JavaScript: Control Statements

OBJECTIVES

In this chapter you will:

- Learn basic problem-solving techniques.
- Develop algorithms through the process of top-down, stepwise refinement.
- Use the `if` and `if...else` selection statements to choose among alternative actions.
- Use the `while` repetition statement to execute statements in a script repeatedly.
- Implement counter-controlled repetition and sentinel-controlled repetition.
- Use the increment, decrement and assignment operators.

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.

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.

7.6 if...else Selection Statement

- Allows you to specify that different actions should be performed when the condition is true and when the condition is false.

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

7.7 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

7.8 Formulating Algorithms: Counter-Controlled Repetition

- Counter-controlled repetition
  - Often called definite repetition, because the number of repetitions is known before the loop begins executing
- A *total* is a variable in which a script accumulates the sum of a series of values
  - Variables that store totals should normally be initialized to zero before they are used in a script
- A *counter* is a variable a script uses to count—typically in a repetition statement

```
1  Set total to zero
2  Set grade counter to one
3
4  While grade counter is less than or equal to ten
5      Input the next grade
6      Add the grade into the total
7      Add one to the grade counter
8
9  Set the class average to the total divided by ten
10 Print the class average
```

Fig. 7.6 | Pseudocode algorithm that uses counter-controlled repetition to solve the class-average problem.

```
1  <!DOCTYPE html>
2
3  <!-- Fig. 7.7: average.html -->
4  <!-- Counter-controlled repetition to calculate a class average. -->
5  <html>
6     <head>
7        <meta charset = "utf-8">
8        <title>Class Average Program</title>
9        <script>
10
11          var total; // sum of grades
12          var gradeCounter; // number of grades entered
13          var grade; // grade typed by user (as a string)
14          var gradeValue; // grade value (converted to integer)
15          var average; // average of all grades
16
17          // initialization phase
18          total = 0; // clear total
19          gradeCounter = 1; // prepare to loop
20
```

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

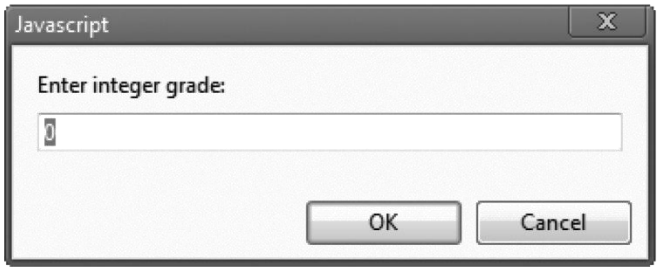
```
21          // processing phase
22          while ( gradeCounter <= 10 ) // loop 10 times
23          {
24
25             // prompt for input and read grade from user
26             grade = window.prompt( "Enter integer grade:", "0" );
27
28             // convert grade from a string to an integer
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.)

Not initializing a variable that will be used in a calculation results in a logic error that produces the value NaN—Not a Number. You must initialize the variable before it is used in a calculation.

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

If the string passed to `parseInt` contains a floating-point numeric value, `parseInt` simply truncates the floating-point part.

For example, the string "27.95" results in the integer 27, and the string "-123.45" results in the integer -123.

If the string passed to `parseInt` is not a numeric value, `parseInt` returns `NaN` (not a number).

How about "123abcde"?

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  <!DOCTYPE html>
2
3  <!-- Fig. 7.9: average2.html -->
4  <!-- Sentinel-controlled repetition to calculate a class average. -->
5  <html>
6     <head>
7        <meta charset = "utf-8">
8        <title>Class Average Program: Sentinel-controlled Repetition</title>
9        <script>
10
11          var total; // sum of grades
12          var gradeCounter; // number of grades entered
13          var grade; // grade typed by user (as a string)
14          var gradeValue; // grade value (converted to integer)
15          var average; // average of all grades
16
17          // initialization phase
18          total = 0; // clear total
19          gradeCounter = 0; // prepare to loop
20
```

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.)

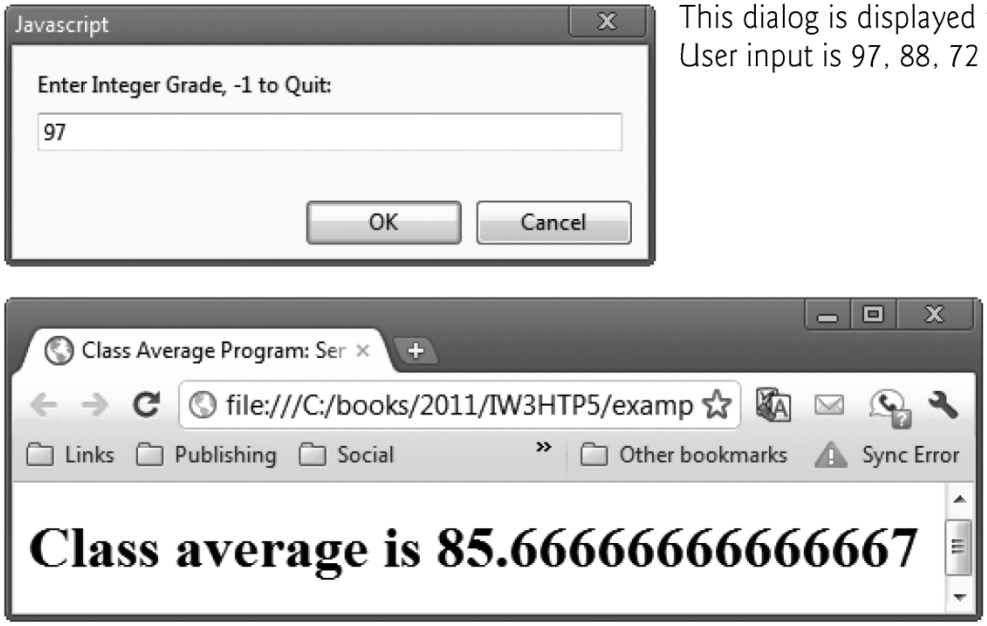


Fig. 7.9 | Sentinel-controlled repetition to calculate a class average. (Part 4 of 4.)

7.11 Assignment Operators

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

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	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.

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

8.3 for Repetition Statement

- 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, and 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.

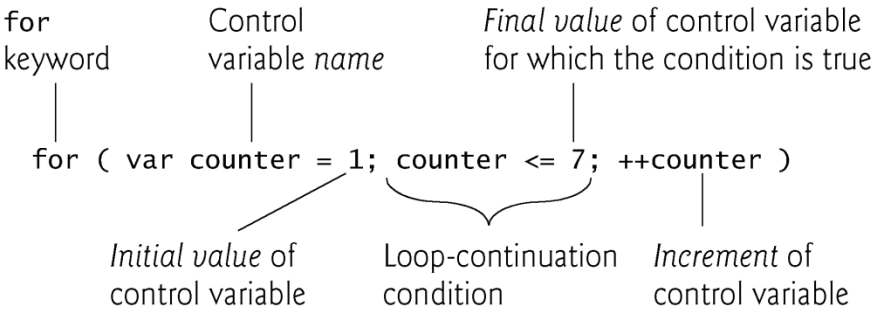


Fig. 8.3 | `for` statement header components.

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

```
<!DOCTYPE html>

<!-- Fig. 8.6: Interest.html -->
<!-- Compound interest calculation with a for loop. -->
<html>
   <head>
      <meta charset = "utf-8">
      <title>Calculating Compound Interest</title>
      <style type = "text/css">
         table         { width: 300px;
                         border-collapse: collapse;
                         background-color: lightblue; }
         table, td, th { border: 1px solid black;
                         padding: 4px; }
         th            { text-align: left;
                         color: white;
                         background-color: darkblue; }
         tr.oddrow     { background-color: white; }
      </style>
```

Fig. 8.6 | Compound interest calculation with a for loop. (Part 1 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.)

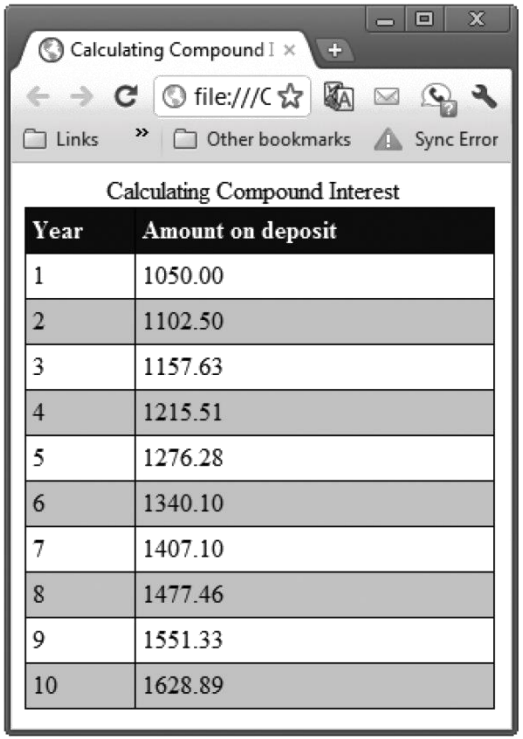


Fig. 8.6 | Compound interest calculation with a `for` loop. (Part 4 of 4.)

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

```
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 1 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.)

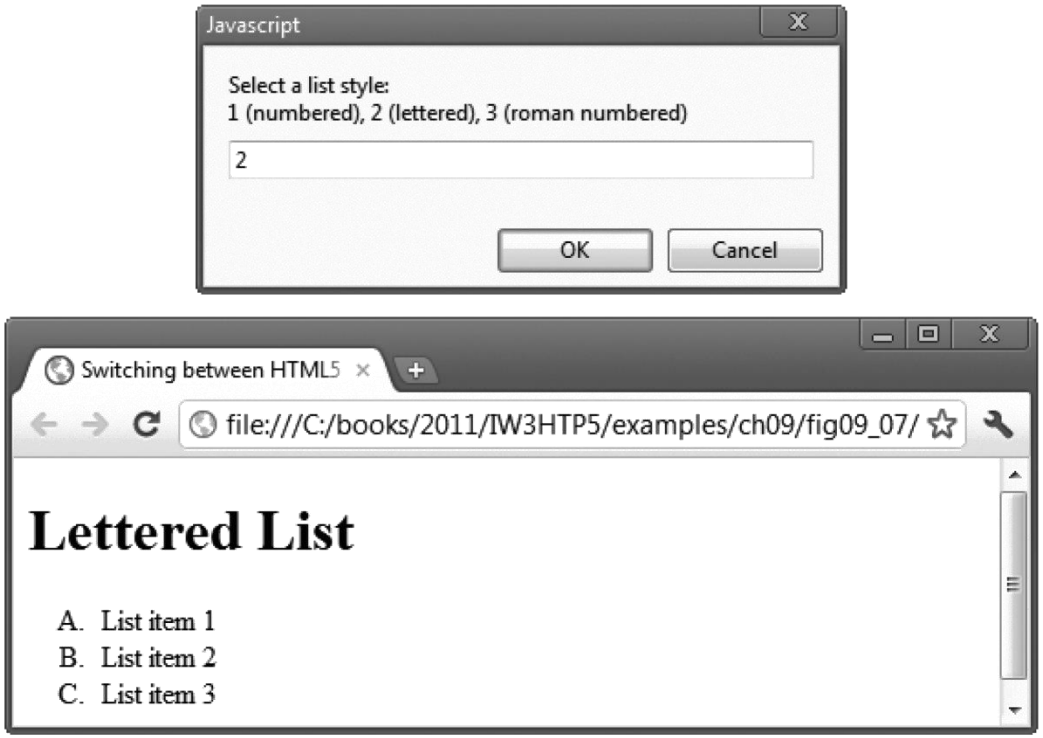


Fig. 8.7 | Using the `switch` multiple-selection statement. (Part 4 of 6.)

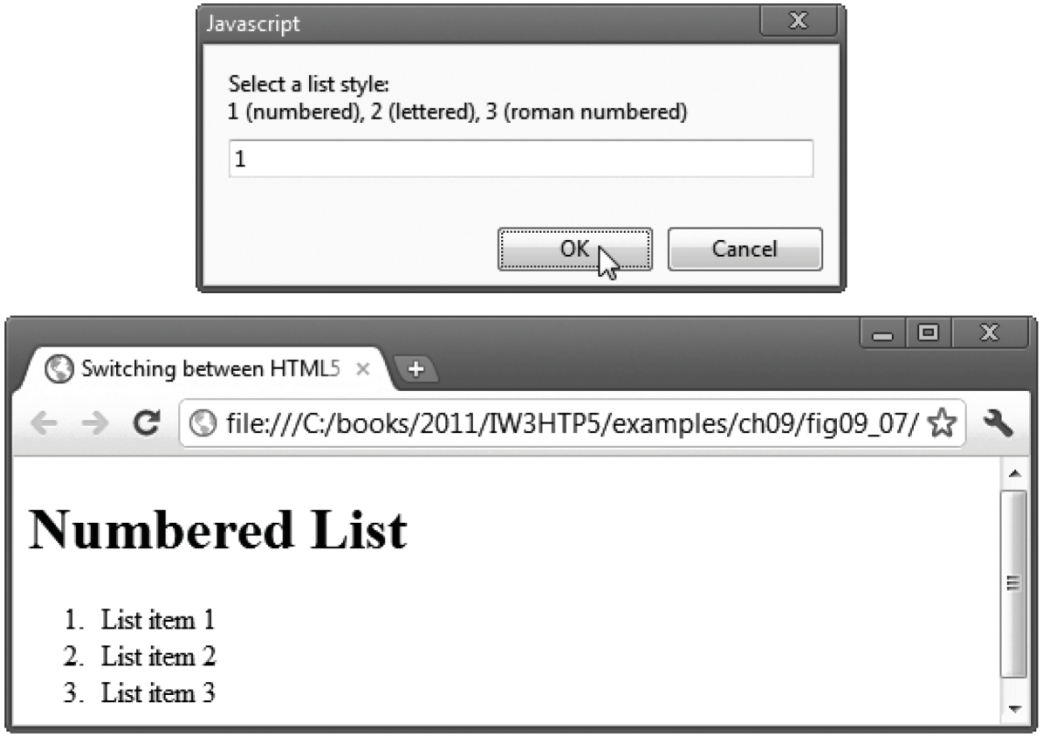


Fig. 8.7 | Using the `switch` multiple-selection statement. (Part 5 of 6.)

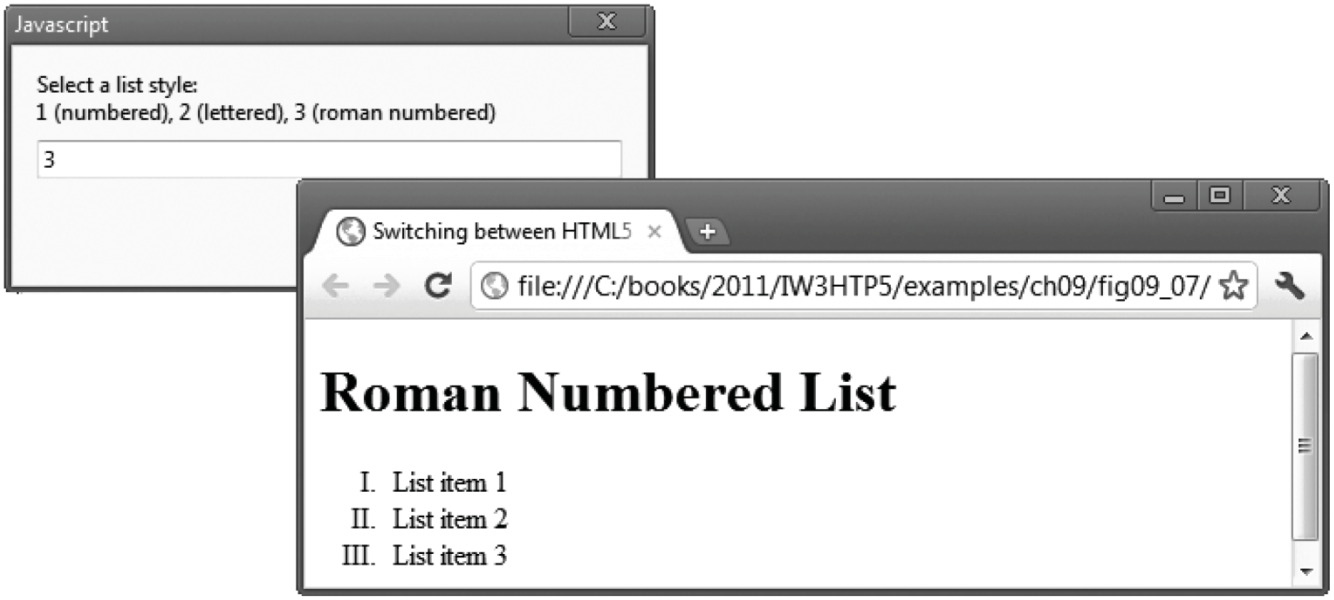


Fig. 8.7 | Using the `switch` multiple-selection statement. (Part 6 of 6.)

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>
```

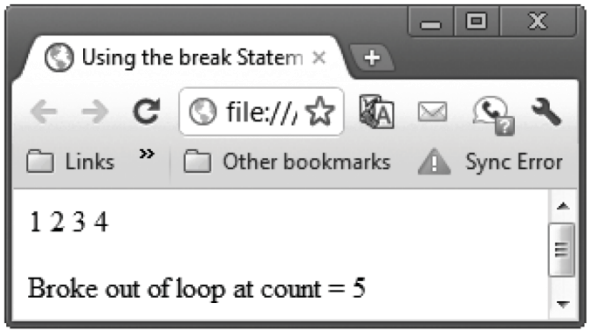


Fig. 8.11 | Using the `break` statement in a `for` statement. (Part 2 of 2.)

```
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>
```

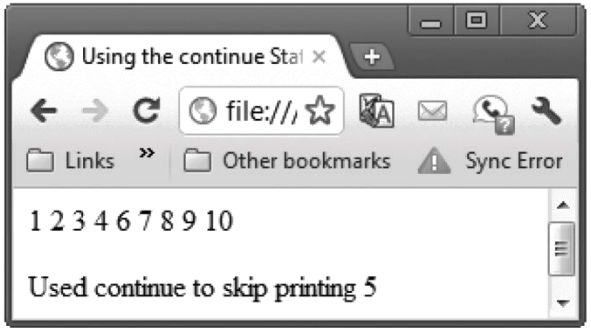


Fig. 8.12 | Using the `continue` statement in a `for` statement. (Part 2 of 2.)

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`

Conversion from Non-boolean values to Boolean Values

- 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`

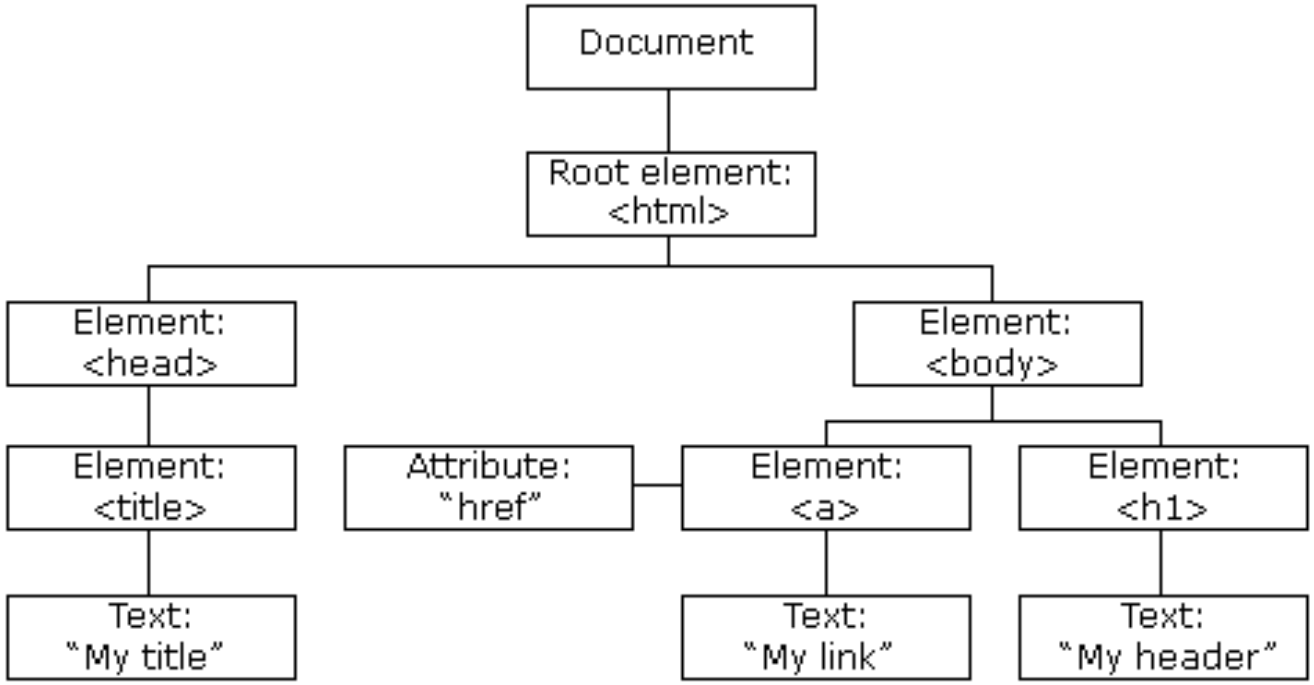
可以讓變數直接作為if的condition！

Logical operators

Logical operators		
Operator	Usage	Description
Logical AND (&&)	expr1 && expr2	Returns expr1 if it can be converted to false; otherwise, returns expr2. Thus, when used with Boolean values, && returns true if both operands are true; otherwise, returns false.
Logical OR (||)	expr1 || expr2	Returns expr1 if it can be converted to true; otherwise, returns expr2. Thus, when used with Boolean values, || returns true if either operand is true; if both are false, returns false.
Logical NOT (!)	!expr	Returns false if its single operand that can be converted to true; otherwise, returns true.

DOM (Document Object Model)1

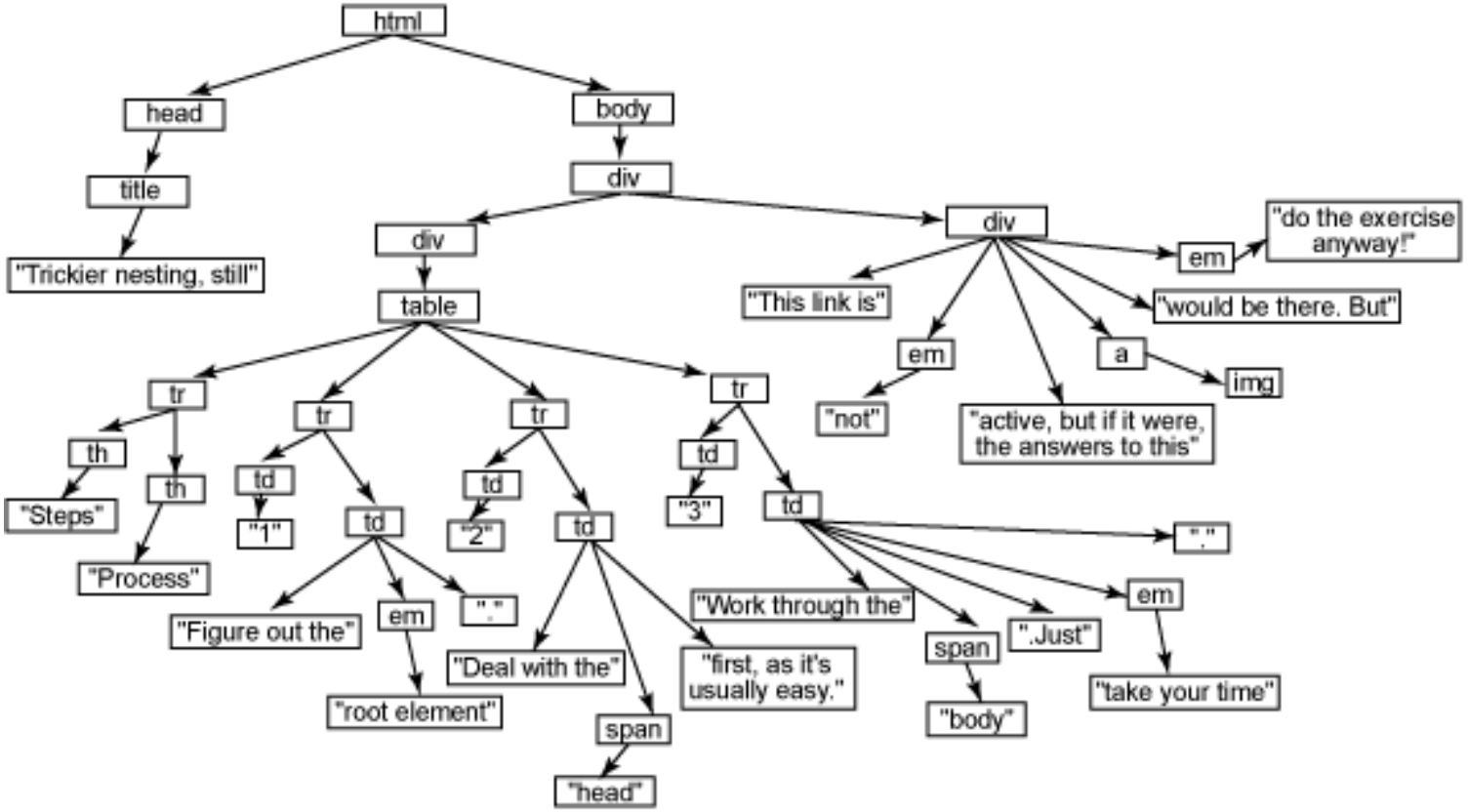
- When a web page is loaded, the browser creates a Document Object Model of the page.
- The HTML DOM model is constructed as a tree of Objects. (樹狀結構的一堆物件)



http://www.w3schools.com/js/js_htmldom.asp

DOM (Document Object Model)2

- A DOM Example for a Page



https://vinaytech.files.wordpress.com/2008/11/domimage.png

在HTML元素上動態顯示訊息1

- 我們若要只改動整張網頁的一個小區塊，我們先要取得那個區塊的元素(element)的物件(object)
  - 這個物件就是文件物件(DOM object)
  - 要能取得某元素的物件，這個物件要先幫他取個id
    - id: 身分識別(identification)，類似身份證字號的概念，是不重複的、唯一的名稱。
    - id只要跟一般變數名稱一樣的風格即可，重點是不要重複。
  - 要用**document.getElementById(id)**取得物件
  - 設定的區塊通常是 元素或<p>元素
    - 用**objectName.innerHTML = "..."** 去動態設定它的內容(請看後續範例)。

在HTML元素上動態顯示訊息2

- 我們可以在元素內**動態**塞入純文字

```
<h2>My First Web Page</h2>
<p>My First Paragraph.</p>
<p id="demo"></p>

<script>
    let area = document.getElementById("demo");
    area.innerHTML = 5 + 6;
</script>
```

My First Web Page

My First Paragraph.

11

https://www.w3schools.com/code/tryit.asp?filename=GR8RQX3COZAO

在HTML元素上動態顯示訊息3

- 也可以在元素內**動態塞入HTML片段**

```
<body>
    <h2>Cute Baby</h2>
    <div id="demo"></div>
    <script>
        let choice = window.prompt("請選擇(1)男孩(2)女孩:");
        if (choice == 1) {
            document.getElementById("demo").innerHTML =
            "<img src = 'http://cdn2.momjunction.com/wp-
content/uploads/2015/02/Hispanic-Baby-Names-For-Your-Little-Boy.jpg'>";
        }
        else if (choice == 2) {
            document.getElementById("demo").innerHTML =
            "<img src = 'http://cdn2.momjunction.com/wp-
content/uploads/2015/04/Top-188-Latest-And-Modern-Hindu-Baby-Girl-Names.jpg'>";
        }
    </script>
</body>
```

https://www.w3schools.com/code/tryit.asp?filen