CHAPTER

3

Adding Logic and Control to Your Programs

So far you've learned about some of JavaScript's basic building blocks. But simply creating a variable and storing a string or number in it doesn't accomplish much. And building an array with a long list of items won't be very useful unless there's an easy way to work your way through the items in the array. In this chapter, you'll learn how to make your programs react intelligently and work more efficiently by using conditional statements, loops, and functions.

Making Programs React Intelligently

Our lives are filled with choices: "What should I wear today?", "What should I eat for lunch?", "What should I do Friday night?", and so on. Many choices you make depend on other circumstances. For example, say you decide you want to go to the movies on Friday night. You'll probably ask yourself a bunch of questions like "Are there any good movies out?", "Is there a movie playing at the right time?", "Do I have enough money to go to the movies (and buy a $17 bag of popcorn)?"

Suppose there *is* a movie that's playing at just the time you want to go. You then ask yourself a simple question: "Do I have enough money?" If the answer is yes, you'll head out to the movie. If the answer is no, you won't go. But on another Friday, you do have enough money, so you go to the movies. This scenario is just a simple example of how the circumstances around us affect the decisions we make.

JavaScript has the same kind of decision-making feature called *conditional statements*. At its most basic, a conditional statement is a simple yes or no question.

If the answer to the question is yes, your program does one thing; if the answer is no, it does something else. Conditional statements are one of the most important programming concepts: they let your programs react to different situations and behave intelligently. You'll use them countless times in your programming, but just to get a clear picture of their usefulness here are a few examples of how they can come in handy:

- **Form validation.** When you want to make sure someone filled out all of the required fields in a form ("Name," "Address," "E-mail", and so on), you'll use conditional statements. For example, if the Name field is empty, don't submit the form.
- **Drag and drop.** If you add the ability to drag elements around your Web page, you might want to check where the visitor drops the element on the page. For example, if he drops a picture onto an image of a trash can, you make the photo disappear from the page.
- **Evaluating input.** If you pop-up a window to ask a visitor a question like "Would you like to answer a few questions about how great this Web site is?", you'll want your script to react differently depending on how the visitor answers the question.

Figure 3-1 shows an example of an application that makes use of conditional statements.

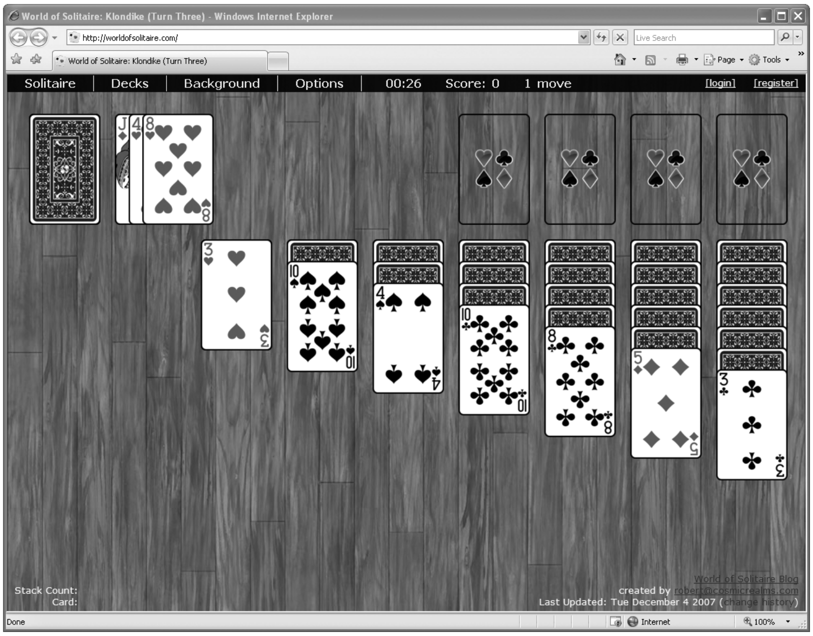
World of Solitaire: Klondike (Turn Three) - Windows Internet Explorer
http://worldofsolitaire.com/
Solitaire
Decks
Background
Options
00:26
Score: 0
1 move
[login]
[register]
Stack Count:
Card:
World of Solitaire Blog
created by robert@cosmicrealms.com
Last Updated: Tue December 4 2007 (change history)
Done
Internet
100%

***Figure 3-1:** It takes a lot of work to have fun. A JavaScript-based game like Solitaire (http://worldofsolitaire.com) demonstrates how a program has to react differently based on the conditions of the program. For example, when a player drags and drops a card, the program has to decide if the player dropped the card in a valid location or not, and then perform different actions in each case.*

Conditional Statement Basics

Conditional statements are also called "if/then" statements, because they perform a task only if the answer to a question is true: "*If* I have enough money *then* I'll go to the movies." The basic structure of a conditional statement looks like this:

```
if ( condition ) {
    // some action happens here
}
```

There are three parts to the statement: *if* indicates that the programming that follows is a conditional statement; the parentheses enclose the yes or no question, called the *condition* (more on that in a moment); and the curly braces ({ }) mark the beginning and end of the JavaScript code that should execute if the condition is true.

Note: In the code listed above, the "// some action happens here" is a JavaScript comment. It's not code that actually runs; it's just a note left in the program, and, in this case, points out to you, the reader, what's supposed to go in that part of the code. See page 71 for more on comments.

In many cases, the condition is a comparison between two values. For example, say you create a game that the player wins when the score is over 100. In this program, you'll need a variable to track the player's score and, at some point, you need to check to see if that score is more than 100 points. In JavaScript, the code to check if the player won could look like this:

```
if (score > 100) {
  alert('You won!');
}
```

The important part is *score > 100*. That phrase is the condition, and it simply tests whether the value stored in the *score* variable is greater than 100. If it is, then a "You won!" dialog box appears; if the player's score is less than or equal to 100, then the JavaScript interpreter skips the alert and moves onto the next part of the program. In addition to > (greater than), there are several other operators used to compare numbers (see Table 3-1).

Tip: Type two spaces (or press the tab key once) before each line of JavaScript code contained within a pair of braces. The spaces (or tab) indent those lines and makes it easier to see the beginning and ending brace and to figure out what code belongs inside the conditional statement. Two spaces is a common technique, but if four spaces make your code easier for you to read, then use four spaces. The examples in this book always indent code inside braces.

More frequently, you'll test to see if two values are equal or not. For example, say you create a JavaScript-based quiz, and one of the questions asks, "How many moons does Saturn have?" The person's answer is stored in a variable named *answer*. You might then write a conditional statement like this:

```
if (answer == 31) {
 alert('Correct. Saturn has 31 moons.');
}
```

The double set of equal signs (==) isn't a typo; it instructs the JavaScript interpreter to compare two values and decide whether they're equal. Remember, in JavaScript, a single equal sign is the *assignment operator* that you use to store a value into a variable:

```
var score = 0; //stores 0 into the variable score
```

Because the JavaScript interpreter already assigns a special meaning to a single equal sign, you need to use two equal signs whenever you want to compare two values to determine if they're equal or not.

You can also use the == (called the *equality operator*) to check to see if two strings are the same. For example, say you let the user type a color into a form, and if they type *'red'*, then you change the background color of the page to red. You could use the conditional operator for that:

```
if (enteredColor == 'red') {
 document.body.style.backgroundColor='red';
}
```

Note: In the code above, don't worry right now about how the page color is changed. You'll learn how to dynamically control CSS properties using JavaScript on page 186.

You can also test to see if two values aren't the same using the *inequality* operator:

```
if (answer != 31) {
 alert("Wrong! That's not how many moons Saturn has.");
}
```

The exclamation mark translates to "not", so != means "not equal to." In this example, if the value stored in *answer* is not 31, then the poor test taker would see the insulting alert message.

Table 3-1. *Use these comparison operators to test values as part of a conditional statement*

Comparison operator	What it means
==	**Equal to.** Compares two values to see if they're the same. Can be used to compare numbers or strings.
!=	**Not equal to.** Compares two values to see if they're *not* the same. Can be used to compare numbers or strings.

Table 3-1. *Use these comparison operators to test values as part of a conditional statement (continued)*

Comparison operator	What it means
>	**Greater than.** Compares two numbers and checks if the number on the left side is greater than the number on the right. For example, 2 > 1 is true, since 2 is a bigger number than 1, but 2 > 3 is false, since 2 isn't bigger than 3.
<	**Less than.** Compares two numbers and checks if the number on the left side is less than the number on the right. For example, 2 < 3 is true, since 2 is a smaller number than 3, but 2 < 1 is false, since 2 isn't less than 1.
>=	**Greater than or equal to.** Compares two numbers and checks if the number on the left side is greater than or the same value as the number on the right. For example, 2 >= 2 is true, since 2 is the same as 2, but 2 >= 3 is false, since 2 isn't a bigger number 3, nor is it equal to 3.
<=	**Less than or equal to.** Compares two numbers and checks if the number on the left side is greater than or the same value as the number on the right. For example, 2 <= 2 is true, since 2 is the same as 2, but 2 <= 1 is false, since 2 isn't a smaller number than 1, nor is 2 equal to 1.

The code that runs if the condition is true isn't limited to just a single line of code as in the previous examples. You can have as many lines of JavaScript between the opening and closing curly braces as you'd like. For example, as part of the JavaScript quiz example, you might keep a running tally of how many correct answers the test-taker gets. So, when the Saturn question is answered correctly, you also want to add 1 to the test-taker's total. You would do that as part of the conditional statement:

```
if (answer == 31) {
 alert('Correct. Saturn has 31 moons.');
 numCorrect = numCorrect + 1;
}
```

And you could add additional lines of JavaScript code between the braces as well—any code that should run if the condition is true.

Adding a Backup Plan

But what if the condition is false? The basic conditional statement in the previous section doesn't have a backup plan for a condition that turns out to be false. In the real world, as you're deciding what to do Friday night and you don't have enough money for the movies, you'd want to do something *else*. An *if* statement has its own kind of backup plan, called an *else clause*. For example, say as part of the

POWER USERS' CLINIC

The Return of the Boolean

On page 42, you learned about the Boolean values–*true* and *false*. Booleans may not seem very useful at first, but you'll find out they're essential when you start using conditional statements. In fact, since a condition is really just a yes or no question, the answer to that question is a Boolean value. For example, check out the following code:

```
var x = 4;
if ( x == 4 ) {
    //do something
}
```

The first line of code stores the number 4 into the variable *x*. The condition on the next line is a simple question: is the value stored in *x* equal to 4? In this case, it is, so the JavaScript between the curly braces runs. But here's what really happens in between the parentheses: the JavaScript interpreter converts the condition into a Boolean value; in programming-speak, the interpreter *evaluates* the condition. If the condition evaluates to *true* (meaning the answer to the question is yes), then the code between the braces runs. However, if the condition evaluates to *false*, then the code in the braces is skipped.

One common use of Booleans is to create what's called a *flag*–a variable that marks whether something is true. For example, when validating a form full of visitor submitted information, you might start by creating a *valid* variable with a Boolean value of *true*–this means you're assuming, at first, that they filled out the form correctly. Then, you'd run through each form field, and if any field is missing information or has the wrong type of information, you change the value in *valid* to *false*. After checking all of the form fields, you test what's stored in *valid*, and if it's still true, you submit the form. If it's not true (meaning one or more form fields were left blank), you display some error messages and prevent the form from submitting:

```
var valid = true;
// lot of other programming gunk happens
in here
// if a field has a problem then you set
valid to false
if (valid) {
 //submit form
} else {
 //print lots of error messages
}
```

JavaScript testing script, you want to notify the test-taker if he gets the answer right, or if he gets it wrong. Here's how you can do that:

```
if (answer == 31) {
 alert('Correct. Saturn has 31 moons.');
 numCorrect = numCorrect + 1;
} else {
 alert("Wrong! That's not how many moons Saturn has.");
}
```

This code sets up an either/or situation; only one of the two messages will appear. If the number 31 is stored in the variable *answer*, then the "correct" alert appears; otherwise, the "wrong" alert appears.

To create an *else* clause, just add "else" after the closing brace for the conditional statement followed by another pair of braces. You add the code that should execute if the condition turns out to be false in between the braces. Again, you can have as many lines of code as you'd like as part of the *else* clause.

Testing More Than One Condition

Sometimes you'll want to test several conditions and have several possible outcomes: think of it like a game show where the host says, "Would you like the prize behind door #1, door #2, or door #3?" You can only pick one. In your day-to-day activities, you also are often faced with multiple choices like this one.

For example, return to the "What should I do Friday night?" question. You could expand your entertainment options based on how much money you have and are willing to spend. For example, you could start off by saying, "If I have more than $50 I'll go out to a nice dinner and a movie (and have some popcorn too)." If you don't have $50, you might try another test: "If I have $35 or more, I'll go to a nice dinner." If you don't have $35, then you'd say, "If I have $12 or more, I'll go to the movies." And finally, if you don't have $12, you might say, "Then I'll just stay at home and watch TV." What a Friday night!

JavaScript lets you perform the same kind of cascading logic using *else if* statements. It works like this: you start with an *if* statement, which is option number 1; you then add one or more *else if* statements to provide additional questions that can trigger additional options; and finally, you use the *else* clause as the fallback position. Here's the basic structure in JavaScript:

```
if (condition) {
 // door #1
} else if (condition2) {
 // door #2
} else {
 // door #3
}
```

This structure is all you need to create a JavaScript "Friday night planner" program. It asks visitors how much money they have, and then determines what they should do on Friday (sound familiar?). You can use the *prompt( )* command that you learned about on page 55 to collect the visitor's response and a series of if/else if statements to determine what he should do:

```
var fridayCash = prompt('How much money can you spend?', '');
if (fridayCash >= 50) {
 alert('You should go out to a dinner and a movie.');
} else if (fridayCash >= 35) {
 alert('You should go out to a fine meal.');
} else if (fridayCash >= 12) {
 alert('You should go see a movie.');
} else {
 alert('Looks like you'll be watching TV.');
}
```

Here's how this program breaks down step-by-step: The first line opens a prompt dialog asking the visitor how much he can spend. Whatever the visitor types is stored in a variable named *fridayCash*. The next line is a test: Is the value the visitor typed 50 or more? If the answer is yes, then an alert appears telling him to go get a meal and see a movie. At this point, the entire conditional statement is done. The JavaScript interpreter skips the next *else if* statement, the following *else if* statement, and the final *else* clause. With a conditional statement, only one of the outcomes can happen, so once the JavaScript interpreter encounters a condition that evaluates to *true*, then it runs the JavaScript code between the braces for that condition and skips everything else within the conditional statement.

Suppose the visitor typed 25. The first condition, in this case, wouldn't be true, since 25 is a smaller number than 50. So the JavaScript interpreter skips the code within the braces for that first condition and continues to the *else if* statement: "Is 25 greater than or equal to 35?" Since the answer is no, it skips the code associated with that condition and encounters the next *else if*. At this point, the condition asks if 25 is greater than or equal to 12; the answer is yes, so an alert box with the message, "You should go see a movie" appears and the program ends, skipping the final *else* clause.

Tip: There's another way to create a series of conditional statements that all test the same variable, as in the *fridayCash* example. *Switch* statements do the same thing, and you'll learn about them on page 499.

More Complex Conditions

When you're dealing with many different variables, you'll often need even more complex conditional statements. For example, when validating a required email address field in a form, you'll want to make sure both that the field isn't empty and that the field contains an email address (and not just random typed letters). Fortunately, JavaScript lets you do these kinds of checks as well.

Making sure more than one condition is true

You'll often need to make decisions based on a combination of factors. For example, you may only want to go to a movie if you have enough money *and* there's a movie you want to see. In this case, you'll go only if two conditions are true; if either one is false, then you won't go to the movie. In JavaScript, you can combine conditions using what's called the *logical AND operator,* which is represented by two ampersands (&&). You can use it between the two conditions within a single conditional statement. For example, if you want to check if a number is between 1 and 10, you can do this:

```
if (a < 10 && a > 1) {
 //the value in a is between 1 and 10
 alert("The value " + a + " is between 1 and 10");
}
```

In this example, there are two conditions: *a < 10* asks if the value stored in the variable *a* is less than 10; the second condition, *a > 1*, is the same as "Is the value in *a* greater than 1?" The JavaScript contained between the braces will run only if *both* conditions are true. So if the variable *a* has the number 0 stored in it, the first condition (*a < 10*) is true (0 is less than 10), but the second condition is false (0 is not greater than 1).

You're not limited to just two conditions. You can connect as many conditions as you need with the && operator:

```
if (b>0 && a>0 && c>0) {
 // all three variables are greater than 0
}
```

This code checks three variables to make sure all three have a value greater than 0. If just one has a value of 0 or less, then the code between the braces won't run.

Making sure at least one condition is true

Other times you'll want to check a series of conditions, but you only need one to be true. For example, say you've added a keyboard control for visitors to jump from picture to picture in a photo gallery. When the visitor presses the N key, the next photo appears. In this case, you want her to go to the next picture when she types either *n* (lowercase) or, if she has the Caps Lock key pressed, *N* (uppercase). You're looking for a kind of either/or logic: either this key *or* that key was pressed. The *logical OR operator*, represented by two pipe characters (||), comes in handy:

```
if (key == 'n' || key == 'N') {
 //move to the next photo
}
```

Tip: To type a pipe character, press Shift-\. The key that types both backslashes and pipe characters is usually located just above the Return key.

With the OR operator, only one condition needs to be true for the JavaScript that follows between the braces to run.

As with the AND operator, you can compare more than two conditions. For example, say you've created a JavaScript racing game. The player has a limited amount of time, a limited amount of gas, and a limited number of cars (each time he crashes he loses one car). To make the game more challenging, you want it to come to an end when any of these three things runs out:

```
if (gas <= 0 || time <= 0 || cars <= 0) {
 //game is over
}
```

When testing multiple conditions, it's sometimes difficult to figure out the logic of the conditional statement. Some programmers group each condition in a set of parentheses to make the logic easier to grasp:

```
 if ((key == 'n') || (key == 'N')) {
  //move to the next photo
}
```

To read this code, simply treat each grouping as a separate test; the results of the operation between parentheses will always turn out to be either true or false.

Negating a condition

If you're a Superman fan, you probably know about Bizarro, an anti-hero who lived on a cubical planet named Htrae (Earth spelled backwards), had a uniform with a backwards S, and was generally the opposite of Superman in every way. When Bizarro said "Yes," he really meant "No"; and when he said "No," he really meant "Yes."

JavaScript programming has an equivalent type of character called the *NOT* operator, which is represented by an exclamation mark (*!*). You've already seen the NOT operator used along with the equal sign to indicate "not equal to": !=. But the NOT operator can be used by itself to completely reverse the results of a conditional statement; in other words, it can make false mean true, and true mean false.

You use the NOT operator when you want to run some code based on a negative condition. For example, say you've created a variable named *valid* that contains a Boolean value of either *true* or *false* (see the box on page 80). You use this variable to track whether a visitor correctly filled out a form. When the visitor tries to submit the form, your JavaScript checks each form field to make sure it passes the requirements you set up (for example, the field can't be empty and it has to have an email address in it). If there's a problem, like the field is empty, you could then set *valid* to false (*valid = false*).

Now if you want to do something like print out an error and prevent the form from being submitted, you can write a conditional statement like this:

```
if (! valid) {
 //print errors and don't submit form
}
```

The condition *! valid* can be translated as "if not valid," which means if *valid* is false, then the *condition* is *true*. To figure out the results of a condition that uses the NOT operator, just evaluate the condition without the NOT operator, then reverse it. In other words, if the condition results to *true*, the ! operator changes it to *false*, so the conditional statement doesn't run.

As you can see the NOT operator is very simple to understand (translated from Bizarro-speak: it's very confusing, but if you use it long enough, you'll get used to it).

Nesting Conditional Statements

In large part, computer programming entails making decisions based on information the visitor has supplied or on current conditions inside a program. The more decisions a program makes, the more possible outcomes and the "smarter" the program seems. In fact, you might find you need to make further decisions *after* you've gone through one conditional statement.

Suppose, in the "What to do on Friday night?" example, you want to expand the program to include every night of the week. In that case, you need to first determine what day of the week it is, and then figure out what to do on that day. So you might have a conditional statement asking if it's Friday, and if it is, you'd have another series of conditional statements to determine what to do on that day:

```
if (dayOfWeek == 'Friday') {
 var fridayCash = prompt('How much money can you spend?', '');
 if (fridayCash >= 50) {
  alert('You should go out to a dinner and a movie.');
 } else if (fridayCash >= 35) {
  alert('You should go out to a fine meal.');
 } else if (fridayCash >= 12) {
  alert('You should go see a movie.');
 } else {
  alert('Looks like you'll be watching TV.');
 }
}
```

In this example, the first condition asks if the value stored in the variable *dayOfWeek* is the string 'Friday'. If the answer is yes, then a prompt dialog appears, gets some information from the visitor, and another conditional statements is run. In other words, the first condition *(dayOfWeek == 'Friday')* is the doorway to another series of conditional statements. However, if *dayOfWeek* isn't 'Friday', then the condition is false and the nested conditional statements are skipped.

Tips for Writing Conditional Statements

The example of a nested conditional statement in the last section may look a little scary. There are lots of ( ), {}, elses, and ifs. And if you happen to mistype one of the crucial pieces of a conditional statement, your script won't work. There are a few things you can do as you type your JavaScript that can make it easier to work with conditional statements.

- **Type both of the curly braces before you type the code inside them.** One of the most common mistakes programmers make is forgetting to add a final brace to a conditional statement. To avoid this mistake, type the condition and the braces first, then type the JavaScript code that executes when the condition is true. For example, start a conditional like this:

```
if (dayOfWeek=='Friday') {


}
```

In other words, type the *if* clause and the first brace, hit Return twice, and then type the last brace. Now that the basic syntax is correct, you can click in the empty line between the braces and add JavaScript.

- **Indent code within braces.** You can better visualize the structure of a conditional statement if you indent all of the JavaScript between a pair of braces:

```
if (a < 10 && a > 1) {
  alert("The value " + a + " is between 1 and 10");
}
```

By using several spaces (or pressing the Tab key) to indent lines within braces, it's easier to identify which code will run as part of the conditional statement. If you have nested conditional statements, indent each nested statement:

```
if (a < 10 && a > 1) {
  //first level indenting for first conditional
  alert("The value " + a + " is between 1 and 10");
  if (a==5) {
    //second level indenting for 2nd conditional
    alert(a + " is half of ten.");
  }
}
```

- **Use == for comparing equals.** When checking whether two values are equal, don't forget to use the equality operator, like this:

```
if (name == 'Bob') {
```

A common mistake is to use a single equal sign, like this:

```
if (name = 'Bob') {
```

A single equal sign stores a value into a variable, so in this case, the string 'Bob' would be stored in the variable *name*. The JavaScript interpreter treats this step as true, so the code following the condition will always run.

Tutorial: Using Conditional Statements

Conditional statements will become part of your day-to-day JavaScript toolkit. In this tutorial, you'll try out conditional statements to control how a script runs.

Note: See the note on page 27 for information on how to download the tutorial files.