LAB 9

JavaScript 2: DOM and Events

What You Will Learn

- Accessing and modifying DOM HTML elements using JavaScript
- Creating event listeners to react to events
- Tools and tricks to help you develop JavaScript

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

THE DOCUMENT OBJECT MODEL (DOM)

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 `lab09` to your course folder created in step one.

In the previous lab, you used the `document.write()` function as a way to manipulate markup using JavaScript. While fine from a learning JavaScript perspective, it is generally not how one typically uses JavaScript. In the first part of this lab, you will learn how to use the Document Object Model (DOM) with JavaScript as a "better" way of manipulating content.

Exercise 9.1 — BASIC DOM SELECTION

1. Examine `lab09-walkthrough01.html` in your browser and then in your editor of choice.

   Notice that it references a JavaScript file at the end of the markup. To better understand the rest of the steps in this exercise, you should continually refer back to the markup.

2. Edit js/`lab09-walkthrough01.js` by adding the following and then testing `lab09-walkthrough01.html` in browser.

   ```
   var msg = document.getElementById("msgArea");
   msg.value = "here is some text";
   ```

   This selects the <textarea> element and sets its value.

3. Add the following code and test.

   ```
   var title = document.getElementById("pageTitle");
   title.innerHTML = "New Title";
   ```

4. Change the previous code to the following:

   ```
   document.getElementById("pageTitle").innerHTML = "Even newer title";
   ```

   This accomplishes the same thing without using a temporary variable.

5. Add the following code and test.

   ```
   msg.value = document.getElementById("first").innerHTML;
   ```

   Notice that the innerHTML property of a node returns all the HTML within it.

6. Change the previous code to the following and test.

   ```
   msg.value = document.getElementById("first").innerText;
   ```

   Notice that the innerText property of a node returns the text content stripped of any tags.

7 Add the following code and test.

```
msg.value = "My class is " + msg.getAttribute("class");
```

As you can see, the getAttribute method returns the value of the specified attribute (in this case the class attribute).

8 Add the following code and test.

```
var elem = document.getElementsByTagName("textarea");
elem[0].value = "selected by tag name";
```

The getElementByTagName returns an array of matching nodes. In this case it is an element with only one element.

9 Add the following and test.

```
elem = document.getElementsByTagName("span");
for (i=0; i<elem.length; i++) {
    document.getElementById("msgArea").value += "\n" + elem[i].innerText;
}
```

This selects all the elements and adds their contents to the <textarea>.

10 Add the following and test:

```
document.getElementById("msgArea").value = "";
var counts = document.getElementsByClassName("count-number");
for (i=0; i<counts.length; i++) {
    document.getElementById("msgArea").value += "\n" + counts[i].innerText;
}
```

This uses the getElementsByClassName() function to find all elements with matching classes.

11 Add the following and test:

```
var ps = document.querySelector("#first p");
document.getElementById("msgArea").value = ps.innerHTML;
```

The querySelector() function lets you use any CSS selector to select an element. If there are multiple elements that could match the selector, it returns only the first match.

12 Add the following and test.

```
ps = document.querySelectorAll("#first p");
for (i=0; i<ps.length; i++) {
    document.getElementById("msgArea").value += ps[i].innerHTML + "\n";
}
```

The querySelector() function returns an array of all matching elements.

13 Add the following code and test.

```
document.getElementById("msgArea").setAttribute("class", "hidden");
```

This sets the class attribute of the `<textarea>` element to hidden (this is a helper CSS class defined within Bootstrap).

14 Finally, add the following code and test.

```
document.getElementById("features").style.backgroundColor = "green";
```

This sets the background color of the `<div>` element of the `id` named “features”.

In the next exercise, you will be exposed to the different methods for "moving" around the tree structure of the DOM.

EXERCISE 9.2 — DOM Family Relations

1. Open `lab09-walkthrough02.html` in browser and open the JavaScript console.
2. Enter the following into the console:

   ```
   document.getElementById("first")
   ```

   This should display the contents of the `<div>` element named "first".
3. Enter the following into the console:

   ```
   document.getElementById("first").firstChild
   ```

   You might be surprised by the result. You might have expected the first child to be the <h1> element, but instead it is a text block which corresponds to the white space (returns and spaces/tabs) between the two elements.
4. Enter the following into the console:

   ```
   document.getElementById("first").firstChild.nextSibling
   ```

   This displays the next element at the same "level".
5. Enter the following into the console.

   ```
   document.getElementById("first").parentNode
   ```
6. Enter the following into `lab09-walkthrough02.js` and then test.

   ```
   var stories = document.getElementById("stories");
   for (i=0;i<stories.childNodes.length; i++)
   {
       console.log("child " + i + " =" + stories.childNodes[i] +
                   " nodeType=" + stories.childNodes[i].nodeType);
   }
   ```

   This steps through the child nodes of the `<div>` element named "first".
7. Modify your loop to change the background color of the non-text nodes of the `<div>` element named "first". The result should look similar to that shown in Figure 9.1.

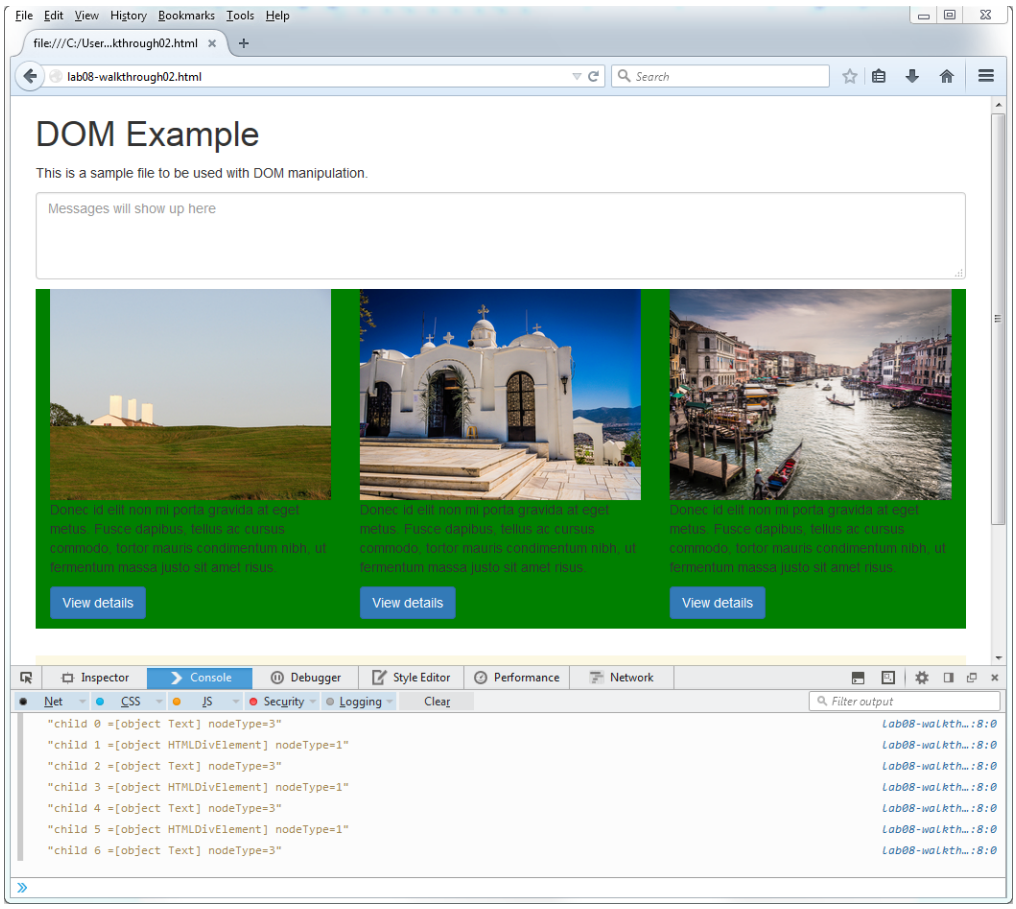

Figure 9.1 – Finished Exercise 9.2

EXERCISE 9.3 — MODIFYING THE DOM

1 Open `lab09-walkthrough03.js` and add the following.

```
/* code goes here */
var first = document.getElementById("first");

var text = document.createTextNode("this is programmatically created");
var p = document.createElement("p");

p.appendChild(text);
first.appendChild(p);
```

This creates a `<p>` element, adds some text to it, and then appends the new `<p>` element to the `<div>` element with the name first.

2 Test in browser. Use the inspect tools to examine the generated HTML.

3 Add the following lines and test.

```
var icons = document.getElementById("iconList");
icons.removeChild( document.getElementById("deleteThisOne") );
```

This deletes one of the `<li>` elements.

Event Handling

One of the most common uses of JavaScript is to use it to respond to different user-initiated actions. These are generally referred to as **events** and the JavaScript that responds to these events are called **event handlers**. As the textbook indicates, there are several ways of writing event handlers in JavaScript. In this lab, we stick with the preferred event listener approach.

EXERCISE 9.4 — Simple Event Handling

1 Open `lab09-walkthrough04.html` and test in browser.

2 Open `lab09-walkthrough04.js`, add the following, and then test.

```
/* code goes here */
function simpleHandler(event) {
    alert("button was clicked");
}

var btn = document.getElementById("testButton");
btn.addEventListener("click", simpleHandler);
```

This defines an event handler function and assigns it to the click event of the button. It is quite common to instead use an anonymous function for the event handler, as shown in the next steps.

3 Add the following code and test.

```
var img = document.getElementById("mainImage");
/* changes the style of the image when it is moused over */
img.addEventListener("mouseover", function (event) {
    img.className = "makeItGray";
});
```

This changes the class of the image when the mouse moves over the image. We now need to remove the class styling when the mouse leaves the image.

4 Add the following code and test.

```
/* remove the styling when mouse leaves image */
img.addEventListener("mouseout", function (event) {
   img.className = "makeItNormal";
});
```

5 Change the `simpleHandler()` function as follows and test.

```
function simpleHandler(event) {
    var content = document.getElementById("content");
    if (btn.innerHTML == "Hide") {
        content.style.display = "none";
        btn.innerHTML = "Show";
    }
    else {
        content.style.display = "block";
        btn.innerHTML = "Hide";
    }
}
```

This makes the button toggle the visibility of the content paragraph.

6 Instead of simply hiding/showing the paragraph, we will make use of CSS3 transitions as well as the JavaScript delay function. Change your code to the following and test.

```
function simpleHandler(event) {
    var content = document.getElementById("content");
    if (btn.innerHTML == "Hide") {
        btn.innerHTML = "Show";
        content.className = "makeItDisappear";
        // change the display mode after a 1000 millisecond delay
        setTimeout(function(){
            content.style.display = "none";
        },1000);
    }
    else {
        btn.innerHTML = "Hide";
        content.style.display = "block";

        // change the class after a 500 millisecond delay
        setTimeout(function(){
            content.className = "makeItNormal";
        },500);
    }
}
```

EXERCISE 9.5 — RESPONDING TO LOAD EVENTS

1 Open `lab09-walkthrough05.html` and test in browser.

2 Open `lab09-walkthrough05.js` and add the following.

```
var divToGet = document.getElementById("div1");
alert(divToGet.innerHTML);
```

3 Display the JavaScript console in your browser and then display Open `lab09-walkthrough05.js`.

If we save and refresh our page we might expect the alert to say "This is div 1" since that is what is contained within the div with id=div1. However, nothing is alerted, and we get an error (the message will differ in different browsers).

4 The reason for the above error is that the script is executed **before** the DOM is fully loaded. To make the script execute **after** the DOM is loaded we must make use of the `onload` event. Try this yourself by moving the code inside of a listener for the `onload` event as follows (and then test):

```
window.addEventListener("load", function(){
    var divToGet = document.getElementById("div1");
    alert(divToGet.innerHTML);
});
```

EXERCISE 9.6 — DEBUGGING EVENTS

1. Open `lab09-walkthrough05.html` and display the JavaScript console.

2. In Chrome, to access the debugger, you have to click on the Sources tab within the Console and then open the JavaScript file that you wish to debug.

   You will now add a breakpoint to your script by clicking to the left of the line that contains the call to alert().

3. Try refreshing the page. The line with the breakpoint will be highlighted as shown in Figure 9.2. You can now examine the state of local and global variables.

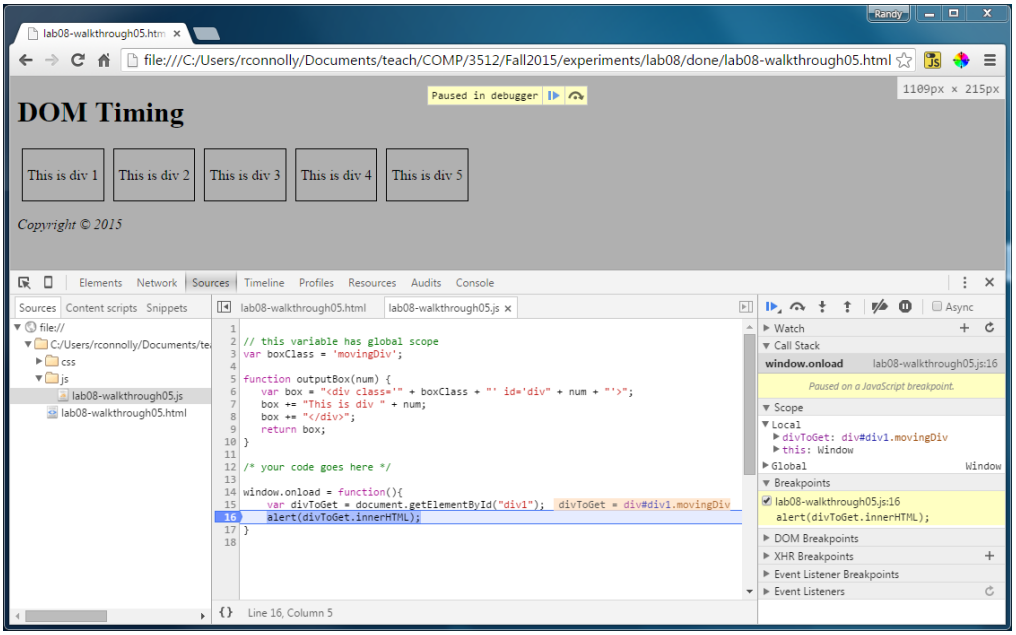

Figure 9.2 – Chrome Debugging

4. In FireFox, to access the debugger, you have to click on the Debugger tab within the Developer tools and then open the JavaScript file that you wish to debug.

   You can add a breakpoint to your script by clicking to the left of the line that contains the call to alert().

5. Try refreshing the page. The line with the breakpoint will be highlighted as shown in Figure 9.3. You can now examine the state of local and global variables.

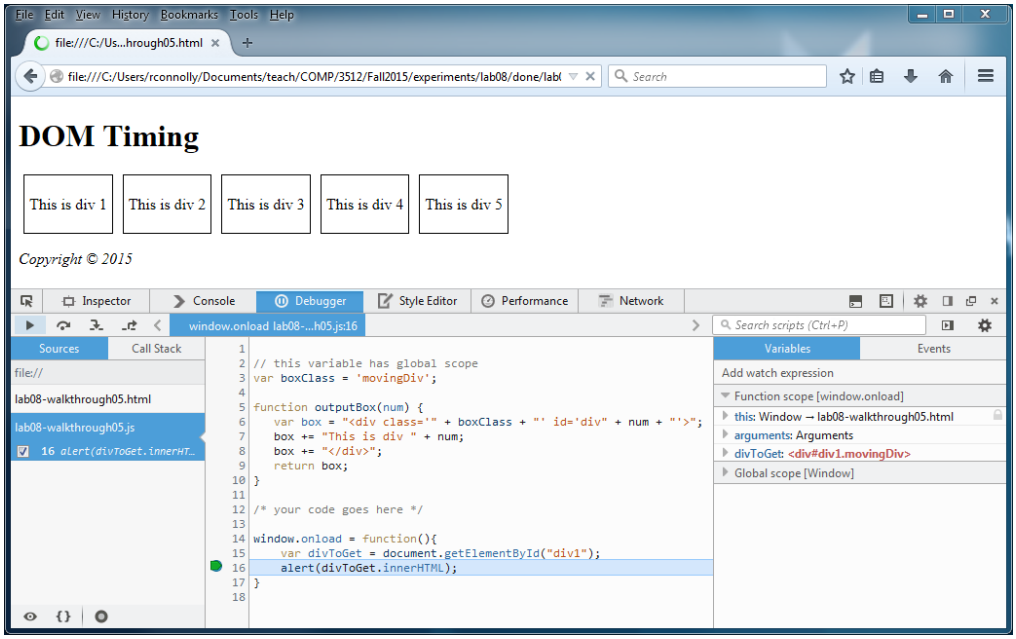

Figure 9.3 – FireFox Debugging

6 Once you have set a breakpoint, you can resume execution of the script or continue execution line-by-line using the execution buttons:

7 Edit `lab09-walkthrough05.js` and edit the code as follows:

```
window.addEventListener("load", function(){
    var divs = document.querySelectorAll(".movingDiv");
    for (i=0; i<divs.length; i++)
    {
        divs[i].addEventListener("mouseover", function (e) {
            alert("triggered by " + e.target.id);
        });
    }
});
```

8 Test in browser by moving your mouse over the different `<div>` boxes.

9 Use the debugger and set a watchpoint before the `alert()` call. Examine the contents of the `e` variable and `e.target` property.

EXERCISE 9.7 — WORKING WITH FORMS

1 Open `lab09-walkthrough07.html` and test in browser.

2 Add the following function to `lab09-walkthrough07.js`:

```
function setBackground(e) {
    if (e.type == "focus") {
        e.target.style.backgroundColor = "#FFE393";
    }
    else if (e.type == "blur") {
        e.target.style.backgroundColor = "white";
    }
}
```

This function is going to get called every time the focus or blur events are triggered in one of our form's input elements.

3 Add the following code:

```
window.addEventListener("load", function(){
    var cssSelector = "input[type=text],input[type=password]";
    var fields = document.querySelectorAll(cssSelector);

    for (i=0; i<fields.length; i++)
    {
        fields[i].addEventListener("focus", setBackground);
        fields[i].addEventListener("blur", setBackground);
    }
});
```

This assigns an anonymous function to the load event of the browser. The function assigns the `setBackground()` to the blur and focus events of the relevant `<input>` elements.

4 Test in browser. Tab between the different elements.

EXERCISE 9.8 — FORM VALIDATION

1 Examine `lab09-walkthrough08.html` and then add the following code to `lab09-walkthrough08.js`.

```
/* responsible for setting up event listeners on page */
function init() {
    document.getElementById("sampleForm").addEventListener("submit",
                                                    checkForEmptyFields);
}

/* initialize handlers once page is ready */
window.addEventListener("load", init);

/* ensures form fields are not empty */
function checkForEmptyFields(e) {
    var cssSelector = "input[type=text],input[type=password]";
    var fields = document.querySelectorAll(cssSelector);

    // loop thru the input elements looking for empty values
    var fieldList = [];
    for (i=0; i<fields.length; i++) {
        if (fields[i].value == null || fields[i].value == "") {
            // since a field is empty prevent the form submission
            e.preventDefault();
            fieldList.push(fields[i]);
```

```
        }
    }

    // now set up the error message
    var msg = "The following fields can't be empty: ";
    if (fieldList.length > 0) {
        for (i=0; i<fieldList.length; i++) {
            msg += fieldList[i].id + ",";
        }
        alert(msg);
    }
}
```

2 Test in browser. Experiment by filling in different fields.

3 Modify the `checkforEmptyFields()` function by adding the following:

```
function checkForEmptyFields(e) {
    // hide the error message element
    var errorArea = document.getElementById("errors");
    errorArea.className = "hidden";

    var cssSelector = "input[type=text],input[type=password]";
    var fields = document.querySelectorAll(cssSelector);

    // loop thru the input elements looking for empty values
    var fieldList = [];
    for (i=0; i<fields.length; i++) {
        if (fields[i].value == null || fields[i].value == "") {
            // since a field is empty prevent the form submission
            e.preventDefault();
            fieldList.push(fields[i]);
        }
    }
    // now set up the error message
    var msg = "The following fields can't be empty: ";
    if (fieldList.length > 0) {
        for (i=0; i<fieldList.length; i++) {
            msg += fieldList[i].id + ",";
        }
        errorArea.innerHTML = "<p>" + msg + "</p>";
        errorArea.className = "visible";
    }
}
```

Instead of displaying the error message inside an alert box this places it within a `<div>` element. The result should look similar to that shown in Figure 9.4.

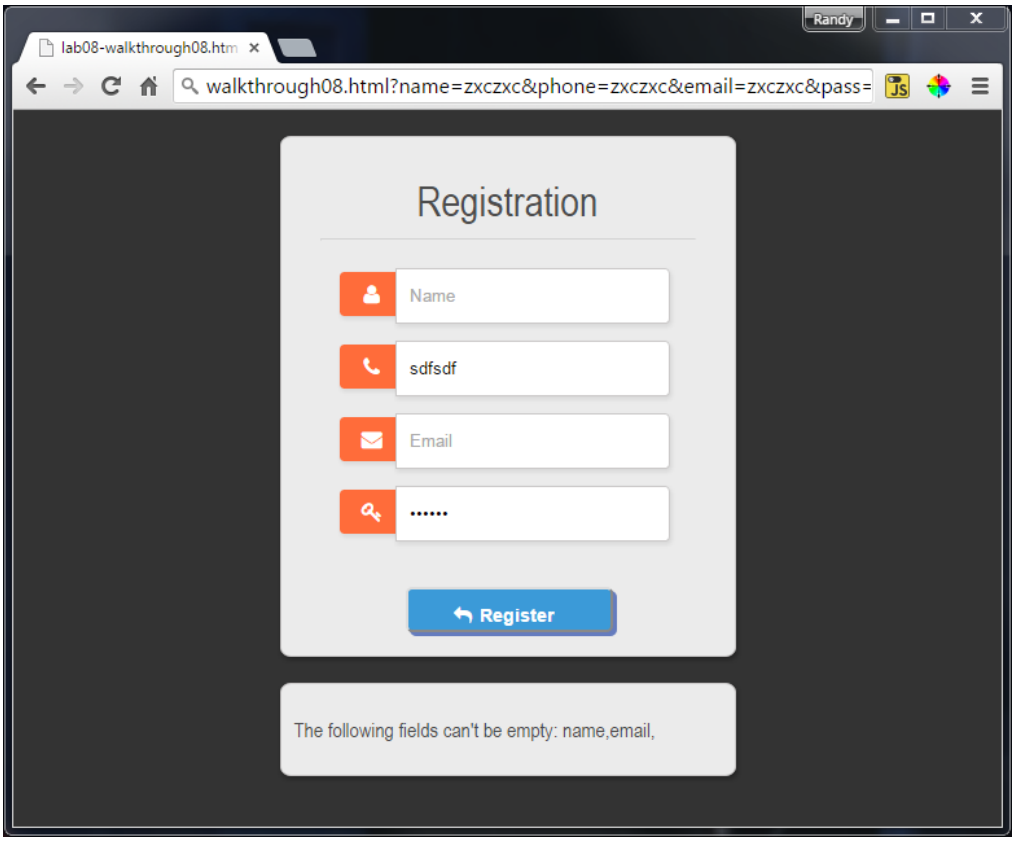

Figure 9.4 – Finished Exercise 9.8