Basic JavaScript Part 1 : Functions

October 22nd, 2010

Functions are a very important concept in most programming languages, but they are even more important in JavaScript. When you read this as a C# or VB developer you probably won't see this as a big deal. But there are some essential things you can do with functions that you should be aware of when doing JavaScript development.

Self-executing functions

As the name implies, these are functions that are called after they are defined. Let's look at an example:

```
(
  function() {
    alert("Hello Mars");
  }
)()
```

This basically is an anonymous function that is invoked immediately, caused by the last pair of parentheses. It's also possible to specify arguments to the anonymous function like so:

```
(
  function(planet) {
    alert("Hello " +  planet);
  }
)("Jupiter")
```

Why is this useful? Well, you can do all sorts of stuff with self-executing functions like one-off initialization without having to mess around with global variables. They are also commonly used for variable scoping. Variables declared inside a self-executing function cannot be accessed from outside the function. This way it's possible to provide proper encapsulation, which I might come back in a later blog post.

Inner functions

This feature reminds me to the early stages of career as a software developer when I was still programming Delphi. Just like Delphi, JavaScript makes it possible to define functions inside another function.

```
function show() {
  function getPlanet() {
    return "Saturn";
  }

  alert("Hello " + getPlanet());
}
```

The function *show()* is publicly available, but the function *getPlanet()* is a private function and can only be called from code that lives inside the *show()* function. So the main benefit here is encapsulation.

Self-rewriting functions

It's possible to assign functions to a variable. Nothing unusual there. But we can also assign a new function to the same variable from within the function that was originally assigned to the variable. You can think of it as a function that rewrites itself from the inside when it is executed. This probably sounds more complicated that it really is so I'll show you an example.

```
var doSomething = function() {
  alert("Doing something useful");

  doSomething = function() {
    alert("But I already did something!");
  }
}
```

Again, this technique is quite useful when you need to do some one-off initialization or perhaps feature sniffing on different browsers.

Closures

In order to understand closures in JavaScript, it is also vital to understand scoping. Unlike other programming languages, JavaScript has function scope instead of block scope. This is also called scope chain.

```
function foo() {
  var x = 8;
  if(x > 0)
  {
    var y = 12;
  }

  (
    function bar() {
      alert(x + y);
    }
  )();
}
```

This might come of a little bit strange, but this code is completely legit. Both the variables *x*and *y* are available inside our self-executing function *bar()*. Variable *y* is available in function*foo()* after its definition inside the *if* block. This is why it's a best practice in JavaScript to declare all variables at the top of the function in order to prevent some nastiness later on.

JavaScript functions also have lexical scope. This means that a function doesn't create its scope at runtime but when its defined. I have to admit that I still run into some issues regarding this concept from time to time.

Now let's get back to closures by looking at a very basic example.

```
function foo()
{
  var x = 15;
  return function() {
    return x;
  }
}
```

This might look familiar when you've worked with lambdas in C#. Calling the function *foo()*returns another function that returns the value of the local variable *x*. Closures are often used when doing OO-style programming in JavaScript. An example of this are getters and setters.

Closing

These are just some of the nice little tidbits that are very powerful tools in your JavaScript tool belt. Actually sitting down and learning about JavaScript as a programming language can be a real eye opener and a very rewarding experience.