c ); // prints 5
         // prints 5 then increments
         document.writeln( " " + c++ );
         document.writeln( " " + c + "</p>" ); // prints 6

```

Fig. 7.14 | Preincrementing and postincrementing. (Part 1 of 2.)

```
20            c = 5;
21            document.writeln( "<h3>Preincrementing</h3>" );
22            document.writeln( "<p>" + c ); // prints 5
23            // increments then prints 6
24            document.writeln( " " + ++c );
25            document.writeln( " " + c + "</p>" ); // prints 6
26
27         </script>
28      </head><body></body>
29   </html>
```

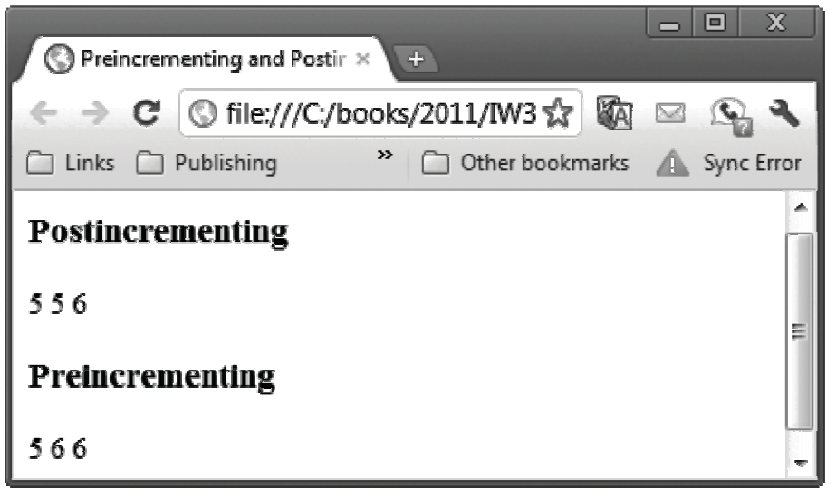


Fig. 7.14 | Preincrementing and postincrementing. (Part 2 of 2.)

7.12 Increment and Decrement Operators (Cont.)

- When incrementing or decrementing a variable in a statement by itself, the preincrement and postincrement forms have the same effect, and the predecrement and postdecrement forms have the same effect
- When a variable appears in the context of a larger expression, preincrementing the variable and postincrementing the variable have different effects. Predecrementing and postdecrementing behave similarly.

Operator	Associativity	Type
`++ --`	right to left	unary
`* / %`	left to right	multiplicative
`+ -`	left to right	additive
`< <= > >=`	left to right	relational
`== != === !===`	left to right	equality
`?:`	right to left	conditional
`= += -= *= /= %=`	right to left	assignment

Fig. 7.15 | Precedence and associativity of the operators discussed so far.