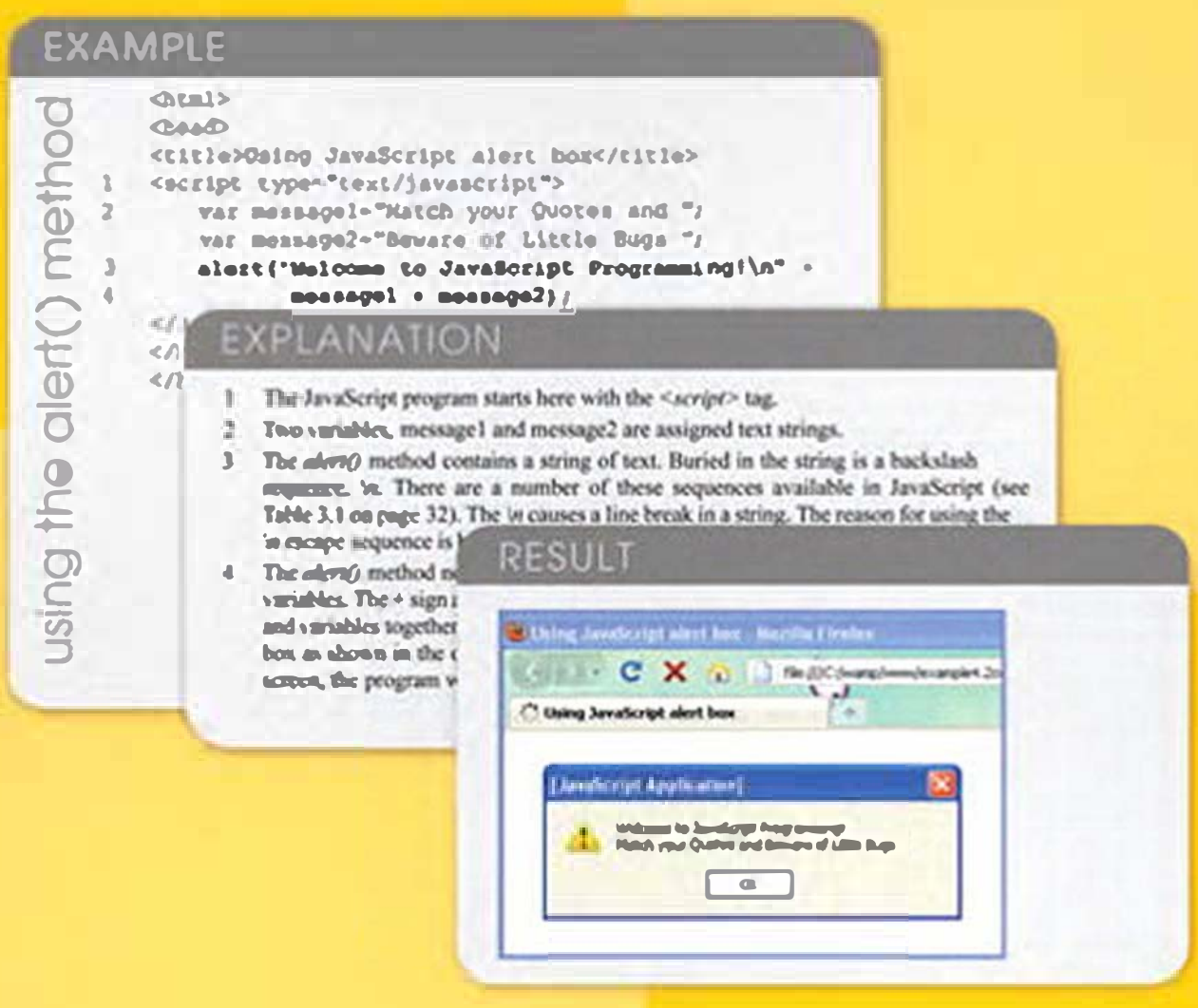
PRENTICE

JavaScript *by* Example

Second Edition



Ellie Quigley

Library of Congress Cataloging-in-Publication Data

Quigley, Ellie.

JavaScript by example / Ellie Quigley.—2nd ed.
p. cm.
Includes index.
ISBN 978-0-13-705489-3 (pbk. : alk. paper)
1. JavaScript (Computer program language) I. Title.
QA76.73.J39Q54 2010
005.13'3—dc22

2010020402

ISBN-13: 978-0-13-705489-3
ISBN-10: 0-13-705489-0
Text printed in the United States on recycled paper at Edwards Brothers in Ann Arbor, Michigan.
First printing, October 2010

Editor-in-Chief
Mark L. Taub

Managing Editor
John Fuller

Full-Service Production Manager
Julie B. Nahil

Production Editor
Dmitri Korzh
Techne Group

Copy Editor
Teresa Horton

Indexer
Potomac Indexing, LLC

Proofreader
Beth Roberts

Editorial Assistant
Kim Boedigheimer

Cover Designer
Anne Jones

Composition
Techne Group

Contents

Preface

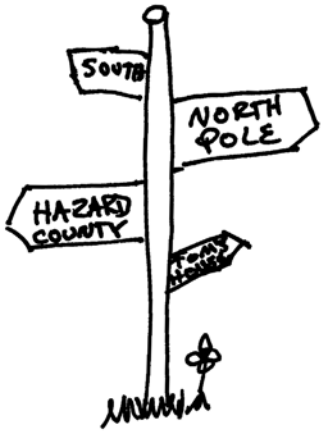
This second edition of *JavaScript by Example* is really more than a new edition; it is a new book! So much has changed since the first edition in 2002, and now with the newfound popularity of Ajax, JavaScript is on a roll! Almost every personal computer has JavaScript installed and running and it is the most popular Web scripting language around, although it comes under different aliases, including Mocha, LiveScript, JScript, and ECMAScript. There are a lot of books out there dedicated to some aspect of the JavaScript language and if you are new to JavaScript, it would be difficult to know where to start. This book is a "one size fits all" edition, dedicated to those of you who need a balance between the technical side of the language and the fun elements, a book that addresses cross-platform issues, and a book that doesn't expect that you are already a guru before you start. This edition explains how the language works from the most basic examples to the more complex, in a progression that seemlessly leads you from example to example until you have mastered the basics all the way to the more advanced topics such as CSS, the DOM, and Ajax.

Because I am a teacher first, I found that using my first edition worked well in the classroom, but I needed more and better examples to get the results I was looking for. Many of my students have been designers but not programmers, or programmers who don't understand design. I needed a text that would accommodate both without leaving either group bored or overwhelmed. This huge effort to modernize the first edition went way beyond where I had expected or imagined. I have learned much and hope that you will enjoy sharing my efforts to make this a fun and thorough coverage of a universally popular and important Web programming language.

chapter

6

Under Certain Conditions



6.1 Control Structures, Blocks, and Compound Statements

If you were confronted with the above signpost, you'd have to decide which direction to take. People control their lives by making decisions, and so do programs. In fact, according to computer science books, a good language allows you to control the flow of your program in three ways. It lets you

- Execute a sequence of statements.
- Branch to an alternative sequence of statements, based on a test.
- Repeat a sequence of statements until some condition is met.

Well, then JavaScript must be a good language. We've already used programs that execute a sequence of statements, one after another.

Now we will examine the branching and looping control structures that allow the flow of the program's control to change depending on some conditional expression.

The decision-making constructs (*if*, *if/else*, *if/else if*, *switch*) contain a control expression that determines whether a block of statements will be executed. The looping constructs (*while, for*) allow the program to execute a statement block repetitively until some condition is satisfied.

A compound statement or block consists of a group of statements surrounded by curly braces. The block is syntactically equivalent to a single statement and usually follows an *if*, *else*, *while*, or *for* construct.

6.2 Conditionals

Conditional constructs control the flow of a program. If a condition is true, the program will execute a block of statements and if the condition is false, flow will go to an alternate block of statements. Decision-making constructs (*if, else, switch*) contain a control

expression that determines whether a block of expressions will be executed. If the condition after the *if* is met, the result is true, and the following block of statements is executed; otherwise the result is false and the block is not executed.

FORMAT

```
if (condition){
   statements;
}
```

EXAMPLE

```
if ( age > 21 ){
   alert("Let's Party!");
}
```

The block of statements (or single statement) is enclosed in curly braces. Normally, statements are executed sequentially. If there is only one statement after the conditional expression, the curly braces are optional.

6.2.1 *if/else*

"You better pay attention now, or else . . . " Ever heard that kind of statement before? JavaScript statements can be handled the same way with the *if/else* branching construct. This construct allows for a two-way decision. The *if* evaluates the expression in parentheses, and if the expression evaluates to true, the block after the opening curly braces is executed; otherwise the block after the *else* is executed.

FORMAT

```
if (condition){
   statements1;
}
else{
   statements2;
}
```

EXAMPLE

```
if ( x > y ){
   alert( "x is larger");
}
else{
    alert( "y is larger");
}
```

EXAMPLE 6.1

```
<html>
  <head>
    <title>Conditional Flow Control</title>
  </head>
  <body>
    <h3>
1     <script type="text/javascript">
        <!--  Hiding JavaScript from old browsers
2       var age=prompt("How old are you? ","");
3       if( age >= 55 ){
4         document.write("You pay the senior fare! ");
5       }
6       else{
7         document.write("You pay the regular adult fare. ");
        }
        //-->
8     </script>
    </h3>
  </body>
</html>
```

EXPLANATION

1 JavaScript program starts here.

2 The prompt dialog box will display the message "*How old are you?*". Whatever the user types into the box will be stored in the variable *age* (see Figure 6.1).

3, 4 If the value of the variable *age* is greater than or equal to *55*, line 4 is executed (see Figure 6.2).

5 This closing curly brace closes the block of statements following the *if* expression. When there is only one statement in the block, the curly braces are not required.

6, 7 The *else* statement, line number 7, is executed if the expression in line 3 is false.

8 This tag marks the end of the JavaScript program.

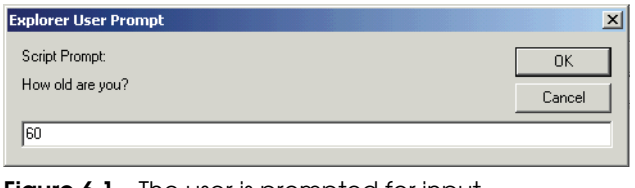


Figure 6.1 The user is prompted for input.

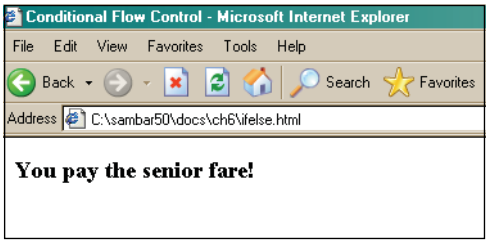


Figure 6.2 If the age entered was greater than 55, this message is displayed.

The Conditional Operator. The conditional operator, called a ternary operator, was discussed in Chapter 5, "Operators." Because it is often used as a shortcut for the *if/else* conditional statement, it is reviewed again here.

FORMAT

```
conditional expression ? expression : expression
```

EXAMPLE

```
x ? y : z     If x evaluates to true, the value of the expression
              becomes y, else the value of the expression becomes z

big = (x > y) ? x : y      If x is greater than y, x is assigned to
                           variable big, else y is assigned to
                           variable big
```

An *if/else* statement instead of the conditional statement:

```
if (x > y) {
   big = x;
}
else{
   big = y;
}
```

EXAMPLE 6.2

```
   <html>
      <head>
         <title>Conditional Operator</title>
      </head>
      <body bgcolor="lightblue">
         <big>
         <script type ="text/javascript">
1           var age = prompt("How old are you? ", "");
2           var price = (age > 55 ) ? 0 : 7.50;
```

EXAMPLE 6.2 (CONTINUED)

```
3         alert("You pay $" + price + 0);
        </script>
        </big>
      </body>
    </html>
```

EXPLANATION

1 The user is prompted for input. The value he or she enters in the prompt box is assigned to the variable *age*.

2 If the value of *age* is greater than 55, the value to the right of the *?* is assigned to the variable *price*; if not, the value after the *:* is assigned to the variable *price*.

3 The alert dialog box displays the value of the variable *price*.

6.2.2 *if/else if*

"If you've got $1, we can go to the Dollar Store; else if you've got $10, we could get a couple of movies; else if you've got $20 we could buy a CD . . . else forget it!" JavaScript provides yet another form of branching, the *if/else if* construct. This construct provides a multiway decision structure.

FORMAT

```
if (condition) {
   statement(s);
}
else if (condition)  {
   statement(s);
}
else if (condition)  {
   statement(s);
}
else{
   statement(s);
}
```

If the first conditional expression following the *if* keyword is true, the statement or block of statements following the expression is executed and control starts after the final *else* block. Otherwise, if the conditional expression following the *if* keyword is false, control branches to the first *else if* and the expression following it is evaluated. If that expression is true, the statement or block of statements following it are executed, and if false, the next *else if* is tested. All *else if*s are tested and if none of their expressions are true, control goes to the *else* statement. Although the *else* is not required, it normally serves as a default action if all previous conditions were false.

EXAMPLE 6.3

```
<html>
  <head>
    <title>Conditional Flow Control</title>
  </head>
  <body>
    <h2>
1     <script type="text/javascript">
        <!--
2       var age=eval( prompt("How old are you? ",""));
3       if( age > 0 && age <= 12 ){
4         alert("You pay the child's fare. ");
        }
5       else if( age > 12 && age < 60 ){
6         alert("You pay the regular adult fare. ");
        }
        else {
7         alert("You pay the senior fare! ");
        }
        //-->
8     </script>
    </h3>
  </body>
</html>
```

EXPLANATION

1 JavaScript program starts here.

2 The prompt dialog box will display the message "*How old are you?* ". Whatever the user types into the box will be converted to a number by the *eval()* method and then stored in the variable *age*.

3, 4 If the value of the variable *age* is greater than *0* and *age* is also less than or equal to *12*, then line 4 is executed and the program continues at line 8.

5, 6 If the expression on line 3 is false, the JavaScript interpreter will test this line, and if the age is greater than 12 and also less than 60, the block of statements that follow will be executed and control goes to line 8. You can have as many *else if*s as you like.

7 The *else* statement, line number 7, is executed if all of the previous expressions test false. This statement is called the default and is not required.

8 This tag marks the end of the JavaScript program.

6.2.3 *switch*

The *switch* statement is an alternative to *if/else if* conditional construct (commonly called a "case statement") and may make the program more readable when handling multiple options.

FORMAT

```
switch (expression){
case label :
   statement(s);
   break;
case label :
   statement(s);
   break;
   ...
default : statement;
}
```

EXAMPLE

```
switch (color){
case "red":
   alert("Hot!");
   break;
case "blue":
   alert("Cold.");
   break;
default:
   alert("Not a good choice.");
   break;
}
```

The value of the *switch* expression is matched against the expressions, called labels, following the *case* keyword. The *case* labels are constants, either string or numeric. Each label is terminated with a colon. The default label is optional, but its action is taken if none of the other cases match the *switch* expression. After a match is found, the statements after the matched label are executed for that case. If none of the cases are matched, the control drops to the *default* case. The default is optional. If a *break* statement is omitted, all statements below the matched label are executed until either a *break* is reached or the entire *switch* block exits.

EXAMPLE 6.4

```
<html>
   <head>
      <title>The Switch Statement</title>
   </head>
   <body>
```

Continues

EXAMPLE 6.4 (CONTINUED)

```
        <script type="text/javascript">
          <!--
1         var day_of_week=Math.floor((Math.random()* 7)+1);
            // Get a random number between 1 and 7
            // Monday is 1, Tuesday is 2, etc.
2         switch(day_of_week){
3           case 1:
            case 2:
            case 3:
            case 4:
4             alert("Business hours Monday through Thursday are from
                     9am to 10pm");
5             break;
            case 5:
              alert("Business hours on Friday are from 9am to 6pm");
              break;
            case 6:
              alert("Business hours on Saturday are from
                     11am to 3pm");
              break;
6           default:
              alert("We are closed on Sundays and holidays");
7             break;
8         }
          //-->
        </script>
      </body>
    </html>
```

EXPLANATION

1 The random number function generates a random number between 1 and 7 inclusive when the script is executed. The random number is stored in a variable called *day_of_week*.

2 The *day_of_week* value of the *switch* expression is matched against the values of each of the *case* labels below.

3 The first *case* that is tested is *1*. If the random number is 1, the message "*Business hours Monday through Thursday are from 9am to 10pm*" will be displayed in the alert dialog box. The same is true for case 2, 3, and 4.

4 This statement is executed if case 1, 2, 3, or 4 are matched. Note there are no *break* statements associated with any of these 4 case statements. Program control just drops from one case to the next, and if cases 1, 2, 3, or 4 are not matched, execution control goes to the next case (case 5) for testing.

5 The *break* statement causes program control to continue after line 8. Without it, the program would continue executing statements into the next *case*, *"yellow"*, and continue doing so until a *break* is reached or the *switch* ends—and we don't want that. The *break* statement sends control of the program to line 8.

EXPLANATION (CONTINUED)

6 The default statements are executed if none of the cases are matched.

7 This final *break* statement is not necessary, but is good practice in case you should decide to replace the *default* with an additional *case* label.

8 The final curly brace ends the *switch* statement. Figure 6.3 displays examples of the output.

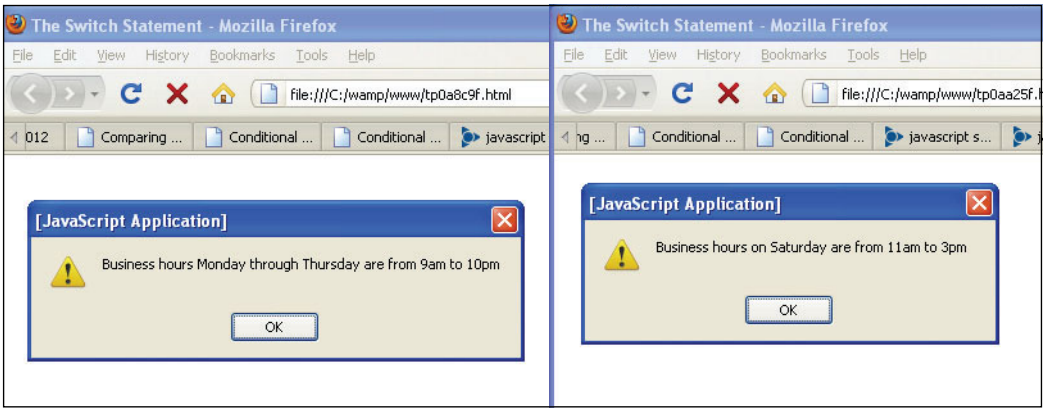


Figure 6.3 A random number between 1 and 7 determines which *case* is matched and executed.

6.3 Loops

Loops are used to execute a segment of code repeatedly until some condition is met. JavaScript's basic looping constructs are

- *while*
- *for*
- *do/while*

6.3.1 The *while* Loop

The *while* statement executes its statement block as long as the expression after the while evaluates to true; that is, nonnull, nonzero, nonfalse. If the condition never changes and is true, the loop will iterate forever (infinite loop). If the condition is false, control goes to the statement right after the closing curly brace of the loop's statement block.

The *break* and *continue* functions are used for loop control.

FORMAT

```
while (condition) {
   statements;
   increment/decrement counter;
}
```

EXAMPLE 6.5

```
   <html>
     <head>
       <title>Looping Constructs</title>
     </head>
     <body>
       <h2>While Loop</h2>
       <font size="+2">
1      <script type="text/javascript">
2        var i=0;    // Initialize loop counter
3        while ( i < 10 ){      // Test
4          document.writeln(i);
5          i++;    // Increment the counter
6        }     // End of loop block
7      </script>
       </font>
     </body>
   </html>
```

EXPLANATION

1 The JavaScript program starts here.

2 The variable *i* is initialized to *0*.

3 The expression after the *while* is tested. If *i* is less than *10*, the block in curly braces is entered and its statements are executed. If the expression evaluates to false, (i.e., *i* is not less than *10*), the loop block exits and control goes to line 6.

4 The value of *i* is displayed in the browser window (see Figure 6.4).

5 The value of *i* is incremented by 1. If this value never changes, the loop will never end.

6 This curly brace marks the end of the *while* loop's block of statements.

7 The JavaScript program ends here.

Figure 6.4 Output from Example 6.5.

6.3.2 The *do/while* Loop

The *do/while* statement executes a block of statements repeatedly until a condition becomes false. Owing to its structure, this loop necessarily executes the statements in the body of the loop at least once before testing its expression, which is found at the bottom of the block. The *do/while* loop is supported in Mozilla/Firefox and Internet Explorer 4.0, JavaScript 1.2, and ECMAScript v3.

FORMAT

```
do
   { statements;}
while (condition);
```

EXAMPLE 6.6

```
   <html>
      <head>
         <title>Looping Constructs</title>
      </head>
      <body>
         <h2>Do While Loop</h2>
         <font size="+2">
         <script type="text/javascript">
1           var i=0;
2           do{
3              document.writeln(i);
4              i++;
5           } while ( i < 10 )
         </script>
         </font>
      </body>
   </html>
```

EXPLANATION

1 The variable *i* is initialized to *0*.
2 The *do* block is entered. This block of statements will be executed before the *while* expression is tested. Even if the *while* expression proves to be false, this block will be executed the first time around.
3 The value of *i* is displayed in the browser window (see Figure 6.5).
4 The value of *i* is incremented by 1.
5 Now, the *while* expression is tested to see if it evaluates to true (i.e., is *i* less than *10*?). If so, control goes back to line 2 and the block is re-entered.

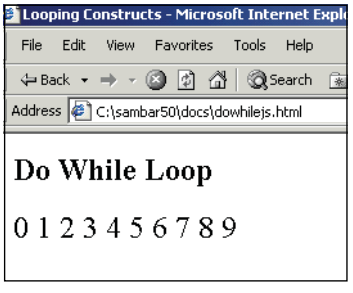


Figure 6.5 Output from Example 6.6, the *do/while* loop.

6.3.3 The *for* Loop

The *for* loop consists of the *for* keyword followed by three expressions separated by semicolons and enclosed within parentheses. Any or all of the expressions can be omitted, but the two semicolons cannot. The first expression is used to set the initial value of variables and is executed just once, the second expression is used to test whether the loop should continue or stop, and the third expression updates the loop variables; that is, it increments or decrements a counter, which will usually determine how many times the loop is repeated.

FORMAT

```
for(Expression1;Expression2;Expression3)
   {statement(s);}
for (initialize; test; increment/decrement)
   {statement(s);}
```

The preceding format is equivalent to the following *while* statement:

```
Expression1;
while( Expression2 )
   { Block; Expression3};
```

EXAMPLE 6.7

```
<html>
  <head>
    <title>Looping Constructs</title>
  </head>
  <body>
    <h2>For Loop</h2>
    <font size="+2">
    <script type="text/javascript">
1     for( var i = 0; i < 10; i++ ){
2       document.write(i);
3     }
    </script>
    </font>
  </body>
</html>
```

EXPLANATION

1 The *for* loop is entered. The expression starts with step 1, the initialization of the variable *i* to *0*. This is the only time this step is executed. The second expression, step 2, tests to see if *i* is less than *10*, and if it is, the statements after the opening curly brace are executed. When all statements in the block have been executed and the closing curly brace is reached, control goes back into the *for* expression to the last expression of the three. *i* is now incremented by one and the expression in step 2 is retested. If true, the block of statements is entered and executed.

2 The value of *i* is displayed in the browser window (see Figure 6.6).

3 The closing curly brace marks the end of the *for* loop.

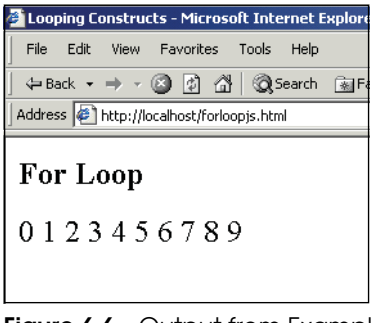


Figure 6.6 Output from Example 6.7.

6.3.4 The *for/in* Loop

The *for/in* loop is like the *for* loop, except it is used with JavaScript objects. Instead of iterating the statements based on a looping condition, it operates on the properties of an object. This loop is discussed in Chapter 9, "JavaScript Core Objects," and is only mentioned here in passing, because it falls into the category of looping constructs.

6.3.5 Loop Control with *break* and *continue*

The control statements, *break* and *continue*, are used to either break out of a loop early or return to the testing condition early; that is, before reaching the closing curly brace of the block following the looping construct.

Table 6.1 Control Statements

Statement	***What It Does***
break	Exits the loop to the next statement after the closing curly brace of the loop's statement block.
continue	Sends loop control directly to the top of the loop and re-evaluates the loop condition. If the condition is true, enters the loop block.

EXAMPLE 6.8

```
  <html>
    <head>
      <title>Looping Constructs</title>
    </head>
    <body>
1     <script type="text/javascript">
2       while(true) {
3         var grade=eval(prompt("What was your grade? ",""));
4         if (grade < 0 || grade > 100) {
            alert("Illegal choice!");
5           continue;   // Go back to the top of the loop
          }
          if(grade > 89 && grade < 101)
6           {alert("Wow! You got an A!");}
7         else if (grade > 79 && grade < 90)
            {alert("You got a B");}
          else if (grade > 69 && grade < 80)
            {alert("You got a C");}
          else if (grade > 59 && grade < 70)
            {alert("You got a D");}
8         else {alert("Study harder. You Failed.");}
9         answer=prompt("Do you want to enter another grade?","");
10        if(answer != "yes"){
11          break;    // Break out of the loop to line 12
          }
12      }
      </script>
    </body>
  </html>
```

EXPLANATION

1 The JavaScript program starts here.

2 The *while* loop is entered. The loop expression will always evaluate to true, causing the body of the loop to be entered.

3 The user is prompted for a grade, which is assigned to the variable *grade*.

4 If the variable *grade* is less than 0 or more than 100, "*Illegal choice*" is printed.

5 The *continue* statement sends control back to line 2 and the loop is re-entered, prompting the user again for a grade.

6 If a valid grade was entered, and it is greater than 89 and less than 101, the grade "*A*" is displayed (see Figure 6.7).

7 Each *else/if* branch will be evaluated until one of them is true.

8 If none of the expressions are true, the *else* condition is reached and "*You Failed*" is displayed.

9 The user is prompted to see if he or she wants to enter another grade.

10, 11 If the answer is not *yes*, the *break* statement takes the user out of the loop, to line 12.

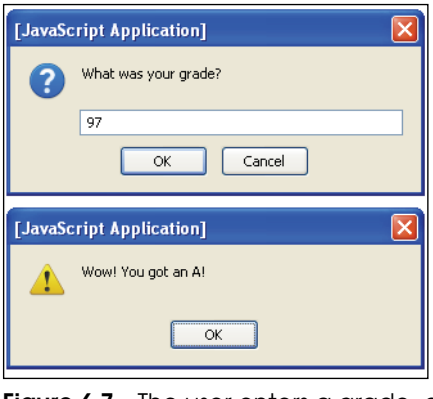


Figure 6.7 The user enters a grade, clicks OK, and gets another alert box.

6.3.6 Nested Loops and Labels

Nested Loops. A loop within a loop is a nested loop. A common use for nested loops is to display data in rows and columns. One loop handles the rows and the other handles the columns. The outside loop is initialized and tested, the inside loop then iterates completely through all of its cycles, and the outside loop starts again where it left off. The inside loop moves faster than the outside loop. Loops can be nested as deeply as you wish, but there are times when it is necessary to terminate the loop owing to some condition.

EXAMPLE 6.9

```
<html>
  <head>
    <title>Nested loops</title>
  </head>
  <body>
    <script type="text/javascript">
      <!--  Hiding JavaScript from old browsers
1     var str = "@";
2     for ( var row = 0; row < 6; row++){
3       for ( var col=0; col < row; col++){
          document.write(str);
        }
4       document.write("<br />");
      }
      //-->
    </script>
  </body>
</html>
```

EXPLANATION

1 The variable *str* is assigned a string "@".

2 The outer *for* loop is entered. The variable *row* is initialized to *0*. If the value of *row* is less than *6*, the loop block (in curly braces) is entered (i.e., go to line 3).

3 The inner *for* loop is entered. The variable *col* is initialized to *0*. If the value of *col* is less than the value of *row*, the loop block is entered and an @ is displayed in the browser. Next, the value of *col* will be incremented by 1, tested, and if still less than the value of *row*, the loop block is entered, and another @ displayed. When this loop has completed, a row of @ symbols will be displayed, and the statements in the outer loop will start up again.

4 When the inner loop has completed looping, this line is executed, producing a break in the rows (see Figure 6.8).

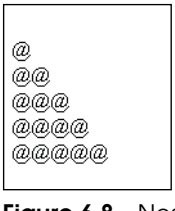


Figure 6.8 Nested loops: rows and columns. Output from Example 6.9.

Labels. Labels allow you to name control statements (*while*, *do/while*, *for*, *for/in*, and *switch*) so that you can refer to them by that name elsewhere in your program. They can be named the same as any other legal identifier that is not a reserved word. By themselves, labels do nothing. Labels are optional, but are often used to control the flow of a loop. A label looks like this, for example:

```
topOfLoop:
```

Normally, if you use loop-control statements such as *break* and *continue*, the control is directed to the innermost loop. There are times when it might be necessary to switch control to some outer loop. This is where labels most often come into play. By prefixing a loop with a label, you can control the flow of the program with *break* and *continue* statements as shown in Example 6.10. Labeling a loop is like giving the loop its own name.

EXAMPLE 6.10

```
  <script type="text/javascript">
1   outerLoop: for ( var row = 0; row < 10; row++){
2     for ( var col=0; col <= row; col++){
        document.write("row "+ row +"|column " + col, "<br />");
3       if(col==3){
          document.write("Breaking out of outer loop at column
4               " + col +"<br />");
5         break outerLoop;
        }
      }
6     document.write("************<br />");
7   }    // end outer loop block
  </script>
```

EXPLANATION

1 The label *outerLoop* labels the *for* loop that follows it. It's like giving the for loop its own name so that it can be referenced by that name later.

2 This is a nested *for* loop. As the program executes the row and column numbers are displayed.

3 If the expression is true, the *break* statement, with the label, causes control to go to line 8; it breaks out of the *outer:* loop. A *break* statement without a label would cause the program to exit just the loop to which it belongs.

4 The value of *row* and *col* are displayed as the inner loop iterates.

5 The *break* statement with the label causes control to go to line 8.

6 Each time the inner loop exits, this row of stars will be printed (see Figure 6.9). Notice that the row of stars is not printed when the loop is exited on line 5.

7 The closing curly brace closes the outer *for* loop block on line 1.

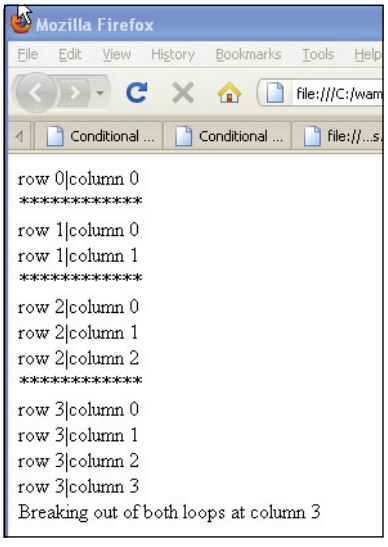


Figure 6.9 Using a label with a loop.

6.4 What You Should Know

"Two roads diverged in a wood, and I—" wrote Robert Frost. This chapter was about making decisions about the flow of your program, what road to take, how to repeat a sequence of statements, and how to stop the repetition. At this point, you should understand:

1. How to use conditional constructs to control the flow of your program; *if/else*, *switch*, and so on.
2. What a block is and when to use curly braces.
3. How and why you would use a *switch* statement.
4. How the *while* and the *do/while* loops differ.
5. How to use a *for* loop.
6. How to use *break* and *continue* with loops.
7. The purpose of nested loops.
8. How to make an infinite loop and how to get out of it.
9. The purpose of labels in loops.
10. How *else/if*s work.

Exercises

1. Create a *while* loop that displays numbers as: 10 9 8 7 6 5 4 3 2 1. Put the numbers in HTML table cells.

2. Ask the user what the current hour is. If the hour is between 6 and 9 a.m., tell the user, "Breakfast is served." If the hour is between 11 a.m. and 1 p.m., tell the user, "Time for lunch." If the hour is between 5 and 8 p.m., tell the user, "It's dinner time." For any other hours, tell the user, "Sorry, you'll have to wait, or go get a snack."

3. Create a conversion table using the following formula:

   ```
   C = (F - 32) / 1.8;
   ```

   Start with a Fahrenheit temperature of 20 degrees and end with a temperature of 120 degrees; use an increment value of 5. The table will have two columns, one for Fahrenheit temperature values and one for those same temperatures converted to Celsius.

4. Ask the user for the name of the company that developed the JavaScript language. Alert the user when he or she is wrong, and then keep asking the user until he or she gets the correct answer. When the user gets it right, confirm it.

5. Use a *switch* statement to rewrite the following JavaScript code. Prompt the user for the number of a month rather than setting it to 8.

   ```
   <script type=text/javascript>
       month = 8;

       if (month == 1) {
           alert("January");
       }
        else if (month == 2) {
           alert("February");
       }
        else if (month == 3) {
           alert("March");
       }
        else if (month == 4) {
           alert("April");
       }
        else if (month == 5) {
           alert("May");
       }
   ```

```
    else if (month == 6) {
        alert("June");
    }
     else if (month == 7) {
        alert("July");
    }
    else if (month == 8) {
        alert("August");
    }
    else if (month == 9) {
        alert("September");
    }
    else if (month == 10) {
        alert("October");
    }
    else if (month == 11) {
        alert("November");
    }
    else if (month == 12) {
        alert("December");
    }
    else{
        alert("Invalid month");
    }
</script>
```

6. Consider the following example:

```
var start_time = (day == weekend) ? 12 : 9;
```

Rewrite the conditional statement using an *if/else* construct.

Index

C

D

E

F

G

H

I

J

M

N

S

T

X

Y

Z