JavaScript: Control Statements I
Let’s all move one place on.
—Lewis Carroll

The wheel is come full circle.
—William Shakespeare

How many apples fell on Newton’s head before he took the hint!
—Robert Frost
OBJECTIVES

In this chapter you will learn:

- Basic problem-solving techniques.
- To develop algorithms through the process of top-down, stepwise refinement.
- To use the if and if else selection statements to choose among alternative actions.
- To use the while repetition statement to execute statements in a script repeatedly.
- Counter-controlled repetition and sentinel-controlled repetition.
- To use the increment, decrement and assignment operators.
7.1 Introduction
7.2 Algorithms
7.3 Pseudocode
7.4 Control Structures
7.5 if Selection Statement
7.6 if else Selection Statement
7.7 while Repetition Statement
7.8 Formulating Algorithms: Counter-Controlled Repetition
7.9 Formulating Algorithms: Sentinel-Controlled Repetition
7.10 Formulating Algorithms: Nested Control Statements
7.11 Assignment Operators
7.12 Increment and Decrement Operators
7.13 Wrap-Up
7.14 Web Resources
7.1 Introduction

• The techniques you will learn here are applicable to most high-level languages, including JavaScript.
7.2 Algorithms

• Any computable problem can be solved by executing a series of actions in a specific order.

• A procedure for solving a problem in terms of the actions to execute and the order in which the actions are to execute is called an algorithm.

• Specifying the order in which statements are to be executed in a computer program is called program control.
7.3 Pseudocode

- Pseudocode is an artificial and informal language that helps programmers develop algorithms.
- Carefully prepared pseudocode may be converted easily to a corresponding JavaScript program.
- Pseudocode normally describes only executable statements—the actions that are performed when the program is converted from pseudocode to JavaScript and executed.
Software Engineering Observation 7.1

Pseudocode is often used to “think out” a program during the program-design process. Then the pseudocode program is converted to a programming language such as JavaScript.
7.4 Control Structures

• Normally, statements in a program execute one after the other, in the order in which they are written. This process is called sequential execution.

• Various JavaScript statements enable the programmer to specify that the next statement to be executed may be other than the next one in sequence. This process is called transfer of control.

• All programs can be written in terms of only three control structures, namely, the sequence structure, the selection structure and the repetition structure.
7.4 Control Structures (Cont.)

• A flowchart is a graphical representation of an algorithm or of a portion of an algorithm. Flowcharts are drawn using certain special-purpose symbols, such as rectangles, diamonds, ovals and small circles; these symbols are connected by arrows called flowlines, which indicate the order in which the actions of the algorithm execute.
7.4 Control Structures (Cont.)

• In a flowchart that represents a complete algorithm, an oval symbol containing the word “Begin” is the first symbol used; an oval symbol containing the word “End” indicates where the algorithm ends.

• In a flowchart that shows only a portion of an algorithm, the oval symbols are omitted in favor of using small circle symbols, also called connector symbols.

• Perhaps the most important flowcharting symbol is the diamond symbol, also called the decision symbol, which indicates that a decision is to be made.
Fig. 7.1 | Flowcharting JavaScript's sequence structure.

```javascript
// Increment the counter
counter = counter + 1;

// Update the total
total = total + grade;
```

Indicates a portion of an algorithm
7.4 Control Structures (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 structure 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 structure 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 structure because it selects among many different actions or groups of actions.
7.4 Control Structures (Cont.)

- JavaScript provides four repetition statements, namely, *while*, *do...while*, *for* and *for...in*.
- Keywords cannot be used as identifiers (e.g., for variable names).
Common Programming Error 7.1

Using a keyword as an identifier is a syntax error.
### JavaScript Keywords

<table>
<thead>
<tr>
<th>break</th>
<th>case</th>
<th>catch</th>
<th>continue</th>
<th>default</th>
</tr>
</thead>
<tbody>
<tr>
<td>delete</td>
<td>do</td>
<td>else</td>
<td>false</td>
<td>finally</td>
</tr>
<tr>
<td>for</td>
<td>function</td>
<td>if</td>
<td>in</td>
<td>instanceof</td>
</tr>
<tr>
<td>new</td>
<td>null</td>
<td>return</td>
<td>switch</td>
<td>this</td>
</tr>
<tr>
<td>throw</td>
<td>true</td>
<td>try</td>
<td>typeof</td>
<td>var</td>
</tr>
<tr>
<td>void</td>
<td>while</td>
<td>with</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*Keywords that are reserved but not used by JavaScript*

<table>
<thead>
<tr>
<th>abstract</th>
<th>boolean</th>
<th>byte</th>
<th>char</th>
<th>class</th>
</tr>
</thead>
<tbody>
<tr>
<td>const</td>
<td>debugger</td>
<td>double</td>
<td>enum</td>
<td>export</td>
</tr>
<tr>
<td>extends</td>
<td>final</td>
<td>float</td>
<td>goto</td>
<td>implements</td>
</tr>
<tr>
<td>import</td>
<td>int</td>
<td>interface</td>
<td>long</td>
<td>native</td>
</tr>
<tr>
<td>package</td>
<td>private</td>
<td>protected</td>
<td>public</td>
<td>short</td>
</tr>
<tr>
<td>static</td>
<td>super</td>
<td>synchronized</td>
<td>throws</td>
<td>transient</td>
</tr>
<tr>
<td>volatile</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
7.4 Control Structures (Cont.)

• Single-entry/single-exit control structures make it easy to build programs.

• Control structures are attached to one another by connecting the exit point of one control structure to the entry point of the next. This procedure is called control-structure stacking.

• There is only one other way control structures may be connected: control-structure nesting.
7.5 if Selection Statement

• The JavaScript interpreter ignores white-space characters: blanks, tabs and newlines used for indentation and vertical spacing. Programmers insert white-space characters to enhance program clarity.

• A decision can be made on any expression that evaluates to a value of JavaScript’s boolean type (i.e., any expression that evaluates to true or false).

• The indentation convention you choose should be carefully applied throughout your programs. It is difficult to read programs that do not use uniform spacing conventions.
Good Programming Practice 7.1

Consistently applying reasonable indentation conventions throughout your programs improves program readability. We suggest a fixed-size tab of about 1/4 inch or three spaces per indent.
Fig. 7.3 | Flowcharting the single-selection if statement.

If grade >= 60, execute this statement

grade >= 60

true

print "Passed"

false

Otherwise
Software Engineering Observation 7.2

In JavaScript, any nonzero numeric value in a condition evaluates to `true`, and 0 evaluates to `false`. For strings, any string containing one or more characters evaluates to `true`, and the empty string (the string containing no characters, represented as `""`) evaluates to `false`. Also, a variable that has been declared with `var` but has not been assigned a value evaluates to `false`. 
7.6 if...else Selection Statement

• Allows the programmer to specify that a different action is to be performed when the condition is true than when the condition is false.
Good Programming Practice 7.2

Indent both body statements of an `if...else` statement.
7.6 *if...else* Selection Statement (Cont.)

- The conditional operator (`?:`) is closely related to the `if...else` statement.
- It is 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` and the third is the value for the conditional expression if the boolean expression evaluates to `false`. 
Fig. 7.4 | Flowcharting the double-selection `if else` statement.
Nested `if...else` statements test for multiple cases by placing `if...else` statements inside other `if...else` structures.

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 in the body of an `if` statement, enclose the statements in braces (`{` and `}`). A set of statements contained within a pair of braces is called a block.
Good Programming Practice 7.3

If there are several levels of indentation, each level should be indented the same additional amount of space.
A block can be placed anywhere in a program that a single statement can be placed.
Software Engineering Observation 7.4

Unlike individual statements, a block does not end with a semicolon. However, each statement within the braces of a block should end with a semicolon.
Common Programming Error 7.2

Forgetting one or both of the braces that delimit a block can lead to syntax errors or logic errors.
7.6 if...else Selection Statement (Cont.)

• A logic error has its effect at execution time.
• A fatal logic error causes a program to fail and terminate prematurely.
• A nonfatal logic error allows a program to continue executing, but the program produces incorrect results.
Good Programming Practice 7.4

Some programmers prefer to type the beginning and ending braces of blocks before typing the individual statements within the braces. This helps avoid omitting one or both of the braces.
Software Engineering Observation 7.5

Just as a block can be placed anywhere a single statement can be placed, it is also possible to have no statement at all (the empty statement) in such places. The empty statement is represented by placing a semicolon (;) where a statement would normally be.
Common Programming Error 7.3

Placing a semicolon after the condition in an if structure leads to a logic error in single-selection if structures and a syntax error in double-selection if structures (if the if part contains a nonempty body statement).
7.7 while Repetition Statement

• The while repetition structure allows the programmer 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. At this point, the repetition terminates, and the first pseudocode statement after the repetition structure executes.
Common Programming Error 7.4

If the body of a \texttt{while} statement never causes the while statement’s condition to become \texttt{false}, a logic error occurs. Normally, such a repetition structure will never terminate—an error called an infinite loop. Both Internet Explorer and Firefox show a dialog allowing the user to terminate a script that contains an infinite loop.
Common Programming Error 7.5

Remember that JavaScript is a case-sensitive language. In code, spelling the keyword `while` with an uppercase \( W \), as in `While`, is a syntax error. All of JavaScript’s reserved keywords, such as `while`, `if` and `else`, contain only lowercase letters.
Fig. 7.5 | Flowcharting the while repetition statement.

- **Product**: `product <= 1000`
  - **True**: `product = 2 * product`
  - **False**: Break out of the cycle when the statement is false

- Evaluate if this statement is still **true**
- If this statement is **true**...
- Execute this statement...
7.8 Formulating Algorithms: Counter-Controlled Repetition

• Counter-controlled repetition is 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 program.

• A counter is a variable a script uses to count.
7.8 Formulating Algorithms: Counter-Controlled Repetition

• Uninitialized variables used in mathematical calculations result in logic errors and produce the value NaN (not a number).

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

Set total to zero
Set grade counter to one

While grade counter is less than or equal to ten

Input the next grade
Add the grade into the total
Add one to the grade counter

Set the class average to the total divided by ten

Print the class average
Counter-controlled repetition to calculate a class average (Part 1 of 3).

Stores the sum of grades

Sets total to 0

Sets gradeCounter to 1 in preparation for the loop

Continues the cycle until gradeCounter is greater than 10
// 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

// Termination Phase
average = total / 10; // calculate the average

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

// -->
</script>
</head>
<body>
<p>Click Refresh (or Reload) to run the script again</p>
</body>
</html>

Fig. 7.7 | Counter-controlled repetition to calculate a class average (Part 2 of 3).

Increments gradeCounter by 1 after each iteration of the loop.
This dialog is displayed 10 times. User input is 100, 88, 93, 55, 68, 77, 83, 95, 73 and 62.

Fig. 7.7 | Counter-controlled repetition to calculate a class average (Part 3 of 3).
Common Programming Error 7.6

Not initializing a variable that will be used in a calculation results in a logic error that produces the value \textit{NaN—Not a Number}. You must initialize the variable before it is used in a calculation.
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` is not a numeric value, `parseInt` returns `NaN` (not a number).
Common Programming Error 7.7

Using floating-point numbers in a manner that assumes they are represented precisely can lead to incorrect results. Real numbers are represented only approximately by computers. For example, no fixed-size floating-point representation of $\pi$ can ever be precise, because $\pi$ is a transcendental number whose value cannot be expressed as digits in a finite amount of space.
• In sentinel-controlled repetition, a special value called a sentinel value (also called a signal value, a dummy value or a flag value) indicates the end of data entry. Sentinel-controlled repetition often is called indefinite repetition, because the number of repetitions is not known in advance.

• It is necessary to choose a sentinel value that cannot be confused with an acceptable input value.
Choosing a sentinel value that is also a legitimate data value results in a logic error and may prevent a sentinel-controlled loop from terminating properly.
• Top-down, stepwise refinement is a technique that is essential to the development of well-structured algorithms.

• This approach begins with pseudocode of the top, the statement that conveys the program’s overall purpose.

• Next we divide the top into a series of smaller tasks and list them in the order in which they need to be performed. This is the first refinement.

• The second refinement commits to specific variables.
Software Engineering Observation 7.7

Each refinement, as well as the top itself, is a complete specification of the algorithm; only the level of detail varies.
Error-Prevention Tip 7.1

When performing division by an expression whose value could be zero, explicitly test for this case, and handle it appropriately in your program (e.g., by printing an error message) rather than allowing the division by zero to occur.
Good Programming Practice 7.5

Include completely blank lines in pseudocode programs to make the pseudocode more readable. The blank lines separate pseudocode control structures and separate the program phases.
Many algorithms can be divided logically into three phases: an initialization phase that initializes the program variables, a processing phase that inputs data values and adjusts program variables accordingly, and a termination phase that calculates and prints the results.
Initialize total to zero
Initialize gradeCounter to zero

Input the first grade (possibly the sentinel)
While the user has not as yet entered the sentinel
   Add this grade into the running total
   Add one to the grade counter
   Input the next grade (possibly the sentinel)

If the counter is not equal to zero
   Set the average to the total divided by the counter
   Print the average
Else
   Print “No grades were entered”

Fig. 7.8 | Sentinel-controlled repetition to solve the class-average problem.
Software Engineering Observation 7.9

The programmer terminates the top-down, stepwise refinement process after specifying the pseudocode algorithm in sufficient detail for the programmer to convert the pseudocode to a JavaScript program. Then, implementing the JavaScript program will normally be straightforward.
Good Programming Practice 7.6

When converting a pseudocode program to JavaScript, keep the pseudocode in the Java-Script program as comments.
Experience has shown that the most difficult part of solving a problem on a computer is developing the algorithm for the solution. Once a correct algorithm is specified, the process of producing a working JavaScript program from the algorithm is normally straightforward.
Many experienced programmers write programs without ever using program-development tools like pseudocode. As they see it, their ultimate goal is to solve the problem on a computer, and writing pseudocode merely delays the production of final outputs. Although this approach may work for simple and familiar problems, it can lead to serious errors in large, complex projects.
7.9 Formulating Algorithms: Sentinel-Controlled Repetition (Cont.)

• Control structures may be stacked on top of one another in sequence.
Fig. 7.9
Sentinel-controlled repetition to calculate a class average (Part 1 of 3).

Set `gradeCounter` to 0 in preparation for the loop.
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

// 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>
<p>Click Refresh (or Reload) to run the script again</p>
</body>
</html>
This dialog is displayed four times. User input is 97, 88, 72 and -1.

Fig. 7.9 | Sentinel-controlled repetition to calculate a class average (Part 3 of 3).
Good Programming Practice 7.7

In a sentinel-controlled loop, the prompts requesting data entry should explicitly remind the user what the sentinel value is.
Common Programming Error 7.9

Omitting the braces that delineate a block can lead to logic errors such as infinite loops.
7.10 Formulating Algorithms: Nested Control Statements

• Control structures may be nested inside of one another.
Initialize passes to zero
Initialize failures to zero
Initialize student to one

While student counter is less than or equal to ten
  Input the next exam result
  If the student passed
    Add one to passes
  Else
    Add one to failures
  Add one to student counter

Print the number of passes
Print the number of failures
If more than eight students passed
  Print “Raise tuition”

Fig. 7.10 | Examination-results problem pseudocode.
Fig. 7.11 | Examination-results calculation (Part 1 of 3).
31 // termination phase
32 document.writeln("<h1>Examination Results</h1>" );
33 document.writeln("Passed: "+passes +"<br />Failed: "+failures );
34
35 // -->
36 </script>
37 </head>
38 <body>
39 </html>

Fig. 7.11 Examination-results calculation (Part 2 of 3).

Additional control structure

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

Fig. 7.11 | Examination-results calculation (Part 3 of 3).
Good Programming Practice 7.8

When inputting values from the user, validate the input to ensure that it is correct. If an input value is incorrect, prompt the user to input the value again.
7.11 Assignment Operators

- JavaScript provides the arithmetic assignment operators `+=`, `-=` `*=` `/=` `and %=`, which abbreviate certain common types of expressions.
Performance Tip 7.1

Programmers can write programs that execute a bit faster when the arithmetic assignment operators are used, because the variable on the left side of the assignment does not have to be evaluated twice.
Performance Tip 7.2

Many of the performance tips we mention in this text result in only nominal improvements, so the reader may be tempted to ignore them. Significant performance improvement often is realized when a supposedly nominal improvement is placed in a loop that may repeat a large number of times.
<table>
<thead>
<tr>
<th>Assignment operator</th>
<th>Initial value of variable</th>
<th>Sample expression</th>
<th>Explanation</th>
<th>Assigns</th>
</tr>
</thead>
<tbody>
<tr>
<td>+=</td>
<td>c = 3</td>
<td>c += 7</td>
<td>c = c + 7</td>
<td>10 to c</td>
</tr>
<tr>
<td>-=</td>
<td>d = 5</td>
<td>d -= 4</td>
<td>d = d - 4</td>
<td>1 to d</td>
</tr>
<tr>
<td>*=</td>
<td>e = 4</td>
<td>e *= 5</td>
<td>e = e * 5</td>
<td>20 to e</td>
</tr>
<tr>
<td>/=</td>
<td>f = 6</td>
<td>f /= 3</td>
<td>f = f / 3</td>
<td>2 to f</td>
</tr>
<tr>
<td>%=</td>
<td>g = 12</td>
<td>g %= 9</td>
<td>g = g % 9</td>
<td>3 to g</td>
</tr>
</tbody>
</table>
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.
### Fig. 7.13
Increment and decrement operators.

<table>
<thead>
<tr>
<th>Operator</th>
<th>Example</th>
<th>Called</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>++</td>
<td>++a</td>
<td>preincrement</td>
<td>Increment a by 1, then use the new value of a in the expression in which a resides.</td>
</tr>
<tr>
<td>++</td>
<td>a++</td>
<td>postincrement</td>
<td>Use the current value of a in the expression in which a resides, then increment a by 1.</td>
</tr>
<tr>
<td>--</td>
<td>--b</td>
<td>predecrement</td>
<td>Decrement b by 1, then use the new value of b in the expression in which b resides.</td>
</tr>
<tr>
<td>--</td>
<td>b--</td>
<td>postdecrement</td>
<td>Use the current value of b in the expression in which b resides, then decrement b by 1.</td>
</tr>
</tbody>
</table>
The predecrement and postdecrement JavaScript operators cause the W3C XHTML Validator to incorrectly report errors. The validator attempts to interpret the decrement operator as part of an XHTML comment tag (<!-- or -->). You can avoid this problem by using the subtraction assignment operator ( -= ) to subtract one from a variable. Note that the validator may report many more (nonexistent) errors once it improperly parses the decrement operator.
Fig. 7.14  
Preincrementing and postincrementing (Part 1 of 2).

<?xml version = "1.0" encoding = "utf-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<!-- Fig. 7.14: increment.html -->
<!-- Preincrementing and Postincrementing. -->
<html xmlns = "http://www.w3.org/1999/xhtml">
  <head>
    <title>Preincrementing and Postincrementing</title>
    <script type = "text/javascript">
      <!--
      var c;
      c = 5;
      document.writeln( "<h3>Postincrementing</h3>" );
      document.writeln( c ); // prints 5
      // prints 5 then increments
      document.writeln( "<br />" + c++ );
      document.writeln( "<br />" + c ); // prints 6
      c = 5;
      document.writeln( "<h3>Preincrementing</h3>" );
      document.writeln( c ); // prints 5
      // increments then prints 6
      document.writeln( "<br />" + ++c );
      document.writeln( "<br />" + c ); // prints 6
      // -->
    </script>
  </head>
  <body></body>
</html>

Prints value of \( c \), then increments it

Increments \( c \), then prints its value
Fig. 7.14 | Preincrementing and postincrementing (Part 2 of 2).
For readability, unary operators should be placed next to their operands, with no intervening spaces.
• When incrementing or decrementing a variable in a statement by itself, the preincrement and postincrement forms have the same effect, and the predecrement and postdecrement forms have the same effect. It is only when a variable appears in the context of a larger expression that preincrementing the variable and postincrementing the variable have different effects. Predecrementing and postdecrementing behave similarly.
Common Programming Error 7.10

Attempting to use the increment or decrement operator on an expression other than a left-hand-side expression—commonly called an lvalue—is a syntax error. A left-hand-side expression is a variable or expression that can appear on the left side of an assignment operation. For example, writing `++(x + z)` is a syntax error, because `(x + z)` is not a left-hand-side expression.
Fig. 7.15
Precedence and associativity of the operators discussed so far.

<table>
<thead>
<tr>
<th>Operator</th>
<th>Associativity</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-</code></td>
<td>right to left</td>
<td>unary</td>
</tr>
<tr>
<td><code>/</code></td>
<td>left to right</td>
<td>multiplicative</td>
</tr>
<tr>
<td><code>-</code></td>
<td>left to right</td>
<td>additive</td>
</tr>
<tr>
<td><code>&lt;=</code></td>
<td>left to right</td>
<td>relational</td>
</tr>
<tr>
<td><code>&gt;</code></td>
<td>left to right</td>
<td>equality</td>
</tr>
<tr>
<td><code>&gt;=</code></td>
<td>left to right</td>
<td>conditional</td>
</tr>
<tr>
<td><code>+=</code></td>
<td>right to left</td>
<td>assignment</td>
</tr>
<tr>
<td><code>-=</code></td>
<td>right to left</td>
<td></td>
</tr>
<tr>
<td><code>*=</code></td>
<td>right to left</td>
<td></td>
</tr>
<tr>
<td><code>/=</code></td>
<td>right to left</td>
<td></td>
</tr>
<tr>
<td><code>%=</code></td>
<td>right to left</td>
<td></td>
</tr>
</tbody>
</table>