Chapter 7 JavaScript: Control Statements, Part 1

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

7.4 Control Statements

- Sequential execution
  - Execute statements in the order they appear in the code
- Transfer of control
  - Changing the order in which statements execute
- All scripts can be written in terms of only three control statements
  - sequence
  - selection
  - repetition

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.

Common Programming Error 7.1

Using a keyword as an identifier (e.g., for variable names) is a syntax error.

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`

Example

- The following statement contains a conditional expression that evaluates to the string "Passed" if the condition studentGrade >= 60 is true and evaluates to the string "Failed" if the condition is false.

- document.writeln(studentGrade >= 60 ? "Passed" : "Failed");

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

Dangling-else Problem

- It’s important to note that the JavaScript interpreter always associates an else with the previous if, unless told to do otherwise by the placement of braces ({}).

Dangling-else Problem

- This statement tests whether x is greater than 5. If so, execution continues by testing whether y is also greater than 5. If the second condition is true, the proper string—"x and y are > 5"—is displayed. However, if the second condition is false, the string "x is <= 5" is displayed, even though we know that x is greater than 5.

```
if ( x > 5 )
  if ( y > 5 )
    document.writeln( "<p>x and y are > 5</p>" );
  else
    document.writeln( "<p>x is <= 5</p>" );
```

Dangling-else Problem

- To force the *first* nested if statement to execute as it was intended originally, we must write it as follows:

```
if ( x > 5 )
{
  if ( y > 5 )
    document.writeln( "<p>x and y are > 5</p>" );
}
else
  document.writeln( "<p>x is <= 5</p>" );
```

- The braces ({}) indicate to the JavaScript interpreter that the second if statement is in the body of the first if statement and that the else is matched with the first if statement.

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.

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.)

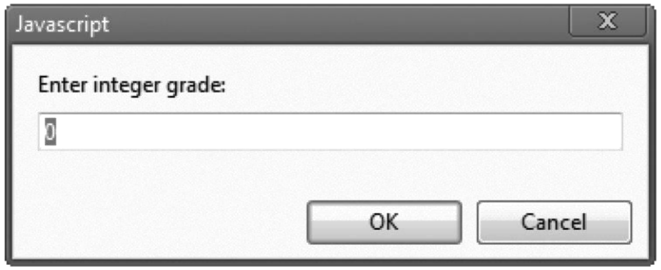
```
      // processing phase
      while ( gradeCounter <= 10 ) // loop 10 times
      {

         // prompt for input and read grade from user
         grade = window.prompt( "Enter integer grade:", "0" );

         // convert grade from a string to an integer
         gradeValue = parseInt( grade );

         // add gradeValue to total
         total = total + gradeValue;

         // add 1 to gradeCounter
         gradeCounter = gradeCounter + 1;
      } // end while

```

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

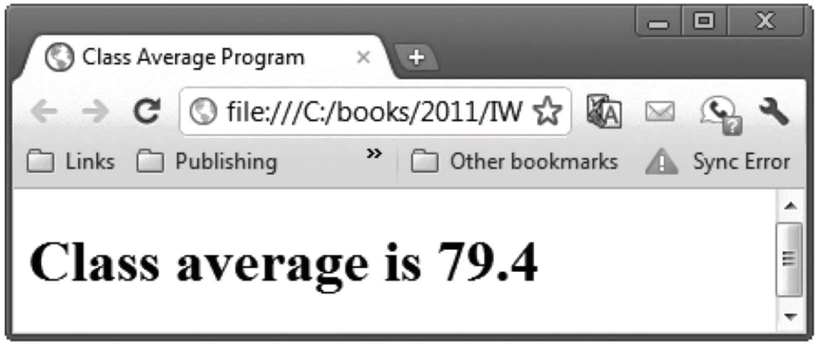
```
            // termination phase
            average = total / 10;   // calculate the average

            // display average of exam grades
            document.writeln(
               "<h1>Class average is " + average + "</h1>" );

         </script>
      </head><body></body>
   </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



Class average is 79.4

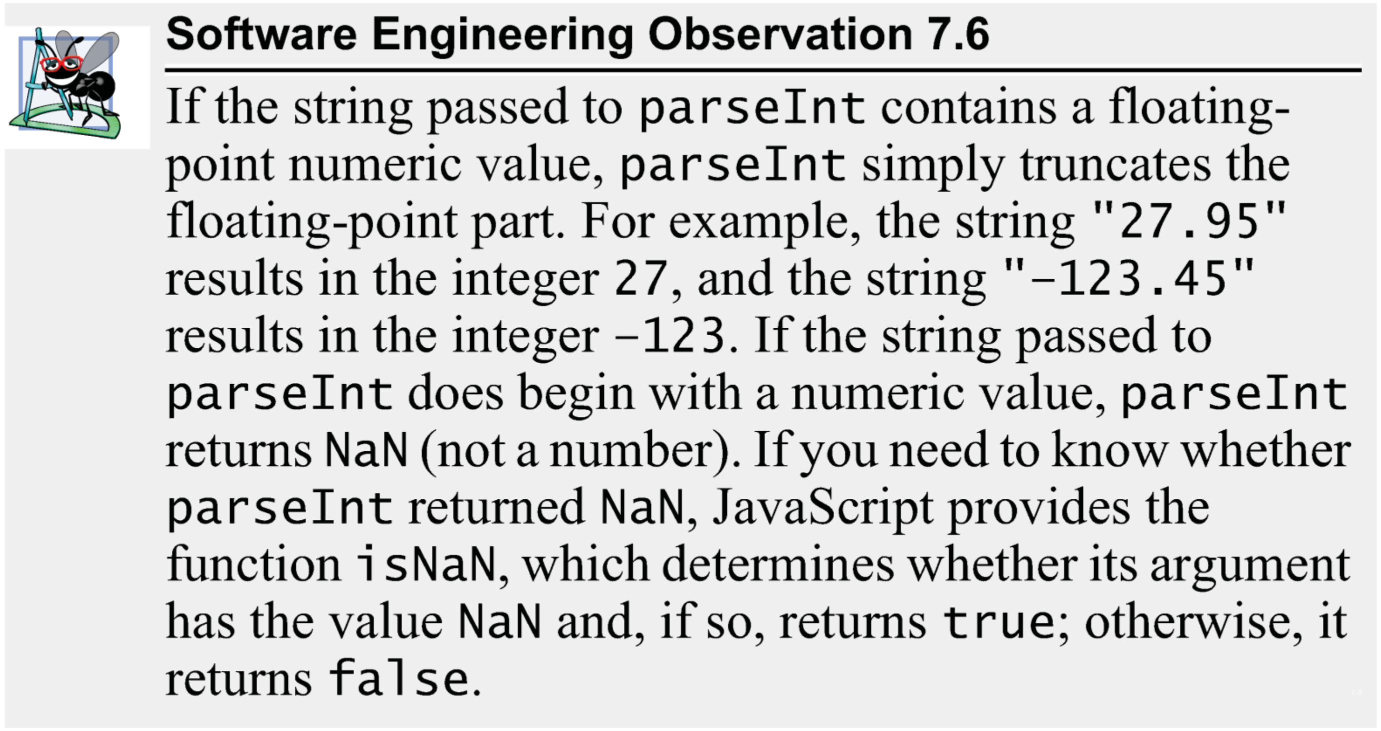
Fig. 7.7 | Counter-controlled repetition to calculate a class average. (Part 4 of 4.)

Common Programming Error 7.3

Not initializing a variable that will be used in a calculation results in a logic error that produces the value `NaN` ("Not a Number").

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



Software Engineering Observation 7.6

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` does begin with a numeric value, `parseInt` returns `NaN` (not a number). If you need to know whether `parseInt` returned `NaN`, JavaScript provides the function `isNaN`, which determines whether its argument has the value `NaN` and, if so, returns `true`; otherwise, it returns `false`.

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

```
<!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

```

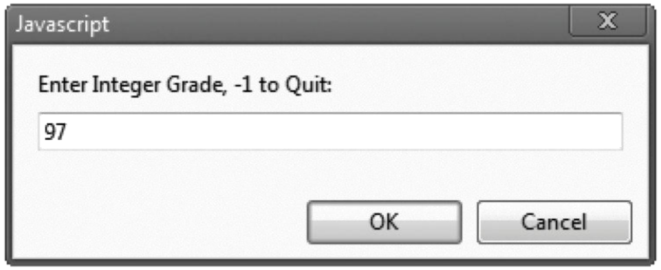
Fig. 7.9 | Sentinel-controlled repetition to calculate a class average. (Part 1 of 4.)

```
         // processing phase
         // prompt for input and read grade from user
         grade = window.prompt(
              "Enter Integer Grade, -1 to Quit:", "0" );

         // convert grade from a string to an integer
         gradeValue = parseInt( grade );

         while ( gradeValue != -1 )
         {
            // add gradeValue to total
            total = total + gradeValue;

            // add 1 to gradeCounter
            gradeCounter = gradeCounter + 1;

            // prompt for input and read grade from user
            grade = window.prompt(
                 "Enter Integer Grade, -1 to Quit:", "0" );

            // convert grade from a string to an integer
            gradeValue = parseInt( grade );
         } // end while
```

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

```

         // termination phase
         if ( gradeCounter != 0 )
         {
            average = total / gradeCounter;

            // display average of exam grades
            document.writeln(
               "<h1>Class average is " + average + "</h1>" );
         } // end if
         else
            document.writeln( "<p>No grades were entered</p>" );

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

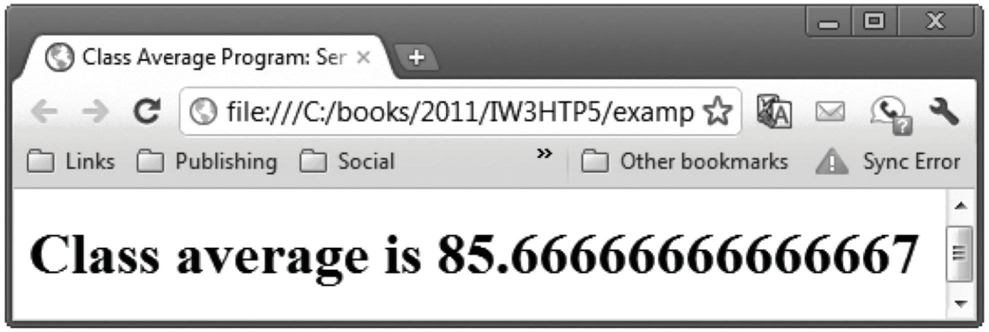
Fig. 7.9 | Sentinel-controlled repetition to calculate a class average. (Part 3 of 4.)



Enter Integer Grade, -1 to Quit:

97

This dialog is displayed four times. User input is 97, 88, 72 and –1.



file:///C:/books/2011/IW3HTP5/examp

Class average is 85.66666666666667

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

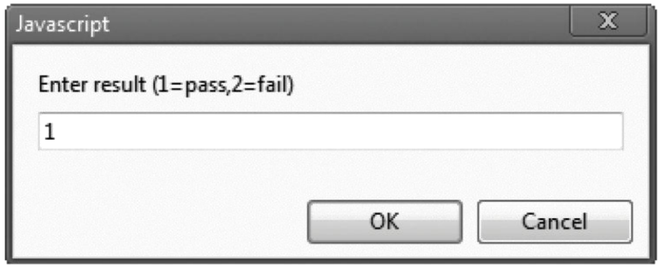
```
<!DOCTYPE html>

<!-- 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.)

```
            // process 10 students; counter-controlled loop
            while ( student <= 10 )
            {
               result = window.prompt( "Enter result (1=pass,2=fail)", "0" );

               if ( result == "1" )
                  passes = passes + 1;
               else
                  failures = failures + 1;

               student = student + 1;
            } // end while

            // termination phase
            document.writeln( "<h1>Examination Results</h1>" );
            document.writeln( "<p>Passed: " + passes +
               "; Failed: " + failures + "</p>" );

            if ( passes > 8 )
               document.writeln( "<p>Bonus to instructor!</p>" );

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

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

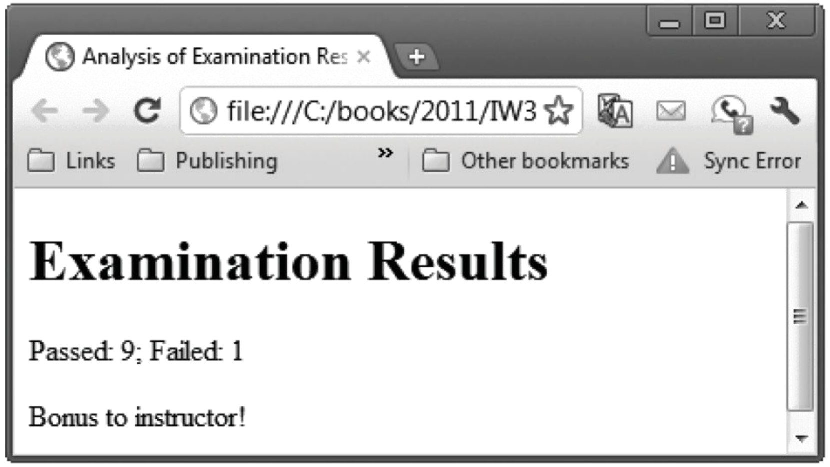
a) This dialog is displayed 10 times. User input is 1, 2, 1, 1, 1, 1, 1, 1, 1 and 1.



Enter result (1=pass,2=fail)

1

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



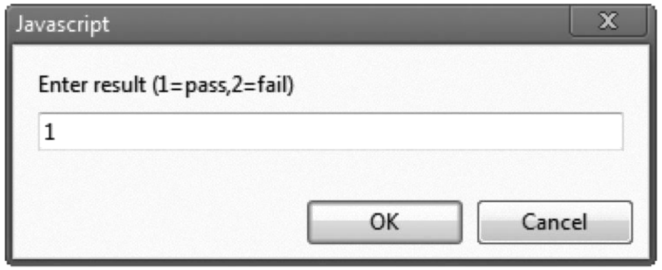
Examination Results

Passed: 9; Failed: 1

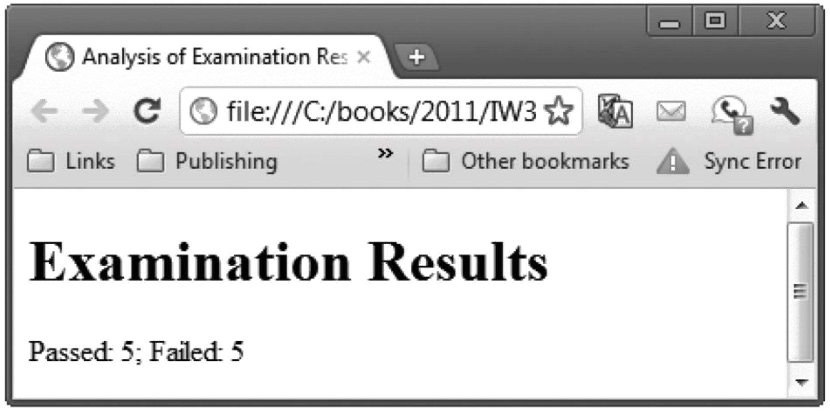
Bonus to instructor!

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.



Examination Results

Passed: 5; Failed: 5

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

7.11 Assignment Operators

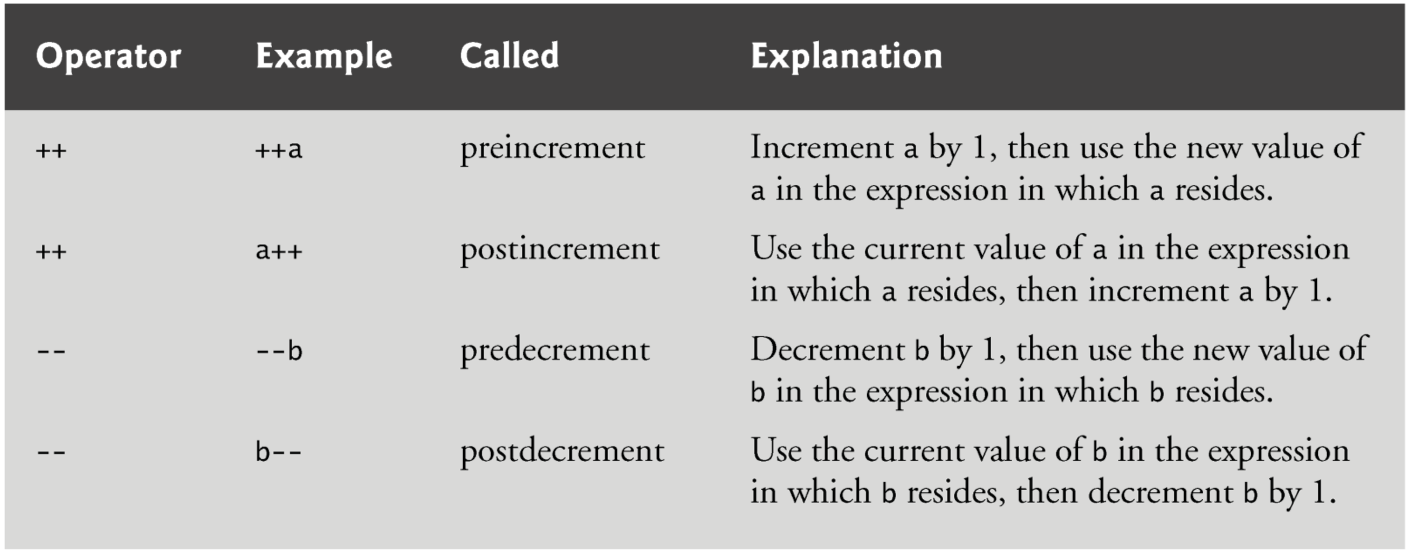
- JavaScript provides the arithmetic assignment operators +=, -=, *=, /= and %=, which abbreviate certain common types of expressions.

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.



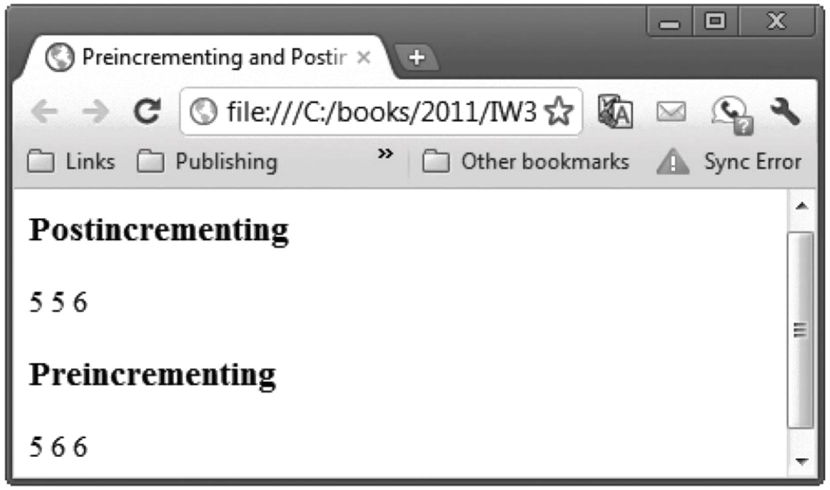
Operator	Example	Called	Explanation
++	++a	preincrement	Increment a by 1, then use the new value of a in the expression in which a resides.
++	a++	postincrement	Use the current value of a in the expression in which a resides, then increment a by 1.
--	--b	predecrement	Decrement b by 1, then use the new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

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

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



Postincrementing

5 5 6

Preincrementing

5 6 6

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

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.