**Variables: Local and Global**

Variables come in two flavors: **local** and **global**.

**Local variables** are temporary. A local variable is declared, initialized, and used for a short period of time, then thrown away. A local variable in JavaScript can only be used within a single function; no other functions have access to that particular variable; when a function is finished executing, the local variables for that function are thrown away.

**Global variables** are persistent. A global variable is created when your web page is loaded, can be accessed by many different processes and functions, and lasts until your web page is unloaded from the browser window.

Again, global variables last for the duration of your web page or program; local variables are used by one process or function, then thrown away.

The distinction between local and global variables is very important in JavaScript programming. We will not have time to explore all of the ramifications of local and global variables in this brief introduction. However, I will be referring to local and global variables frequently in the upcoming sections.

Global variables in JavaScript are declared at the base level of the SCRIPT tag.

Example:

<script type="text/javascript">

/\* <![CDATA[ \*/

var george = 1;

var fred = "Hi";

/\*]]>\*/

</script>

In the above example, both george and fred are GLOBAL variables because they have been declared at the base level of the SCRIPT tag. Global variables can be accessed within all functions.

Example (abbreviated):

<script type="text/javascript">

/\* <![CDATA[ \*/

var george = 1;

var fred = "Hi";

function doStuff() {

alert(george);

}

function doSomethingElse() {

alert(fred);

}

/\*]]>\*/

</script>

Example (in context):

<html>

<head>

<title>Sample JavaScript</title>

<script type="text/javascript">

/\* <![CDATA[ \*/

var george = 1;

var fred = "Hi";

function doStuff() {

alert(george);

}

function doSomethingElse() {

alert(fred);

}

/\*]]>\*/

</script>

</head>

<body>

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

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

</body>

</html>

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

I can also change the contents of (or operate upon or manipulate) global variables from within functions.

Example (abbreviated):

<script type="text/javascript">

/\* <![CDATA[ \*/

var george = 1;

var fred = "Hi";

function doStuff() {

alert(george);

george = "Boo";

fred = "Howl";

}

function doSomethingElse() {

alert(fred);

george = "Yum";

fred = "Delicious!";

}

/\*]]>\*/

</script>

Example (in context):

<html>

<head>

<title>Sample JavaScript</title>

<script type="text/javascript">

/\* <![CDATA[ \*/

var george = 1;

var fred = "Hi";

function doStuff() {

alert(george);

george = "Boo";

fred = "Howl";

}

function doSomethingElse() {

alert(fred);

george = "Yum";

fred = "Delicious!";

}

/\*]]>\*/

</script>

</head>

<body>

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

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

</body>

</html>

Here is the above example [**displayed**](http://fog.ccsf.cc.ca.us/~srubin/js3a.html).

As you can see, each time we call one of the functions, we are changing the contents of the global variables. Try clicking the links a few times to see what I mean.

Local variables are declared WITHIN a function declaration ONLY. Local variables will last only as long as the function is executing. When the function has finished executing, its local variables are thrown away. Functions may NOT access the local variables of other functions.

Example (abbreviated):

<script type="text/javascript">

/\* <![CDATA[ \*/

function doStuff() {

var george = "Wow";

alert(george);

}

function doSomethingElse() {

var fred = "How about that!";

alert(fred);

}

/\*]]>\*/

</script>

In the above example, the variable george is a local variable of doStuff(), while the variable fred is a local variable of doSomethingElse(). Because fred and george are local variables in different functions, I can not access fred from doStuff(), nor can I access george from doSomethingElse(); either of these actions would result in a JavaScript error.

Example (INCORRECT):

<script type="text/javascript">

/\* <![CDATA[ \*/

function doStuff() {

var george = "Wow";

alert(fred);

}

function doSomethingElse() {

var fred = "How about that!";

alert(george);

}

/\*]]>\*/

</script>

In the above INCORRECT example, I am trying to access george from doSomethingElse() and fred from doStuff(), neither of which are possible actions.

Example (POSSIBLE):

<script type="text/javascript">

/\* <![CDATA[ \*/

function doStuff() {

var george = "Wow";

var fred = "How about that!";

alert(george);

alert(fred);

}

/\* <![CDATA[ \*/

</script>

In the above example, george and fred are both local variables of the SAME function; any process within that function, then, could access either george and/or fred.

We'll talk more about handling local variables in the next sections.

**One Final Note**

Each function may contain local variables which have the SAME names as local variables contained in other functions. For instance, the following code is perfectly CORRECT:

function doStuff() {

var myNumber = 1;

alert(myNumber);

}

function doSomethingElse() {

var myNumber = 10;

alert(myNumber);

}

Because local variables are temporary, only existing for the duration of a given function, other functions may use the**SAME** local variable names. The technical reason for this is called **PROTECTED NAMESPACE**, which we don't need to go into here. Suffice it to say, that each function is like a separate universe, with its own collection of local variables, and that these local variables are separated or protected from the local variables in other functions.

So, even though I am referring to the local variable myNumber in BOTH doStuff() and doSomethingElse(), these two variables, because they are both LOCAL variables, have NO relationship to one another whatsoever.