Javascript - A Tutorial

Introduction

This tutorial was created in response to a request on the GoG.com community forums. It is aimed at people with some understanding of HTML, but no experience with C-like programming languages.

The focus is on avoiding typical pitfalls that people usually encounter when learning JavaScript for the browser by themselves.

Tools

There are very good tools for complex projects (I myself use Aptana), but while learning, such tools more often that not get in the way. Start with a simple code editor, like SciTE or whatever editor (that has syntax highlighting) you're comfortable with. Otherwise you'll spend more time fighting the IDE than actually coding.

As testing environment, I'd recommend Firefox as the error handling is pretty good and you usually get meaningful error messages when you do anything wrong. Take a look at Tools/Error Console. Another important tool is the DOMInspector which gives you an overview of the current document, including all Javascript methods and properties for a given node.

Don't try to start with a library like Mootools or jQuery. These libraries force you to use a certain pattern and will make switching to a different library more difficult later on. It's best to learn native JavaScript first, and switch to whatever library you want to use lateron..

First program

OK, first the obligatory "Hello World":
Open your XHTML file and add a script element inside the <head>...</head>:

```
<script type="text/javascript"> </script>
```

Don't add it to the body, as browsers without JavasScript support might actually display your code as text (there are very few browsers around that still do that, but it's better to be safe than sorry).

The script element has two important attributes: type and src (source). The type tells the browser what kind of language to be use. This is Javascript by default, but some browsers also accept other languages (like VBScript on MSIE).

The src attribute is much more important:
If it exists, the value is assumed to be an url and the browser will load the script and execute it. The text inside the script element than becomes the fallback text, should Javascript not be supported. If no src attribute is present, the text inside the script element is assumed to be the code.

So now we have a code block that does nothing yet. Let's make it do something:

```
<script type="text/javascript">
        window.alert("Hello World");
</script>
```

© 2009 Hans Schmucker

We can see a couple of things here:
1. How functions/methods are called. This happens in the usual C-style:

```
functionName(parameters);
```

Also, notice that each instruction has to be terminated with a ";" (this is actually optional in many cases, but it's considered very bad style to leave it out).

2. Literal values:
Literal values, like numbers, strings or booleans can be created in place, simply by writing the value. Strings need to be enclosed in single or double quotes and the usual escaping rules with "\" apply. Most important is that quotes and the backslash itself have to be prefixed with a backslash should they be part of a string.
For example if you intend to write
"C:\config.sys" (with quotes)
you'll need to actually write

```
"\"C:\\config.sys\""
```

3. Methods/properties of an object are accessed through the "." operator. In this case, the JS interpreter first looks for the window object (which is global and predefined, so it's always there) and then a property/method named like the part after the dot, in this case "alert". So the form is usually object.property or object.method(). (methods are properties in JavaScript too, but we'll get to that later). Alternatively, you can use array notation with square brackets and a string object["property"]. Both are equal, but since the second one takes a string you can dynamically construct calls. So the above code could equally well be written like this:

```
<script type="text/javascript">
	window["alert"]("Hello World");
</script>
```

Variables

All values can also be stored in variables and used latero n. Variables are created using the "var" keyword (yes, you can actually create global variables without it, but you really shouldn't because this will cause conflicts down the line). Variables can be initialized to a given value directly or lateron in the code. Assignment happens with the "=" operator.

```
<script type="text/javascript">
	var text="Hello";
	var text2;
	text2="World";
	window.alert(text);
	window.alert(text2);
</script>
```

© 2009 Hans Schmucker

Will display first Hello, then World.
Variables can be changed as often as you like. Nothing is constant.
Assignments are always processed right to left, which means that whatever is right of the "=" is always assigned to what's left of it. An assignment has also a result, which is the assigned value, so you can include multiple "=" in an expression:

```
<script type="text/javascript">
	var a;
	var b;
	var c=10;
	a=b=c;
</script>
```

will assign c to b and a.
A few more things to remember: A variable name is case sensitive, so Foo is not the same as foo. Also, variable names can contain "$", "_" and numbers. However, the first character can never be a number.

String and mathematical operations

There are a few tools you can use to manipulate strings, numbers and so on. For example, one string is appended to another one using the "+" operator:

```
"Hello"+" "+"World"
```

is the same as

```
"Hello World"
```

Mathematical operations like "+", "-", "*" and "/" can just be written like you normally would. Brackets can be used to further describe the order in which an expression should be processed.
For example:

```
<script type="text/javascript">
	window.alert(("Hello"+" "+"World ")+(4/2));
</script>
```

will display "Hello World 2" (without quotes). The order in which this happens is dictated by the brackets: Combine "Hello"+" "+"World " into "Hello World ". Solve 4 divided by 2 with 2. Combine "Hello World " and 2 to "Hello World 2". In addition to the brackets, normal rules like "*","/" first, "+", "-" later also apply.
So:

```
<script type="text/javascript">
	window.alert(4+3/2);
</script>
```

© 2009 Hans Schmucker

will correctly display 5.5 (4+(3/2)) and not 3.5.

Control Statements

Conditions and loops are mostly identical with C. For example:
do{action}while(condition);
Will repeat action, while the condition is true (but at least once, as the condition is checked after action has taken place, if you want to check first, use while(condition){action} instead). For example:

```
<script type="text/javascript">
	var num=0;
	do{
		window.alert(num);
		num=num+1;
	}while(num<3);
</script>
```

will display "0","1","2" . After "2" has been displayed, the value will be 3. Since our condition is that number is smaller than 3, that condition will no longer be true and Javascript will exit the loop. There is a number of comparison operators that you can use like "<" (smaller than), ">" (greather than), "==" (equals), "<=" (smaller or equal to), ">=" (greater or equal to) and "!=" (not equal to). However you're not limited to conditions. Also other types can be converted to condition statements. For example:
strings are always true, unless the string has zero length: ""
numbers are always true, except 0
So for example another way to do what we have done above would be

```
<script type="text/javascript">
	var num=-3;
	do{
		window.alert((num+3));
		num=num+1;
	}while(num);
</script>
```

The loop would count up towards zero and exit when it reaches it.
Another important operator is NOT, which is written as a "!". It simply changes true to false and vice versa.
A way to write our first sample using the not operator would have been

```
<script type="text/javascript">
 var num=0;
 do{
   window.alert(num);
   num=num+1;
 }while(!(num>=3));
</script>
```

© 2009 Hans Schmucker

There are different kinds of loops for different purposes and we'll talk about them in a second but since we're already talking about conditions anyway, I think it's a good time to talk about branching.
Every once in a while you might want to take a certain action only if a condition is true or false. The easiest construct to use here is the "if" directive:
if(condition){action}
the action will only be executed if the condition is true. Example:

```
<script type="text/javascript">
 var num=0;
 do{
   if(num>=2){
     window.alert(num);
   }
   num=num+1;
 }while(num<3);
</script>
```

Will do the same thing as our previous sample: count up from 0 to 2. However, we have included the output method in a condition, namely that the number has to be greater or equal to 2. Which only happens for one case: 2.
If you want to take another action if the condition is not true, simply add an else{action} block after your if statement:

```
<script type="text/javascript">
	var num=0;
	do{
		if(num>=2){
			window.alert(num);
		}else{
			window.alert("Too small");
		}
		num=num+1;
	}while(num<3);
</script>
```

You can also chain else if conditions if you need to check for more cases:

```
<script type="text/javascript">
	var num=0;
	do{
		if(num>=2){
			window.alert(num);
		}else if(num==0){
			window.alert("Nothing");
		}else{
```

© 2009 Hans Schmucker

```
                window.alert("Too small");
            }
            num=num+1;
        }while(num<3);
</script>
```

A shortcut for a lot of else if when you really only want to check for different values is the switch/case statement.

```
switch(input){
        case value1:
            action1
            break;
        case value2:
            action2
            break;
        default:
            defaultaction:
            break;
}
```

Let's put it into action. We want something special to happen for values of 0 and 1 and a standard message for the rest. So we just add two cases and a default section that will be executed for all other cases.:

```
<script type="text/javascript">
        var num=0;
        do{
            switch(num){
                case 0:
                    window.alert("Virtually non-existent");
                    break;
                case 1:
                    window.alert("Average");
                    break;
                default:
                    window.alert("Pretty big");
                    break;
            }
            num=num+1;
        }while(num<5);
</script>
```

Easy, isn't it?
Now, lets return to the loops for one second. There's one (or two, depending on how you count) types of loops that we haven't touched yet, namely the "for" loops.

© 2009 Hans Schmucker

Let's start with the simple one:
for(initialization;condition;increment){action}
OK, this looks very frightening, but it's not that bad really. It's just another way to format a normal while loop.
The equivalent would be

```
initialization
while(condition){
	action
}
```

So why do we write it differently. Because one very common case looks a lot more confusing using the while loop than when using the for loop notation: Sequences
You probably already noticed that our while loop was a bit hard to read, so here's how it would look as a for loop:

```
<script type="text/javascript">
	for(var num=0;num<3;num=num+1){
		window.alert(num);
	}
</script>
```

Much more readable, right? And with a bit of special magic called the increment operator (which is fancy talk for "add one") "++" it looks even better:

```
<script type="text/javascript">
	for(var num=0;num<3;num++){
		window.alert(num);
	}
</script>
```

Functions

While the statements above allow us to branch out and therefore theoretically allow for a complete program to be written like this, it does get tiresome (and unreadable) to repeat the same stuff over and over again. That's what functions are for. A function is a code snippet that you can simply call over and over again. For example window.alert is a function that provides us with a messagebox.
We can define our own function with a very simple pattern:

```
function functionName(parameter1, parameter2){
	action
	return (value);
}
```

A function is defined using the function keyword and an (optional, but we'll get to that in the next chapter) function name. After that there have to be brackets, which contain any number of parameters (including none). A parameter is a variable that will be created when the function is called and assigned whatever value was put into the bracket during

© 2009 Hans Schmucker

the call. So if I was to run this function with functionName("Hello","World"); then parameter1 would have the value "Hello" and parameter2 the value "World".
"action" is only a placeholder for whatever code is supposed to run.
Finally, the return statement can return anything back. The call then becomes the return value. For example if I had a function addOne(num) that increases the parameter by one and then returns it, then window.alert(addOne(2)); would display "3":

```
<script type="text/javascript">
	function addOne(num){
		return (num+1);
	}

	window.alert(addOne(2));
</script>
```

There's one other thing that's special about functions and it has to do with variables. We learned to use the "var" statement to create variables, right? Well, you might have noticed that it also works when you leave out the "var". In fact, outside of any functions, these two ways have identical results: They create a global variable and assign it a value. (still, keep the "var", because if you stop using it here, you'll forget to add it when it's needed).
I keep mentioning "global" variables and you can guess that I'm doing this not only because I like the word, but also because I want to separate them from the "local" variables.
The difference is relatively simple: A global variable is available everywhere (for example "window"), while a local variable is only visible inside the function. That almost makes it sound like global variables were "better". Far from it: Because the name is used globally, there's a good chance that two functions have the same variable names. When that happens, the one function will overwrite the values from the other function, which usually results in utter chaos.
"USE VAR", I can't stress that enough. Example:

```
<script type="text/javascript">
	var sum=2;
	function addOne(num){
		var sum=(num+1);
		return (sum);
	}

	window.alert(addOne(sum));
	window.alert(sum);
</script>
```

If we hadn't included the "var" inside the function, Javascript would have assumed that we wanted to use the global variable "sum" and would have altered that. Sum would also have increased by one outside of the function.

Types

While Javascript is not a strongly typed languages (variables can adapt to the type of value assigned to them, so a variable that held a string first can hold a number later on), there are types of objects. The normal types are

© 2009 Hans Schmucker

"string", "number" and "boolean"
plus, there are some more complex types, specifically
"object", "function", "regexp" and "undefined"
String, number and boolean work pretty much like you'd expect them to. They hold a string ("Hello World"), a number (256) and a boolean (true or false).
functions is where it gets complicated. You've seen what we have done in the previous chapter? We created a function suing the name notation, which automatically creates a global variable with that name and assigns it the function as value. But that's not the only way to create a function. You can also just assign it to a variable, which will then be of type "function". You can copy it around if you want, you can pass it as argument to the function. You can do pretty much anything with it that you could do with a string or a number.
A popular mechanism that's based on this principle are so called "callbacks". Callbacks are arguments that are passed to a function, which will then run them if appropriate.
For example, we can have a

© 2009 Hans Schmucker