



Chapter 8

JavaScript: Control

Statements, Part 2

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



OBJECTIVES

In this chapter you'll:

- Learn the essentials of counter-controlled repetition
- Use the **for** and **do...while** repetition statements to execute statements in a program repeatedly.
- Perform multiple selection using the **switch** selection statement.
- Use the **break** and **continue** program-control statements
- Use the logical operators to make decisions.



8.1 Introduction

8.2 Essentials of Counter-Controlled Repetition

8.3 `for` Repetition Statement

8.4 Examples Using the `for` Statement

8.5 `switch` Multiple-Selection Statement

8.6 `do...while` Repetition Statement

8.7 `break` and `continue` Statements

8.8 Logical Operators

8.9 Web Resources



8.2 Essentials of Counter-Controlled Repetition

- ▶ Counter-controlled repetition requires
 - name of a control variable
 - initial value of the control variable
 - the increment (or decrement) by which the control variable is modified each time through the loop
 - the condition that tests for the final value of the control variable to determine whether looping should continue



8.2 Essentials of Counter-Controlled Repetition (Cont.)

- ▶ The double-quote character delimits the beginning and end of a string literal in JavaScript
 - it cannot be used in a string unless it is preceded by a \ to create the escape sequence \"



8.2 Essentials of Counter-Controlled Repetition (Cont.)

- ▶ HTML5 allows either single quotes ('') or double quotes ("") to be placed around the value specified for an attribute
- ▶ JavaScript allows single quotes to be placed in a string literal



```
1 <!DOCTYPE html>
2
3 <!-- Fig. 8.1: WhileCounter.html -->
4 <!-- Counter-controlled repetition. -->
5 <html>
6   <head>
7     <meta charset = "utf-8">
8     <title>Counter-Controlled Repetition</title>
9     <script>
10
11       var counter = 1; // initialization
12
13       while ( counter <= 7 ) // repetition condition
14       {
15         document.writeln( "<p style = 'font-size: " +
16                           counter + "ex'>HTML5 font size " + counter + "ex</p>" );
17         ++counter; // increment
18     } //end while
19
20   </script>
21 </head><body></body>
22 </html>
```

Fig. 8.1 | Counter-controlled repetition. (Part 1 of 2.)



Fig. 8.1 | Counter-controlled repetition. (Part 2 of 2.)



8.3 for Repetition Statement

- ▶ **for statement**
 - Specifies each of the items needed for counter-controlled repetition with a control variable
 - Can use a block to put multiple statements into the body
- ▶ If the loop's condition uses a < or > instead of a <= or >=, or vice-versa, it can result in an off-by-one error
- ▶ **for statement header contains three expressions**
 - Initialization
 - Condition
 - Increment Expression
- ▶ The increment expression in the for statement acts like a stand-alone statement at the end of the body of the for statement
- ▶ Place only expressions involving the control variable in the initialization and increment sections of a for statement



```
1 <!DOCTYPE html>
2
3 <!-- Fig. 8.2: ForCounter.html -->
4 <!-- Counter-controlled repetition with the for statement. -->
5 <html>
6   <head>
7     <meta charset="utf-8">
8     <title>Counter-Controlled Repetition</title>
9     <script>
10
11       // Initialization, repetition condition and
12       // incrementing are all included in the for
13       // statement header.
14       for ( var counter = 1; counter <= 7; ++counter )
15         document.writeln( "<p style = 'font-size: " +
16                           counter + "ex'>HTML5 font size " + counter + "ex</p>" );
17
18     </script>
19   </head><body></body>
20 </html>
```

Fig. 8.2 | Counter-controlled repetition with the for statement.

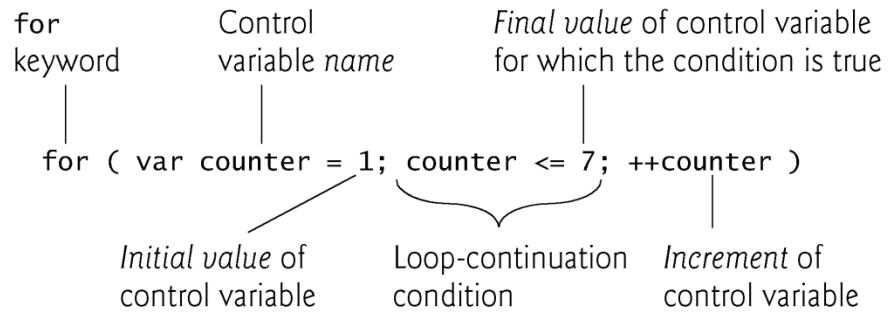


Fig. 8.3 | for statement header components.



8.3 for Repetition Statement (Cont.)

- ▶ The three expressions in the for statement are optional
- ▶ The two semicolons in the for statement are required
- ▶ The initialization, loop-continuation condition and increment portions of a for statement can contain arithmetic expressions



8.3 for Repetition Statement (Cont.)

- ▶ The part of a script in which a variable name can be used is known as the variable's scope
- ▶ The “increment” of a for statement may be negative, in which case it is called a decrement and the loop actually counts downward
- ▶ If the loop-continuation condition initially is false, the body of the for statement is not performed
 - Execution proceeds with the statement following the for statement



Error-Prevention Tip 8.1

Although the value of the control variable can be changed in the body of a `for` statement, avoid changing it, because doing so can lead to subtle errors.

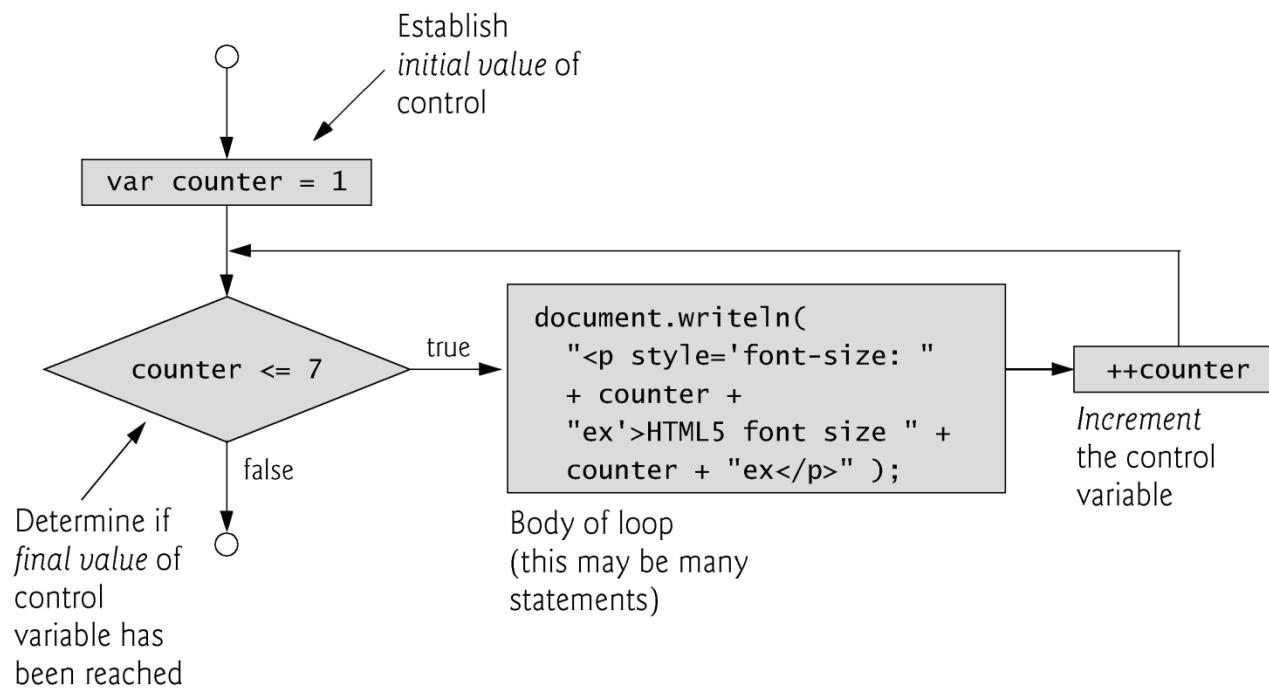


Fig. 8.4 | for repetition statement flowchart.



8.4 Examples Using the for Statement

- ▶ Figure 8.5 uses the for statement to sum the even integers from 2 to 100.



Common Programming Error 8.1

Not using the proper relational operator in the loop-continuation condition of a loop that counts downward (e.g., using $i \leq 1$ instead of $i \geq 1$ in a loop that counts down to 1) is a logic error.



```
1 <!DOCTYPE html>
2
3 <!-- Fig. 8.5: Sum.html -->
4 <!-- Summation with the for repetition structure. -->
5 <html>
6   <head>
7     <meta charset = "utf-8">
8     <title>Sum the Even Integers from 2 to 100</title>
9     <script>
10
11       var sum = 0;
12
13       for ( var number = 2; number <= 100; number += 2 )
14         sum += number;
15
16       document.writeln( "The sum of the even integers " +
17                     "from 2 to 100 is " + sum );
18
19     </script>
20   </head><body></body>
21 </html>
```

Fig. 8.5 | Summation with the for repetition structure. (Part I of 2.)



Fig. 8.5 | Summation with the for repetition structure. (Part 2 of 2.)



Good Programming Practice 8.1

Although statements preceding a **for** statement and in the body of a **for** statement can often be merged into the **for** header, avoid doing so, because it makes the program more difficult to read.



```
1 <!DOCTYPE html>
2
3 <!-- Fig. 8.6: Interest.html -->
4 <!-- Compound interest calculation with a for loop. -->
5 <html>
6   <head>
7     <meta charset = "utf-8">
8     <title>Calculating Compound Interest</title>
9     <style type = "text/css">
10    table { width: 300px;
11          border-collapse: collapse;
12          background-color: lightblue; }
13    table, td, th { border: 1px solid black;
14                  padding: 4px; }
15    th { text-align: left;
16          color: white;
17          background-color: darkblue; }
18    tr.oddrow { background-color: white; }
19  </style>
```

Fig. 8.6 | Compound interest calculation with a for loop. (Part I of 4.)



```
20 <script>
21
22     var amount; // current amount of money
23     var principal = 1000.00; // principal amount
24     var rate = 0.05; // interest rate
25
26     document.writeln("<table>"); // begin the table
27     document.writeln(
28         "<caption>Calculating Compound Interest</caption>");
29     document.writeln(
30         "<thead><tr><th>Year</th>" ); // year column heading
31     document.writeln(
32         "<th>Amount on deposit</th>"); // amount column heading
33     document.writeln( "</tr></thead><tbody>" );
34
```

Fig. 8.6 | Compound interest calculation with a for loop. (Part 2 of 4.)



```
35 // output a table row for each year
36 for ( var year = 1; year <= 10; ++year )
37 {
38     amount = principal * Math.pow( 1.0 + rate, year );
39
40     if ( year % 2 !== 0 )
41         document.writeln( "<tr class='oddrow'><td>" + year +
42                         "</td><td>" + amount.toFixed(2) + "</td></tr>" );
43     else
44         document.writeln( "<tr><td>" + year +
45                         "</td><td>" + amount.toFixed(2) + "</td></tr>" );
46 } //end for
47
48 document.writeln( "</tbody></table>" );
49
50 </script>
51 </head><body></body>
52 </html>
```

Fig. 8.6 | Compound interest calculation with a for loop. (Part 3 of 4.)

Calculating Compound I ×

file:///C|

Links Other bookmarks Sync Error

Calculating Compound Interest

Year	Amount on deposit
1	1050.00
2	1102.50
3	1157.63
4	1215.51
5	1276.28
6	1340.10
7	1407.10
8	1477.46
9	1551.33
10	1628.89

Fig. 8.6 | Compound interest calculation with a for loop. (Part 4 of 4.)



8.4 Examples Using the for Statement (cont.)

- ▶ JavaScript does not include an exponentiation operator
 - Math object's pow method for this purpose.
`Math.pow(x, y)` calculates the value of x raised to the y th power.



8.5 switch Multiple-Selection Statement

- ▶ **switch multiple-selection statement**
 - Tests a variable or expression separately for each of the values it may assume
 - Different actions are taken for each value
- ▶ **CSS property list-style-type**
 - Allows you to set the numbering system for a list
 - Possible values include
 - decimal (numbers—the default)
 - lower-roman (lowercase roman numerals)
 - upper-roman (uppercase roman numerals)
 - lower-alpha (lowercase letters)
 - upper-alpha (uppercase letters)
 - others



8.5 switch Multiple-Selection Statement (Cont.)

- ▶ **switch statement**
 - Consists of a series of `case` labels and an optional `default` case
 - When control reaches a `switch` statement
 - The script evaluates the controlling expression in the parentheses
 - Compares this value with the value in each of the `case` labels
 - If the comparison evaluates to true, the statements after the `case` label are executed in order until a `break` statement is reached
- ▶ The `break` statement is used as the last statement in each case to exit the `switch` statement immediately
- ▶ The `default` case allows you to specify a set of statements to execute if no other case is satisfied
 - Usually the last case in the `switch` statement



8.5 switch Multiple-Selection Statement (Cont.)

- ▶ Each case can have multiple actions (statements)
- ▶ Braces are not required around multiple actions in a case of a switch
- ▶ The break statement is not required for the last case because program control automatically continues with the next statement after the switch
- ▶ Having several case labels listed together (e.g., case 1: case 2: with no statements between the cases) executes the same set of actions for each case



```
1 <!DOCTYPE html>
2
3 <!-- Fig. 8.7: SwitchTest.html -->
4 <!-- Using the switch multiple-selection statement. -->
5 <html>
6   <head>
7     <meta charset = "utf-8">
8     <title>Switching between HTML5 List Formats</title>
9     <script>
10
11       var choice; // user's choice
12       var startTag; // starting list item tag
13       var endTag; // ending list item tag
14       var validInput = true; // true if input valid else false
15       var listType; // type of list as a string
16
17       choice = window.prompt( "Select a list style:\n" +
18         "1 (numbered), 2 (lettered), 3 (roman numbered)", "1" );
19
```

Fig. 8.7 | Using the switch multiple-selection statement. (Part I of 6.)



```
20     switch ( choice )
21     {
22         case "1":
23             startTag = "<ol>";
24             endTag = "</ol>";
25             listType = "<h1>Numbered List</h1>";
26             break;
27         case "2":
28             startTag = "<ol style = 'list-style-type: upper-alpha'>";
29             endTag = "</ol>";
30             listType = "<h1>Lettered List</h1>";
31             break;
32         case "3":
33             startTag = "<ol style = 'list-style-type: upper-roman'>";
34             endTag = "</ol>";
35             listType = "<h1>Roman Numbered List</h1>";
36             break;
37         default:
38             validInput = false;
39             break;
40     } //end switch
41
```

Fig. 8.7 | Using the `switch` multiple-selection statement. (Part 2 of 6.)



```
42     if ( validInput === true )
43     {
44         document.writeln( listType + startTag );
45
46         for ( var i = 1; i <= 3; ++i )
47             document.writeln( "<li>List item " + i + "</li>" );
48
49         document.writeln( endTag );
50     } //end if
51     else
52         document.writeln( "Invalid choice: " + choice );
53
54     </script>
55     </head><body></body>
56 </html>
```

Fig. 8.7 | Using the switch multiple-selection statement. (Part 3 of 6.)

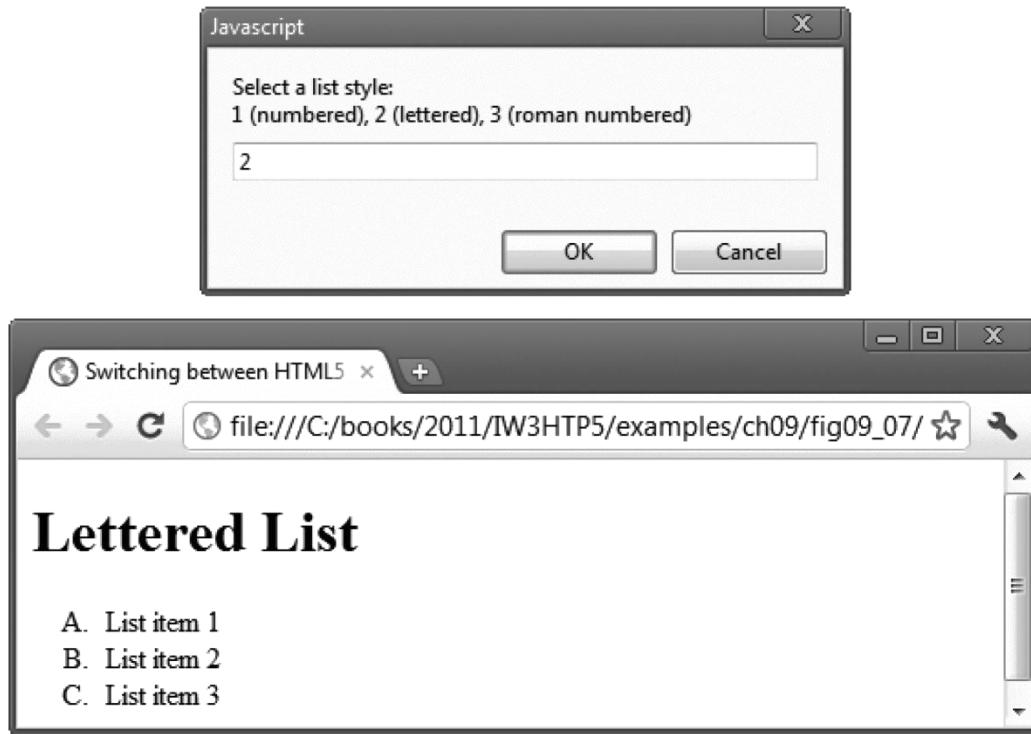


Fig. 8.7 | Using the switch multiple-selection statement. (Part 4 of 6.)

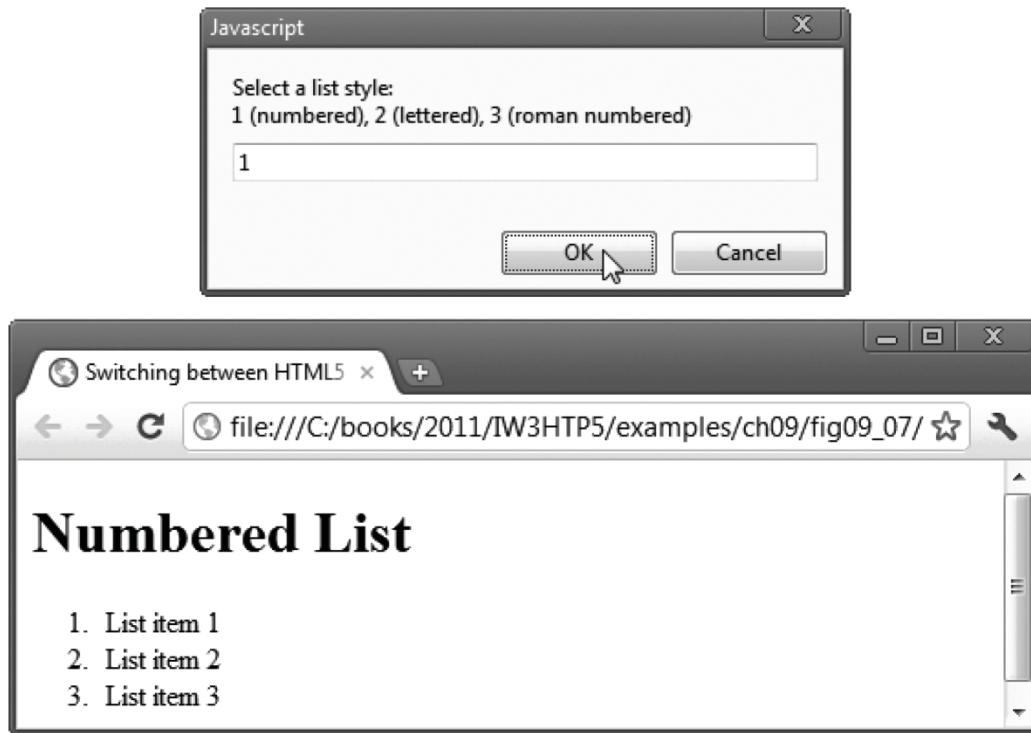


Fig. 8.7 | Using the switch multiple-selection statement. (Part 5 of 6.)

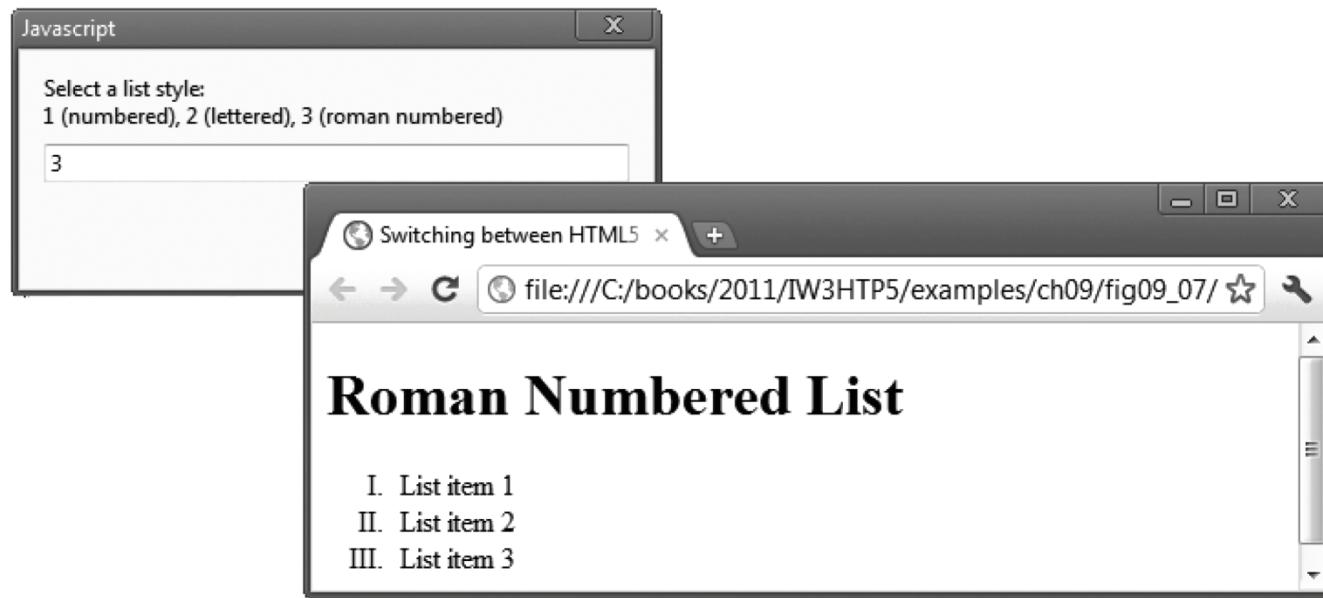


Fig. 8.7 | Using the switch multiple-selection statement. (Part 6 of 6.)

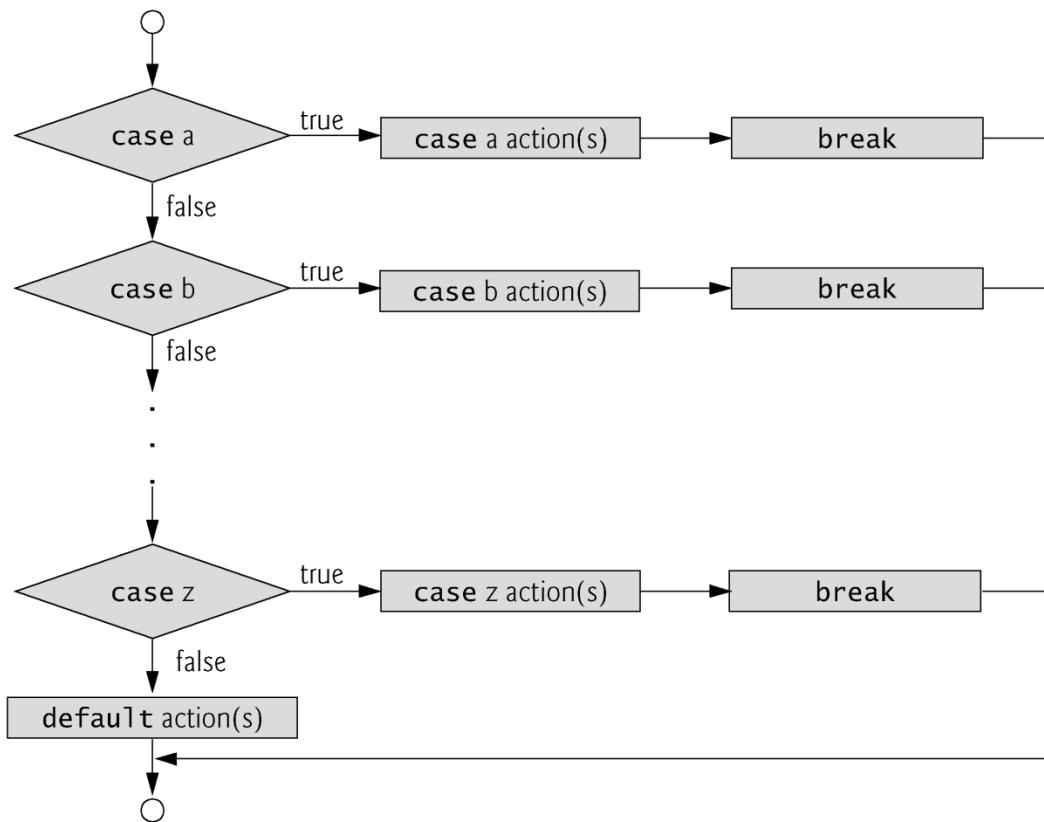


Fig. 8.8 | switch multiple-selection statement.



8.6 do...while Repetition Statement

- ▶ do...while statement
 - tests the loop-continuation condition *after* the loop body executes
 - *The loop body always executes at least once*



```
1  <!DOCTYPE html>
2
3  <!-- Fig. 8.9: DoWhileTest.html -->
4  <!-- Using the do...while repetition statement. -->
5  <html>
6      <head>
7          <meta charset = "utf-8">
8          <title>Using the do...while Repetition Statement</title>
9          <script>
10
11             var counter = 1;
12
13         do {
14             document.writeln( "<h" + counter + ">This is " +
15                 "an h" + counter + " level head" + "</h" +
16                     counter + ">" );
17             ++counter;
18         } while ( counter <= 6 );
19
20     </script>
21
22     </head><body></body>
23 </html>
```

Fig. 8.9 | Using the do...while repetition statement. (Part 1 of 2.)

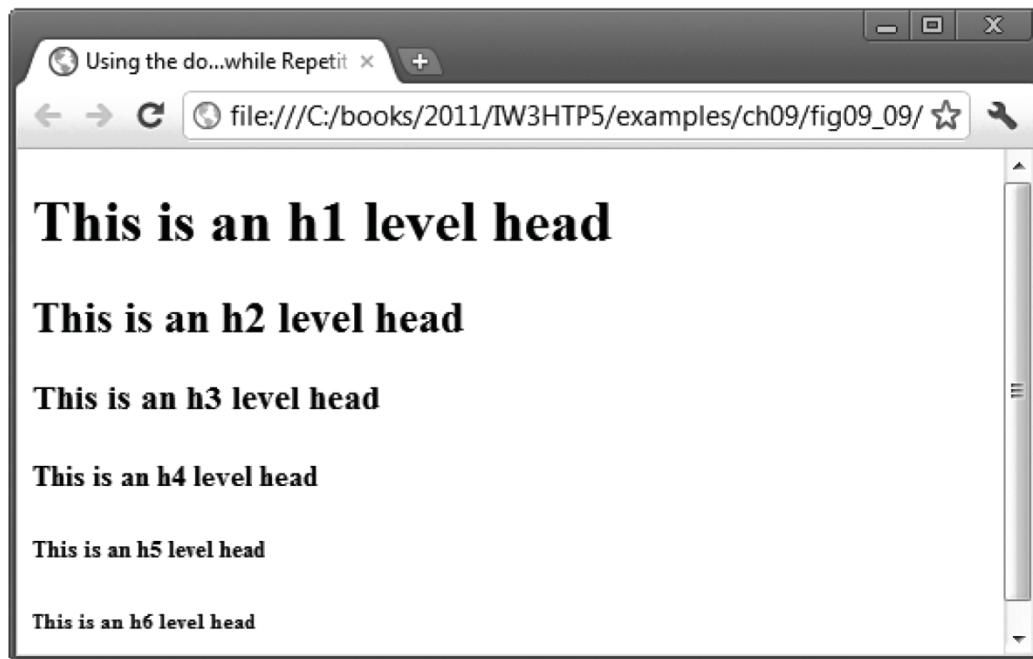


Fig. 8.9 | Using the do...while repetition statement. (Part 2 of 2.)

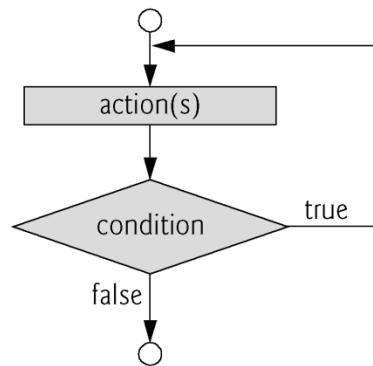


Fig. 8.10 | do...while repetition statement flowchart.



Common Programming Error 8.2

Infinite loops are caused when the loop-continuation condition never becomes `false` in a `while`, `for` or `do...while` statement. To prevent this, make sure that there's not a semicolon immediately after the header of a `while` or `for` statement. In a counter-controlled loop, make sure that the control variable is incremented (or decremented) in the body of the loop. In a sentinel-controlled loop, the sentinel value should eventually be input.



8.7 break and continue Statements

- ▶ break statement in a while, for, do...while or switch statement
 - Causes *immediate exit* from the statement
 - Execution continues with the next statement in sequence
- ▶ break statement common uses
 - Escape early from a loop
 - Skip the remainder of a switch statement



8.7 break and continue Statements (Cont.)

- ▶ continue statement in a while, for or do...while
 - skips the remaining statements in the body of the statement and proceeds with the next iteration of the loop
 - In while and do...while statements, the loop-continuation test evaluates immediately after the continue statement executes
 - In for statements, the increment expression executes, then the loop-continuation test evaluates



```
1 <!DOCTYPE html>
2
3 <!-- Fig. 8.11: BreakTest.html -->
4 <!-- Using the break statement in a for statement. -->
5 <html>
6   <head>
7     <meta charset = "utf-8">
8     <title>
9       Using the break Statement in a for Statement
10    </title>
11    <script>
12
13      for ( var count = 1; count <= 10; ++count )
14      {
15        if ( count == 5 )
16          break; // break loop only if count == 5
17
18        document.writeln( count + " " );
19      } //end for
20
```

Fig. 8.11 | Using the break statement in a for statement. (Part 1 of 2.)

```
21     document.writeln(  
22         "<p>Broke out of loop at count = " + count + "</p>" );  
23  
24     </script>  
25 </head><body></body>  
26 </html>
```

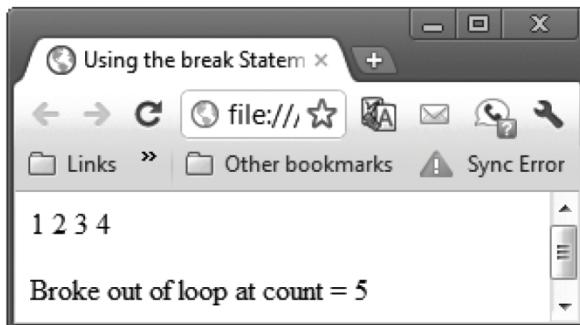


Fig. 8.11 | Using the break statement in a for statement. (Part 2 of 2.)



```
1 <!DOCTYPE html>
2
3 <!-- Fig. 8.12: ContinueTest.html -->
4 <!-- Using the continue statement in a for statement. -->
5 <html>
6   <head>
7     <meta charset = "utf-8">
8     <title>
9       Using the continue Statement in a for Statement
10    </title>
11
12   <script>
13
14     for ( var count = 1; count <= 10; ++count )
15     {
16       if ( count == 5 )
17         continue; // skip remaining loop code only if count == 5
18
19       document.writeln( count + " " );
20     } //end for
21
```

Fig. 8.12 | Using the continue statement in a for statement. (Part 1 of 2.)

```
22     document.writeln( "<p>Used continue to skip printing 5</p>" );
23
24 </script>
25
26 </head><body></body>
27 </html>
```

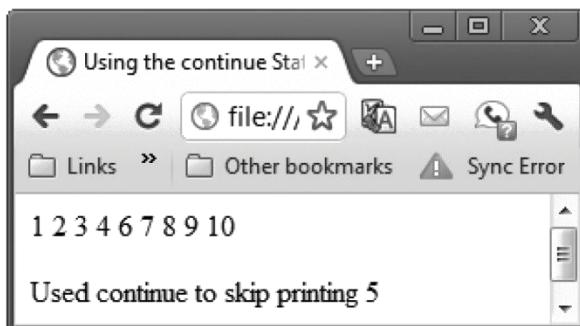


Fig. 8.12 | Using the `continue` statement in a `for` statement. (Part 2 of 2.)



8.8 Logical Operators

- ▶ Logical operators can be used to form complex conditions by combining simple conditions
 - `&&` (logical AND)
 - `||` (logical OR)
 - `!` (logical NOT, also called logical negation)
- ▶ The `&&` operator is used to ensure that two conditions are both true before choosing a certain path of execution
- ▶ JavaScript evaluates to `false` or `true` all expressions that include relational operators, equality operators and/or logical operators



expression1	expression2	expression1 && expression2
false	false	false
false	true	false
true	false	false
true	true	true

Fig. 8.13 | Truth table for the **&&** (logical AND) operator.



8.8 Logical Operators (Cont.)

- ▶ The `||` (logical OR) operator is used to ensure that either or both of two conditions are true before choosing choose a certain path of execution



expression1	expression2	expression1 expression2
false	false	false
false	true	true
true	false	true
true	true	true

Fig. 8.14 | Truth table for the `||` (logical OR) operator.



8.8 Logical Operators (Cont.)

- ▶ The `&&` operator has a higher precedence than the `||` operator
- ▶ Both operators associate from left to right.
- ▶ An expression containing `&&` or `||` operators is evaluated only until truth or falsity is known
 - This is called short-circuit evaluation



8.8 Logical Operators (Cont.)

- ▶ ! (logical negation) operator
 - reverses the meaning of a condition (i.e., a true value becomes false, and a false value becomes true)
 - Has only a single condition as an operand (i.e., it is a unary operator)
 - Placed before a condition to evaluate to true if the original condition (without the logical negation operator) is false

expression	<code>!expression</code>
false	true
true	false

Fig. 8.15 | Truth table for operator `!` (logical negation).



8.9 Logical Operators (Cont.)

- ▶ Most nonboolean values can be converted to a boolean true or false value
- ▶ Nonzero numeric values are considered to be true
- ▶ The numeric value zero is considered to be false
- ▶ Any string that contains characters is considered to be true
- ▶ The empty string is considered to be false
- ▶ The value null and variables that have been declared but not initialized are considered to be false
- ▶ All objects are considered to be true

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

Fig. 8.16 | Precedence and associativity of the operators discussed so far.