JavaScript Functions

- Defining Functions
- Nesting Functions
- Closures

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

What is a Function?

- A **function** is a chunk of code that you define once but can call as many times as needed
- We refer to a function that is part of an object as a **method**
- Functions can accept **parameters** and optionally return a value

```
1    function sum(a, b)
2    {
3       return a+b;
4    }
5
6    var result = sum(5, 8);
```

In other languages, functions are referred to as "subroutines", "procedures" or subprograms.

Functions can use a predefined variable named "arguments", that represents the parameters actually passed to the function. This object contains a "length" property containing the number of arguments.

```
function function7(a, b, c)
{
  // displays 1
   alert(arguments.length);
}
function7(12);
```

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Functions are Objects

- In JavaScript, functions are a type of **object**, and thus can be passed as parameters to another function or returned
- We can thus say that JavaScript functions are **first-class functions** as opposed to languages like Java (pre Java v8) in which functions (methods) can only be defined and invoked

Java version 8 does support first-class methods.

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Defining Functions

- JavaScript provides two syntaxes for defining functions:
    - Function declaration
    - Function definition expression
- The difference is that the **definition expression** syntax assigns the function object to a variable

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Function Declarations

- **Function declarations** let you define and name functions

```
1    // function declaration syntax
2    function function1(p1, p1)
3    {
4      alert(p1+p2);
5    }
6
7    function function2()
8    {
9      return "hi";
10   }
11
12   function1(2, 3);
13   var x = function2();
```

Behind the scenes, this syntax creates a variable with the function name and assigns the function object to the variable. Thus calling code that invokes the function is really using the variable.

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Function Definition Expression

- **Function definition expressions** let you define a function, optionally name it and optionally invoke it immediately

```
1    // function definition expression
2    var function3 = function(p1) {return p1;}
3
4    // naming is useful for recursion
5    var f4 = function function4() {return function4};
6
7    function3(12);
8    var x = f4();    // references function4
9    // function4();  // not visible
```

The syntax where you use both a variable and a function name is useful for writing recursive functions - in the function's body, you can invoke the function itself. Note that the function name is only defined within the body - outside code must use the variable to invoke the method.

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Function Definition Expression, cont'd

- You can use the function definition expression syntax to define and invoke **anonymous** functions on the fly

```
1    // define and invoke a function in one fell swoop
2    var f5 = function(){alert("ho");}();  // "ho"
3
4    // define and invoke an anonymous function
5    // note the extra set of parens
6    (function(p1){alert(p1);}(42));  // 42
7
8    // pass an anonymous function to another function
9    function function6(obj){alert(obj(72));}
10   function6(function(p1) {return p1;});  // 72
```

In the line that defines and invokes the anonymous function, the extra set of parenthesis is required to avoid a syntax error.

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Global Functions vs Methods

- Functions that are part of an object are known as **methods**
- JavaScript provides methods with an implicit **this** reference that refers to the object on which the method is running

```
1    var sue = {
2      name: "Sue",
3      age:  32
4    };
5
6    sue.ageInDogYears = function() {
7       return this.age * 7;
8    }
9
10   alert("Age in dog years: " + sue.ageInDogYears());
```


Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Scope of Variables Review

- JavaScript supports **global** and **local** variables
- Variables defined within a function (method) are **local** and can only be used by that method

```
1    var a = "My global variable";
2
3    function function1()
4    {
5      var b = "My local variable";
6
7      alert(a);  // access global
8    }
9
10   function function2()
11   {
12     alert(a);  // access global
13     alert(b);  // error - out of scope
14   }
```

Global variables are actually implemented as properties on the so-called "global object". In client-side JavaScript in a Web browser, the "window" object acts as the global object.

A function is a block of code with a name and optional arguments. To declare a function, use the "function" keyword, provide a proper identifier, open-and-close parenthesis and a pair of "curly braces".

Unlike languages such as Java and C, JavaScript doesn't have "block scope", in which variables defined within a block are local to that block.

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Defining Functions within Functions

- JavaScript programs can nest functions within other functions – the nested functions can access variables defined in the enclosing function

```
1    //  a nested function can access
2    //  variables defined by enclosing function
3      function f1() {
4         var x = 12;
5
6         function f2() {
7            alert(x);
8         }
9
10        f2();
11     }
12
13     f1();  // 12
14     f2();  // undefined
```

Note that the nested function is not directly visible outside of its enclosing function.

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Closures

- An enclosing function can return a nested function back to a caller, which can then invoke the nested function
- The nested function can still access local variables defined by the enclosing function, even though no one called the enclosing function – this is known as a **closure**
- Closures are a fundamental part of **functional programming**, in which you write programs that consist of functions that don't modify any external state, and thus have no **side effects**
- A complete discussion of functional programming is beyond the scope of this course

---

The benefit of functional programming is that programs are easier to understand, maintain and debug, since there's no mutable "global" data. In addition to JavaScript, other languages that support functional programming include Lisp, Scheme, Haskell, Clojure and Scala.

There is a useful library that helps you implement functional programming in JavaScript at:

`http://osteele.com/sources/javascript/functional/`

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Closures, cont'd

```
1       function f1()
2       {
3          var x = 12;
4
5          function f2()
6          {
7             alert(x);
8          }
9
10         return f2;
11      }
12
13      // demonstrate a closure in action
14      var f2 = f1();
15      f2();             // 12
```

JavaScript internally implements closures by maintaining a "scope chain" - when a nested function references a variable, JavaScript looks first in the nested function itself, and then in the enclosing function.

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Chapter Summary

In this chapter, you learned:

- The fundamentals of JavaScript functions
- How to implement a **closure**

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum