Book 3: JavaScript - Functions

Book for Weeks 8-9: JavaScript Functions

Site: Insight2 @ CCSF
Course: CNIT133-META-Interactive Web Pages-Rubin-JavaScri-Spring 2014
Book: Book 3: JavaScript - Functions
Printed by: Thuong Ho
Date: Thursday, February 13, 2014, 9:54 PM

Table of contents

1 JavaScript Objects and Functions: An Overview

Textbook References

Flanagan book: p. 163-197 (Ch. 8)

Gosselin book: p. 73-82

McDuffie book: Ch. 8, 10

Deitel book: Ch. 9

Because JavaScript is an object-oriented programming language, I need to tell you a little bit about JavaScript "objects" and the structure of the language generally.

In any programming language, you write lines of code which are executed, left-to-right, top-to-bottom, just like in HTML. In an object-oriented programming language such as JavaScript, however, much of this code is organized in OBJECTS. In fact, most of the built-in features of JavaScript use objects of one sort or another, which means that you need to understand basic object syntax before you can do much of anything, including reading the JavaScript reference books.

In an HTML page in a JavaScript-enabled web browser, most or even all things on the page are mirrored in JavaScript objects. There is a window object which refers to the browser window itself, there is a document object which refers to the HTML page itself (and, potentially, everything contained in that HTML page), there is an images object which has to do with pictures, etc. By manipulating these objects in JavaScript code, you can affect the behavior and appearance of an HTML page, call up dialog boxes or other resources, and invoke a fairly sophisticated set of commands to do your bidding.

Every object in JavaScript has two main components: a set of PROPERTIES, and a set of METHODS. Properties are generally passive in nature, describing the state of some feature of the object. Methods, on the other hand, are generally active, performing some sort of task.

Over the years, I have used something called the "object-oriented car", a common metaphor in programming instruction, to explain object-oriented programming principles. As I have grown more sophisticated in my explanations, however, the "car" object example has begun to break down. Now, instead of describing the entire car in my lectures, I describe only the STEERING WHEEL of the car using object-oriented programming terms.

Let us create a SteeringWheel object (and notice how the name of the object has NO SPACES!). This SteeringWheel object has one property, "direction", and one method, "honk()".

SteeringWheel Object

```
// properties
direction
```

```
// methods
honk()
```

The direction property of the SteeringWheel object says what direction the SteeringWheel is pointed. Since this is a VERY simple SteeringWheel, only left, right, or center would be legal values for this property.

The honk() method of the SteeringWheel object honks the car's horn; we'll talk more about this method later on.

The code that defines the SteeringWheel object, with all of its properties and methods, is called a CONSTRUCTOR. The exact syntax for creating your own object constructor is really beyond the scope of this introductory lecture. However, you WILL be invoking pre-built constructors when you use JavaScript, so we'll talk about how to do this.

The SteeringWheel constructor merely defines the basic structure of the SteeringWheel object; it does NOT actually create a SteeringWheel which you can manipulate directly. To get a SteeringWheel object that you can use, you need to create an INSTANCE of the SteeringWheel object (also known as INSTANTIATING a SteeringWheel object). When you create an instance of an object, you are creating ONE copy of the object which you have control over.

In JavaScript, you will need to instantiate an object by using a variable to contain a copy of the object. First, you'll declare a variable to hold the instance of the SteeringWheel object; then you'll initialize the variable as a "new" SteeringWheel object using the "new" keyword.

Example:

```
var theWheel = new SteeringWheel();
```

Whenever you instantiate an object, you MUST use the keyword, new, before the call to the object constructor, as you see above. Also, notice the opening/closing parentheses characters following the call to the SteeringWheel constructor (and prior to the semi-colon end-of-line marker); these parentheses are called the "function call" operator (which calls/invokes a function), and are an essential part of the object instantiation process.

A FUNCTION, in JavaScript, is a collection of lines of code, a program, which does something (like switch an image, or pop up a new window, or BUILD AN OBJECT, etc). In order to execute a function, to make a function "run" in JavaScript, you must state the name of the function, followed by the "function call" operator (the opening/closing parentheses characters); this will cause the function to execute when your code is executed. Note: we'll talk about making our OWN functions a little later on in this module.

It turns out that an object constructor, such as the SteeringWheel object constructor in the above example, is made in JavaScript out of a FUNCTION. The same function syntax that you will use to create miniature programs later on is also used to make object constructors!

Just as a side note, it turns out that FUNCTIONS are also OBJECTS themselves; this is because almost everything in JavaScript is built out of objects. I'm sure that this seems confusing now, but don't worry about it; as you get deeper into JavaScript, over time, it will all start to make much more sense to you.

Here's the code again:

```
var theWheel = new SteeringWheel();
```

Whenever you instantiate an object, you will DECLARE a variable, then INITIALIZE that variable to be a NEW instance of whatever OBJECT CONSTRUCTOR you want. An object constructor is really a FUNCTION, which you must execute in order to create your object; this is done by stating the name of the object constructor function (in this case, SteeringWheel), followed by the function call operator (the parentheses characters). Again, the function call operator is necessary in order to execute the object constructor function. As in all lines of code in JavaScript, you must end the line with a semi-colon end-of-line marker.

Once you have created an instance of your desired object, you may access the PROPERTIES and METHODS of the object by using DOT SYNTAX.

Dot syntax is a way of accessing child elements of a parent object (remember parents and children?) by using period/dot (.) characters between parent and child elements to indicate a path through a complex tree of relationships, much as the slash (/) character is used in URL syntax to indicate a path through a complex directory structure on a web server.

Here's a much simpler example. If I wanted to put $10.00 cash into an empty cash register in a local coffee house in Willits, CA, USA, using JavaScript dot syntax to indicate the path to the cash register, the line of JavaScript code might look something like this:

earth.northAmerica.usa.cal.mendoCo.willits.pjCoffee.cashRegister = 10;

Dot syntax always creates a path from the most general to the most specific element, reading from left-to-right, and, as you can see from the above example, there are NO SPACES IN THE NAMES of the elements or between the dots!

Back to the instance of our SteeringWheel object, theWheel:

The SteeringWheel object has one property, direction. Object properties can be accessed via the instance of the object using dot syntax. In the following examples, I am assuming that theWheel has already been declared and instantiated.

Example:

```
theWheel.direction = "left";
```

In the above example, I have set the direction property of theWheel object equal to the string, "left". Surprise! An object property is really a VARIABLE which is attached to a particular object; this property may be filled with whatever data we like (in this case, a string).

Having executed the above line of code, theWheel is now pointed towards the left.

Example:

```
theWheel.direction = "right";
```

Now, theWheel is pointed towards the right.

Note: At this point, students often point out to me that a property is supposed to be static, yet the direction property of theWheel seems to be turning the steering wheel of the car rather than merely reflecting which direction theWheel is pointing! Well, oftentimes object properties are not as static as I have led you to believe (remember, I said "usually" static!). In a real JavaScript object, however, I would

probably have written a method called "turnWheel()" which actually turned the wheel, and the direction property would merely have been a string which indicated the current direction of the wheel.

Please cut me some slack, I'm trying to make a simple example here!

Now, if I wanted to honk the horn of theWheel, I would need to call the honk() method of theWheel.

A METHOD is really a FUNCTION which is attached to an object, just as a PROPERTY is really a VARIABLE attached to an object (again, a function is a program which performs some action)! Therefore, I use the function call operator (opening/closing parentheses) after the name of the method (and dot syntax, of course).

Example:

```
theWheel.honk();
```

The above example executes the honk() method of theWheel, causing theWheel's horn to honk.

Example (complete code):

```
var theWheel = new SteeringWheel(); theWheel.direction = "left"; theWheel.direction = "right"; theWheel.honk();
```

Boy, if we drove like this down the road, we'd probably get a ticket for reckless driving!

You can create more than one instance of an object, simply by declaring and instantiating more copies of the object.

Example:

```
var theWheel = new SteeringWheel(); var theSecondWheel = new SteeringWheel(); theWheel.direction = "left"; theSecondWheel.direction = "center"; theWheel.honk();
```

In the above example, there are two copies of the SteeringWheel object (hopefully attached to two separate cars!). The direction property of theWheel is set to "left", while the direction property of theSecondWheel is set to "center". Then, I honked the horn of theWheel, but I left theSecondWheel silent.

Each instance of an object is separate, and contains separate information. Even though both objects in the above example are SteeringWheel objects, they are completely independant from one another!

I can make as many instances of an object as I want, but I think TWO SteeringWheel objects is enough for now...

2 JavaScript Objects and Functions: Part Two

Let's look at a REAL example of JavaScript objects in action. In this example, I am going to instantiate two Image objects, which are used to represent GIF/JPEG pictures for use on a web page. I am then going to set the src (source) properties of the Image objects to appropriate URLs; these src properties are identical to the SRC attribute of an IMG tag.

Example:

```
var picture0 = new Image();
var picture1 = new Image();
picture0.src = "capitalA.gif";
picture1.src = "capitalB.gif";
```

Now, let's say that I have an IMG tag named "fred" on my web page.

Example:

```
<img src="capitalD.gif" width="54" height="54" name="fred" />
```

I could access the image, fred, on my web page using something called the "simplified document object model", which allows me to use dot syntax to point to an element on an HTML page.

Example (abbreviated):

```
window.document.fred
```

Example (abbreviated; looking at SRC attribute of fred):

```
window.document.fred.src
```

Example (full line of code):

```
window.document.fred.src = picture0.src;
```

window is the instance of the Window object which represents the current web browser window. document is the instance of the Document object which represents the current HTML page in the web browser window. fred is the name of the IMG tag on the HTML page. src is the SRC attribute of the fred IMG tag.

In the above example, I am setting the src property for fred equal to the src property of the picture0 Image object, thus changing fred's picture from a graphic of the letter D to a graphic of the letter A! I've just performed an image switch! Hooray!

Now, I'm going to switch fred's image to picture1, the graphic of the capital letter B:

```
window.document.fred.src = picture1.src;
```

Woowee! I've switched that image again! I'm programming in JavaScript, doing something useful! Wow!

Alright, so this is not ALL the code that you'll need to deal with image switching, but it outlines the basic principles. First, you instanciate image objects with src properties set to URLs for the images that

you want. Then, you refer to the named IMG tag on the page using dot syntax and the simplified document object model, and reset the src property of the named IMG tag equal to the src property of the desired instance of the Image object.

Almost every web site in creation uses image switching to make rollover effects on graphical hyper-references! Be warned, however: all images involved in a switch MUST have the same dimensions, the same width and height!

The SteeringWheel Object Revisited

We need to talk more about the honk() method of the SteeringWheel object, to illustrate additional basic JavaScript programming principles.

Our computerized car horn is capable of doing more than merely honking; it can play tunes, as well! There are two tunes it can make, along with the generic honk sound: Beethoven's Fifth Symphony Theme, and Mary Had a Little Lamb.

Here are the string values representing these car horn tunes:

"5th" for Beethoven's Fifth Symphony (dah dah dah DAAAAHHHH!).
"MHALL" for Mary Had a Little Lamb.
"honk" for the generic honking sound.

Now, I need to pass these string values into the horn() method of the SteeringWheel object. To do this, I need to place the desired string BETWEEN the parentheses of the function call operator for the horn method in my JavaScript command. When you do this, you are said to be "passing a parameter" or "passing an argument" to the method.

Example:

```
theWheel.horn("5th");
```

In the above example, I am passing the string value "5th" as an ARGUMENT or PARAMETER (these words are often used interchangeably here) to the horn method of theWheel. This causes the horn in theWheel to play Beethoven's Fifth rather than the standard honking sound. Note: everything I say about parameters and arguments in this section may ALSO be applied to functions, since methods are really functions attached to an object!

Let's have the horn play Mary Had a Little Lamb.

Example:

```
theWheel.horn("MHALL");
```

Ah, what a sweet sound!

By passing different arguments to a method, I can cause that method to behave in different ways.

In JavaScript generally, you will pass ARGUMENTS or PARAMETERS for many different reasons and to many different effects. You can pass any data type you like as an argument to a method, whether it be number or string or boolean or whatever. Each built-in JavaScript method will require different sorts of arguments, and will perform different kinds of actions.

Example:

```
theWheel.horn(false);
```

I passed the boolean value, false, to the horn method. The way the horn method is written, a false value passed as an argument prevents the horn from honking. Many methods can accept boolean values of true or false in this way.

Many methods are programmed to accept more than one argument. If you wish to pass more than one argument to a method, you must separate each argument with a comma (and an optional space, if you wish).

Example:

```
theWheel.horn("5th", "MHALL");
```

In the above example, the horn method first plays Beethoven's Fifth, then plays Mary Had a Little Lamb.

Example:

```
theWheel.horn("honk", "5th", "MHALL", "honk");
```

In the above example, the horn method plays a sequence of sounds, starting with a honk, moving to Beethoven's Fifth, then Mary Had a Little Lamb, and ending with another honk. In JavaScript, methods may accept one or more arguments, or a changeable number of arguments, depending upon how that method is written.

Let's look again at our real world example, using the Image object.

Example:

```
var picture0 = new Image(54, 54);
```

It turns out that the Image object constructor can accept TWO arguments, both integers. The first argument represents the WIDTH of the image in pixels, while the second argument represents the HEIGHT of the image in pixels. These arguments initialize the width and height properties of the instance of the Image object in question.

Example:

```
var picture0 = new Image(54, 100);
var myWidth = picture0.width;
var myHeight = picture0.height;
```

The variable myWidth is now equal to 54, which is the value of the width property of picture0. The variable myHeight is now equal to 100, which is the value of the height property of picture0.

One Last Note on Variables

Variables are boxes for information. By stating a variable name in JavaScript code, you are passing or manipulating the information contained in that variable.

Remember this example from a previous section?

```
var louie = 10.25;
var josie = louie;
```

By stating louie's name in the second line of code above, I was able to copy or pass the information from louie into josie by using the "gets" operator. In the same way, I can pass information from a variable into a method by stating the name of the variable between the parentheses of the function call operator.

Example:

```
var myHornSound = "5th";
var theWheel = new SteeringWheel();
theWheel.horn(myHornSound);
```

I just love to hear Beethoven's Fifth Symphony!

Example:

```
var myHornSound0 = "5th";
var myHornSound1 = "MHALL";
var theWheel = new SteeringWheel();
theWheel.horn(myHornSound1, myHornSound0, "honk");
```

In the above example, I passed THREE arguments to the horn method of theWheel; two of the arguments were variable names, and one was an actual string with no variable involved. Some of my arguments can be variables, while others of my arguments can be direct data; this is perfectly legal.

The following example is NOT legal:

```
var myHornSound = "5th";
var theWheel = new SteeringWheel();
theWheel.horn("myHornSound");
```

Notice how I passed the STRING "myHornSound" to the horn method when I REALLY wanted to pass the variable named myHornSound?
"myHornSound" is just a bunch of text characters, a string, while myHornSound is an actual call to the variable itself. The above code would break your JavaScript, since the horn method does not recognize the string, "myHornSound", as a legal argument value.

Summary

A VARIABLE may hold an INSTANCE of an OBJECT. Each object has PROPERTIES and METHODS. Properties are essentially variables attached to an object, while methods are essentially FUNCTIONS attached to an object. A FUNCTION is a program which performs some action. Whenever you wish to execute a function (or a method), you must state the name of the function, followed by the FUNCTION CALL OPERATOR (which is represented by opening and closing parentheses characters). Properties and methods of an object are accessed using DOT SYNTAX, which allows you to trace the path of relationships between parent and child objects. Note: as you might expect, then, properties and methods of an object are child elements of the object. When you execute a method or function, you may pass ARGUMENTS or PARAMETERS to that method or function. Arguments/parameters passed to a method/function can influence the behavior of that method/function in some fashion. Data may be passed as an argument to a method/function either directly (by stating the actual number, string, or boolean value), or by passing a variable containing the data (by stating the variable name).

3 Common Event Handlers

Because of an incompatibility issue with Insight, click here for the next chapter about Common Event Handlers.

4 Basic Function Syntax

Inside a SCRIPT tag, you may define as many FUNCTIONs as you wish. A function is a piece of executable code, a program. A function can be triggered either by a call from an event handler, set in motion by a user, or by a call from another function.

A function requires the following syntax:

```
function doStuff() { };
```

First, you must state the JavaScript keyword, function, which tells the JavaScript interpreter that you want to create a FUNCTION. This is followed by a space, then the NAME of the function, followed by the parentheses characters (which are NOT, strangely, the function call operator here, but, rather, serve another purpose which we'll talk about a little later). The parentheses are followed by another space, then the opening and closing curly-braces we used in CSS; this is all ended, of course, with a semi-colon end-of-line marker.

This statement of a function is called a FUNCTION DECLARATION.

Note: the semi-colon end-of-line marker at the end of a function declaration is RARELY, or NEVER, seen, and is usually omitted entirely.

Example (correct):

```
function doStuff() { }
```

Between the curly-braces, then, we can define lines of code. In fact, just as in CSS, we will break out our lines of code, putting the opening and closing curly-braces on separate lines, and each line of JavaScript code between them on separate lines.

Example:

```
function doStuff() {
 var george = 1;
 alert(george);
}
```

Again, this function would be placed inside of a SCRIPT tag. You may define as many functions as you like inside of your SCRIPT tag.

Example:

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

function doStuff() {
 var george = 1;
 alert(george);
}

function doSomethingElse() {
 alert("Howdy");
}

</script>
```

White Space in JavaScript

You will notice in the above examples that I have indented the lines of JavaScript code inside of the function declaration; this is a customary practice, and, unlike with HTML, it is supported and perfectly legal. Indent child elements (inside of curly-braces) one tab in from parent elements.

As you can see in the above example, I have also used white space to separate my functions. This practice, too, is customary, supported, and legal.

The one thing you have to beware of is placing CARRIAGE RETURN characters anywhere but at the ends of lines of code (or as blank lines). JavaScript can potentially interpret carriage return characters as SEMI-COLON end-of-line markers; if you put a carriage return in the middle of a command, the code may break.

Example (WRONG):

```
function doStuff() {
 alert(
 "Howdy"
 );
}
```

As you can see in the above example, I have placed illegal carriage returns in the middle of my function call operator, the parentheses. This will cause my code to break.

Example (CORRECTED):

```
function doStuff() {
 alert("Howdy");
}
```

Calling a Function from an Event Handler

Once I have declared and defined my functions, I can call those functions from the event handlers or javascript absolute URLs in the BODY of my HTML page.

Example (abbreviated):

```
<a href="#" onClick="doStuff();">Link One</a>
<a href="javascript:doSomethingElse();">Link Two</a>
```

Example (in context):

```
<html>
<head>
<title>Sample JavaScript</title>
<script type="text/javascript">

function doStuff() {
 var george = 1;
 alert(george);
}

function doSomethingElse() {
 alert("Howdy");
}
```

```
</script>
</head>
<body>
<p><a href="#" onClick="doStuff();">Link One</a></p>

<p><a href="javascript:doSomethingElse();">Link Two</a></p>
</body>
</html>
```

Here is the above example displayed.

In the above example, do you notice how the code remains inactive UNTIL you click on the hyper-references in the BODY? When the user interacts with an HTML element, a function is called (from either an event handler or from the javascript absolute URL) and executed.

Organizing our JavaScript code inside functions in the SCRIPT tag allows us to keep our HTML code relatively brief and uncluttered. It also opens up a door to a lot of power which is outside of the scope of this introduction to explore.

I will mention again, however, that a function may be called from within ANOTHER function.

Example (abbreviated):

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

function doStuff() {
 var george = 1;
 alert(george);

 doSomethingElse();
}

function doSomethingElse() {
 alert("Howdy");
}

</script>
```

Example (in context):

```
<html>
<head>
<title>Sample JavaScript</title>
<script type="text/javascript">

function doStuff() {
 var george = 1;
 alert(george);

 doSomethingElse();
}

function doSomethingElse() {
 alert("Howdy");
}

</script>
</head>
<body>
<p><a href="javascript:doStuff();">Link One</a></p>
</body>
```

```
</html>
```

Here is the above example displayed.

Notice how I only called the doStuff() function from within my event handler? The doStuff() function has been rewritten to call the doSomethingElse() function. So, from one event, I can call a function which can call many functions!

I know that this doesn't seem like such a big deal to you right now, but it provides a means of writing the most sophisticated and powerful code imaginable.

I can call both built-in JavaScript methods (such as the alert() method of the window object), and functions which I have defined myself! Pretty slick!

5 Variables: Local and Global

Variables come in two flavors: local and global.

Local variables are temporary. A local variable is declared, initialized, and used for a short period of time, then thrown away. A local variable in JavaScript can only be used within a single function; no other functions have access to that particular variable; when a function is finished executing, the local variables for that function are thrown away.

Global variables are persistent. A global variable is created when your web page is loaded, can be accessed by many different processes and functions, and lasts until your web page is unloaded from the browser window.

Again, global variables last for the duration of your web page or program; local variables are used by one process or function, then thrown away.

The distinction between local and global variables is very important in JavaScript programming. We will not have time to explore all of the ramifications of local and global variables in this brief introduction. However, I will be referring to local and global variables frequently in the upcoming sections.

Global variables in JavaScript are declared at the base level of the SCRIPT tag.

Example:

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

var george = 1;
var fred = "Hi";

</script>
```

In the above example, both george and fred are GLOBAL variables because they have been declared at the base level of the SCRIPT tag. Global variables can be accessed within all functions.

Example (abbreviated):

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

var george = 1;
var fred = "Hi";

function doStuff() {
 alert(george);
}

function doSomethingElse() {
 alert(fred);
}

</script>
```

Example (in context):

```
<html>
<head>
<title>Sample JavaScript</title>
```

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

var george = 1;
var fred = "Hi";

function doStuff() {
 alert(george);
}

function doSomethingElse() {
 alert(fred);
}

</script>
</head>
<body>
<p><a href="#" onClick="doStuff();">Calling doStuff()</a></p>

<p><a href="javascript:doSomethingElse();">Calling doSomethingElse()</a></p>
</body>
</html>
```

Here is the above example displayed.

I can also change the contents of (or operate upon or manipulate) global variables from within functions.

Example (abbreviated):

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

var george = 1;
var fred = "Hi";

function doStuff() {
 alert(george);
 george = "Boo";
 fred = "Howl";
}

function doSomethingElse() {
 alert(fred);
 george = "Yum";
 fred = "Delicious!";
}

</script>
```

Example (in context):

```
<html>
<head>
<title>Sample JavaScript</title>
<script type="text/javascript">

var george = 1;
var fred = "Hi";

function doStuff() {
 alert(george);
 george = "Boo";
 fred = "Howl";
}
```

```
function doSomethingElse() {
 alert(fred);
 george = "Yum";
 fred = "Delicious!";
}

</script>
</head>
<body>
<p><a href="#" onClick="doStuff();">Calling doStuff()</a></p>

<p><a href="javascript:doSomethingElse();">Calling doSomethingElse()</a></p>
</body>
</html>
```

Displayed:

As you can see, each time we call one of the functions, we are changing the contents of the global variables.

Local variables are declared WITHIN a function declaration ONLY. Local variables will last only as long as the function is executing. When the function has finished executing, its local variables are thrown away. Functions may NOT access the local variables of other functions.

Example (abbreviated):

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

function doStuff() {
 var george = "Wow";
 alert(george);
}

function doSomethingElse() {
 var fred = "How about that!";
 alert(fred);
}

</script>
```

In the above example, the variable george is a local variable of doStuff(), while the variable fred is a local variable of doSomethingElse(). Because fred and george are local variables in different functions, I can not access fred from doStuff(), nor can I access george from doSomethingElse(); either of these actions would result in a JavaScript error.

Example (INCORRECT):

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

function doStuff() {
 var george = "Wow";
 alert(fred);
}

function doSomethingElse() {
 var fred = "How about that!";
 alert(george);
}

</script>
```

In the above INCORRECT example, I am trying to access george from doSomethingElse() and fred from doStuff(), neither of which are possible actions.

Example (POSSIBLE):

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

function doStuff() {
 var george = "Wow";
 var fred = "How about that!";
 alert(george);
 alert(fred);
}

</script>
```

In the above example, george and fred are both local variables of the SAME function; any process within that function, then, could access either george and/or fred.

We'll talk more about handling local variables in the next sections.

One Final Note

Each function may contain local variables which have the SAME names as local variables contained in other functions. For instance, the following code is perfectly CORRECT:

```
function doStuff() {
 var myNumber = 1;
 alert(myNumber);
}

function doSomethingElse() {
 var myNumber = 10;
 alert(myNumber);
}
```

Because local variables are temporary, only existing for the duration of a given function, other functions may use the SAME local variable names. The technical reason for this is called PROTECTED NAMESPACE, which we don't need to go into here. Suffice it to say, that each function is like a separate universe, with its own collection of local variables, and that these local variables are separated or protected from the local variables in other functions.

So, even though I am referring to the local variable myNumber in BOTH doStuff() and doSomethingElse(), these two variables, because they are both LOCAL variables, have NO relationship to one another whatsoever.

6 Local and Global Variable Naming Conventions

Officially, it doesn't matter what you name local and global variables, as long as you follow the naming conventions that we have discussed in earlier modules. In practice, however, there are standardized local and global variable naming conventions that many programmers follow.

Here is the naming convention that I follow:

Local variable names always begin with the prefix "my".

Example:

```
myText
myCounter
myPosition
myNumber
```

Global variable names always begin with the prefix "the".

Example:

```
theText
theCounter
thePosition
theNumber
```

Following a local/global naming convention, like the one outlined above, can really help you keep your local and global variables straight when you start writing more complex code. Trust me, it makes a difference!

I also strongly recommend making your variable names reflect their purpose; this will make your code read much more like regular English than like techno-gibberish. For instance, if your variable is holding the result of a calculation, name it myResult or myAnswer rather than goobah or fred. Which of the following code examples is easier to read?

```
var goobah = 1 + 1;
var myAnswer = 1 + 1;
```

If I were dealing with temporary information, I might call my variable temp or myTemp. If I were dealing with some HTML text, I might call my variable myHTMLText or myText or theText (depending on whether it were local or global).

In the same way, if I were writing a function which switched images, I would name that function using verb syntax, reflecting its purpose: switchImage() or handleImageSwitch(), rather than grumble() or snuffBox().

I had one student who named all of her functions after days of the week, and all of her variables after friends. Needless to say, she couldn't remember which function did what, or what the variables were for. Another person had the bright idea of naming a utility function "fillUpMyCokePlease()". Again, no one can ever remember exactly WHAT that function is supposed to do! You get the idea.

Do yourself a favor: name variables and functions after their purpose, not after your dog or to commemorate the purchase of your new lava lamp. You'll thank yourself later, trust me...

7 Functions: Declaring Arguments

You have already passed arguments to a built-in JavaScript method, the alert() method of the window instance of the Window object.

Example:

```
alert("Howdy");
```

Example (conservative syntax):

```
window.alert("Howdy");
```

As mentioned earlier, methods are functions which are attached to objects. Functions, then, can receive arguments. Now that you know how to code a function, I can show you how to declare arguments for your functions.

Example:

```
<html>
<head>
<title>Sample JavaScript</title>
<script type="text/javascript">

function doStuff() {
 alert("Howdy");
}

</script>
</head>
<body>
<p><a href="#" onClick="doStuff('Boo');">Calling doStuff()</a></p>
</body>
</html>
```

In the above example, I have declared one function, doStuff(). In the BODY of my page, I have called doStuff() from the onClick event handler in the hyper-reference; I am passing doStuff one argument, the string "Boo".

Now, how would I access that argument inside of the doStuff() function, and how could I pass the argument's data to the alert() method call in doStuff()?

The easiest way to access an argument inside a function (and the only one we're going to explore in this introductory lecture) is to declare a local variable which will hold the argument data; this local variable MUST be declared within the parentheses characters following the name of the function, doStuff, in the function declaration. Note: You would NOT use the keyword, var, when declaring this type of local "argument" variable.

Example (abbreviated):

```
function doStuff(fred) {
 // some code here...
}
```

By declaring the local variable, fred, within the parentheses of the function declaration in the manner

shown above, I am creating a container to hold the argument value being passed into the function. Within the function, then, I could access the local variable, fred, normally. Whatever argument has been passed to this doStuff function would be accessed via fred.

Example (abbreviated):

```
function doStuff(fred) {
 alert(fred);
}
```

Now, in the above example, what value does fred actually contain, and what string would display in the alert dialog box? The answer is, it depends on what value has been passed as an argument to the doStuff function.

Example:

```
<html>
<head>
<title>Sample JavaScript</title>
<script type="text/javascript">

function doStuff(fred) {
 alert(fred);
}

</script>
</head>
<body>
<p><a href="#" onClick="doStuff('Boo');">Calling doStuff()</a></p>

<p><a href="#" onClick="doStuff('Ouch');">Calling doStuff() Again</a></p>
</body>
</html>
```

Here is the above example displayed.

When the first link calls the doStuff function from its onClick event handler, it passes one argument to doStuff(), the string "Boo". That string value is passed into the local variable, fred, because fred has been declared as a local "argument" variable within the parentheses of the doStuff function declaration. The variable fred now contains the string value "Boo", which is passed to the alert() method in the above code; the alert() method then causes an alert dialog box to pop up which says "Boo".

The second link performs the same actions, only passing the string "Ouch" instead of "Boo".

As you can see, both hyper-references call the SAME function, but each hyper-reference passes that function a different argument value. The function itself always performs the same tasks, but its behavior is altered slightly depending on the value of the argument passed into the local variable, fred.

This is the seed of how you create generalized functions to perform variations on the same task! Without these sorts of local argument variables, a function would be forced to behave identically EVERY time you called it, just like the functions you saw in the previous sections. WITH these local "argument" variables, a function becomes much more flexible in the tasks it can perform.

Just as I can pass more than one argument to a method/function, I can create more than one local "argument" variable by declaring those variables within the parentheses of the function declaration. Each of the local variable names declared between the parentheses of the function declaration would be

separated by a comma.

Example (abbreviated):

```
function doStuff(fred, george) {
 alert(fred);
 alert(george);
}
```

Example (in context):

```
<html>
<head>
<title>Sample JavaScript</title>
<script type="text/javascript">

function doStuff(fred, george) {
 alert(fred);
 alert(george);
}

</script>
</head>
<body>
<p><a href="#" onClick="doStuff('Boo', 'Scared You!');">Calling doStuff()</a></p>

<p><a href="#" onClick="doStuff('Ouch', 'That Hurt!');">Calling doStuff() Again</a></p>
</body>
</html>
```

Here is the above example displayed.

Arguments are always passed into the local "argument" variables for a function in sequence from left to right. The first argument passed to the above function would be passed into the first local argument variable, fred, and the second argument passed to the above function would be passed into the second local argument variable, george. The local "argument" variable name is irrelevant to which argument gets passed into which variable; the arguments are always passed into the function in order, left to right.

Being able to declare local variables to hold arguments for a function is an extremely powerful tool. Obviously, we haven't time to explore all of the ramifications of this capability.

8 The return Keyword

A JavaScript function, when executed, may send some piece of data back to whatever process originally called it using the return keyword.

For instance, I might make a calculation function which takes in numbers as arguments, performs a complex calculation (a mortgage payment calculation would be an appropriate example), and spits out an answer. The alert() method that you have used so far to spit out answers is obviously inadequate to real work; the return keyword is what you will need to use to get information out of a function.

Built-in JavaScript methods use the return keyword to spit out many different types of values, which you will need to intercept and process for one reason or another, placing those returned values into local or global variables so that you can use and manipulate that information. In addition, you will be called upon to return values yourself, not only between functions, but also to the web browser. And it's not only numbers, strings, or booleans which can be returned from a function or method; references to objects can ALSO be returned from a function, such as a reference to a new browser window. I'll talk more about many of these subjects later on.

Let's look at the addMe() function you created in the previous module.

Example:

```
function addMe(myFirstNum, mySecondNum) {

var myAnswer = myFirstNum + mySecondNum;

alert(myAnswer);

}
```

The above example function DOES spit out an answer, but it does so via the medium of an alert dialog box. A function like this would NOT spit out information in an alert dialog box, it would normally be called by another function, which would need to receive the calculation result; this requires the use of the return keyword.

Example (abbreviated):

```
function addMe(myFirstNum, mySecondNum) {

var myAnswer = myFirstNum + mySecondNum;

return myAnswer;

}
```

Example (in context):

```
function doStuff() {

var myNumberOfCats = 4;
```

```
var myNumberOfDogs = 2;

var myTotalNumberOfPets = addMe(myNumberOfCats, myNumberOfDogs);

alert(myTotalNumberOfPets);

}

function addMe(myFirstNum, mySecondNum) {

var myAnswer = myFirstNum + mySecondNum;

return myAnswer;

}
```

In the above example, I have called the addMe() function from inside doStuff(); this call passes myNumberOfCats and myNumberOfDogs as arguments to the addMe() function, which adds these values together (using its local variables, myFirstNum and mySecondNum). Once this calculation has been finished, the variable containing the result (myAnswer) is placed after the return keyword, which sends that information back to the calling function, doStuff(). In effect, the addMe() function call in doStuff() is replaced with whatever value is returned from the addMe() function; in this case, the number 6. That answer, 6, is put into the local variable, myTotalNumberOfPets, by the "gets" operator, whereupon that answer is displayed; for simplicity's sake, I have used the alert() method to display the answer, but a real function would probably be doing something much more involved with the information.

This is an over-simplified demonstration of the return keyword. Obviously, you would never need to go through this much trouble just to add two numbers together. I am trying to show you the mechanics of this process using code that is extremely easy to analyze.

Note: If a function has just one return statement then that statement MUST be the LAST statement in a function, because once the return keyword has sent its value back to the calling function, the function containing the return keyword ENDS, regardless of whether there is any more code following that keyword or not.

Example:

```
function addMe(myFirstNum, mySecondNum) {

var myAnswer = myFirstNum + mySecondNum;

return myAnswer;

alert(myAnswer);

}
```

In the above example, the call to the alert() method would NOT be executed because it follows the statement containing the return keyword. Again, after the return keyword statement has been executed, any lines of code following that statement are ignored.

9 Checking for Valid Numeric Input

In order to validate whether a user has entered a numeric value, you can use the isNaN() function.

The isNaN() function determines whether a value is an illegal number (Not-a-Number).

This function returns true if the value is NaN, and false if not.

Let's take an example. Assume you want the user to enter an integer in an input text field before processing the input with calculations. Here are the main steps:

1. Obtain the input from the form as a string.

2. Use parseInt() to change the numeric string to an integer. If the input is not a numeric string then it's parseint value will now contain the value of NaN (Not a number).

3. Use if..else logic to determine if the new value is a number or not a number.

(**Note that using the above sequence of coding will also validate a number followed by alphabetic or special characters, as numeric, e.g., 22h. If you wish to invalidate this type of input then you should first test for isNaN and then use parseInt() if the the input is totally numeric. More about this in the next chapter.)**

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

function process() {

// define variables

var number, numberP;

// get input number value from form using getElementById

number = document.getElementById("input").value;

// change string to an integer

numberP = parseInt( number );

// if numberP is Not a Number then alert error message, else display number

// Note that the 2nd alert is not normally needed, as it's only here to show the value of a non-numeric input as NaN

if ( isNaN(numberP) )
{
window.alert("Inputted value " + number + " is Not a number - try again");

window.alert("The parseInt value is " + numberP + " - try again");
}
```

```
else {
document.myform.comment.value = ( numberP + " is a number" );
}

}

</script>

</head>

<body>

<section class="body">

<h2>Check for Numeric Input</h2>

<form id="myform" name="myform" action="">

Enter input: <input type="text" id="input" size="10" >

<br ><br>

<input type="button" onclick="process()" value="SUBMIT">
<input type="reset" value="RESET"><br ><br>

Comment: <input type="text" name="comment" size="25" >

</form>
</section>

</body>
```

Displayed

Note that if decimals are to be accepted in your page as numeric input then use parseFoat() instead of parseInt() in the script.

When you check for valid input and find one or more of the input fields to be invalid, it's a good idea to clear the input fields and focus the cursor to the first input field for the convenience of the user.

For example, where hw and fin are variable names of the input fields in the form:

```
function getAvg() {

var hwF, finF, finAvg, grade;

hwF = parseFloat(document.form.hw.value);

finF = parseFloat(document.form.fin.value);
```

```
if (isNaN(hwF) || isNaN(finF) || hwF < 0 || finF < 0 || hwF > 100 || finF > 100) {

document.myform.hw.value = " "; // clears hw input
document.myform.fin.value = " "; //clears final exam input
document.myform.hw.focus(); // focuses cursor to hw input

window.alert("Both values must be numbers and not be less than 0 and not greater than 100!");
}
else
{


// process

}
```

Alternatively:

You can give the reset button an "id" and then can use the following to reset the input fields:

```
document.getElementById("myResetButton").click();
```

or you can invoke the reset() function on the form itself:
document.forms.myform.reset();

Note that <input type="reset"> will only reset the form fields. If your form also had error messages from a previous submission of the form and you want these messages cleared then another method is using location.reload() in your reset statement. For example:

```
<input type="reset" value="RESET" onclick="location.reload();">
```

Instead of using an alert when input is invalid, you can use **innerHTML and getElementById** to display an error message directly on the page, when input is invalid. This page does exactly that.

10 More on parseInt() and isNaN

This chapter was written by Michael Ogi, except for the very last part.

Checking for the empty string ""

If you want to check whether the user entered nothing in the input box, you need to check whether it's equal to "", before you do any conversions:

```
var userInput = document.getElementById("myInputField").value;
if (userInput == "") {
        window.alert("You didn't enter anything in the input field.");
}
```

Value of text fields

Note that the value of a text field is always a string. For example, if the user enters 123 in the text field, the value is the string "123", as opposed to the number 123. If the user enters nothing, the value is the empty string "".

Number conversions

JavaScript provides three functions to convert a value to a number:

1. **Number()** casting function -- converts a non-numeric value to a number
2. **parseInt()** function -- converts a string to an integer
3. **parseFloat** function -- converts a string to a real (floating-point) number

Note that **parseInt()** and **parseFloat()** deal specifically with converting strings to integers or floating-point numbers, respectively.

The parseInt() Function

When the input is a string and you are working with integers, it is usually best to use the **parseInt()** function to try to convert the string to an integer, and then use **isNaN()** on the result to see whether the conversion was successful. The rules for **parseInt()** are as follows:

- Leading white space in the string is ignored.
- If the first non-leading white space isn't a number or minus sign, **NaN** is returned.
- If the first character is a number, then the conversion continues until the first non-numeric character is found.

Here are some examples:

```
parseInt("") // returns NaN
parseInt("123") // returns 123
parseInt("123.45") // returns 123
parseInt(" 123.45") // returns 123
parseInt("123abc") // returns 123
parseInt("-123.45abc") // returns -123
```

The **parseInt()** function also has a second argument that indicates the radix or base of the input string, so it knows whether to interpret the input as binary, octal, decimal, hexadecimal, etc.:

```
parseInt("11", 2) // returns 3 (parsed as binary)
parseInt("11", 8); // returns 9 (parsed as octal)
parseInt("11", 10); // returns 11 (parsed as decimal
parseInt("11", 16); // returns 17 (parsed as hexadecimal)
```

If you don't pass in the second argument, **parseInt()** will assume that a string that begins with "0" followed by a number is an octal value, and a string that begins with "0x" followed by a number is a hexadecimal value:

```
parseInt("031") // returns 25 (parsed as octal)
parseInt("0x2A") // returns 42 (parsed as hexadecimal)
```

For that reason, it is sometimes a good idea to pass in 10 as the second argument to the **parseInt()** function, since normally we are expecting decimal values, and we don't want leading 0s to affect the outcome.

Which to use first: parseInt() or isNaN()?
There's no right answer, but I think normally you would try to convert the string to an integer using **parseInt()**, and then test whether the conversion was successful via **isNaN()**.

isNaN() will try to convert its input to a number via the **Number()** casting function and then test whether the result is **NaN. Number()** differs from **parseInt()** in several ways, but for string input, I think the primary difference is how it deals deals with "" and how it deals with input that starts with numbers followed by non-numeric characters.

Case 1: User input is the empty string.
The **isNaN()** check won't catch this case because **Number("")** returns 0:

```
Number("") // returns 0
isNaN("") // returns false, because "" will convert to 0, and 0 is a number!

parseInt("") // returns NaN
isNaN(parseInt("")) // returns true
```

Case 2: Input starts with a number, with subsequent non-numeric characters.
If you want to reject this kind of input, you could actually use **isNaN()** first; it depends on how strict you want to be. Do you want to be flexible and interpret "123abc" as 123 automatically, or reject the input altogether?

```
Number("123abc") // returns NaN
isNaN("123abc") // returns true

parseInt("123abc") // returns 123
isNaN(parseInt("123abc")) // returns false
```

Examples of validating a number as TOTALY numeric

The following example is similar to the example from the previous chapter. The difference is that isNaN() is used **before** parseInt() because we want to validate input that is totally numeric before continuing processing. In addition, if the input is not **totally numeric** then after the alerts, I clear the input field and focus the cursor to it. Note that I also need to test whether the input is empty. If I fail to do this then an input that is empty will display NaN in the result field.

Here is an example that **uses an alert** to display an error message when input is invalid.

Here is an example that **uses innerHTML and getElementById** to display an error message when input is invalid.

11 Using jQuery to check for Valid Numeric Input

An easy way to check for valid numeric form input is to use jQuery. Some websites suggest using a regular expression for validating numeric form input with jquery. I have discovered a jquery plugin that uses a simpler method for data validation.

Within the head section you would include:

The script for accessing the jquery library:
<script type="text/javascript" src="http://fog.ccsf.edu/~srubin/jquery.js"></script>

The script that validates numeric input:
<script type="text/javascript" src="http://fog.ccsf.edu/~srubin/validate.js"></script>

Now we set the jquery rules. In the following example, I want my form input to be invalid if no data is entered, non-numeric data is entered, or the numeric data is not between 0 and 10.

```
$(document).ready(function(){
$("#myform").validate({
// Rules for each input item
rules:
{
num: { required: true, number: true, min: 0, max: 10}

}
});
});
```

Note that myform is the id of my form statement and num is the name of the input form element. I also code as num the id of the input form element because I am using getElementById in the processing. It should be fairly obvious what the above rules are testing., i.e., the input form data is required, the data must be numeric, and the data must be within the range of 0 to 10.

Then in my processing function, I allow the processing if the data entered is valid. Thus, I would have:

```
function process() {

if ($("#myform").valid()) {

//process data

}
}
```

When data is not entered in the form input element and the submit button is clicked then an error message is automatically displayed next to the form input element, viz., *This field is required.* I can style this error message using CSS.

When non-numeric data is entered in the form input element and the submit button is clicked then an error message is automatically displayed next to the form input element, viz., *Please enter a valid number.*

The above example displayed.

12 jQuery UI

jQuery UI is a curated set of user interface interactions, effects, widgets, and themes built on top of the jQuery JavaScript Library.

http://jqueryui.com/

For an Animation that uses a button to bring up a dialog box
http://jqueryui.com/dialog/#animated

Dialogs may be animated by specifying an effect for the show and/or hide properties. You must include the individual effects file for any effects you would like to use.

You should include these statements in your HTML coding:

<link rel="stylesheet" href="http://code.jquery.com/ui/1.10.3/themes/smoothness/jquery-ui.css" />

<script src="http://code.jquery.com/jquery-1.9.1.js"></script>

<script src="http://code.jquery.com/ui/1.10.3/jquery-ui.js"></script>

Another useful feature of jQuery UI is having tabs as links to the parts of an assignment.

http://jqueryui.com/tabs/

Here is an example of using jQuery UI tabs.

Where applicable, feel free to use any JQuery UI scripts in the homework assignments.

13 Data type conversions

This chapter was written by Michael Ogi.

When the operands are of different types, the equality (==) operator actually converts its operands before determining whether or not they are equal. Here are the basic rules. (This is from "JavaScript: The Definitive Guide," 5ed, p. 67):

-- If one value is null and the other is undefined, they are equal.

-- If one value is a number and the other is a string, convert the string to a number and try the comparison again. (JavaScript converts "" to a 0, as a mentioned in my post regarding the isNaN() function.)

-- If either value is Boolean true, convert it to 1 and try the comparison again. If either value is Boolean false, convert it to 0 and try to the comparison again.

-- If one value is an object and the other is a number or string, convert the object to a primitive via toString() or valueOf(). Built-in classes use valueOf()before toString(), except for the Date class, which performs the toString() conversion.

-- Any other combinations of values are not equal.

Here are some examples:

Group 1:
null == undefined // true

Group 2 (string converted to number)
0 == "" // true!!
"" is converted to 0. 0 equals 0, so this is true.

"123" == 123 // true
"123" converts to 123. 123 equals 123, so this is true.

"1a" == 1 // false
"1a" converts to NaN. NaN is not equal to 1, so this is false.

Group 3 (string converted to number and Boolean converted to number):
"abc" = true // false
true is converted to 1, and then abc" is converted to NaN. 1 is not equal to NaN, so this is false

"1" == true // true!!
true is converted to 1, and then "1" is converted to 1. 1 is equal to 1, so this is true.

"0" == false // true!!
false is converted to 0, and then "0" is converted to 0. 0 is equal to 0, so this is true.

"true" == true // false!!
"true" is converted to NaN, and true is convertes to 1. NaN is not equal to 1, so this false.

What you've bumped up against is that first example in Group 2. "" == 0 evaluates to true, because "" is converted to 0 before the comparison.

There are a couple of ways that you can code this. One is compare the value of the input field to "" before you convert the field to a number. When you retrieve the value of the text field via

var theInput = document.getElementById("q1").value

and a 0 was entered in the text field, theInput variable will actually contain the string "0". Then the comparison:

theInput == ""

will compare "0" and "". Since both operands are strings, no conversion will occur, and the result will be false, as you'd expect.

The other thing you can do is use the JavaScript identity (===) operator. The identity operator will not perform conversions. If the operands have different types, they are NOT identical:

"" === 0 // false because we are comparing a string with a number

Whichever method you choose, just be mindful of the data types of the values you're comparing, and that JavaScript may perform implicit conversions depending on the operators and functions you use on those values.

14 charAt()

The charAt() method returns the character at the specified index in a string.

The index of the first character is 0, and the index of the last character in a string called "txt", is txt.length-1.

Syntax: stringName.charAt(index)

For example:

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

var str = "San Francisco"; // str is the name of the string

window.alert("First character: " + str.charAt(0)); // alerts S
window.alert("Fourth character: " + str.charAt(3)); // alerts
window.alert("Seventh character: " + str.charAt(6)); // alerts a
window.alert("Last character: " + str.charAt(str.length-1)); // alerts o

</script>
```

Displayed

15 Switch statement

Use the switch statement to select one of many blocks of code to be executed. The switch statement alleviates the use of if else if statements and provides another tool for you to use for controlling the logic of conditional statements.

Syntax:

```
switch
{
case 1:
execute code block 1
break;
case 2:
execute code block 2
break;
case 3:
execute code block 3
break;
default:
code to be executed if n is different from case 1 and 2 and 3
}
```

This is how it works: First we have a single expression n (most often a variable), that is evaluated once. The value of the expression is then compared with the values for each case in the structure. If there is a match, the block of code associated with that case is executed. Use break to prevent the code from running into the next case automatically.

Example:

```
<!DOCTYPE HTML>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Switch Example</title>

<script type="text/javascript">

function process() {

var mnumb;

mnumb = document.myform.month.value;

switch (mnumb)
{
```

```
// Display the month name based on the month number

case "01":
document.myform.mname.value=("You were born in January");
break;
case "02":
document.myform.mname.value=("You were born in February");
break;
case "03":
document.myform.mname.value=("You were born in March");
break;
case "04":
document.myform.mname.value=("You were born in April");
break;
case "05":
document.myform.mname.value=("You were born in May");
break;
case "06":
document.myform.mname.value=("You were born in June");
break;
case "07":
document.myform.mname.value=("You were born in July");
break;
case "08":
document.myform.mname.value=("You were born in August");
break;
case "09":
document.myform.mname.value=("You were born in September");
break;
case "10":
document.myform.mname.value=("You were born in October");
break;
case "11":
document.myform.mname.value=("You were born in November");
break;
case "12":
document.myform.mname.value=("You were born in December");
break;
default:
document.myform.mname.value=("Invalid month number!");
}

}

</script>
</head>

<body>
```

```
<h3>Switch Example</h3>

<form name="myform">
Enter your birth month as a number (2 digits) <input type="text" name="month" size="2"><br>
Result: <input type="text" name="mname" size="25"><br><br>

<input type="button" value="submit" onclick="process();">
<input type="reset" value="reset">
</form>

</body>
</html>
```

Displayed.

Note the equivalent if else if logic for the above switch logic

```
if (mnumb == "01")
document.myform.mname.value=("You were born in January");
else if (mnumb == "02")
document.myform.mname.value=("You were born in February");
else if (mnumb == "03")
document.myform.mname.value=("You were born in March);
else if (mnumb == "04")
document.myform.mname.value=("You were born in April);
else if (mnumb == "05")
document.myform.mname.value=("You were born in May");
else if (mnumb == "06")
document.myform.mname.value=("You were born in June");
else if (mnumb == "07")
document.myform.mname.value=("You were born in July");
else if (mnumb == "08")
document.myform.mname.value=("You were born in August");
else if (mnumb == "09")
document.myform.mname.value=("You were born in September");
else if (mnumb == "10")
document.myform.mname.value=("You were born in October");
else if (mnumb == "11")
document.myform.mname.value=("You were born in November");
else if (mnumb == "12")
document.myform.mname.value=("You were born in December");
else
document.myform.mname.value=("Invalid month number!");
```

16 Web Tutorials and Examples - Functions

functions

- http://www.w3schools.com/js/js_functions.asp
- http://www.webteacher.com/javascript/ch01.html

event handlers

- http://www.w3schools.com/js/js_htmldom_events.asp

onload

- http://www.codetoad.com/javascript/miscellaneous/onload_event.asp
- http://www.w3schools.com/jsref/event_onload.asp

onmouseover

- http://www.javascriptkit.com/javatutors/event4.shtml
- http://www.pageresource.com/jscript/jmouse.htm

onfocus and onblur

- http://www.w3schools.com/jsref/event_onfocus.asp
- http://www.w3schools.com/jsref/event_onblur.asp

switch statement

- http://www.w3schools.com/js/js_switch.asp
- http://developer.mozilla.org/en/docs/Core_JavaScript_1.5_Reference:Statements:switch
- http://www.quackit.com/javascript/tutorial/javascript_switch_statement.cfm
- http://www.javascriptkit.com/javatutors/switch.shtml

break statement and continue statement

- http://www.tutorialspoint.com/javascript/javascript_loop_control.htm
- http://www.w3schools.com/js/js_break.asp

string charAt() and isNaN()

- http://www.w3schools.com/jsref/jsref_charAt.asp
- http://www.w3schools.com/jsref/jsref_isnan.asp

setTimeout

- http://www.pageresource.com/jscript/jtimeout.htm

Math.random

- http://www.the-art-of-web.com/javascript/random/
- http://javascriptkit.com/javatutors/randomnum.shtml
- http://www.pageresource.com/jscript/jrandom.htm

17 Short Exercises - Functions

1. Write function named distance that calculates the distance
between two points (x1, y1) and (x2,y2). All numbers and return values should be floating-point values.Incorporate this function into a script that enables the user to enter the coordinates of the points through an XHTML form.

Ans:

```
1 <!DOCTYPE html>
2 <html lang="en">
3
4
5 <head>
6
7 <meta charset="utf-8">
8
9 <title>Solution 10.9</title>
10
11 <script type = "text/javascript">
12
13 function getDistance()
14 {
15 var x1 = parseInt( document.myForm.x1.value );
16 var y1 = parseInt( document.myForm.y1.value );
17 var x2 = parseInt( document.myForm.x2.value );
18 var y2 = parseInt( document.myForm.y2.value );
19 document.myForm.result.value =
20 distance( x1, y1, x2, y2 );
21 }
22
23 function distance( x1, y1, x2, y2 )
24 {
25 return Math.sqrt( Math.pow( Math.abs( x1 - x2 ),
26 2 ) + Math.pow( Math.abs( y1 - y2 ), 2 ) );
27 }
28
29 </script>
30 </head>
31
32 <body>
33 <form name = "myForm">
34 <table>
35 <tr><td>Enter x1 value</td>
36 <td><input name = "x1" type = "text">
37 </td>
38 </tr>
```

```
39 <tr><td>Enter y1 value</td>
40 <td><input name = "y1" type = "text">
41 </td>
42 </tr>
43 <tr><td>Enter x2 value</td>
44 <td><input name = "x2" type = "text">
45 </td>
46 </tr>
47 <tr><td>Enter y2 value</td>
48 <td><input name = "y2" type = "text">
49 </td>
50 </tr>
51 <tr><td><input type = "button" value= "Calculate"
52 onclick = "getDistance()"></td>
53 </tr>
54 <tr><td>Distance</td>
55 <td><input name = "result" type= "text"></td>
56 </tr>
57 </table>
58 </form>
59 </body>
60 </html>
```

2. Answer each of the following questions:

a) What does it mean to choose numbers “at random”?
Ans: Every number has an equal chance of being chosen at any time.

b) Why is the Math.random function useful for simulating games of chance?
Ans: Math.random produces a series of random numbers, and randomness is useful in making simulations appear realistic.

c) Why is it often necessary to scale and/or shift the values produced by Math.random?
Ans: To produce random numbers in a specific range.

d) Why is computerized simulation of real-world situations a useful technique?

Ans: It enables more accurate predictions of random events such as cars arriving at toll booths and people arriving in lines at a supermarket. The results of a simulation can help determine how many toll booths to have open or how many cashiers to have open at a specified time.

3. Write statements that assign random integers to the variable n in the following ranges:

a) 1 = n = 2
Ans: n = Math.floor(1 + Math.random() * 2);

b) 1 = n = 100
Ans: n = Math.floor(1 + Math.random() * 100);

c) 0 = n = 9
Ans: n = Math.floor(Math.random() * 10);

d) 1000 = n = 1112
Ans: n = Math.floor(1000 + Math.random() * 113);

e) –1 = n = 1
Ans: n = Math.floor(-1 + Math.random() * 3);

f) –3 = n = 11
Ans: n = Math.floor(-3 + Math.random() * 15);

4 For each of the following sets of integers,
write a single statement that will print a number at random
from the set:

a) 2, 4, 6, 8, 10.
Ans: document.write((parseInt(Math.random() * 5) + 1) * 2);

b) 3, 5, 7, 9, 11.
Ans: document.write((parseInt(Math.random() * 5) + 1) * 2 + 1);

5. Write program segments that accomplish each of the
following tasks:

a) Calculate the integer part of the quotient when integer a is
divided by integer b.
b) Calculate the integer remainder when integer a is divided by
integer b.
c) Use the program pieces developed in parts (a) and (b) to
write a function displayDigitsthat receives an integer between
1 and 99999 and prints it as a series of digits, each pair of
which is separated by two spaces. For example, the integer
4562 should be printed as 4 5 6 2.
d) Incorporate the function developed in part (c) into a script
that inputs an integer from a prompt dialog and invokes
displayDigits by passing to the function the integer entered.

Ans:

```
1 <!DOCTYPE html>
2 <html lang="en">
3
```

```
4
5 <head>
6
7 <meta charset="utf-8">
8
9 <title>Solution 10.18</title>
10
11 <script type = "text/javascript">
12
13 var input;
14
15 input = window.prompt(
16 "Enter a number between 1 and 99999:", "0" );
17 document.writeln( "<pre>" );
18 displayDigits( parseInt( input ) );
19 document.writeln( "</pre>" );
20
21 // the following functions execute only when called
22 // part a
23 function quotient( a, b )
24 {
25 return Math.floor( a / b );
26 }
27
28 // part b
29 function remainder( a, b )
30 {
31 return a % b;
32 }
33
34 // part c
35 function displayDigits( number )
36 {
37 var divisor = 10000, digit;
38
39 // determine the divisor
40 while ( number % divisor == number )
41 divisor = quotient( divisor, 10 );
42
43 while ( divisor >= 1 ) {
44 digit = quotient( number, divisor );
45 document.write( digit + " " );
46 number = remainder( number, divisor );
47 divisor = quotient( divisor, 10 );
48 }
49 }
50
51 </script>
52 </head>
```

```
53
54 <body>
55 <p>
56 Click Refresh (or Reload) to run this script again.
57 </p>
58 </body>
59 </html>
```

18 Coding Exercises - Functions

Exercise 1 using a function:

Write a script that contains a function that will calculate the area of a rectangle (area = width * length). Use form input text boxes to allow the user to input the width and length. Use a form button to make the function call that will calculate the area. Allow for the area to be displayed in a form input text box. Check for invalid input and then alert an error message.

Click here for the solution and then view source.

Exercise 1a using a function:

Same as Exercise 1, except I use onkeyup on the last input field (rather than a submit button) and then displaying an error message (using innerHTML) in red and bold-faced and with a large font size when there is an invalid input.

Click here for the solution and then view source.

Exercise 1b using a function:

Same as Exercise 1, except I use jQuery to display an error message when the user either enters non-numeric data, or enters a number not between 0-1000, or does not enter data for either width or height.

Click here for the solution and then view source.

Exercise 2 using a function:

Write a script that contains a function that will calculate a person's Body Mass Index. Use the formula: bmi = (weight * 703) / (height in inches * height in inches). Use form input text boxes to allow the user to input the weight and height. Use a form button to make the function call that will calculate the bmi. Allow for the bmi to be displayed in a form input text box. Check for invalid input and then alert an error message. Use jQuery to show/hide instructions.

Click here for the solution and then view source. Here is another version using jQuery validation.

Exercise 3 using a function and the return keyword:

A Web Development Company has 3 employee positions, namely, Web Trainee, Web Designer, and Web Programmer. All positions are paid by the hour.

Create a webpage using a script containing a function that inputs an employee's hourly rate and returns "Position is Web Trainee" if the employee's rate is $10-19 per hour, "Position is Web Designer" if the employee's rate is $20-49 per hour, and returns "Position is Web Programmer" if the employee's rate is $50-75 per hour. If the entered rate is anything else, then return "Invalid Input".

Use a form input text box for inputting the hourly rate. Access the function by using a form button.

Display the employee position in another form input text box. The script should contain a function that calls another function containing if..else if logic and the return keyword.

Click here for the solution and then view source. I have included comments in the source to explain the code.

Exercise 4 - Switch

Create a webpage that contains a script that inputs a one digit rating of a webpage as 1-excellent, 2-good, 3-fair, 4-poor. Use a form text box for input. Using a switch statement display the corresponding rating message on the same page, including "invalid choice" when the input is not between 1 and 4.

Click here for the solution and then view source.

Exercise 5 - Switch using ranges:

Create a webpage that contains a script that inputs a student's homework average and final exam score (both as integers), using form text boxes. Validate the input and alert an error message if the input is invalid. If the input is valid then calculate the student's final average according to the formula:

final average = .7 * hwAvg + .3 * finalExam

Using a switch statement with ranges of values for final average, display the student's final average (rounded to an integer), the final letter grade and relevant comments into a form textarea, using the following:

90-100 A
80-89 B
70-79 C
60-69 D
0-59 F

Click here for the solution and then view source.

Exercise 6a using a function and Math.random() method:

Create a webpage using a script that simulates coin tossing. Let the program toss the coin each time the user clicks the Toss button. Count the number of times each side of the coin appears. Display the results. The program should call a separate function flip that takes no arguments and returns false for tails and true for heads. [Note: If the program realistically simulates the coin tossing, each side of the coin should appear approximately half the time]. Use jQuery to show/hide information about coin flipping.

Click here for the solution and then view source.

Exercise 6b using a function and Math.random() method:

Same as Exercise 6a, except using penny images and jQuery UI to display instructions about coin flipping. Note that the instruction dialog can be dragged to any point on the page.

Click here for the solution and then view source.

Exercise 7 using a function and Math.random() method:

Create a webpage using a script that simulates tossing dice.

Click here for the solution and then view source. I have included extensive comments in the source to explain the code.

Here is another version that simulates tossing dice and also displays the counts of each roll using an array.

Exercise 8 using a function and Math.random() method:

Create a webpage using a script that asks the viewer to guess a number between 1 and 5. After each guess, confirm whether the viewer wants to continue or not.

Click here for the solution.

Exercise 9 creating a Dynamic Page:

Create a dynamic questionnaire page about wine with dependencies among it's questions.

Click here for the solution.

Note - Page created by Paul Strader (former cnit 133 student)
"All I'm doing is modifying the display property of the div tag using javascript. All the div's are pre written with the information. Depending on which radio button is selected, determines what div will be visible."