**Control structures**

***The 'if' statement***

if( myVariable == 2 ) {

myVariable = 1;

} else {

myVariable = 0;

}

If myVariable had been 2 it would now be 1. If it had been anything other than 2 it would now be 0.

'If' statements can also test for the occurence of a child object of an object that may not exist. For example, some browsers provide document.body.style while some older browsers do not even provide document.body. In these browsers, writing 'if( document.body.style )' would just produce an error (see the section on '[Variables](http://www.howtocreate.co.uk/tutorials/javascript/variables)' subsection '[Avoiding errors with variables](http://www.howtocreate.co.uk/tutorials/javascript/objectoriented#avoidvarerror)'). In order to solve this problem, we could write this:

if( document.body ) {

if( document.body.style ) { etc. }

}

However, the && operator has a useful feature that we can use here to combine the two 'if' statements into one:

if( document.body && document.body.style ) { etc. }

The first test would be false, so the browser would not proceed to the second. This is known as a short-circuit. The || operator has a similar feature, but it will only evaluate the second test if the first one fails.

JavaScript understands that if the '{' and '}' (curly brace) characters are left out, then only the next command belongs to that statement:

if( x < 5 )

x++;

window.alert(x);

Here, the alert will always happen reguardless of x, but x will only be incremented if x is less than 5. This may seem convenient, as it allows you to make your code a tiny bit shorter, but I recommend avoiding this syntax. It makes your code harder to read, especially if you start nesting your control structures. It also makes it easy to forget to put them in when you needed them, and also makes debugging code much harder, since you will need to go back through your code to add them so that you can add extra debugging tests. It is best to always use the curly braces, even if they are optional.

As always, there is an exception. Nested 'if' statements like this can start to get difficult to manage:

if( myVariable == 2 ) {

myVariable = 1;

} else {

if( myVariable == 5 ) {

myVariable = 3;

} else {

myVariable = 4;

}

}

By strategically removing curly braces, that can usefully be reduced to this construct (which you may recognise from other programming languages) - note that 'else if' is *not* written as 'elseif':

if( myVariable == 2 ) {

myVariable = 1;

} else if( myVariable == 5 ) {

myVariable = 3;

} else {

myVariable = 4;

}

***The 'for' loop***

This is one of the most common constructs in use. Typically, it is used to cycle through the contents of an array, or to create a specific number of new objects, but it can do many more useful things if needed. The syntax of the 'for' loop is as follows:

for( starting\_initialise; continue\_as\_long\_as\_condition; do\_this\_each\_time )

starting\_initialise

This is where you define new variables that you will use in the loop, typically for use with incremental counting. As with all variables, you must declare them (if you have not done so already). You can define multiple variables if needed, using:

var myVariable1 = value, myVariable2 = another\_value;

These variables are not restricted to being inside the 'for' loop, and will be available to all code after the loop (in the same scope as the loop).

continue\_as\_long\_as\_condition

This is where you define the conditons under which the loop should continue to execute. The syntax is exactly the same as for the 'if' statement, so you can apply more than one continue condition by using the && or || operators:

myVariable1 <= 5 && myVariable2 >= 70;

If the condition is not satisfied when the for loop begins, then it will never loop through it.

do\_this\_each\_time

Once the end of the loop is reached, it will do whatever you tell it to here. Typically, this is used to increment or decrement a stepping variable, and it is possible to perform actions on more than one variable by separating them with a comma:

myVariable1++, myVariable2 -= 4

The following is a full example.

for( var myVariable = 1; myVariable <= 5; myVariable++ ) {

myArray[myVariable] = 1;

}

myArray[1] to myArray[5] are now 1.

***The 'for - in' loop***

The 'for - in' loop is used to cycle through all exposed properties of an object (or array). Every time you create properties or methods on an object, these will be added to the list of properties that will be exposed. Most internal properties (the ones that JavaScript creates) will also be exposed, but JavaScript engines are allowed to hide internal properties and methods if they want to. You should not rely on any specific behaviour here, but note that some browsers will give the internal properties and methods of intrinsic objects, and some will not.

Again, you should declare the variable names that you use, if you have not done so already. The syntax of the 'for - in' loop is as follows:

for( var myVariable in anObjectOrArray ) {

This will run through the loop, once for each exposed property in anObjectOrArray. Each time it loops, it assigns the next property name as a string value to myVariable. You can then use array notation to access the value of that property. The following example writes all the exposed properties of the document object:

for( var myVariable in document ) {

document.write( myVariable + ' = ' + document[myVariable] + '<br>' );

}

Note that if you use this loop on an array, it will list the numbered and named keys, including the internal 'length' property. It is very easy to make mistakes here, so be careful not to mistake these property types for each other.

***The 'while' loop***

The 'while' loop is identical in behaviour to the 'for' loop, only without the initial setup, and loop-end actions. It will continue to run as long as the condition is satisfied:

var myVariable = 1;

while( myVariable <= 5 ) {

myArray[myVariable] = 1;

myVariable++;

}

myArray[1] to myArray[5] are now 1.

Using a feature of the increment (and decrement) operator here, it is possible to shorten the code inside the loop to be just 'myArray[myVariable++] = 1;', and it would have exactly the same effect. Firstly, it would use the value of myVariable to index the array cell, then it would increment myVariable.

This also works in reverse; 'myArray[++myVariable] = 1;'. Firstly, it would increment the value of myVariable, then it would use the new value to index the array cell. If I had done this, myArray[2] to myArray[6] would now be 1.

These features also work outside loops, but this is where you will most commonly see them, so I have included them here.

***The 'do - while' loop***

This is similar to the while loop, but with an important difference. The condition is evaluated at the end of the loop, meaning that even if the condition is never satisfied, it will still run through the loop at least once.

var myVariable = 1;

do {

myArray[myVariable] = 1;

myVariable++;

} while( myVariable <= 5 );

myArray[1] to myArray[5] are now 1.

***The 'switch' statement***

The 'switch' statement is like repeated 'if' statements, testing a single value to see if it matches one of a set of values:

switch(myVar) {

case 1:

//if myVar is 1 this is executed

case 'sample':

//if myVar is 'sample' (or 1, see the next paragraph)

//this is executed

case false:

//if myVar is false (or 1 or 'sample', see the next paragraph)

//this is executed

default:

//if myVar does not satisfy any case, (or if it is

//1 or 'sample' or false, see the next paragraph)

//this is executed

}

If a case is satisfied, the code beyond that case will also be executed unless the break statement is used. In the above example, if myVar is 1, the code for case 'sample', case false and default will all be executed as well. The solution is to use break; as follows (The use of the break statement is described below).

switch(myVar) {

case 1:

//if myVar is 1 this is executed

break;

case 'sample':

//if myVar is 'sample' this is executed

break;

case false:

//if myVar is false this is executed

break;

default:

//if myVar does not satisfy any case, this is executed

//break; is unnecessary here as there are no cases following this

}

***The 'with' statement***

Take for example the following example:

x = Math.round( Math.LN2 + Math.E + Math.pow( y, 4 ) );

Using the 'with' statement, this can be replaced with:

with( Math ) {

x = round( LN2 + E + pow( y, 4 ) );

}

Note that the 'with' statement brings extra variable names into the current scope. In the example above, if I already had a variable called pow before the 'with' statement, this variable would be unavailable inside the with statement, as it would have been replaced by the method of the Math object (as would any other variables that matched property or method names). Once the 'with' statement is complete, the old variables would become available again.

***The quick 'if' statement***

This is known as the conditional or ternary operator, and is an easy way to assign different values to a variable, depending on a condition.

var myVariable = document.getElementById ? 1 : 0;

This is identical to:

if( document.getElementById ) {

var myVariable = 1;

} else {

var myVariable = 0;

}

***The try - catch - finally statement***

* Netscape 4, Internet Explorer 4 and WebTV do not support this structure and will produce errors if you use it.

The 'try - catch - finally' control stucture allows you to detect errors and quietly deal with them without producing error messages or aborting the script, and in fact, without even interrupting the flow of the script that is running. This makes it superior to the original way of handling script errors (without error messages) where scripts are completely aborted: