JavaScript and jQuery Course
Instructor Teresa Pelkie
Session 3

Topic: How to Code Control Statements and More Operators

Chapter 2: Getting started with JavaScript page 72 to end

Control Structures / Conditionals pages 72-75

A *control structure* is a block of programming code that analyzes variables and chooses which code to execute next based on values of the variables. An *if* statement is a commonly used form of a control structure.

The "if" Conditional Statement - pages 74-75

The if statement is used to perform different actions based on different conditions or to make a distinction between different possibilities.

```
if (condition)
{
    // code to be executed if condition is true
}
```

- If the condition **evaluates to true** the code will be executed
- If the condition **evaluates to false** the code will **not** be executed
- there is NO semicolon after the condition
- conditionals are usually used with *relational operators* (also called *comparison operators*) - page 73
- think of the `if` statement as comparing two things
- conditionals are also used with *logical operators* - page 73

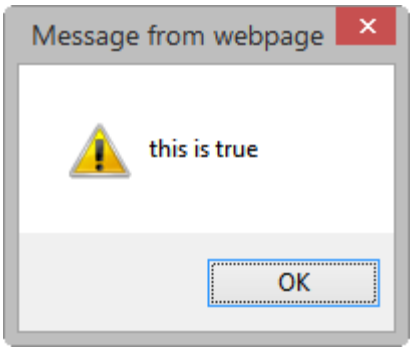
Examples using Relational Operators (also called Comparison Operators) - page 73

Operator	Meaning
==	Equal To
!=	Not Equal To
<	Less Than
>	Greater Than
<=	Less Than or Equal To
>=	Greater Than or Equal To

Example 1 - string comparison for equality

```
var myName = "Teresa";

if (myName == "Teresa")        // note that there is no semicolon here
{
    alert("this is true");     // note that there is a semicolon here
}
```



Note that the `alert()` method can be used to show the results of the `if` comparison.

Example 2 - string comparison for equality

```
var myName = "Mary";

if (myName == "Teresa")
{
    alert("this is true");
}
```

Result: nothing will happen as the condition is not true

Example 3 - testing a numeric variable compared to a literal numeric value

```
var  x = 20;

if (x >= 20)
{
    alert("this is true");
}
```

Result: the `alert()` message is displayed, the condition is true

Examples using Logical Operators - page 73

Operator	Name	Where found on keyboard
&&	AND	Above 7 key (top row)
||	OR	Above \ key (above the Enter key)
!	NOT	Above 1 key (top row)

Example 4 - testing two numeric variables to determine if both comparisons are true (AND)

```
var  x = 6;
var  y = 3

if (x < 10 && y > 1)
{
    alert("this is true");
}
```

Result: the condition is `true` (`x` is less than `10` `AND` `y` is greater than `1`)

Example 5 - test to determine if two numeric variables are not equal (NOT)

```
var  x = 6;
var  y = 3

// the following if shows one way to code this test
if ( !(x == y) )
{
    alert("this is true");
}

// this is another way to code the test
if (x != y)
{
    alert("this is true");
}
```

Result: in the first `if`, the condition is `true` ("`x` is equal to `y`" is not true, so adding the `!` (`not`) operator) means that the condition can be read as "is `x` not equal to `y`" -- which is `true`). In the second `if`, the "not equal" operator is used to directly compare the values.

Example 6 - testing two numeric variables to determine if either comparison is true (OR)

```
var  x = 6;
var  y = 3

if ( x == 5 || y == 3)
{
    alert("this is true");
}
```

Result: the condition is `true`. `y` is equal to `3`. When using an `OR` operator, only one of the conditions has to be true.

Example 7 - A compound condition, possible confusion of order of precedence

```
var x = 6;
var y = 3;
var z = 2;

if ( x == 5 && y == 3 || z == 2)
{
    alert("this is true");
}
```

Question: is this evaluated as

```
[x AND y] OR z
```

or

```
x AND [y OR z]
```

Result: the test is evaluated as `[x AND y] OR z`

The first test that is performed is

```
x == 5 && y == 3
```

That evaluates to `false`, because `x` is not equal to `5` and `y` is equal to `3`. In an `AND` test, both sides of the test have to evaluate to `true` in order for the result of the test to be `true`.

Next, the following test is performed:

```
false OR z == 2
```

This test evaluates to `true`. In an `OR` test, only one side of the test has to evaluate to `true` in order for the test to be `true` (both sides can also be `true`, and the test will be `true`). In this test, the left side is `false`, but the right size (`z == 2`) is true, so the entire condition evaluates to `true`.

Order or precedence rule: if you do not use parentheses to control the order or precedence used in evaluating the condition, the order of precedence is `NOT`, then `AND`, then `OR`.

Example 8 - A compound condition, using parentheses to control the order of precedence

```
var x = 6;
var y = 3;
var z = 2;

if ( (x == 5 && y == 3) || z == 2 )
{
    alert("this is true");
}
```

Result: this condition is evaluated as follows:

```
[x AND y] OR z
```

evaluates to: `[false AND true] OR true`

evaluates to: `false OR true`

evaluates to: `true`

Example 9 - A compound condition, using parentheses to control the order of precedence

```
var x = 6;
var y = 3;
var z = 2;

if ( x == 5 && (y == 3 || z == 2) )
{
    alert("this is true");
}
```

Result: this condition is evaluated as follows:

```
      x AND [y OR z]
```

evaluates to: `false AND [true OR true]`

evaluates to: `false AND true`

evaluates to: `false`

isNaN - "is Not a Number", page 73

isNaN() - JavaScript function used to test if a expression is a valid number

```
alert( isNaN(100) );    // will return false because  it is a number
alert( isNaN("100") ); // will return true because it is not a number,
                       // but a string (the value is in quotations)
```

The "if / else" Conditional Statement - page 75

See also http://www.w3schools.com/js/js_if_else.asp

The `if/else` statement allows us to execute some code when the condition is true and also when the condition is not true.

Example 10: an if/else test that handles 2 possible outcomes

```
if (condition)
{
    // code to be executed if condition is true;
}
else
{
    // code to be executed if condition is not true;
}
```

The "if / else if" Conditional Statement - page 75

You can have additional `else if` conditions

Example 11: an if / else if / else test that handles more than 2 possible outcomes

```
if (condition)
{
    // code to be executed if condition is true;
}
else if (condition2)
{
    // code to be executed if condition2 is true;
}
else
{
    // code to execute if neither is true
}
```

When the **first true condition is found**, the code for that condition is executed and the **remaining tests are skipped** (control passes to the next executable statement after the final closing brace for the `if` condition).

Example 12: alternate brace style (opening braces on the same line)

```
if (condition) {
    // code to be executed if condition is true;
}
else if (condition2) {
    // code to be executed if condition2 is true;
}
else {
    // code to execute if neither is true
}
```

Remarks: you can use **either brace style** in your code (opening brace is on next line or opening brace is on same line). Use the same brace style consistently.

Example: **do not** write code like this:

```
if (condition) {
    // code to be executed if condition is true;
}
else if (condition2)
{   // this brace is on the next line
    // code to be executed if condition2 is true;
}
else {
    // code to execute if neither is true
}
```

Example 13: brace example when there is only one statement for each condition

```
if      (condition ) { // code to be executed if condition is true;  }
else if (condition2) { // code to be executed if condition2 is true; }
else                 { // code to execute if neither is true         }
```

Remarks: if each condition has only one statement to be executed, you can possibly line up each block as shown here. This may make it easier to read the code to determine what happens when a condition is `true`.

Remarks about brace style

You should pick a brace style (opening brace on same line or opening brace on next line) and **be consistent** with your brace style **throughout your code**.

If you use "opening brace on the next line", the opening brace should appear under the statement keyword. **Example**: the opening brace character is aligned under the `if` keyword or the `else if` keyword or the `else` keyword.

The closing brace character should always be aligned under the keyword.

☞ PAY ATTENTION TO HOW YOU FORMAT THE OPENING AND CLOSING BRACES. ☜

It is a **very common error** to omit a closing brace or to have too many closing brace characters. If you line up the closing brace characters under the keyword, it will be easier to match up the opening brace with the required closing brace.

Here is a quick tip that will make sure that you **always** have a closing brace for an opening brace: **as soon as you type the opening brace, type the closing brace** (some editing programs will provide the closing brace for you). **Only after** you type the closing brace, go back and type the statement(s) that are to be included in the block (between the braces).

The "switch" conditional statement - not in the book

see http://www.w3schools.com/js/js_switch.asp

Use the `switch` statement when you have many conditions to evaluate. Use `switch` instead of a long series of `if / else if / else` statements.

```
switch (condition to evaluate)  // note that there is no semicolon here
{
    case 1:      // note that there is a colon here
        // code to execute if case 1 is true
        break;   // note that there is a semicolon here - the break is required

    case 2:
        // code to execute if case 2 is true
        break;

    case 3:
        // code to execute if case 3 is true
        break;

    case 4:
        // code to execute if case 4 is true
        break;

    default:
        // code to execute if none of the above are true - the default
}
```

Example 14: using the switch statement

```
var yourGrade = 'A';

switch (yourGrade)
{
    case 'A':
        document.write("Good job<br>");
        break;

    case 'B':
        document.write("Pretty good<br>");
        break;

    case 'C':
        document.write("Passed<br>");
        break;

    case 'D':
        document.write("Not so good<br>");
        break;

    case 'F':
        document.write("Failed<br>");
        break;

    default:
        document.write("Unknown grade<br>")
}
```

The Ternary Operator - not in the book

see also (end of page) - http://www.w3schools.com/js/js_comparisons.asp

The *ternary operator* shortens an `if/else` statement into a single statement. This provides a way to write the standard `if/else` condition using less code.

You may see it in other people's code so I am including it here.

Example 15: a traditional if/else statement

```
var a = 5;
var b = 3;

if (a > b)
{
    alert("true");
}
else
{
   alert("false");
}
```

Example 16: using a ternary operator in place of the if/else

```
var a = 5;
var b = 3;

(a > b) ? alert("true") : alert("false");
```

Format of the ternary operator

```
(condition to be tested) ? result_if_true : result_if_false;
```

The `result_if_true` and `result_if_false` must contain a single statement. If you need to execute more than one statement, use the `if/else` construct.

Loops - pages 76-79

Loops let you run a block a code over and over again.

In JavaScript, there are two different kind of loops:

- **for** - loops through a block of code *for* a specified number of times
- **while** - loops through a block of code *while* a specified condition is true

For Loop - pages 78-79

Used when you know in advance or can determine how many times the script should run.

There are four parts to a JavaScript `for` loop:

- The **counter variable** is defined and usually used only in the `for` loop to count how many times the `for` loop has looped. **`i`** is a commonly used name for the counter variable ("how many time to *i*terate").
- The **conditional statement** is used to determine how many times the loop executes. The condition usually compares the counter variable to an ending value.
- The counter variable is **incremented after each iteration of the loop** in the **increment section** of the `for` loop.
- The **code that is executed for each loop**

Basic syntax for a for loop

```
for ( initial expression; condition; increment expression) // note the semicolons
{
    code to execute
}
```

Example 17: the for loop

```
var i;

for ( i = 0; i < 5; i++ )
{
    document.write("The number is " + i + "<br>");
}
```

Remarks: This example defines a `for` loop that starts with `i = 0`. The loop will run as long as `i` is less than `5`. `i` will increase by `1` each time the loop is executed (the value of `i` is incremented after each iteration).

Example 18: a for loop, define the counter variable inside the loop

```
for (var i = 0; i < 5; i++)
{
    document.write("The number is " + i + "<br>");
}
```

Remarks: this example shows the most common way to code a `for` loop. In this example the counter variable `i` is defined **inside the loop**. That means that the value of `i` is only known (or "scoped") to the loop. This is the **preferred way to code** the counter variable, as you usually do not need to work with it outside of the loop.

It is always a good idea to limit the scope of a variable to the smallest block of code that it is used in. In this example, the counter variable `i` is only used inside the `for` loop, so it is scoped to only that loop.

While Loop - pages 76-77

A `while` loop executes a block of code as long as a specified condition is `true`. There are three key parts to a JavaScript while loop:

- The **conditional statement which must be true** for the code to be executed.
- The **code to be executed** if the condition is `true`.
- The **increment** operator to increment the counter variable.

When a `while` loop begins, the JavaScript interpreter determines if the condition statement is `true`. If it is, the code between the curly braces is executed. At the end of the block, control is passed back to the condition statement and the loop begins again, as long as the condition evaluates to `true`.

Note: if the condition always evaluates to `true`, you will **never exit** the `while` loop. Be careful when using `while` loops!

Basic Syntax for a while loop

```
while (condition) {
    // code to be executed
    //increment operator
}
```

Example 19: a while loop

```
var i = 0;

while (i < 5)
{
    document.write("The number is " + i + "<br>");
    i++;
}
```

Explanation

The example shows a `while` loop that starts with `i = 0`. The loop will continue to run as long as `i` is less than `5`. `i` will increase by `1` each time the loop runs. The loop is exited when `i` is equal to `5` (the condition evaluates to `false`).

do while loop - page 77

The `do while` loop is a variant of the `while` loop. This loop will execute the block of code once, before checking if the condition is `true`, then it will repeat the loop as long as the condition is `true`.

Note: the `do while` loop will always run at least once!

Basic Syntax for a do while loop

```
do {
    // code to be executed
    // increment operator
}
while (condition);
```

As an alternative, the `while` keyword is often put on the same statement as the closing brace:

```
do {
    // code to be executed
    // increment operator
} while (condition);
```

Example 20: a do while loop

```
var i = 0;

do
{
    document.write("The number is " + i + "<br>");
    i++;
}
while (i < 5);
```

Explanation

This example uses a `do while` loop. The loop is always executed at least once, even if the condition is `false`, because the statements are executed **before** the condition is tested.

prompt() method of the window object - page 61

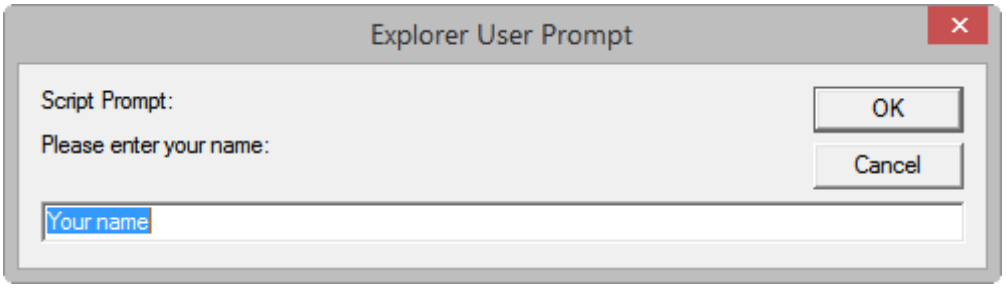
This is used in the textbook examples to obtain input - pages 88-89

The `prompt()` method displays a dialog box that prompts the user for input. This method returns the string that the user enters.

Note that whatever is entered is treated as a string. If you need a numeric value, you can use the `parseInt()` function to return an integer or the `parseFloat()` function to return a value that contains decimal positions.

Example 21: using the prompt() method

```
var theAnswer = prompt("Please enter your name:", "Your name");
alert(theAnswer);
```



Note that the first argument is shown above the input area. The second argument is optional, if you include it, it is shown as the initial text inside the input area. You can use the second argument to display additional instructions to the user or to supply a default value for the prompt.

Finding Errors in Your Code - page 85

Review this section to see how to use the Error Console in FireFox.

Sample Files for this session

File Name	Description
assignment_02_start_page.html	HTML file that contains the starting HTML for the assignment. You will add the JavaScript code that you develop to this page.
03_do_while_loop.html	Page that shows an exmaple of a JavaScript "do-while" loop.
03_for_loop.html	Page that shows an example of a JavaScript "for" loop.
03_if_else_statements.html	Page that shows an example of JavaScript "if/else if/else" statements.
03_if_statements.html	Page that shows an example of a JavaScript "if" statement.
03_innerHTML_DOM.html	Page that shows an example of how to change what is displayed on the page using the innerHTML property.
03_prompt.html	Page that shows an example of how to use the prompt function to get a value from the user.
03_switch.html	Page that shows an example of the switch statement. This can frequently be used as an alternative to if/else statements.
03_ternary_operator.html	Page that shows an example of the ternary operator. This can be used as an alternative to an if/else statement.
03_textbook_mpg.html	This page shows the calculations used for the MPG example.
03_textbook_test_scores.html	This page contains the calculations used for the Test Scores example.
03_while_loop.html	This page shows an example of a JavaScript "while" loop.