Variable scoping

By Cal Henderson

2009-05-11: This article was never completed, and much better explanations can be found elsewhere. Head to Google!
2013-03-25: This article has been translated into Spanish by Maria Ramos at Webhostinghub.com.

What is Scope

Scope refers to the places in which a particular variable is valid. If you create a variable called *bob* in a function, then referencing *bob* later in another function may give unexpected results. This is because variables can be limited in scope.

What is Scope for?

Blah

Scope Types

There are two normal scopes to deal with in javascript: global and function local. globally scoped variables are available at all points in the code. locally scoped function variables are only available in the function you create them in.

The *var* keyword is used to explicitly declare a variable:

```
var my_int = 10;
```

In javascript, variables can also be implicitly declared by assigning a value to them:

```
my_implicit_int = 12;
```

At present, no method exists of disabling implict variable declaration. This can cause problems:

```
var my_variable = 10;
if (my_varaible == 10) alert('10!');
```

The second variable name is mis-spelt, but the code still runs without compilation errors - of course, the *alert()* statement will not fire!

Local and Global Scope

When a variable with the same name as an existing variable comes into scope, the more local variable overrides to more global one (until it falls out of scope). The following four examples illustrate this:

```
a = 5;
func_a();
document.writeln("global a = "+a+"<br>");
function func_a(){
 a = 10;
 document.writeln("local a = "+a+"<br>");
}

b = 5;
func_b();
document.writeln("global b = "+b+"<br>");
function func_b(){
```

```
 var b = 10;
 document.writeln("local b = "+b+"<br>");
}

var c = 5;
func_c();
document.writeln("global c = "+c+"<br>");
function func_c(){
 var c = 10;
 document.writeln("local c = "+c+"<br>");
}

var d = 5;
func_d();
document.writeln("global d = "+d+"<br>");
function func_d(){
 d = 10;
 document.writeln("local d = "+d+"<br>");
}
```

The output of executing this script is not necessarily what you would expect:

```
 local a = 10
global a = 10
local b = 10
global b = 5
local c = 10
global c = 5
local d = 10
global d = 10
```

The upshot of these results can be summed up in some simple rules of scope:
All implicitly declared variables belong to the global scope Explicity defined variables belong to the local block, meaning they are global if defined in a global context and local if defined within a function.

Scope Inheritance

The only scope block that javascript recognises is the function. This means that variables can not be overridden in structural blocks, such as *if* statements:

```
 var e = 1;
if (true){
 var e = 2;
 document.writeln("blocked e = "+e+"<br>");
}
document.writeln("global e = "+e+"<br>");
```

Because the *if* block has the same scope as the code before it, the declaration is a re-declaration (so is ignored) and the old variable is assigned a new value. The output is then:

```
 blocked e = 2
```

```
global e = 2
```

Another scope oddity in javascript is that scope isn't inherited between functions - if a function declares a local variable, nobody but the function can access it - even functions called within that function. The following code demonstrates this:

```
var f = 1;
func_f_1();

function func_f_1(){
 var f = 2;
 func_f_2();
}

function func_f_2(){
 document.writeln("f = "+f+"<br>");
}
```

The global variable declared is in scope for the last function, but the first function's local variable isn't. This output is thus as expected:

```
f = 1
```

There is a slight twist to this though - functions defined inside another function inherit it's scope, even when called from outside the parent function. The child function's scope becomes a subset of the parent's scope and this works on multiple levels. For example:

```
var g = 1;
func_g_1();

function func_g_1(){
 var g = 2;
 func_g_2();

 function func_g_2(){
  document.writeln("g = "+g+"<br>");
 }
}
```

Because func_g_2 inherits scope, we get the following output:

```
g = 2
```

And if we wanted to get crazy, we can nest it deeper:

```
var h = 1;
func_h_1();

function func_h_1(){
 var h = 2;
 func_h_2();
```

```
  function func_h_2(){
   var h = 3;
   func_h_3();

   function func_h_3(){
    document.writeln("h = "+h+"<br>");
   }
  }
}
```

Which predictably outputs:

h = 3

(end of article)