**The return Keyword**

A JavaScript function, when executed, may send some piece of data back to whatever process originally called it using the return keyword.

For instance, I might make a calculation function which takes in numbers as arguments, performs a complex calculation (a mortgage payment calculation would be an appropriate example), and spits out an answer. The alert() method that you have used so far to spit out answers is obviously inadequate to real work; the return keyword is what you will need to use to get information out of a function.

Built-in JavaScript methods use the return keyword to spit out many different types of values, which you will need to intercept and process for one reason or another, placing those returned values into local or global variables so that you can use and manipulate that information. In addition, you will be called upon to return values yourself, not only between functions, but also to the web browser. And it's not only numbers, strings, or booleans which can be returned from a function or method; references to objects can ALSO be returned from a function, such as a reference to a new browser window. I'll talk more about many of these subjects later on.

Let's look at the addMe() function.

Example:

function addMe(myFirstNum, mySecondNum) {

var myAnswer = myFirstNum + mySecondNum;

alert(myAnswer);

}

The above example function DOES spit out an answer, but it does so via the medium of an alert dialog box. A function like this would NOT spit out information in an alert dialog box, it would normally be called by another function, which would need to receive the calculation result; this requires the use of the return keyword.

Example (abbreviated):

function addMe(myFirstNum, mySecondNum) {

var myAnswer = myFirstNum + mySecondNum;

return myAnswer;

}

Example (in context):

function doStuff() {

var myNumberOfCats = 4;

var myNumberOfDogs = 2;

var myTotalNumberOfPets = addMe(myNumberOfCats, myNumberOfDogs);

alert(myTotalNumberOfPets);

}

function addMe(myFirstNum, mySecondNum) {

var myAnswer = myFirstNum + mySecondNum;

return myAnswer;

}

In the above example, I have called the addMe() function from inside doStuff(); this call passes myNumberOfCats and myNumberOfDogs as arguments to the addMe() function, which adds these values together (using its local variables, myFirstNum and mySecondNum). Once this calculation has been finished, the variable containing the result (myAnswer) is placed after the return keyword, which sends that information back to the calling function, doStuff(). In effect, the addMe() function call in doStuff() is replaced with whatever value is returned from the addMe() function; in this case, the number 6. That answer, 6, is put into the local variable, myTotalNumberOfPets, by the "gets" operator, whereupon that answer is displayed; for simplicity's sake, I have used the alert() method to display the answer, but a real function would probably be doing something much more involved with the information.

This is an over-simplified demonstration of the return keyword. Obviously, you would never need to go through this much trouble just to add two numbers together. I am trying to show you the mechanics of this process using code that is extremely easy to analyze.

Note: The return keyword MUST be part of the LAST statement in a function, because once the return keyword has sent its value back to the calling function, the function containing the return keyword ENDS, regardless of whether there is any more code following that keyword or not.

Example:

function addMe(myFirstNum, mySecondNum) {

var myAnswer = myFirstNum + mySecondNum;

return myAnswer;

alert(myAnswer);

}

In the above example, the call to the alert() method would NOT be executed because it follows the statement containing the return keyword. Again, after the return keyword statement has been executed, any lines of code following that statement are ignored.

Many functions and methods in JavaScript return values of one sort or another to a calling process or function. In the remainder of this module, I will be showing you two different uses of return values which merely scratch the surface (once again) of the vast number of things you can do with this functionality.