

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, 3rd Ed

Randy Connolly and Ricardo Hoar

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

Date Last Revised: Sept 12, 2018

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 Copy the folder titled lab08 to your course folder created in step one. This lab08 folder could be provided by your instructor, or you could clone it from GitHub repo for this lab.

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-exh02.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.

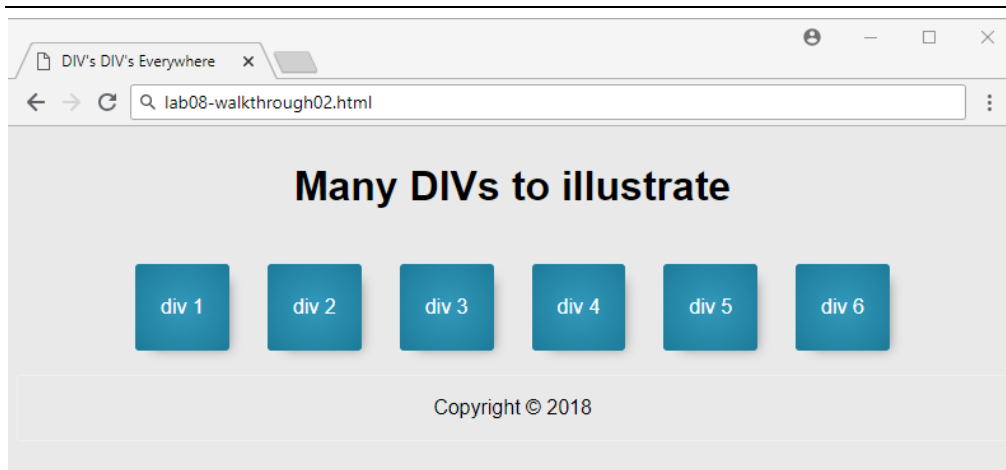


Figure 8.1 – Beginning Exercise 8.2

- 3 Add the following to the lab08-ex02.html file inside of the <head> tags:

```
<script>
  /* 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 and/or version, you will see something similar to Figure 8.2.

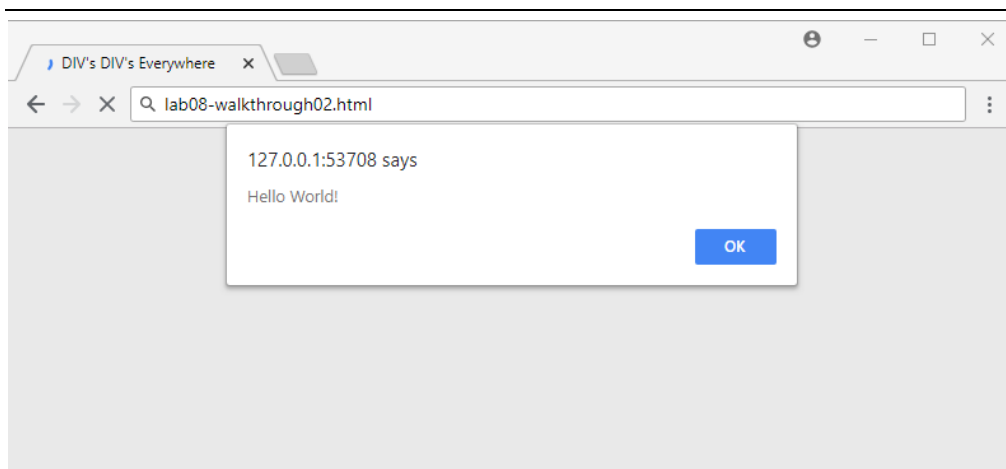


Figure 8.2 – Finished Exercise 8.2

EXERCISE 8.3 — EXTERNAL JAVASCRIPT

- 1 Create a file named `hello.js` in the same location as the `lab08-ex02.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.

Depending on your editor and whether it is using a linter such as JSLint or ESLint, it is possible you may see warnings or error messages here about `document` not being defined. You can change that behavior by configuring the linter to use the browser as the environment; alternately, simply ignore those messages.

- 2 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 src="hello.js"></script>
<header class=centered>
  <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.

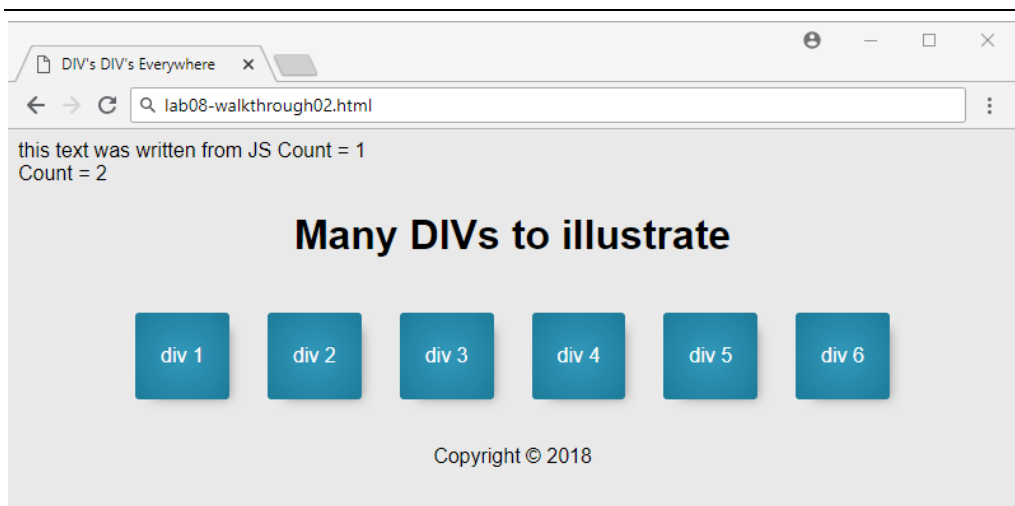


Figure 8.3 – Finished Exercise 8.3

EXERCISE 8.4 — USING NOSCRIPT

- 1 Although you have now used embedded and external JavaScript, you still may want to 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 page requires JavaScript be enabled</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 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 `doccument` so that our line of code reads.

```
doccument.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 Chrome, then select the ellipse button  button, then choose More Tools | Developer Tools menu. Once the developer tools are visible (either on the left or bottom of page or in a floating window), click the Console tab. As shown in Figure 8.4, the Console will show the specific error that `doccument` is not defined..

If you are using FireFox, use the Tools | Web Developer | Web Console menu option.

If you are using Edge, then select the ellipse button, and choose Developer Tools.

If you are using Safari, you will have to enable the Developer tools first within Preferences | Advanced and turn on the Show Develop menu check box. Once you do so, you can use the Develop | Show JavaScript Console menu option.

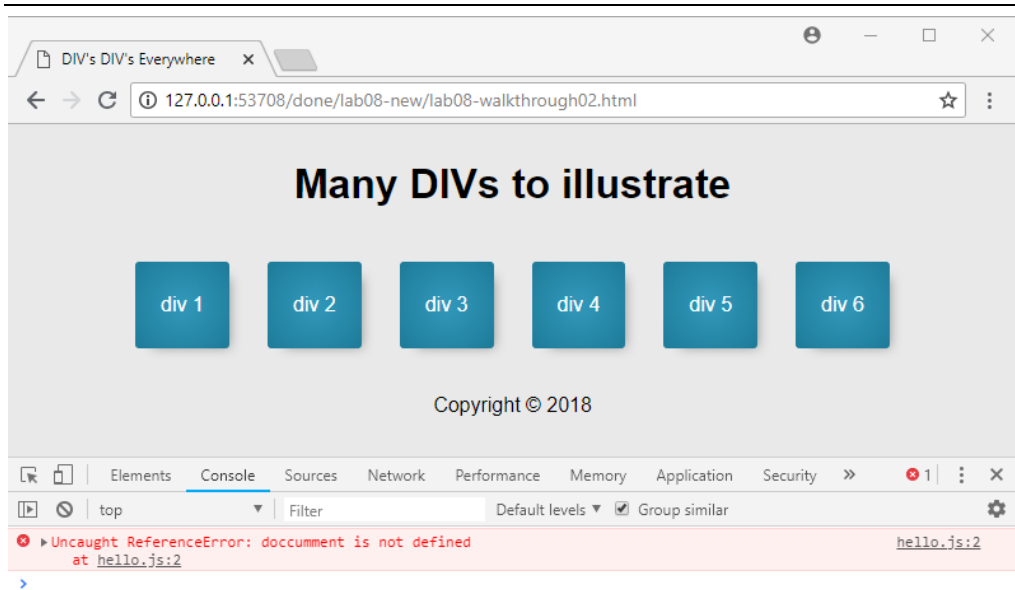


Figure 8.4 – Chrome Console

- 7 Fix the error by spelling document correctly and then refreshing the browser.
- 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 as shown below. 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. Several examples in this and future labs will make use of this (and the other) console functions.

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.

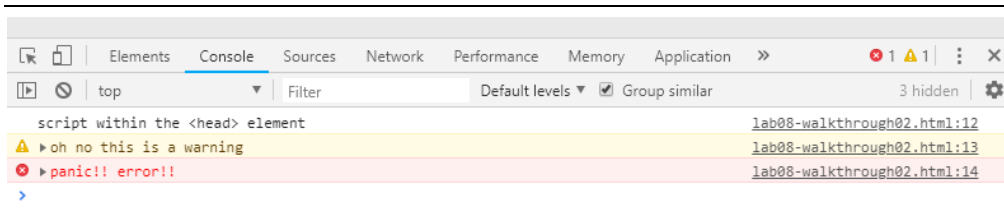


Figure 8.5 –Using the console object

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 VARIABLES

- 1 Examine lab08-ex06.html and then open lab08-ex06.js in your editor.

In this exercise, we will be using the console for output.

- 2 Enter the following code in lab08-ex06.js and **test** (that is, save the file, then view lab08-ex06.html in a browser, and then view the JavaScript console).

```
var a1 = 23;
console.log(a1);
```

```
const a2 = 24;
console.log(a2);
```

There are several ways to declare variables in JavaScript. With `const`, variables are immutable (their contents can't change).

- 3 Modify the code by adding the following and test.

```
var a1 = 23;
a1 = 50;
console.log(a1);
```

```
const a2 = 24;
a2 = 51;
console.log(a2);
```

Trying to change `a2` should generate an exception: as a `const`, variables can't be changed.

- 4 Modify the code by adding the following (note removed attempt to modify `a2`) and test.

```
const a2 = 24;
// a2 = 51;
console.log(a2);

var isDifficult = true;
var isNull = null;
var whatIsThis;

console.log(isDifficult);
console.log(isNull);
console.log(whatIsThis);
```

```
console.log(xyz);
```

Notice that a defined variable with no content is equal to “undefined”, which is different from the null value. In JavaScript, null is a special value used to indicate a variable with no value; undefined is used to indicate a declared variable hasn't had a value assigned. Both null and undefined are different cases that using variable that hasn't been defined, which, this example illustrates, generates a run-time error.

- 5 Modify the code by adding the following and test.

```
//console.log(xyz);
```

```
var foo = "Randy";
console.log(foo);
foo = 99;
console.log(foo);
foo = 'Sue';
console.log(foo);
```

Notice that variables can contain different data types over time. Notice also that string literals can use single or double quotes.

- 6 Modify the code by adding the following and test.

```
foo = "10";
console.log(typeof foo);
console.log(typeof isDifficult);
console.log(typeof whatIsThis);
```

JavaScript does have data types! Type is determined by the type of data it contains.

- 7 Modify the code and test. Can you guess the data type of canYouGuess?

```
foo = "10";
var canYouGuess = a1 + foo;
console.log(typeof foo);
console.log(typeof canYouGuess);
console.log(canYouGuess);
```

In JavaScript the + operator means either addition (if data type is number) or concatenation (if one of the data types is string). JavaScript has performed type coercion on a1 and turned it into a string.

- 8 Modify the code (notice you are removing the quotes) and test. Can you guess the data type of canYouGuess?

```
foo = 10;
var canYouGuess = a1 + foo;
console.log(typeof foo);
console.log(typeof canYouGuess);
```

Since the type of foo is numeric the + operator here acted as addition rather than concatenation.

- 9 Add the following code and test.

```
var age = prompt("What is your age?");
console.log('age=' + age + ' data type=' + typeof age);
```

This example illustrates the simple prompt() function for retrieving a value from the user. Notice that no matter what you type in, it assumes it is a string.

- 10 Modify the code as follows and test entering valid and not valid numbers.

```
var age = prompt("What is your age?");
var iAge = Number(age);
console.log('age=' + iAge + ' data type=' + typeof iAge);
```

Now iAge either contains a number or the special value Nan (not-a-number).

TEST YOUR KNOWLEDGE #1

Examine lab08-test01.html and then open lab08-test01.js in your editor. Modify the JavaScript file to implement the following functionality.

- 1 Provide a prompt to the user to enter a bill total.
- 2 Convert this user input to a (don't worry about error handling for non-numbers).
- 3 Calculate the tip amount assuming 10% (simply multiply the user input by 0.1). Use a const to define the 10% tip percentage.
- 4 Display the bill total and tip amount on the same console output line, e.g.

For bill of \$20 the tip should be \$2

EXERCISE 8.7 — CONCATENATION

- 1 Examine lab08-ex07.html and then open lab08-ex07.js in your editor.

In this exercise, we will be using the console for output.

- 2 Enter the following code in lab08-ex07.js and test.

```
var country = "France";
var capital = "Paris";
var population = 67;
```

```
var msg="The population of " + country + " is " + population + " million";
console.log(msg);
```

- 3 Add the following and test.

```
msg = capital + " is the capital of " + country ;
console.log(msg);
msg += " and has " + population + "mil people";
console.log(msg);
```

Notice the second example uses the += operator to add a string to an existing string.

- 4 Add the following and test.

```
var msg2 = `The population of ${country} is ${population} million`;
console.log(msg2);
```

*This example uses **template literals**, a new feature in ES6. The literal character here is the back-tick (the key to the left of the 1 key on most keyboards) not a single quote. With template literals you can include variable references within the literal, thereby avoiding the concatenation operators. As you can see, these two approaches produce the same output; some programmers prefer concatenation, others prefer template literals.*

EXERCISE 8.8 — USING THE CONSOLE AND NATIVES

- 1 In this example, we won't bother with a pre-defined file. Instead you will simply code directly in the console. In your browser, display the JavaScript Console.

- 2 Enter the following code into the console.

```
var s1 = "hello"
```

When you press Enter in the console, you will likely see the message "undefined". The console always returns the result of your expression. In this case, the line doesn't evaluate to a string or number or Boolean, which is why it displays undefined. But don't worry, it's still defined and initialized as a variable.

- 3 Enter the following into the console.

```
s1  
typeof s1
```

This will display the value and data type of s1.

- 4 Enter the following into the console.

```
var s2 = new String("hello")  
s2  
typeof s2
```

While s2 looks like s1, internally it is quite different: it is a JavaScript object.

- 5 Enter the following into the console.

```
var d = new Date()  
d  
d.getFullYear()
```

Most objects have a variety of properties and methods you can invoke.

- 6 Enter the following into the console.

```
s2.toUpperCase()  
s2
```

Notice how toUpperCase() didn't change the contents of s2.

- 7 Enter the following into the console.

```
s1.toUpperCase()  
s1
```

You should be a bit puzzled here. The variable s1 is a string primitive, while s2 is a string object. Yet both have properties and methods. What is happening is JavaScript is automatically "boxing" the primitive variable into a temporary object so you can make use of the native object's properties and methods.

- 8 Enter the following into the console.

```
var n = 34  
typeof n  
n.toFixed(2)
```

As you can see, number primitives will also be implicitly boxed into an object to give you access to the Number object properties and methods.

EXERCISE 8.9 — USING CONDITIONALS

- 1 Examine lab08-ex09.html and then open lab08-ex09.js in your editor.

In this exercise, we will be using the console for output.

- 2 Enter the following code in lab08-ex09.js and test.

```
const age = 65;
const age2 = "65";

if (age == 65)
  console.log("you are eligible for retirement");
else
  console.log("you are too young for retirement");
```

- 3 Change the conditional by adding {} braces:

```
if (age == 65) {
  console.log("you are eligible for retirement");
} else {
  console.log("you are too young for retirement");
}
```

- 4 Add the following and test.

```
if (age == age2) console.log("these are the same");
```

This surprisingly displays the “these are the same” message. Why? Since two different data types are being compared, one of the variables is being coerced into a different type to match the comparison variable.

- 5 Add the following and test.

```
if (age === age2) console.log("these shouldn't be triple equal");
```

The strict equality operator (===) compares without type coercion; if the types of the two sides of the operator are different, then they are considered unequal.

- 6 Add the following and test.

```
var userAge = prompt("What is your age?");
var status = (userAge >= 65) ? 'You can be retired' : 'You keep working';
console.log(status);
```

*This illustrates a **ternary operator**, which provides a more concise syntax than an if...else block for expressing a conditional that executes a single line of code based on a condition.*

JavaScript conditionals evaluate to true or to false. Most conditional expressions have clear truthness or falseness, e.g., `age > 30`). However, JavaScript programmers sometimes skip part of the comparison and rely on whether a single value is **truthy** or **falsey**. The falsy values in JavaScript are `undefined`, `null`, `0`, `""`, and `NaN`.

EXERCISE 8.10 — TRUTHY AND FALSY

- 1 Examine lab08-ex10.html and then open lab08-ex10.js in your editor.

In this exercise, we will be using the console for output.

- 2 Enter the following code in lab08-ex10.js and test.

```
var age;  
if (age)  
  console.log('this is truthy');  
else  
  console.log('this is falsy');
```

Since age has a value of undefined, it evaluates to false.

- 3 Change the code as follows and test.

```
var age = 23;  
if (age)  
  console.log('this is truthy');  
else  
  console.log('this is falsy');
```

Since age has a non-zero, integer value, it evaluates to true.

- 4 Change the code as follows and test.

```
var age = 'hello';
```

- 5 Change the code as follows and test.

```
var age = '';
```

TEST YOUR KNOWLEDGE #2

Modify your results from previous Test Your Knowledge and implement the following functionality.

- 1 Display an error message to the console if the user input is not a valid number.

Exercise 8.11 — ARRAYS AND ITERATION

- 1 Examine lab08-ex11.html and then open lab08-ex11.js in your editor.
In this exercise you will programmatically add elements to an array, and then print the array to the browser.

- 2 Enter the following code in lab08-ex11.js and test.

```
var days = new Array("Mon", "Tues", "Wed", "Thur", "Fri");
var months = ['jan', 'feb', 'mar'];
var years = [];
years[0]=1999;
years[1]=2000;
```

```
console.log(days);
console.log(months);
console.log(years);
```

This illustrates three different ways of creating and populating arrays in JavaScript.

- 3 Arrays are actually objects in JavaScript, meaning they have many properties and methods that you can invoke. For instance, add the following and test.

```
console.log(months.length);
days.push("Sat");
days.unshift("Sun");
console.log(days);
months.pop();
console.log(months);
```

The push() method adds a new element to the end of the array, unshift() adds an element to the front of an array, while pop() removes the last element

- 6 To iterate through the contents of an array, the classic approach is to use the for loop. Add the following loop and test.

```
for (let i=0; i < days.length; i++) {
  console.log('index=' + i + ' value=' + days[i])
}
```

Notice the Let: you could have used var as well. The advantage of Let is that the variable becomes block scoped, meaning it's only available within the loop block.

- 7 More recent versions of JavaScript provide an alternative syntax for looping through an array called the **for-of loop**. Add this loop and test.

```
for (let mon of months) {
  console.log(mon);
}
```

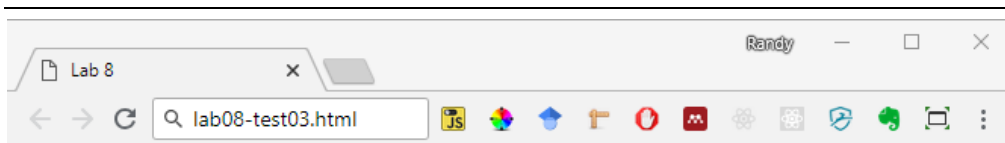
The traditional for equivalent of this for-of loop would be:

```
for (let i=0; i < months.length; i++) {
  let mon = months[i];
  console.log(mon);
}
```

TEST YOUR KNOWLEDGE #3

Modify your results from previous Test Your Knowledge (or create a copy of previous version) and implement the following functionality.

- 1 Comment out code retrieving and validating bill total from user (we are going to replace user input with data from array)
- 2 Define an array called `billTotals` that contains an array of numeric values, e.g. values of 50, 150, 20, 500, etc.
- 3 Define a new empty array called `tips`.
- 4 Loop through this array and first determine the tip percentage for each number in the `billTotals` array using this logic: if total > 75 then tip% = 10%, if total between 30 and 75 then tip% = 20%, else if total < 30 then tip% = 30%
- 5 Calculate tip by multiplying individual `billAmount` element by the appropriate tip percentage.
- 6 Add (push) this tip to the `tips` array.
- 7 Once all the tips are calculated, then output to the console each bill total and tip amount on a separate console line using the same format as you used in Test Your Knowledge #1 (see Figure 8.6). This will require another loop (a `for` loop) which will iterate the `billTotals` array but reference both `billTotals` and `tips` arrays.



Test Your Knowledge #3

Output will be in the console

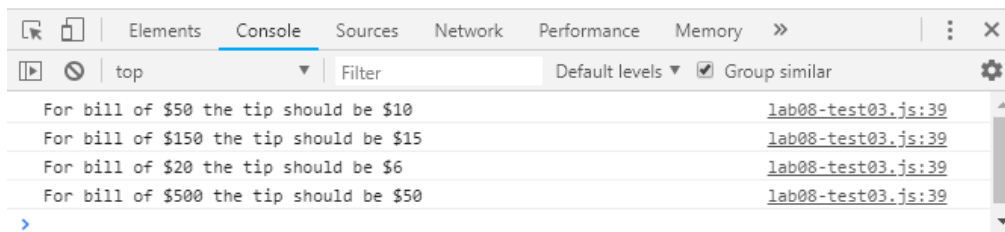


Figure 8.6 – Test your Knowledge #3

FUNCTIONS

Exercise 8.12 — JAVASCRIPT FUNCTIONS

- 1 Examine lab08-ex12.html and then open lab08-ex12.js in your editor and add the following function definition to lab08-ex12.js.

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

- 2 In lab08-ex12.html, 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 in 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 8.7.

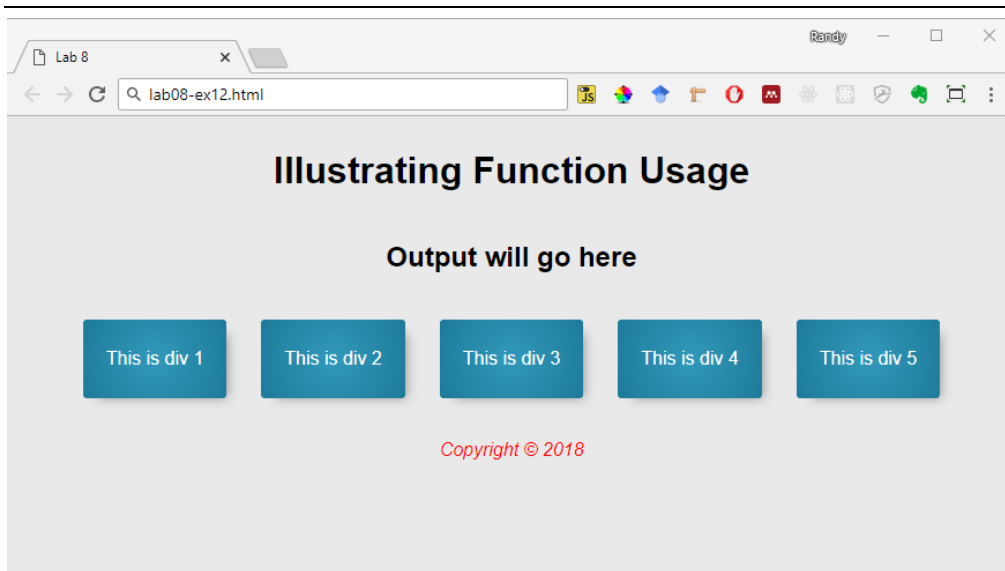


Figure 8.7 – Using JavaScript functions

TEST YOUR KNOWLEDGE #4

Modify your results from previous Test Your Knowledge (or create a copy of previous version) and implement the following functionality.

- 1 Define a function named `calculateTip` that takes a single parameter named `total` which contains the individual bill total for which the tip is going to be calculated.
- 2 In the function, calculate the tip using the same logic as the previous Test Your Knowledge. Your function should return the tip.
- 3 Change your previous code so that your loop uses this new function to calculate the tip for each number in the array.

Exercise 8.13 — SCOPE

- 1 Examine `lab08-ex13.html` and then open `lab08-ex13.js` in your editor and add the following code to `lab08-ex13.js` and test.

```
function calculateTotal(price, quantity) {  
    var amount = price * quantity;  
    return amount;  
}
```

```
var t1 = calculateTotal(15,2);  
console.log(t1);
```

This defines and invokes a simple function. Notice the local variable `amount` within the function. This has function scope, meaning it is only available within this function.

- 2 Modify the code as follows and test.

```
console.log(t1);
console.log(amount);
```

This will generate an exception: amount is not available outside the function.

- 3 Modify the code as follows and test.

```
var amount = 0;

function calculateTotal(price, quantity) {
  amount = price * quantity;
  return amount;
}
```

```
var t1 = calculateTotal(15,2);
console.log(t1);
console.log(amount);
```

Notice that you have removed the var from the amount calculation, meaning we are using the variable defined outside the function.

- 4 Modify the code as follows and test.

```
function calculateTotal(price, quantity) {
  var amount = price * quantity;
  return amount;
}
```

There is now two amount variables: one defined within the function and one outside it (i.e., one with global scope). Notice that the console.log references the one in global scope.

- 5 Modify the code as follows and test.

```
//var amount = 0;
var tax = 0.20;

function calculateTotal(price, quantity) {
  var amount = price * quantity;
  amount = amount - (amount * tax);
  return amount;
}
```

```
var t1 = calculateTotal(15,2);
console.log(t1);
console.log(tax);
```

Scope rules allow functions to access variables declared in their parent's (or ancestor's) scope. Ideally, your functions should be self-contained and not have dependencies outside its parameters; nonetheless, you will often see JavaScript code that references variables in its ancestors so you should be familiar with how scope works in JavaScript.

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 8.14 — FUNCTIONS AS OBJECTS

- 1 Examine lab08-ex14.html and then open lab08-ex14.js in your editor and add the following code.

```
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-ex14.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

You have just seen that functions are actually objects in JavaScript. Objects are essential in JavaScript because almost everything else in JavaScript is an object.

Exercise 8.15 — CREATING JAVASCRIPT OBJECTS

- 1 Examine lab08-ex15.html and then open lab08-ex15.js in your editor. Add the following code and then test.

```
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 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 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

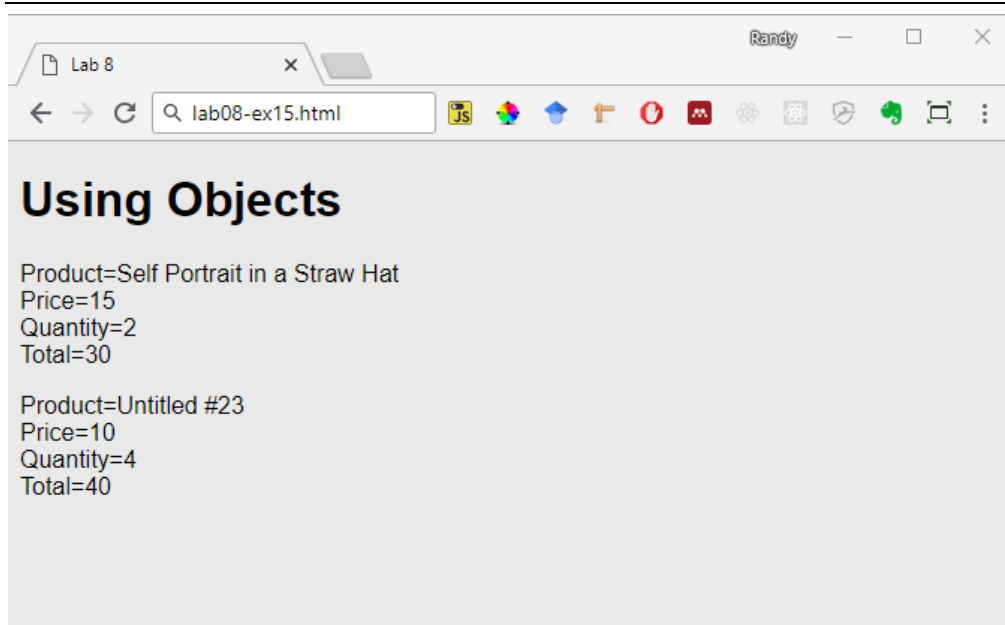


Figure 8.8 – Result of Exercise 8.15.

Exercise 8.16 — ARRAYS OF OBJECTS

- 1 Examine lab08-ex16.html and then open lab08-ex16.js in your editor. It contains some approaches for creating an array of objects. Add the following code and then test.

```
for (let c of countries) {  
    document.write(c.name);  
    document.write("<br>");  
}
```

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

```
for (let c of countries) {  
    document.write(c["name"]);  
    document.write("<br>");  
}
```

- 3 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.

- 4 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 (let c of countries) {
  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>");
}
```

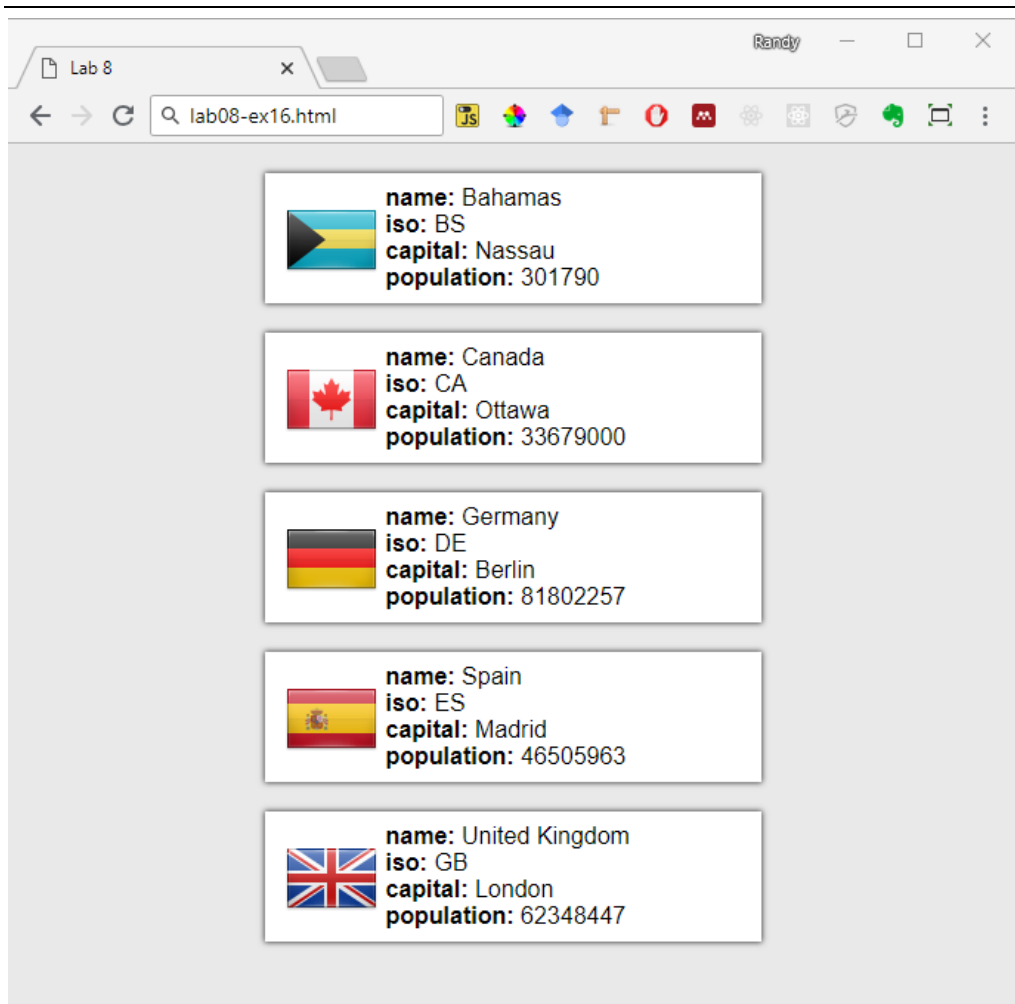


Figure 8.9 – Result of Exercise 8.16.

TEST YOUR KNOWLEDGE #5

Modify your results from previous Test Your Knowledge (or create a copy of previous version) and implement the following functionality.

- 1 Define a new object named `Tipper`. This object must be defined using a function constructor notation. It is going to contain the data arrays, calculation functions, and output code from previous Test Your Knowledge together into a single object.
- 2 Move the `billTotals` and `tips` arrays into `Tipper` as properties (remember to use this keyword).
- 3 Move the `calculateTip` function into `Tipper`.
- 4 Create a new function named `calculate` in `Tipper` that loops through the `billTotals` and calculates the tip and adds the results to the `tips` array (you have this code from the previous Test Your Knowledge exercise). Remember to use this keyword.
- 5 Create a new function named `output` in `Tipper` that outputs to the console each bill total and tip amount on a separate console line (you have this code from the previous Test Your Knowledge exercise).
- 6 Finally, instantiate a new object using this function constructor and call its `generate()` method.

The output should look identical to the previous Test Your Knowledge exercise.