

LAB 6 Part 1

JAVASCRIPT 1: LANGUAGE FOUNDATIONS

What You Will Learn

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

EXERCISE 6.1. — USING BUILT-IN JAVASCRIPT OBJECTS

- 1 Examine lab06-walkthrough01.html.

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

- 2 Open lab06-walkthrough01.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.

- 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 6.1 (of course, the result of Math.random() will be different).

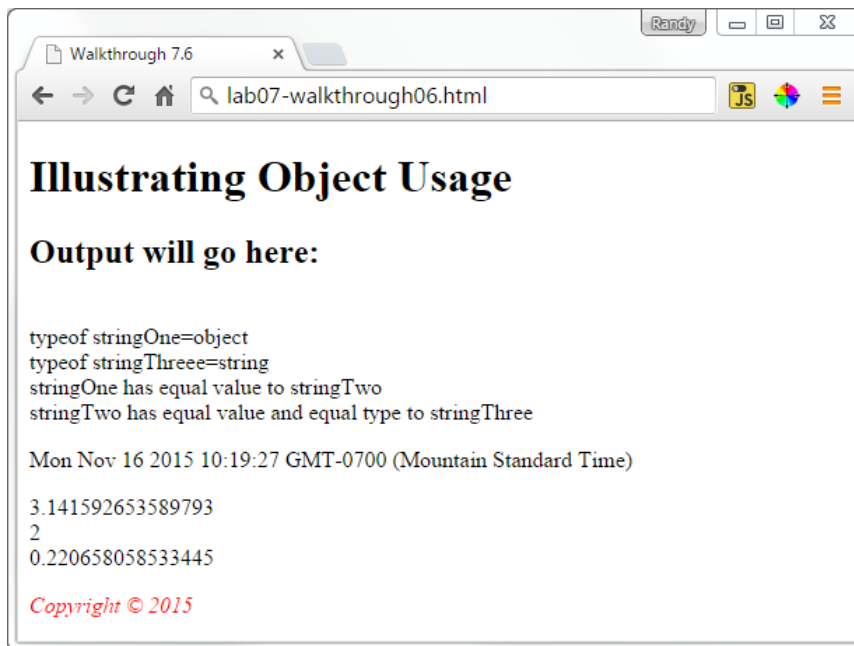


Figure 6.1 – Finished Exercise 6.1

Exercise 6.2 — 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.

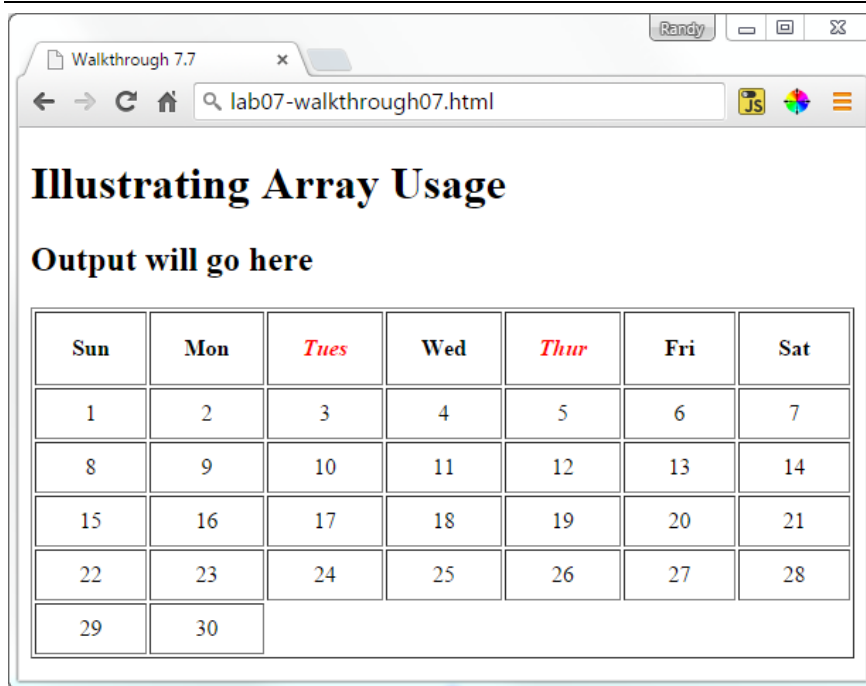
```

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 6.2 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 6.2 – 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 6.3. — JAVASCRIPT FUNCTIONS

- 1 Open `js/lab06-exercise03.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 `lab06-exercise03.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 6.3.

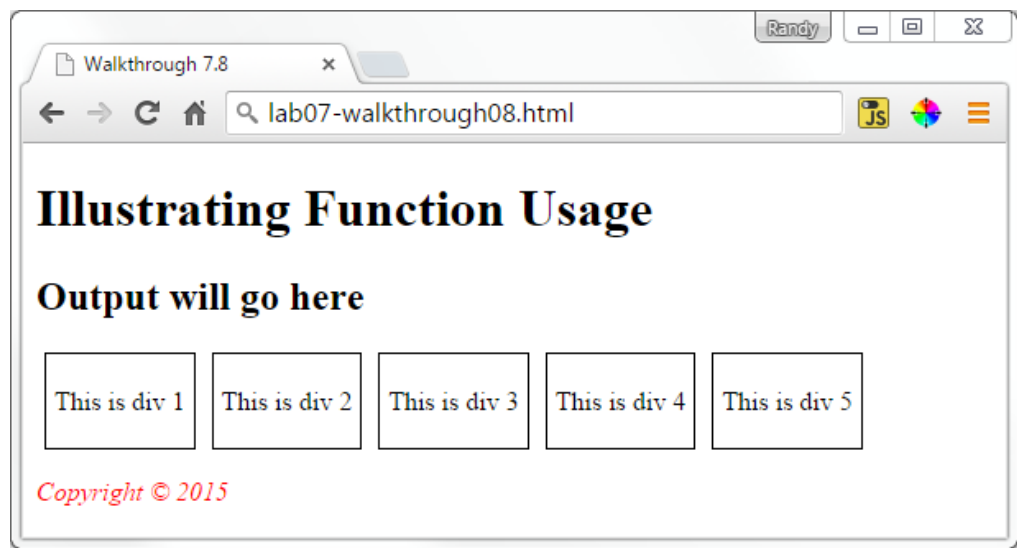


Figure 6.3 – Using JavaScript functions

- 8 Add the following variable definition to `js/lab06-exercise03.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.

```
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 6.4 — FUNCTIONS AS OBJECTS

- 1 Open `js/lab06-exercise04.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 `lab06-exercise04.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 6.5 — CREATING JAVASCRIPT OBJECTS

- 1 Open `js/lab06-exercise05.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 lab06-exercise05.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 6.4



Figure 6.4 – Result of Exercise 6.5.

