**Basic Function Syntax**

Inside a SCRIPT tag, you may define as many FUNCTIONs as you wish. A function is a piece of executable code, a program. A function can be triggered either by a call from an event handler, set in motion by a user, or by a call from another function.

A function requires the following syntax:

function doStuff() { };

First, you must state the JavaScript keyword, function, which tells the JavaScript interpreter that you want to create a FUNCTION. This is followed by a space, then the NAME of the function, followed by the parentheses characters (which are NOT, strangely, the function call operator here, but, rather, serve another purpose which we'll talk about a little later). The parentheses are followed by another space, then the opening and closing curly-braces we used in CSS; this is all ended, of course, with a semi-colon end-of-line marker.

This statement of a function is called a FUNCTION DECLARATION.

Note: the semi-colon end-of-line marker at the end of a function declaration is RARELY, or NEVER, seen, and is usually omitted entirely.

Example (correct):

function doStuff() { }

Between the curly-braces, then, we can define lines of code. In fact, just as in CSS, we will break out our lines of code, putting the opening and closing curly-braces on separate lines, and each line of JavaScript code between them on separate lines.

Example:

function doStuff() {

var george = 1;

alert(george);

}

Again, this function would be placed inside of a SCRIPT tag. You may define as many functions as you like inside of your SCRIPT tag.

Example:

<script type="text/javascript">

/\* <![CDATA[ \*/

function doStuff() {

var george = 1;

alert(george);

}

function doSomethingElse() {

alert("Howdy");

}

/\*]]>\*/

</script>

**White Space in JavaScript**

You will notice in the above examples that I have indented the lines of JavaScript code inside of the function declaration; this is a customary practice, and, unlike with HTML, it is supported and perfectly legal. Indent child elements (inside of curly-braces) one tab in from parent elements.

As you can see in the above example, I have also used white space to separate my functions. This practice, too, is customary, supported, and legal.

The one thing you have to beware of is placing CARRIAGE RETURN characters anywhere but at the ends of lines of code (or as blank lines). JavaScript can potentially interpret carriage return characters as SEMI-COLON end-of-line markers; if you put a carriage return in the middle of a command, the code may break.

Example (WRONG):

function doStuff() {

alert(

"Howdy"

);

}

As you can see in the above example, I have placed illegal carriage returns in the middle of my function call operator, the parentheses. This will cause my code to break.

Example (CORRECTED):

function doStuff() {

alert("Howdy");

}

**Calling a Function from an Event Handler**

Once I have declared and defined my functions, I can call those functions from the event handlers or javascript absolute URLs in the BODY of my HTML page.

Example (abbreviated):

<a href="#" onClick="doStuff();">Link One</a>

<a href="javascript:doSomethingElse();">Link Two</a>

Example (in context):

<html>

<head>

<title>Sample JavaScript</title>

<script type="text/javascript">

/\* <![CDATA[ \*/

function doStuff() {

var george = 1;

alert(george);

}

function doSomethingElse() {

alert("Howdy");

}

/\*]]>\*/

</script>

</head>

<body>

<p><a href="#" onClick="doStuff();">Link One</a></p>

<p><a href="javascript:doSomethingElse();">Link Two</a></p>

</body>

</html>

Here is the above example.

In the above example, do you notice how the code remains inactive UNTIL you click on the hyper-references in the BODY? When the user interacts with an HTML element, a function is called (from either an event handler or from the javascript absolute URL) and executed.

Organizing our JavaScript code inside functions in the SCRIPT tag allows us to keep our HTML code relatively brief and uncluttered. It also opens up a door to a lot of power which is outside of the scope of this introduction to explore.

I will mention again, however, that a function may be called from within ANOTHER function.

Example (abbreviated):

<script type="text/javascript">

/\* <![CDATA[ \*/

function doStuff() {

var george = 1;

alert(george);

doSomethingElse();

}

function doSomethingElse() {

alert("Howdy");

}

/\*]]>\*/

</script>

Example (in context):

<html>

<head>

<title>Sample JavaScript</title>

<script type="text/javascript">

/\* <![CDATA[ \*/

function doStuff() {

var george = 1;

alert(george);

doSomethingElse();

}

function doSomethingElse() {

alert("Howdy");

}

/\*]]>\*/

</script>

</head>

<body>

<p><a href="javascript:doStuff();">Link One</a></p>

</body>

</html>

Here is the above example [**displayed**](http://fog.ccsf.edu/~srubin/js3.html).

Notice how I only called the doStuff() function from within my event handler? The doStuff() function has been rewritten to call the doSomethingElse() function. So, from one event, I can call a function which can call many functions!

I know that this doesn't seem like such a big deal to you right now, but it provides a means of writing the most sophisticated and powerful code imaginable.

I can call both built-in JavaScript methods (such as the alert() method of the window object), and functions which I have defined myself! Pretty slick!