

LAB 8

JAVASCRIPT 1: LANGUAGE FOUNDATIONS

WHAT YOU WILL LEARN

- Linking JavaScript into your HTML files
- The basics of JavaScript syntax
- Working with functions and events

APPROXIMATE TIME

The exercises in this lab should take approximately 90 minutes to complete.

FUNDAMENTALS OF WEB DEVELOPMENT, 2ND ED

Randy Connolly and Ricardo Hoar

Textbook by Pearson
<http://www.funwebdev.com>

Date Last Revised: Feb 9, 2017

QUICK TOUR OF JAVASCRIPT

PREPARING DIRECTORIES

- 1 If you haven't done so already, create a folder in your personal drive for all the labs for this book.
- 2 From the main labs folder (either downloaded from the textbook's web site using code provided with book or in a common location provided by your instructor), copy the folder titled lab08 to your course folder created in step one.

Now we are ready to program in Javascript.

Exercise 8.1 — ENABLING/DISABLING JAVASCRIPT

- 1 Before you start to develop with JavaScript, it's important to know how to turn it on and off in your browser. The details of exactly where to enable and disable differ from browser to browser, and change from version to version, so the details are left as an exercise for your particular browser. You might look into settings, options, or preferences.
If you use Chrome or Firefox, you might look into installing an extension that enables you to quickly turn JavaScript on or off.
- 2 Visit <http://examples.funwebdev.com/JavaScriptTest.php> with JavaScript enabled and you will see a message saying "You have JavaScript enabled".
Linux servers are case sensitive so you will likely need to follow exactly the capitalization on this URL.
- 3 Turn JavaScript off and visit the same page. You should now see the message "You have JavaScript disabled".
Now you are able to turn JavaScript on and off as required for testing and development.

The next exercise shows you how to include JavaScript using the embedded technique.

Exercise 8.2 — EMBEDDED JAVASCRIPT

- ~~1 Examine lab08-walkthrough02.html in your editor of choice. Some common editors include emacs, notepad++, brackets, sublime, and eclipse.~~
- ~~2 Preview this file in your browser and it should resemble Figure 8.1.~~

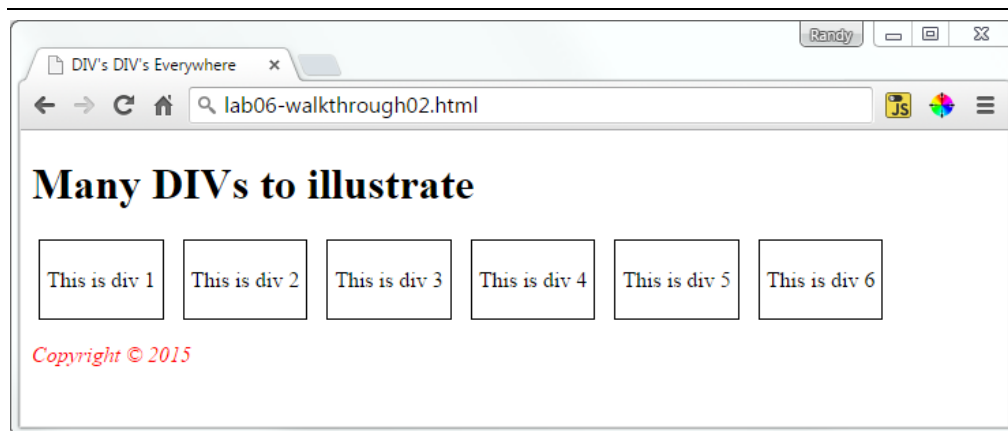


Figure 8.1 – Beginning Exercise 8.2

- 3 ~~Add the following to the lab06-walkthrough02.html file inside of the <head> tags:~~

```
<script type="text/javascript">
  /* Your first script */
  alert("Hello World!");
</script>
```

- 4 ~~Save and test in browser.~~

Notice that a popup is displayed with the text "Hello World!", and requires you to click ok before seeing the web page underneath. Depending on your browser you will see something similar to Figure 8.2.

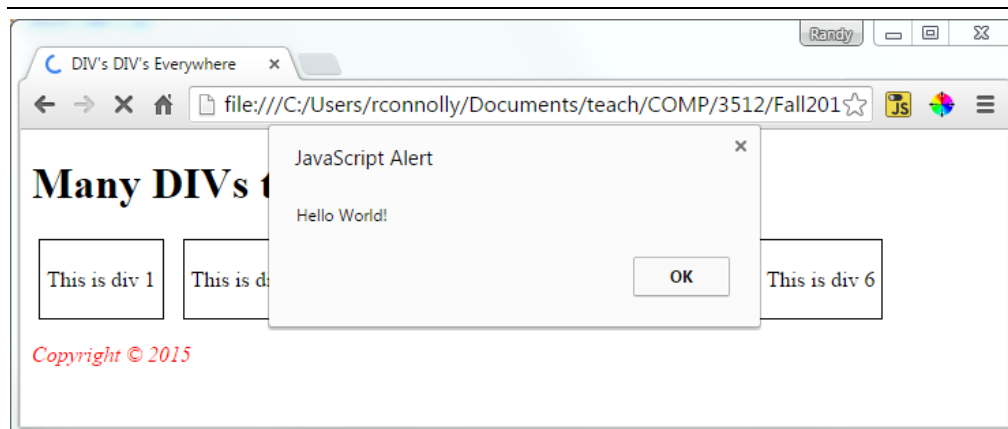


Figure 8.2 – Finished Exercise 8.2

EXERCISE 8.3 — EXTERNAL JAVASCRIPT

- 1 ~~Modify your script from Exercise 8.2 to include an external JavaScript file. Just after the <body> tag, add the following link to the external file:~~

```
</head>
<body>
<script type="text/javascript" language="javascript" src="hello.js">
</script>
<header>
  <h1>Many DIVs to illustrate</h1>
</header>
```

The above lines tells the browser to load a file in the same relative directory, named hello.js. Since the file does not yet exist, if you save and refresh the page nothing will change.

- 2 ~~Create a file named hello.js in the same location as the lab08-walkthrough02.html file. Inside hello.js add the following line of JavaScript and then save the file:~~

```
document.write("this text was written from JS");
var count = 1;
document.write(" Count = " + count);

count++;
var output = "<br>Count = " + count;
document.write(output);
```

When you refresh the browser you should see this text at the top of your page as shown in Figure 8.3.

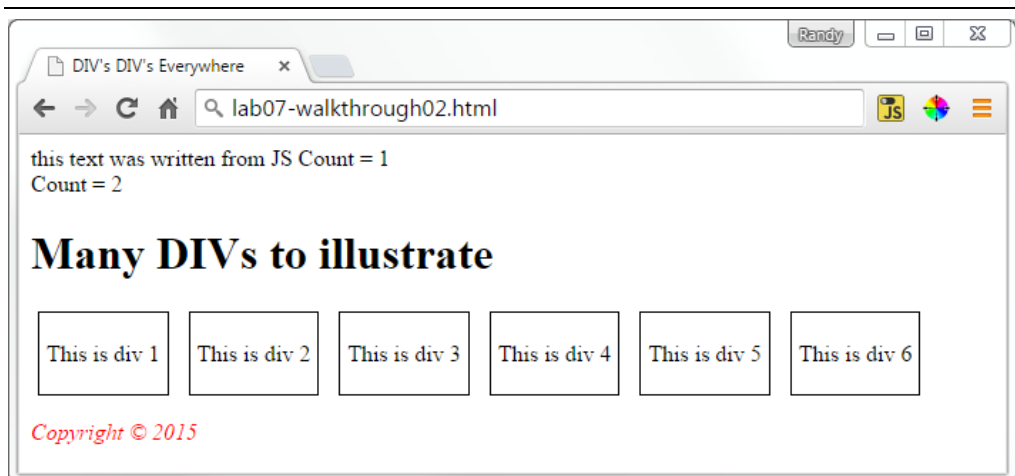


Figure 8.3 – Finished Exercise 8.3

EXERCISE 8.4 — USING NOSCRIPT

- 1 Although you have now used embedded and external JavaScript, we must still account for users without JavaScript enabled. Use the files from the previous two examples as a starting point and add the following markup after the `<script>` include from the previous exercise:

```
<noscript>This text only shown if JS is turned off</noscript>
```

- 2 Save changes and test in browser.

In order to see if your tag is working correctly you must disable JavaScript (see example 1). Once JavaScript is turned off, refresh the page and you should see the noscript message printed at the top of the page.

For the next exercise we will build on the example from Exercise 8.4 so you can continue working in those files. Because we will eventually have multiple JavaScript files to manage we will now organize them into a folder like we did for our image and css files.

Exercise 8.5 — USING THE BROWSER CONSOLE

- ~~1 You have been supplied with a subdirectory named js inside of your working directory. Move hello.js into that new location.~~

- ~~2 Change the reference in the HTML file to reference the new location as follows:~~

```
<script type="text/javascript" language="javascript"
src="js/hello.js"></script>
```

- ~~3 Test in browser.~~

The result should look the same as before, but now we have a better file organization going forward.

- 4 Now we will purposely make a syntactic error in our JS file so we can learn to identify and fix the error.

Modify the code inside the hello.js file so that you misspell document as documment so that our line of code reads.

```
documment.write ("this text was written from JS");
```

Now refresh the page and notice that the text at the top of the page is missing.

However, despite the text missing there is no immediate notification that an error has occurred!

- 5 To get better feedback about programming errors, you will want to use some type of JavaScript debugger/developer tool within your browser.

If you are using FireFox, use the Tools | Web Developer | Web Console menu option. As shown in Figure 8.4, the Console will show the specific error that documment is not defined.

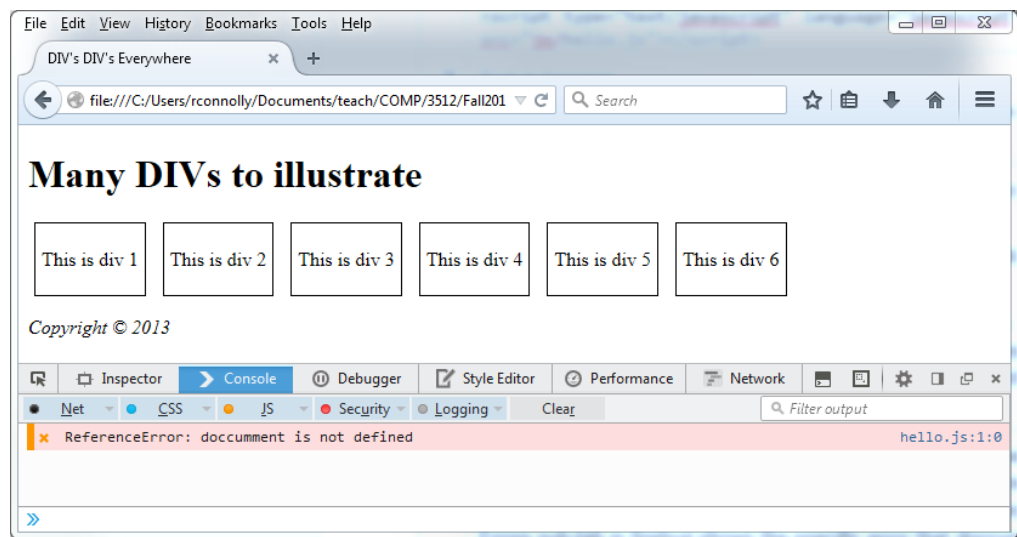


Figure 8.4 – FireFox Web Console


- 6 If you are using Chrome, then select the  button, then choose More Tools | Developer Tools menu. Once the developer tools are visible (either on the left or bottom of page), click the Console tab, and you should see something similar to that shown in Figure 8.5.
- 7 Fix the error by spelling document correctly and see the correct output of your simple script. Test.



Figure 8.5 – Chrome Web Console

- 8 Return to the JavaScript console. You can use the console as well to run and test JavaScript. This can be especially useful way to interrogate the state of JavaScript variables.
- 9 Enter the following into the browser's console: `count`
 After you press `Enter`, the console will display the current value of the `count` variable (which should be 2).

- 10 Enter the following into the browser's console: output
- 11 Enter the following into the browser's console: `var temp = count * 7;`
You can also enter any valid line of JavaScript into the console as well. The console will display the message `undefined`. It displays `undefined` because the console evaluates every expression entered into the console: this assignment does not produce/return a value so it displays `undefined`.
- 12 Enter the following into the browser's console: `temp`
This will display the current value of `temp` (which should be 14).
- 13 Switch to your source code editor, comment out the call to the `alert()` function, and add a call to the `console.log()` function. Test in browser.
- ```
<script type="text/javascript">
 /* Your 1st script */
 //alert("Hello World!");
 console.log("script within the <head> element");
</script>
```
- The `console.log()` function outputs directly to the browser's JavaScript console. This can be a helpful technique for debugging JavaScript.*
- 14 Add the following lines and test.
- ```
<script type="text/javascript">
  /* Your 1st script */
  //alert("Hello World!");
  console.log("script within the <head> element");
  console.warn("oh no this is a warning");
  console.error("panic!! error!!");
</script>
```
- This will display messages in the console similar to those shown in Figure 8.6. You will learn more about debugging JavaScript in a future lab.*

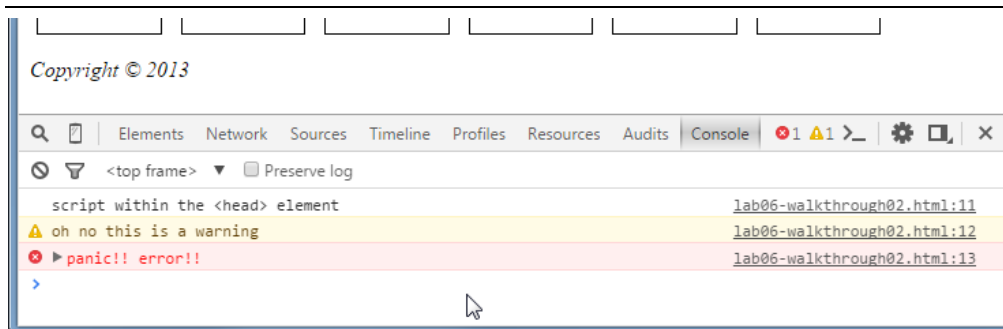


Figure 8.6 –Using the Console object

For the next exercise we will manipulate JavaScript objects including Strings and Dates to illustrate how to construct, modify and output objects in JavaScript.

EXERCISE 8.6 — USING BUILT-IN JAVASCRIPT OBJECTS

- 1 Examine lab08-walkthrough06.html.

The HTML defines a few div's which we will use to demonstrate output from JavaScript Objects.

- 2 Open lab08-walkthrough06.js, add the following lines and test:

```
var stringOne = new String("Test");
document.write(stringOne);
```

This code creates a *String* object, and outputs its value to the HTML page. Note, if your editor is using JSLint (a code analysis tool), you might see the message “Do not use *String* as a constructor”. The reason for this is a bit complicated: the *new* keyword returns a *string* object which in JavaScript is different from a *string* literal.

- 3 Change the code as follows and test:

```
var stringOne = new String("Test");
var stringTwo = "Test";
var stringThree = "Test";
var stringFour = new String("Test");

document.write("<br>typeof stringOne=" + typeof stringOne);
document.write("<br>typeof stringThree=" + typeof stringThree);
```

Why does this matter? The next exercise will illustrate.

- 4 Add the following code after the previous code and test:

```
if (stringOne == stringTwo)
    document.write("<br>stringOne has = value to stringTwo");

if (stringOne == stringFour)
    document.write("<br>stringOne has = value to stringFour");

if (stringOne === stringTwo)
    document.write("<br>stringOne has = value and = type to stringTwo");

if (stringTwo === stringThree)
    document.write("<br>stringTwo has = value and = type to stringThree");

if (stringTwo === stringFour)
    document.write("<br>stringTwo has = value and = type to stringThree");
```

Notice how the *string* object and literals compare differently! As well, notice that you cannot compare for equality between two *string* objects (but you can compare with literals). In general, you will want to create strings as literals.

- 5 Since strings are pretty boring, we will next create a *Date* object in JavaScript. Add the following to your code and test.

```
var dateOne = new Date();
document.write("<p>" + dateOne + "</p>");
```

- 6 To illustrate the *Math* class, add the code lines below and test.

```
document.write(Math.PI+"<br/>");
document.write(Math.sqrt(4)+"<br/>");
document.write(Math.random()+"<br/>");
```

The result will look similar to that shown in Figure 8.7 (of course, the result of *Math.random()* will be different).

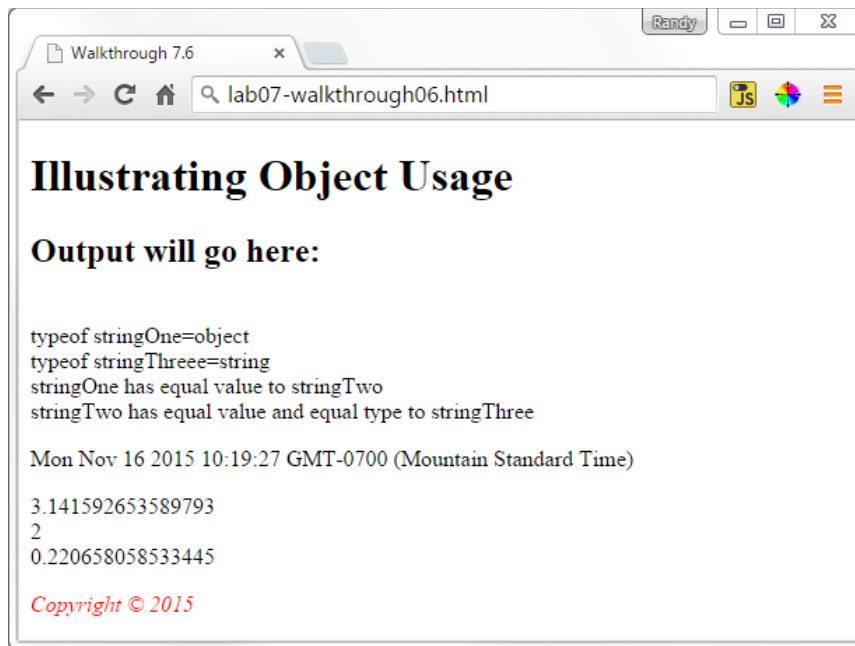


Figure 8.7 – Finished Exercise 8.6

Exercise 8.7 — ARRAYS

- 1 In this exercise you will programmatically add elements to an array, and then print the array to the browser.
- 2 To begin create an array in `/js/lab08-walkthrough07.js`, populate it with the weekdays "Mon" through "Fri" as follows, and then test:


```
var daysOfWeek = new Array("Mon", "Tues", "Wed", "Thur", "Fri");
document.write(daysOfWeek + "<br>");
```
- 3 You may have noticed we are missing the weekend days. To fix this let us first add Saturday to the end of the array using the `push()` method as follows and then test.


```
var daysOfWeek = new Array("Mon", "Tues", "Wed", "Thur", "Fri");
daysOfWeek.push("Sat");
document.write(daysOfWeek + "<br/>");
```
- 4 If we wanted to add Sunday to the end of the list, we could use the `push()` method again. However, we want to add it to the front of the array, so we will use `unshift()` instead.


```
daysOfWeek.unshift("Sun");
document.write(daysOfWeek + "<br/>");
```
- 5 Comment out the `document.write` of the `daysOfWeek` array.
- 6 Although printing each individual element within the element can work, a better approach will be to iterate through the array as follows, then test:

```
document.write("<table border=1>");
document.write("<tr>");
for (var i = 0; i < daysOfWeek.length; i++){
    document.write("<th>" + daysOfWeek[i] + "</th>");
```

```

}
document.write("</tr>");
document.write("</table>");

```

- 7 Modify the loop as follows and then test.

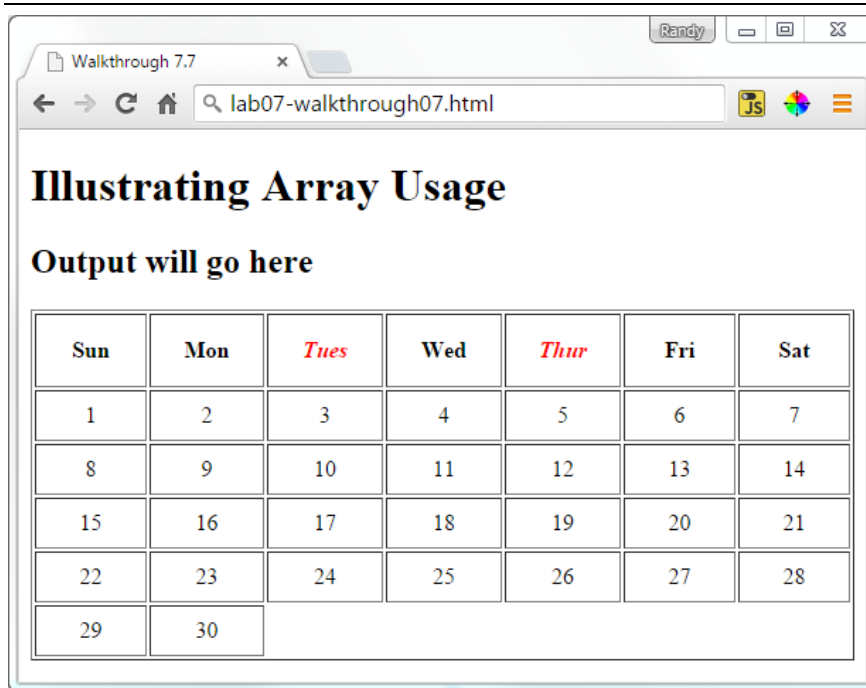
```

for (i = 0; i < daysOfWeek.length; i++) {
    if (daysOfWeek[i].length < 4)
        day = daysOfWeek[i];
    else
        day = "<em>" + daysOfWeek[i] + "</em>";
    document.write("<th>" + day + "</th>");
}

```

This uses a conditional expression to modify the appearance of the weekday labels that are longer than 3 characters (i.e., Tues and Thur).

- 8 Complete the table being created by adding other loops to output the values 1 through 30, assuming that the month starts on Sunday. Your output should look similar to Figure 8.8 below.



Sun	Mon	<i>Tues</i>	Wed	<i>Thur</i>	Fri	Sat
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30					

Figure 8.8 – Output of a table using arrays, conditionals, and loops

FUNCTIONS

Like any other programming language, functions are used to create modular code. Unlike other programming languages, functions in JavaScript are used for many other purposes, including the definition of objects.

Exercise 8.8 — JAVASCRIPT FUNCTIONS

- 1 Open `js/lab08-exercise08.js` and add the following function definition:

```
function outputBox() {  
    document.write("<div class='movingDiv' id='div1'>");  
    document.write("This is div 1");  
    document.write("</div>");  
}
```

- 2 Open `lab08-exercise08.html` and add the following code to the `<script>` element.

```
<script>  
    // add function calls here  
    outputBox();  
</script>
```

- 3 Test in browser.

If your function is correct, you will see a border-lined box.

- 4 Modify the function as follows.

```
function outputBox() {  
    var box = "<div class='movingDiv' id='div1'>";  
    box += "This is div 1";  
    box += "</div>";  
    return box;  
}
```

Instead of having the function perform the output, the function now returns a populated string.

- 5 Modify the `<script>` element as follows and test.

```
<script>  
    // add function calls here  
    document.write(outputBox());  
</script>
```

In terms of the output, nothing should have changed.

- 6 Modify the function as follows.

```
function outputBox(num) {  
    var box = "<div class='movingDiv' id='div" + num + "'>";  
    box += "This is div " + num;  
    box += "</div>";  
    return box;  
}
```

This adds a parameter to the function.

- 7 Modify the `<script>` element as follows and test.

```
<script>  
    // add function calls here  
    for (count=1; count<6; count++) {  
        document.write(outputBox(count));  
    }  
</script>
```

Your result should look similar to that shown in Figure 8.9.

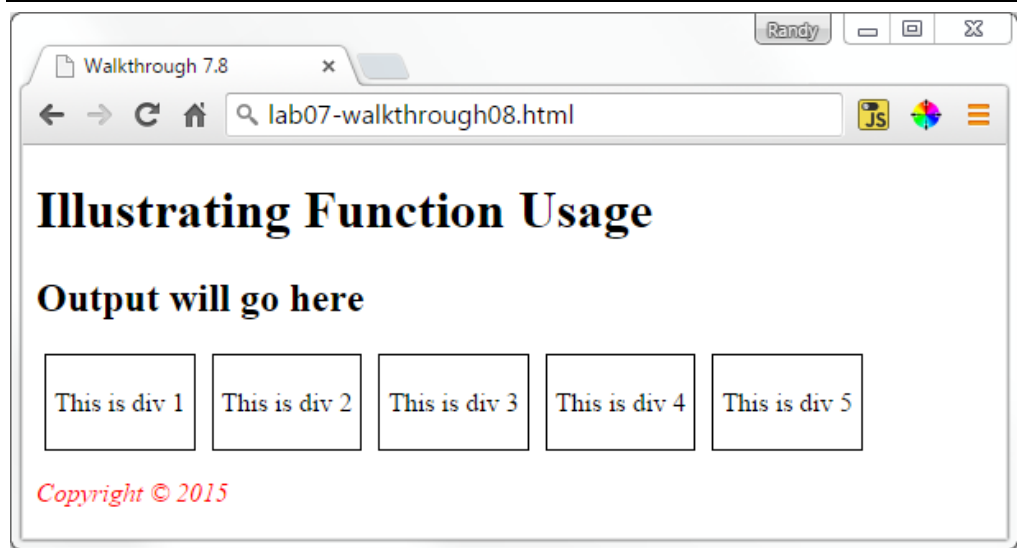


Figure 8.9 – Using JavaScript functions

- 8 Add the following variable definition to `js/lab08-exercise08.js` and modify the function as follows (and test):

```
// this variable has global scope
var boxClass = 'movingDiv';
```

```
function outputBox(num) {
    var box = "<div class='" + boxClass + "' id='div" + num + "'>";
```

In general, in JavaScript (as in other programming languages) you should try to minimize your usage of global variables. We will at times use global variables in this (and future) lab exercises for simplicity sake.

- 9 Add the following line after the for loop and test. Examine the browser error console (see Exercise 8.5).

```
console.log(box);
```

Since the variable box was defined using the var statement within a function, its scope is limited to that function.

- 10 Modify the `outputBox()` function and remove the `var` from the `box` variable definition and test (be sure to examine the console).

Notice that defining a variable without a var gives that variable global scope. Mistakenly introducing (or overriding) global variables in this way is a common source of bugs. Thus you should always explicitly declare variables with the var keyword.

Unlike most other programming languages, **functions in JavaScript are also objects**. This means that you can assign a function definition to a variable and then manipulate that variable.

Exercise 8.9 — FUNCTIONS AS OBJECTS

- 1 Open `js/lab08-exercise09.js` and add the following.

```
var isCanadian = true;

function taxRate() {
  // remember : variables defined outside of a function have global scope
  if (isCanadian) {
    return 0.05;
  } else {
    return 0.0;
  }
}

function calculateTax(amount) {
  return amount * taxRate();
}

function calculateTotal(price, quantity) {
  return (price * quantity) + calculateTax(price * quantity);
}
```

We are going to use and manipulate these JavaScript functions in this exercise.

- 2 Add the following to the `<script>` element in `lab08-exercise09.html` and test.

```
<td>
  <script>
    var amount = calculateTotal(15,2);
    document.write("$" + amount.toFixed(2));
  </script>
</td>
```

The `toFixed` method returns the amount as a string formatted with two decimal places.

- 3 Let's assume the `taxRate()` function is only ever used by the `calculateFunction()`. In order to reduce the possibility of function name conflicts in the future, we can nest one function within the other. Try this by moving your `taxRate()` function within `calculateTax()` as follows. Test (everything should work as before).

```
var isCanadian = true;

function calculateTax(amount) {
  return amount * taxRate();

  function taxRate() {
    if (isCanadian) {
      return 0.05;
    } else {
      return 0.0;
    }
  }
}
```

- 4 Instead of defining a nested named function we could instead define the function as an object. You can try this by changing your code as follows and test:

```
function calculateTax(amount) {

  // define a function as an object
```

```

    var tax = function taxRate() {
        if (isCanadian) {
            return 0.05;
        } else {
            return 0.0;
        }
    };

    // now invoke the function using the object
    return amount * tax();
}

```

- 5 Instead of defining a named function we could instead make the function definition anonymous. You can try this by removing the function name as follows and test:

```

// define an anonymous function as an object
var tax = function () {

```

- 6 Because functions can be objects, we can define functions anywhere we could use a normal variable. For example, change your code as follows and test (the output will continue to look the same as before).

```

function calculateTax(amount, tax) {
    return amount * tax();
}

function calculateTotal(price, quantity) {
    var amount = price * quantity;
    return amount + calculateTax(amount, function () {
        if (isCanadian) {
            return 0.05;
        } else {
            return 0.0;
        }
    });
}

```

Notice that here we are passing a function as a parameter; the passed function object is anonymous. This fact that functions can be passed as objects will likely seem confusing at first. It is an essential technique in almost all real-world JavaScript.

OBJECTS

Objects are essential in JavaScript because almost everything in JavaScript is an object.

Exercise 8.10 — CREATING JAVASCRIPT OBJECTS

- 1 Open `js/lab08-exercise10.js` and add the following code.

```

var order = new Object();

order.product = "Self Portrait in a Straw Hat";
order.price = 15.0;
order.quantity = 2;
order.total = function() { return this.price * this.quantity; };

document.write("Product=" + order.product);

```

```
document.write("<br>Price=" + order.price);
document.write("<br>Quantity=" + order.quantity);
document.write("<br>Total=" + order.total());
```

- 2 Test by opening lab08-exercise10.html in the browser.
- 3 Change the last line from step 1 to the following (i.e., remove brackets after the function name) and then test.

```
document.write("<br>Total=" + order.total);
```

Notice that without the trailing brackets, JavaScript returns the content of the property rather than executing the function!

- 4 Restore the brackets to the function call and then change the first line to the following:

```
var order = {};
```

This is an alternate way of defining an empty object.

- 5 Comment out your previous object property definitions and recreate them using the following:

```
order["product"] = "Self Portrait in a Straw Hat";
order["price"] = 15.0;
order["quantity"] = 2;
order["total"] = function() { return this.price * this.quantity; };
```

- 6 Comment out your previous object property definitions and recreate them using the following:

```
var order = {
  product: "Self Portrait in a Straw Hat",
  price: 15.0,
  quantity: 2,
  total: function() { return this.price * this.quantity; }
};
```

This approach is often referred to as object literal notation.

- 7 You can also create objects using constructor functions. This allows you to create multiple instances of the same object type. Comment out your previous code and add the following:

```
function order(product, price, quantity) {
  this.product = product;
  this.price = price;
  this.quantity = quantity;
  this.total = function() { return this.price * this.quantity; }
}
```

- 8 Now create two instances using this constructor:

```
var example1 = new order("Self Portrait in a Straw Hat", 15, 2);
var example2 = new order("Untitled #23", 10, 4);
```

- 9 And test these variable using the following (and then test in browser):

```
document.write("<p>Product=" + example1.product);
document.write("<br>Price=" + example1.price);
document.write("<br>Quantity=" + example1.quantity);
document.write("<br>Total=" + example1.total());

document.write("<p>Product=" + example2.product);
```

```
document.write("<br>Price=" + example2.price);  
document.write("<br>Quantity=" + example2.quantity);  
document.write("<br>Total=" + example2.total());
```

- 10 Finally, let's improve our code by commenting out the code from step 9 and then modifying the constructor function as follows:

```
function order(product, price, quantity) {  
    this.product = product;  
    this.price = price;  
    this.quantity = quantity;  
    this.total = function() { return this.price * this.quantity; },  
  
    this.output = function() {  
        document.write("<p>Product=" + this.product);  
        document.write("<br>Price=" + this.price);  
        document.write("<br>Quantity=" + this.quantity);  
        document.write("<br>Total=" + this.total());  
    }  
}
```

Don't forget to add the comma after the total() function definition.

Note: in a later lab, we will instead use JavaScript prototypes as a more efficient way to add methods/functions to an object.

- 11 Test the function by adding the following code (and then test in browser):

```
var example1 = new order("Self Portrait in a Straw Hat", 15, 2);  
var example2 = new order("Untitled #23", 10, 4);
```

```
example1.output();  
example2.output();
```

The result should look similar to that shown in Figure 8.10



Figure 8.10 – Result of Exercise 8.10.

Exercise 8.11 — ARRAYS OF OBJECTS

- 1 Open and examine `js/lab08-exercise11.js` to see some approaches for creating an array of objects. Add the following code to iterate through the array and test.

```
for (i = 0; i < countries.length; i++) {  
    document.write(countries[i].name);  
    document.write("<br>");  
}
```

- 2 Try instead manipulating a temporary object variable as shown in the following:

```
for (i = 0; i < countries.length; i++) {  
    var c = countries[i];  
    document.write(c.name);  
    document.write("<br>");  
}
```

This creates the same output but (perhaps) makes for more understandable code.

- 3 Recall that you can also access object properties through bracket notations, as shown in the following:

```
for (i = 0; i < countries.length; i++) {  
    var c = countries[i];  
    document.write(c["name"]);  
    document.write("<br>");  
}
```

- 4 Change the loop as follows:

```
var temp = "name";  
document.write(c[temp]);
```

One advantage of bracket notation is that we can use the content of variables to access an object property.

- 5 Using bracket notation and the `for...in` loop we can iterate through the properties of an object. To see this, change your loop as follows and test.

```
for (i = 0; i < countries.length; i++) {  
    var c = countries[i];  
    document.write("<div class='box'>");  
    document.write("<img src='flags/' + c.iso + '.png' class='boxImg'>");  
    for (var propertyName in c) {  
        document.write("<strong>");  
        document.write(propertyName + ": ");  
        document.write("</strong>");  
        document.write(c[propertyName]);  
        document.write("<br>");  
    }  
    document.write("</div>");  
}
```

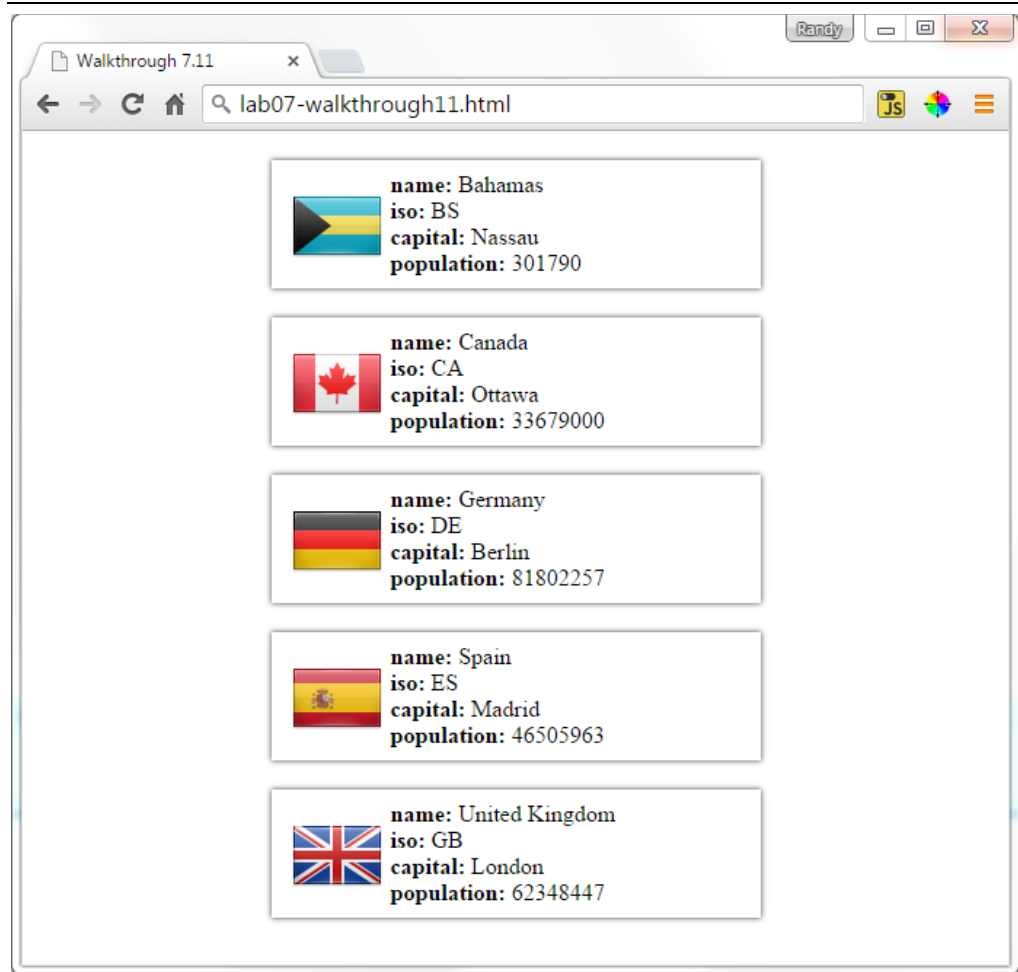


Figure 8.11 – Result of Exercise 8.11.