

7

JavaScript: Control Statements I

© 2008 Pearson Education, Inc. All rights reserved.

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

© 2008 Pearson Education, Inc. All rights reserved.

- ### 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.
- © 2008 Pearson Education, Inc. All rights reserved.

Outline

- 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

© 2008 Pearson Education, Inc. All rights reserved.

7.1 Introduction

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

© 2008 Pearson Education, Inc. All rights reserved.

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

© 2008 Pearson Education, Inc. All rights reserved.

7.3 Pseudocode

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

© 2008 Pearson Education, Inc. All rights reserved.

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.

© 2008 Pearson Education, Inc. All rights reserved.

7.4 Control Structures

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

© 2008 Pearson Education, Inc. All rights reserved.

7.4 Control Structures (Cont.)

• Flowchart

- A graphical representation of an algorithm or of a portion of an algorithm
- Drawn using certain special-purpose symbols, such as rectangles, diamonds, ovals and small circles
- Symbols are connected by arrows called flowlines, which indicate the order in which the actions of the algorithm execute

© 2008 Pearson Education, Inc. All rights reserved.

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.

© 2008 Pearson Education, Inc. All rights reserved.

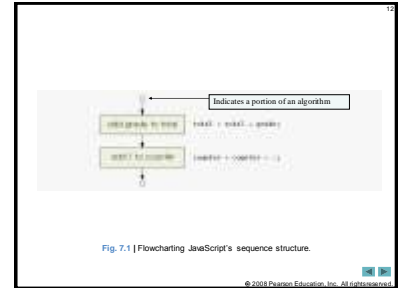


Fig. 7.1 | Flowcharting JavaScript's sequence structure.

© 2008 Pearson Education, Inc. All rights reserved.

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.

© 2008 Pearson Education, Inc. All rights reserved.

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).

© 2008 Pearson Education, Inc. All rights reserved.

Common Programming Error 7.1

Using a keyword as an identifier is a syntax error.

© 2008 Pearson Education, Inc. All rights reserved.

Fig. 7.2 | JavaScript keywords.

JavaScript 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		
<i>Keywords that are reserved but not used by JavaScript</i>				
abstract	boolean	byte	char	class
const	debugger	double	enum	export
extends	final	float	goto	implements
import	int	interface	long	native
package	private	protected	public	short
static	super	synchronized	throws	transient
volatile				

© 2008 Pearson Education, Inc. All rights reserved.

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.
 - Control-structure stacking.
- There is only one other way control structures may be connected
 - Control-structure nesting

© 2008 Pearson Education, Inc. All rights reserved.

7.5 if Selection Statement

- The JavaScript interpreter ignores white-space characters
 - Manks, 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

© 2008 Pearson Education, Inc. All rights reserved.

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.

© 2008 Pearson Education, Inc. All rights reserved.

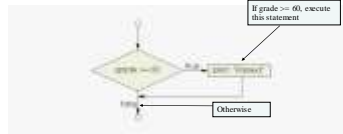


Fig. 7.3 | Flowcharting the single-selection **if** statement.

© 2008 Pearson Education, Inc. All rights reserved.

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**.

© 2008 Pearson Education, Inc. All rights reserved.

7.6 **if...else** Selection Statement

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

© 2008 Pearson Education, Inc. All rights reserved.

Good Programming Practice 7.2

Indent both body statements of an **if...else** statement.

© 2008 Pearson Education, Inc. All rights reserved.

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**

© 2008 Pearson Education, Inc. All rights reserved.

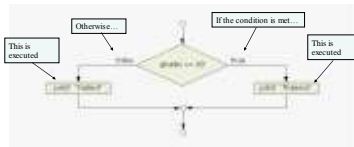


Fig. 7.4 | Flowcharting the double-selection **if...else** statement.

© 2008 Pearson Education, Inc. All rights reserved.

7.6 **if...else** Selection Statement (Cont.)

- 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, enclose the statements in braces (**{}** and **}**)
 - A set of statements contained within a pair of braces is called a block

© 2008 Pearson Education, Inc. All rights reserved.

Good Programming Practice 7.3

If there are several levels of indentation, each level should be indented the same additional amount of space.

© 2008 Pearson Education, Inc. All rights reserved.

Software Engineering Observation 7.3

A block can be placed anywhere in a program that a single statement can be placed.

© 2008 Pearson Education, Inc. All rights reserved.

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.

© 2008 Pearson Education, Inc. All rights reserved.

Common Programming Error 7.2

Forgetting one or both of the braces that delimit a block can lead to syntax errors or logic errors.

© 2008 Pearson Education, Inc. All rights reserved.

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.

© 2008 Pearson Education, Inc. All rights reserved.

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.

© 2008 Pearson Education, Inc. All rights reserved.

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.

© 2008 Pearson Education, Inc. All rights reserved.

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).

© 2008 Pearson Education, Inc. All rights reserved.

7.7 `while` Repetition Statement

- **while**
 - 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 and repetition terminates

© 2008 Pearson Education, Inc. All rights reserved.

Common Programming Error 7.4

If the body of a `while` statement never causes the `while` statement's condition to become true, 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.

© 2008 Pearson Education, Inc. All rights reserved.

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).

© 2008 Pearson Education, Inc. All rights reserved.

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 π can ever be precise, because π is a transcendental number whose value cannot be expressed as digits in a finite amount of space.

© 2008 Pearson Education, Inc. All rights reserved.

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

© 2008 Pearson Education, Inc. All rights reserved.

Common Programming Error 7.8

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.

© 2008 Pearson Education, Inc. All rights reserved.

7.9 Formulating Algorithms: Sentinel-Controlled Repetition (Cont.)

- Top-down, stepwise refinement
 - A technique that is essential to the development of well-structured algorithms
 - Approach begins with pseudocode of the top, the statement that conveys the program's overall purpose
 - Divide the top into a series of smaller tasks and list them in the order in which they need to be performed—the first refinement
 - Second refinement commits to specific variables

© 2008 Pearson Education, Inc. All rights reserved.

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.

© 2008 Pearson Education, Inc. All rights reserved.

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.

© 2008 Pearson Education, Inc. All rights reserved.

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.

© 2008 Pearson Education, Inc. All rights reserved.

Software Engineering Observation 7.8

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.

© 2008 Pearson Education, Inc. All rights reserved.

```

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 JavaScript program as comments.

Software Engineering Observation 7.10

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.

Software Engineering Observation 7.11

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) -->
<!-- Fig. 7.8: average.html -->
<!-- Sentinel-controlled repetition to calculate a class average -->
<!-- http://www.it-ebooks.info -->
<html>
<script type="text/javascript">
<!--
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

// sentinel/initial phase
total = 0; // clear total
gradeCounter = 0; // prepare to loop

// processing phase
// prompt the user 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 );

```

Fig. 7.9 | Sentinel-controlled repetition to calculate a class average (Part 1 of 3).

```

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

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

// prompt the user 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 );

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

// display average of class grades
document.write(
    "Average is " + total / gradeCounter );

// end program
} // end of program
</script>
</html>
<!-- Click here to go to the script source -->

```

Fig. 7.9 | Sentinel-controlled repetition to calculate a class average (Part 2 of 3).

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.

© 2008 Pearson Education, Inc. All rights reserved.

Common Programming Error 7.9

Omitting the braces that delineate a block can lead to logic errors such as infinite loops.

© 2008 Pearson Education, Inc. All rights reserved.

7.10 Formulating Algorithms: Nested Control Statements

- Control structures may be nested inside of one another

© 2008 Pearson Education, Inc. All rights reserved.

```
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.

© 2008 Pearson Education, Inc. All rights reserved.

```
<code></code>
```

Fig. 7.11 | Examination-results calculation (Part 1 of 3).

© 2008 Pearson Education, Inc. All rights reserved.

```
<code></code>
```

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

© 2008 Pearson Education, Inc. All rights reserved.

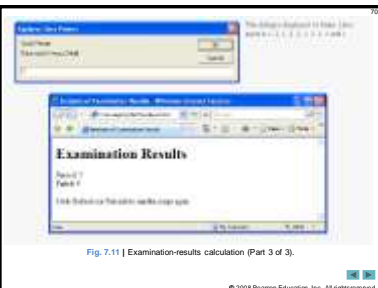


Fig. 7.11 | Examination-results calculation (Part 3 of 3).

© 2008 Pearson Education, Inc. All rights reserved.

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.

© 2008 Pearson Education, Inc. All rights reserved.

7.11 Assignment Operators

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

© 2008 Pearson Education, Inc. All rights reserved.

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.

© 2008 Pearson Education, Inc. All rights reserved.

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.

© 2008 Pearson Education, Inc. All rights reserved.

Fig. 7.12 | Arithmetic assignment operators.

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

© 2008 Pearson Education, Inc. All rights reserved.

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 postfix to the variable, the variable is used in its expression, then incremented or decremented by 1.

© 2008 Pearson Education, Inc. All rights reserved.

Fig. 7.13 | Increment and decrement operators.

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.

© 2008 Pearson Education, Inc. All rights reserved.

Error-Prevention Tip 7.2

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.

© 2008 Pearson Education, Inc. All rights reserved.

Fig. 7.14 | Preincrementing and postincrementing (Part 1 of 2).

```

1 <html xmlns="http://www.w3.org/1999/xhtml" >
2 <body onload="init()" >
3 <script src="http://www.w3.org/2001/xml/1.0/xhtml.js" >
4 </script >
5 <!-- Fig. 7.14: Preincrementing and postincrementing -->
6 <!-- Preincrementing and postincrementing -->
7 <html xmlns="http://www.w3.org/2001/xhtml" >
8 <body >
9 <!-- Preincrementing and postincrementing -->
10 <script type="text/javascript" >
11 <!--
12
13 c = 3;
14 document.write( "<pre>Preincrementing/Postincrementing</pre>" );
15 document.write( c ); // prints 3
16 // prints 3 then increments
17 document.write( "c++" ); // prints 4
18 document.write( "c++" ); // prints 4
19
20 c = 11;
21 document.write( "<pre>Preincrementing/Postincrementing</pre>" );
22 document.write( c ); // prints 11
23 // increments then prints 4
24 document.write( "++c" ); // prints 4
25 document.write( "++c" ); // prints 4
26 // -->
27 </script >
28 </body >
29 </html >

```

© 2008 Pearson Education, Inc. All rights reserved.

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

© 2008 Pearson Education, Inc. All rights reserved.

Good Programming Practice 7.9

For readability, unary operators should be placed next to their operands, with no intervening spaces.

© 2008 Pearson Education, Inc. All rights reserved.

7.12 Increment and Decrement Operators (Cont.)

- 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
- When a variable appears in the context of a larger expression, preincrementing the variable and postincrementing the variable have different effects. Predecrementing and postdecrementing behave similarly.

© 2008 Pearson Education, Inc. All rights reserved.

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 + 1)` is a syntax error, because `(x + 1)` is not a left-hand-side expression.

© 2008 Pearson Education, Inc. All rights reserved.

Fig. 7.15 | Precedence and associativity of the operators discussed so far.

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

© 2008 Pearson Education, Inc. All rights reserved.