JavaScript: Introduction to Scripting

6.1 Introduction

- JavaScript
  - Scripting language 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

- 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

```
1  <!DOCTYPE html>
2
3  <!-- Fig. 6.1: welcome.html -->
4  <!-- Displaying a line of text. -->
5  <html>
6     <head>
7        <meta charset = "utf-8">
8        <title>A First Program in JavaScript</title>
9        <script type = "text/javascript">
10
11          document.writeln(
12             "<h1>Welcome to JavaScript Programming!</h1>" );
13
14       </script>
15    </head><body></body>
16 </html>
```

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

6.2 Your First Script (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**
  - **<script type = "text/javascript">**

6.2 Your First Script (cont.)

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

6.2 Your First Script (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
  - http://www.w3schools.com/jsref/dom_obj_document.asp

6.2 Your First Script

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

6.2 Your First Script

- 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

6.2 Your First Script (cont.)

- The document object's **writeln** method
  - Writes a line of HTML5 text 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
  - **<script>**

    document.write("Hello World!");

    document.writeln("Have a nice day!");

    </script>
  - http://www.w3schools.com/jsref/met_doc_writeln.asp

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

Software Engineering Observation 6.1

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

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.

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. Each browser has its own way to display JavaScript Errors. For example, Firefox has the Error Console (in its Web Developer menu) and Chrome has the JavaScript console (in its Tools menu). To view script errors in IE9, select *Internet Options...* from the *Tools* menu. In the dialog that appears, select the *Advanced* tab and click the checkbox labeled *Display a notification about every script error* under the *Browsing* category.

Error-Prevention Tip 6.1

When the interpreter reports a syntax error, sometimes the error is not in the line 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.

6.2 Your First Script

- 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

Common Programming Error 6.3

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

Common Programming Error 6.4

Many people confuse the writing of HTML5 text with the rendering of HTML5 text. Writing HTML5 text creates the HTML5 that will be rendered by the browser for presentation to the user.

6.3 Modifying Your First Script (Cont.)

- 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
    - http://www.w3schools.com/jsref/met_win_alert.asp
    - **window.alert("Welcome to\nJavaScript\nProgramming!");**
    - http://www.w3schools.com/jsref/obj_window.asp

```
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.)

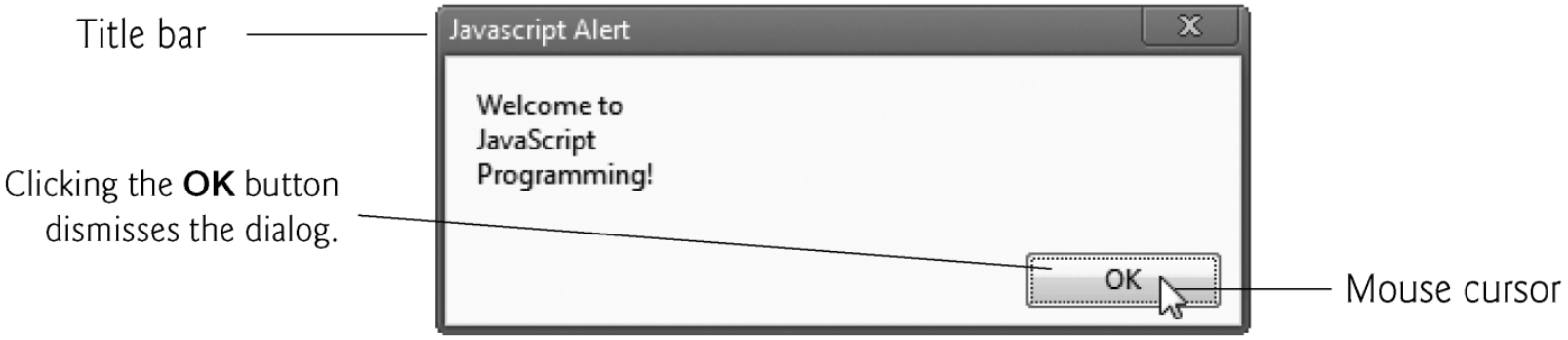

Fig. 6.3 | Alert dialog displaying multiple lines. (Part 2 of 2.)

6.3 Modifying Your First Script (Cont.)

- 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
  - Allow the user to enter a value that the script can use
- Figure 6.5 presents the script and sample output

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)
  - **var name;**
  - **var y = 5;**
  - **var x = y + 2;**
  - Several variables may be declared in one declaration or in multiple declarations

6.4.1 Dynamic Welcome Page (cont.)

- 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
  - **var person = prompt("Please enter your name", "Harry Potter");**
  - http://www.w3schools.com/jsref/met_win_prompt.asp
- Script can then use the value that the user inputs

6.4.1 Dynamic Welcome Page (cont.)

- The user can type anything in the text field of the prompt dialog
  - If the user clicks the **Cancel** button, no string value is sent to the script
  - Instead, the prompt dialog submits the value **null**
- **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"

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
  - **name = window.prompt("Please enter your name");**

6.4.1 Dynamic Welcome Page (cont.)

- The expression inside the parentheses uses the operator + to "add"
  - A string (the literal "<h1>Hello, ")
  - The variable name
  - 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 to be combined
- The result of this operation is a new string

6.4.1 Dynamic Welcome Page (cont.)

- As you'll see 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 "

6.4.1 Dynamic Welcome Page (cont.)

- After the browser interprets the **<head>** section of theHTML5 document (which contains the JavaScript)
  - It then interprets the **<body>** of the HTML5 document and renders the HTML5
  - The HTML5 page is *not* rendered until the **prompt** is dismissed because the **prompt** pauses execution in the head, before the body is processed
  - If you reload the page after entering a name, the browser will execute the script again and so you can change the name

```
<!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.)

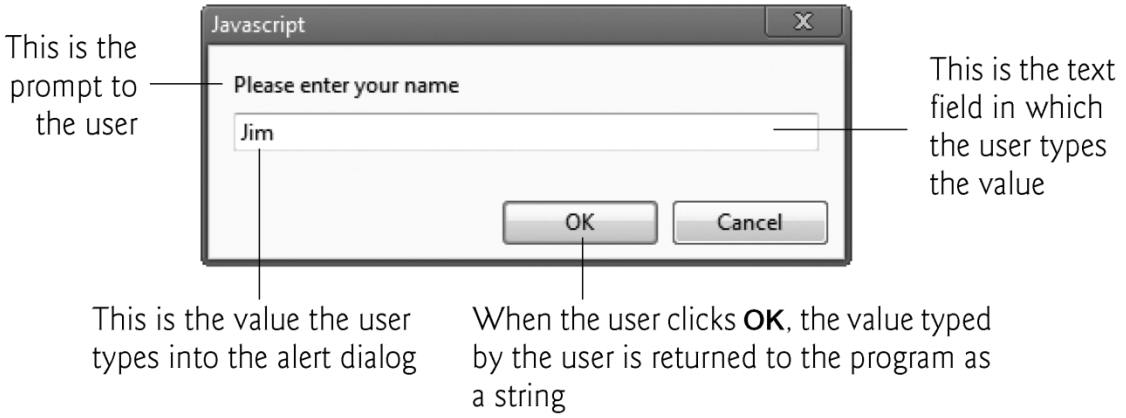

Fig. 6.6 | Prompt dialog displayed by the `window` object's `prompt` method.

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).

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`.

Good Programming Practice 6.4

Although it's not required, declare each variable on a separate line. This allows for easy insertion of a comment next to each declaration. This is a widely followed professional coding standard.

Good Programming Practice 6.5

Place a space on each side of a binary operator. This format makes the operator stand out and makes the script more readable.

Common Programming Error 6.5

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

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
- Function **parseInt** converts its string argument to an integer
- http://www.w3schools.com/jsref/jsref_parseint.asp
- http://www.w3schools.com/jsref/jsref_obj_global.asp

```
<!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.)

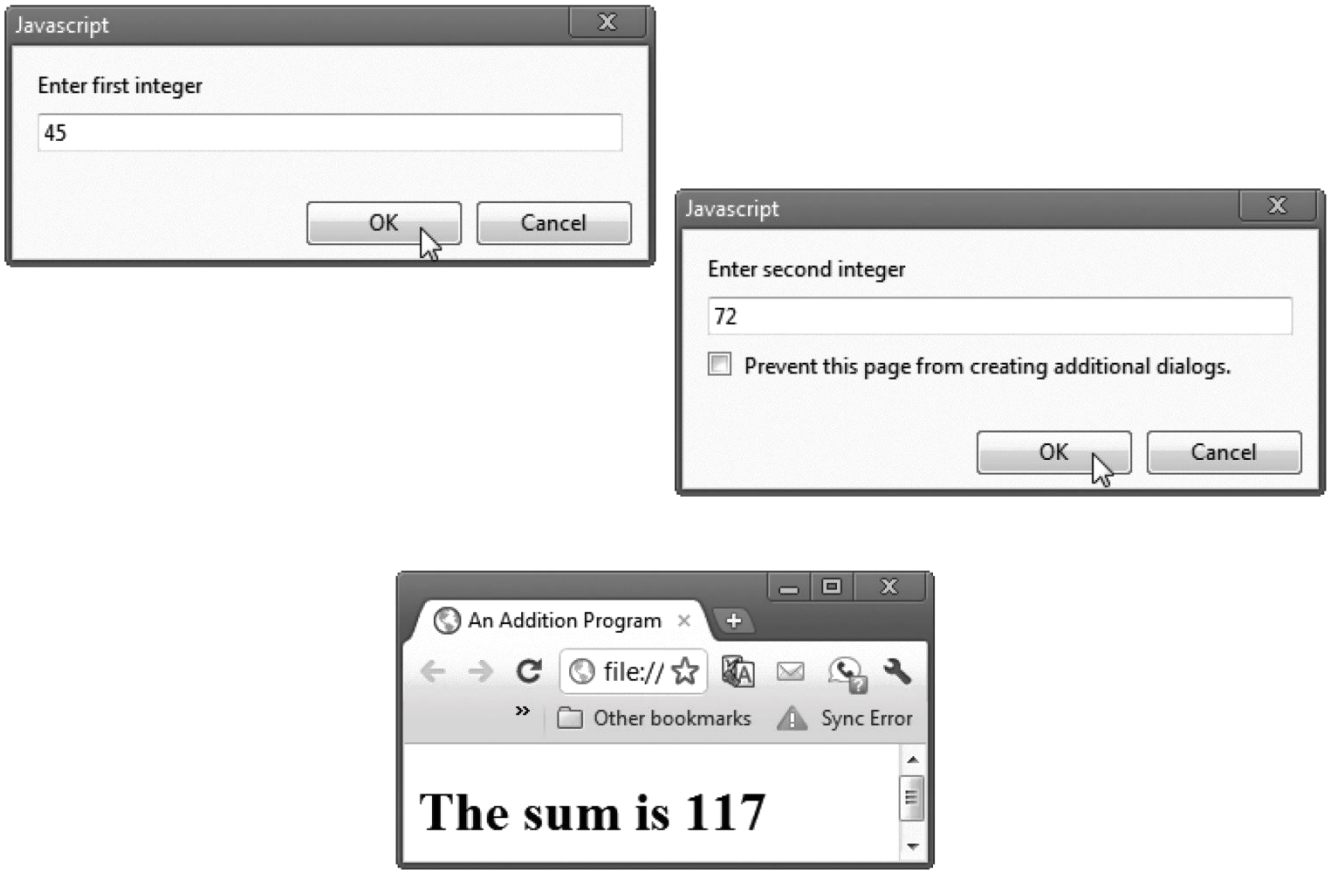

Fig. 6.7 | Addition script. (Part 3 of 3.)

Common Programming Error 6.6

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 calculated.

6.4.2 Adding Integers

- As discussed in the Preface, we validated our code using HTML5, CSS3 and JavaScript validation tools
- Browsers are generally forgiving and don't typically display error messages to the user
- As a programmer, you should thoroughly test your web pages and validate them
- Validation tools report two types of messages—*errors* and *warnings*
- Typically, you must resolve errors; otherwise, your web pages probably won't render or execute correctly

6.4.2 Adding Integers

- Pages with warnings normally render and execute correctly
  - Some organizations have strict protocols indicating that all pages must be free of both warnings and errors before they can be posted on a live website
- When you validate this example at www.javascriptlint.com, lines 24–25 produce the warning message
  - Function **parseInt** has an optional second parameter, known as the *radix*, that specifies the base number system that's used to parse the number (e.g., 8 for octal, 10 for decimal and 16 for hexadecimal)
  - The default is base 10, but you can specify any base from 2 to 32
  - **number1 = parseInt(firstNumber, 10);**

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 script

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

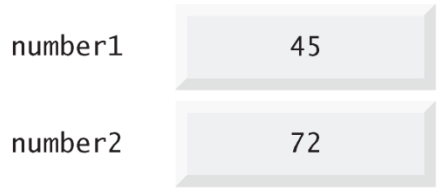

Fig. 6.9 | Memory locations after inputting values for variables `number1` and `number2`.

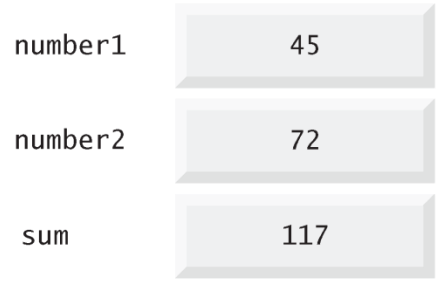

Fig. 6.10 | Memory locations after calculating the `sum` of `number1` and `number2`.

6.5 Memory Concepts (Cont.)

- 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
  - http://www.w3schools.com/jsref/jsref_operators.asp

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

6.6 Arithmetic (Cont.)

- 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

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

JavaScript:
```
m = ( a + b + c + d + e ) / 5;
```

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

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

Common Programming Error 6.7

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."

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
  - http://www.w3schools.com/jsref/jsref_obj_date.asp

```
1  <!DOCTYPE html>
2
3  <!-- Fig. 6.14: welcome5.html -->
4  <!-- Using equality and relational operators. -->
5  <html>
6     <head>
7        <meta charset = "utf-8">
8        <title>Using Relational Operators</title>
9        <script type = "text/javascript">
10          <!--
11          var name; // string entered by the user
12          var now = new Date();       // current date and time
13          var hour = now.getHours(); // current hour (0-23)
14
15          // read the name from the prompt box as a string
16          name = window.prompt( "Please enter your name" );
17
18          // determine whether it's morning
19          if ( hour < 12 )
20             document.write( "<h1>Good Morning, " );
21
```

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.)

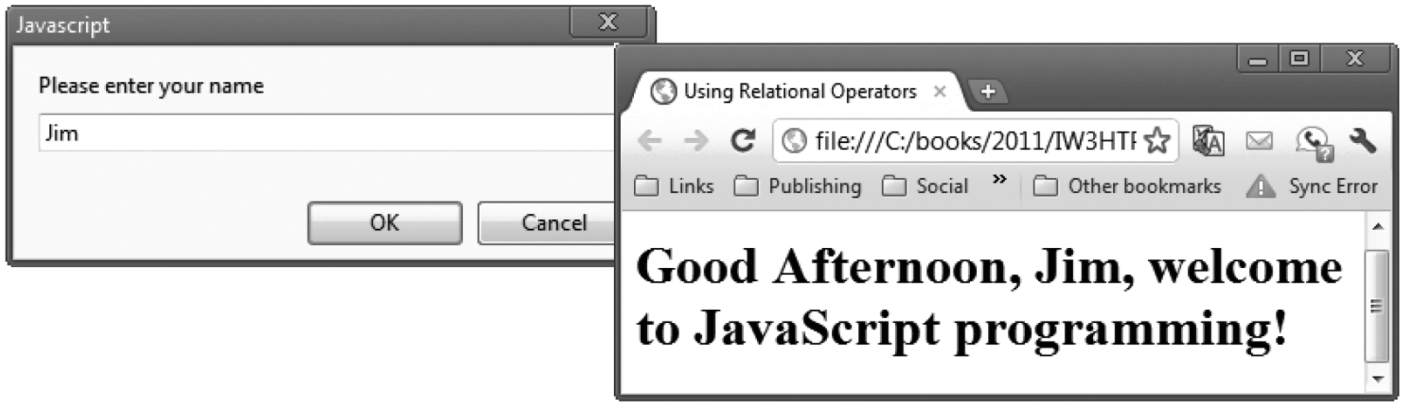

Fig. 6.14 | Using equality and relational operators. (Part 3 of 3.)

Good Programming Practice 6.6

Include comments after the closing curly brace of control statements (such as `if` statements) to indicate where the statements end, as in line 35 of Fig. 6.14.

Good Programming Practice 6.7

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

Good Programming Practice 6.8

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're uncertain about the order of evaluation, 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.

Error-Prevention Tip 6.2

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.

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

Fig. 6.15 | Precedence and associativity of the operators discussed so far.

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

- JavaScript can convert between types for you. This includes
- cases in which you're comparing values
  - The comparison "75" == 75 yields the value true because JavaScript converts the string "75" to the number 75 before performing the equality (==) comparison
  - To prevent implicit conversions in comparisons, which can lead to unexpected results, JavaScript provides the **strict equals (===)** and **strict does not equal (!==)** operators
  - The comparison "75" === 75 yields the value false because one operand is a string and the other is a number
  - Similarly, "75" !== 75 yields true because the operand's types are not equal, therefore the values are not equal

6.8 Web Resources

- The Deitel JavaScript Resource Center contains links to some of the best JavaScript resources on the web
  - http://www.deitel.com/javascript
- 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