Chapter 5
Functions

Adapted from
JavaScript: The Complete Reference 2nd Edition
by
Thomas Powell & Fritz Schneider

© 2004 Thomas Powell, Fritz Schneider, McGraw-Hill
Function Basics

• Functions are used to create code fragments that can be used over and over again. Hopefully, these are abstract reusable components, but this is up to the programmer.

• function functionname(parameterlist)
  {
    statement(s)
  }

where

  – Functionname must be well-formed JavaScript identifier
  – Parameterlist is a list of JavaScript identifiers separated by commas. The list may also be empty
Function Example 1

Simple function with no parameters

```javascript
function sayHello()
{
    alert("Hello there");
}
sayHello();    // invoke the function
```

Note: You generally will be unable to call a function before it is defined. This suggests that you should define your functions in the `<head>` of your (X)HTML document. However, in some JavaScript implementations you can forward reference with the same `<script>` block.
Function Example 2: Parameters

```javascript
function sayHello(name)
{
    if (name != "")
        alert("Hello there "+name);
    else
        alert("Don’t be shy. ");
}

/* Make some calls */
sayHello("George");
sayHello();
```
Example 3: Multiple Parameters & Return

```javascript
function addThree(arg1, arg2, arg3)
{
    return (arg1 + arg2 + arg3);
}

var x = 5, y = 7, result;
result = addThree(x, y, 11);
alert(result);
```
Example 4: Multiple Returns

```javascript
function myMax(arg1, arg2)
{
    if (arg1 >= arg2)
        return arg1
    else
        return arg2;
}
```

**Note:** Functions always return some value whether or not a `return` is explicitly provided. Usually it is a value of `undefined`.
Parameter Passing

- Primitive Data types are passed by value, in other words a copy of the data is made and given to the function

```javascript
function fiddle(arg1)
{
    arg1 = 10;
    document.write("In fiddle arg1 = "+arg1+"<br />");
}
var x = 5;
document.write("Before function call x = "+x+"<br />");
fiddle(x);
document.write("After function call x = "+x+"<br />");
```
Parameter Passing 2

- Composite types are passed by reference in JS

```javascript
function fiddle(arg1)
{
    arg1[0] = "changed";
    document.write("In fiddle arg1 = "+arg1+"<br />
    ");
}

var x = ["first", "second", "third"];
document.write("Before function call x = "+x+"<br />
    ");
fiddle(x);
document.write("After function call x = "+x+"<br "/")
```

Global and Local Variables

• A global variable is one that is known throughout a document
• A local variable is limited to the particular function it is defined in
• All variables defined outside a function are global by default
• Variables within a function defined using a var statements are local
Global and Local Example

// Define x globally
var x = 5;
function myFunction()
{
    document.write("Entering function<br /> x="+x+" <br />");
document.write("Changing x <br />");
x = 7;
document.write("x="+x+"<br /> Leaving function<br />");
}
document.write("Starting Script<br />");
document.write("x="+x+"<br />");
myFunction();

document.write("Returning from function<br />");
document.write("x="+x+"<br />");
document.write("Ending Script");
Local Variable Example

```javascript
function myFunction() {
    var y = 5; // define a local variable

    document.write("Within function y="+y);
}

myFunction();
document.write("After function y="+y);

**Note:** This example will throw an error, but that’s the point. You could use an if statement to avoid problems like

```javascript
if (window.y)
    document.write("After function y="+y);
else
    document.write("Y is undefined");
```
Mask Out

- Be careful when you have local and global variables of the same name, you may get an undesirable effect called mask out.

```javascript
var x = "As a global I am a string";
function maskDemo()
{
    var x = 5;
    document.write("In function maskDemo x="+x+"<br />");
}

document.write("Before function call x="+x+"<br />");
maskDemo();
document.write("After function call x="+x+"<br />");
```
Local Functions

function testFunction()
{
    function inner1() { document.write("testFunction-inner1<br />"); }
    function inner2() { document.write("testFunction-inner2<br />"); }
    document.write("Entering testFunction<br />");
    inner1();
    inner2();
    document.write("Leaving testFunction<br />");
}

document.write("About to call testFunction<br />");
testFunction();
document.write("Returned from testFunction<br />");

/* Call inner 1 or inner2 here and error */
inner1();
Functions as Objects

• Like nearly everything in JS, functions are objects and can be created and accessed as such

```javascript
var sayHello = new Function("alert('Hello there');");
sayHello();
```

• This allows us to even reuse functions in an interesting way.

```javascript
var sayHelloAgain = sayHello;
sayHelloAgain();
```
Functions as Objects

• You can also define functions with parameters in this fashion.

```javascript
var sayHello2 = new Function("msg","alert('Hello there '+msg);"),
sayHello2('Thomas');
```

• The general syntax is

```javascript
var functionName = new Function("argument 1", …"argument n", "statements for function body");
```
Useful Function Features

• As objects you can reference the length of functions, thus find out the number of arguments

```javascript
function myFunction(arg1, arg2, arg3) {
    // do something
}
alert("Number of parameters for myFunction = " + myFunction.length);
```
Arguments and Length

- You can examine not just defined arguments but actual passed parameters

```javascript
function myFunction()
{
    document.write("Number of parameters defined = " + myFunction.length + "<br />");
    document.write("Number of parameters passed = " + myFunction.arguments.length + "<br />");
    for (i=0; i<arguments.length; i++)
        document.write("Parameter "+i+" = " + myFunction.arguments[i]+"<br />");
}

myFunction(33, 858, 404);
```
Variable Arguments

• Given arguments and length you can write more adaptive functions that take variable arguments

```javascript
function sumAll()
{
    var total=0;

    for (var i=0; i< sumAll.arguments.length; i++)
        total+=sumAll.arguments[i];

    return(total);
}

alert(sumAll(3,5,3,5,3,2,6));
```
Literal and Anonymous Functions

function simpleRobot(robotName)
{
    this.name = robotName;
    this.sayHi = function () { alert('Hi my name is '+this.name); };
    this.sayBye = function () { alert('Bye!'); };
    this.sayAnything = function (msg)
    {
        alert(this.name+' says '+msg);
    }
}

fred.sayHi();
fred.sayAnything("I don't know what to say");
fred.sayBye();
Recursive Functions

- JS supports recursive functions that call themselves
- Factorial $n! = n*(n-1)*(n-2) * \ldots * 1$

```javascript
function factorial(n) {
    if (n == 0)
        return 1;
    else
        return n* factorial(n -1);
}

alert(factorial(5));
```

- Demo this with negative value in Internet Explorer
Tips on Using Functions

• Define all functions for a script first
• Name functions well
• Consider using linked .js files for functions
• Use explicit return statements
• Write stand-alone functions
• Check arguments carefully
• Comment your functions
Summary

• Functions are useful for defining reusable blocks of code

• Functions in JavaScript pass data by value typically though complex types are passed by reference

• Functions can support local variables

• Functions in JavaScript are powerful
  – Variable arguments, anonymous and literal functions, recursion, etc.