JavaScript for Developers

Labs and Solutions

55244AC / MAX003AC

COPYRIGHT

© 2017 MAX Technical Training, Inc. All Rights Reserved. This manual and any training materials supplied with it are copyrighted with all rights reserved. No part of this publication may be reproduced, transmitted, transcribed, stored in a retrieval system or translated into any language in any form by any means without the written permission of MAX Technical Training, Inc.

TRADEMARKS

Brand names, company names and product names used herein are trademarks or registered trademarks of their respective companies.

MAX Technical Training, Inc.   
4900 Parkway Drive, Suite 160   
Mason, OH 45040  
513.322.8888

55244AC / MAX003AC A0 08/30/2017

[Lab Notes 3](#_Toc491983152)

[Module 1 Lab – Basic Scripting 4](#_Toc491983153)

[Questions 4](#_Toc491983154)

[Hands on Lab 5](#_Toc491983155)

[Exercise 1 – Experimenting with the F12 Developer Tools 5](#_Toc491983156)

[Exercise 2 – Update the time on the page by using JavaScript. 9](#_Toc491983157)

[Exercise 3 – Add JavaScript to respond to an event. 10](#_Toc491983158)

[Module 1 Lab – Answers 13](#_Toc491983159)

[Questions 13](#_Toc491983160)

[Hands on Lab 14](#_Toc491983161)

[Module 2 Lab - The Document Object Model 16](#_Toc491983162)

[Questions 16](#_Toc491983163)

[Hands on Lab 19](#_Toc491983164)

[Exercise 1 – Add an animation to the home page. 19](#_Toc491983165)

[Exercise 2 – Move the JavaScript to a File 20](#_Toc491983166)

[Exercise 3 – Add JavaScript to Supply MouseOver Tips to the Menu Tabs 21](#_Toc491983167)

[Module 2 Lab - Answers 26](#_Toc491983168)

[Questions 26](#_Toc491983169)

[Hands on Lab 30](#_Toc491983170)

[Exercise 1 Code: 30](#_Toc491983171)

[Exercise 3 Code: 31](#_Toc491983172)

[Module 3 Lab – jQuery 33](#_Toc491983173)

[Questions 33](#_Toc491983174)

[Hands on Lab 34](#_Toc491983175)

[Exercise 1 – Add the jQuery Library to the Project 34](#_Toc491983176)

[Exercise 2 – Create New Elements using jQuery 34](#_Toc491983177)

[Module 3 Lab – Answers 36](#_Toc491983178)

[Questions 36](#_Toc491983179)

[Module 4 Lab - Ajax and Web Services 38](#_Toc491983180)

[Questions 38](#_Toc491983181)

[Hands on Lab 39](#_Toc491983182)

[Exercise 1 – Exploring Data Files 39](#_Toc491983183)

[Exercise 2 – Exploring a REST Web Service 40](#_Toc491983184)

[Exercise 3 – Load Data from a Web Service 41](#_Toc491983185)

[Module 4 Lab – Answers 48](#_Toc491983186)

[Questions 48](#_Toc491983187)

[Module 5 Lab – Diving in Deeper 49](#_Toc491983188)

[Questions 49](#_Toc491983189)

[Hands on Lab 49](#_Toc491983190)

[Exercise 1 – Exploring Types and Objects 50](#_Toc491983191)

[Exercise 2 - Exploring Default Objects 52](#_Toc491983192)

[Exercise 3 – Debugging using the “console” Object 53](#_Toc491983193)

[Exercise 4 - Working with Scope 56](#_Toc491983194)

[Exercise 5 – Functions and Event Handlers 57](#_Toc491983195)

[Exercise 6 - Closure 61](#_Toc491983196)

[Module 5 Lab – Answers 65](#_Toc491983197)

[Questions 65](#_Toc491983198)

[Hands on Lab 66](#_Toc491983199)

[Exercise 1 66](#_Toc491983200)

[Exercise 3 – Debugging using the “console” Object 67](#_Toc491983201)

[Exercise 4 - Working with Scope 67](#_Toc491983202)

[Exercise 5 – Functions and Event Handlers 68](#_Toc491983203)

[Exercise 6 - Closure 68](#_Toc491983204)

## Lab Notes

Try each lab from the basic steps first, and then review the lab solutions if you need some help.

**Your Environment Might Not Look the Same**

The screens and features depicted in this courseware and in the lab environment may differ due to:

* Drive letters or network paths used for lab files.
* Version of the editor being used.
* Browser versions.

**Your instructor should provide you with the following, if needed:**

URL to your server: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

User Name: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Password: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Path to sample files: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## Module 1 Lab – Basic Scripting

### Questions

❑ Which of the following are names for the browser’s scripting language?

* LiveScript
* VisualScript
* JavaScript
* ECMAScript
* JScript

❑ Which of the following are names for JavaScript routines to deal with older browsers?

* Slices
* Shims
* Wedges
* Shives
* Pollyfills

❑ Which of the following is the most commonly used shortcut key for launching the browser’s debugging tools?

* F1
* F2
* F5
* F10
* F12

❑ A JavaScript variable is formally created using:

* Dim
* var
* let
* new

❑ Which of the following are valid JavaScript comments?

* // this is a comment
* \\ this is a comment
* !- this is a comment
* <!-- this is a comment -->
* /\* this is a comment \*/
* /\*  
   this is a comment  
  \*/
* REM this is a comment

❑ Which of the following are used to compare two variables?

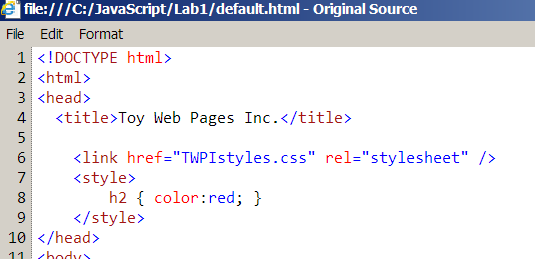
* =
* ==
* ===
* >
* !=
* <>

### Hands on Lab

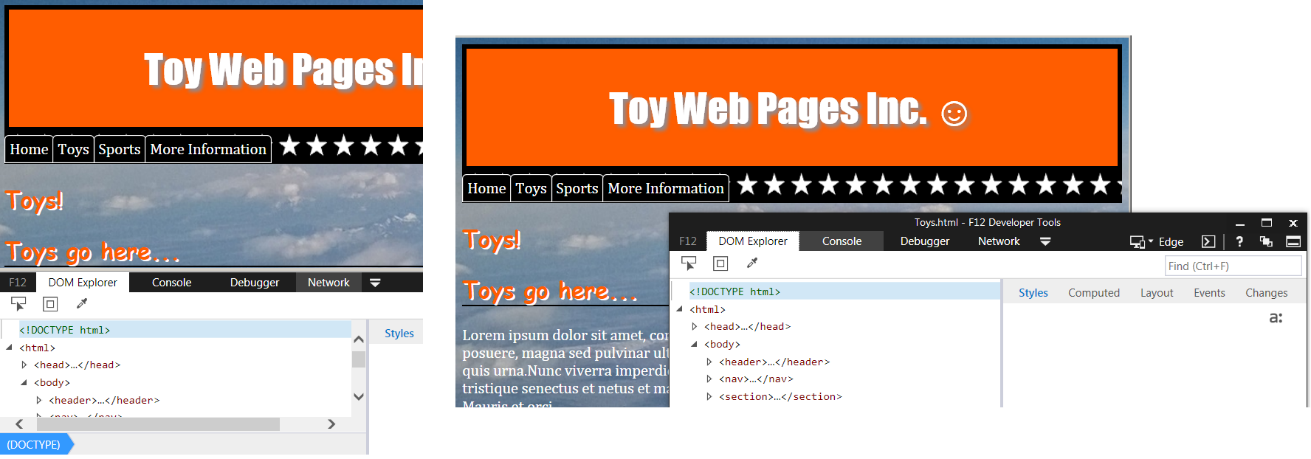
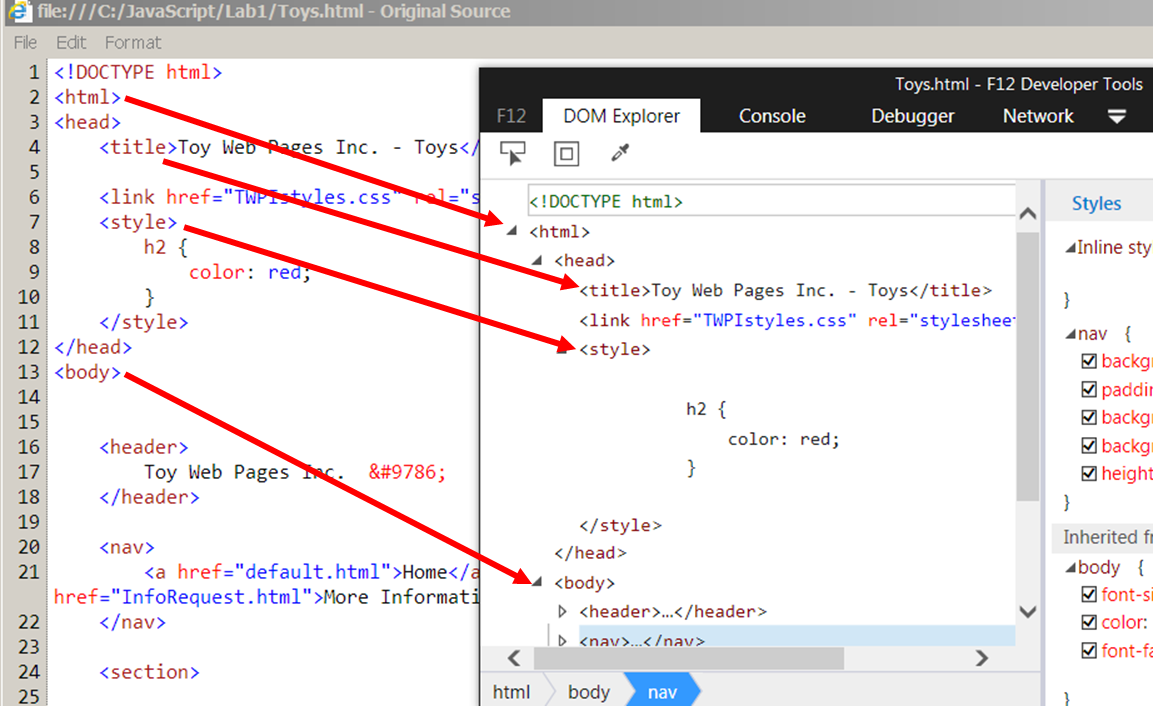
#### Exercise 1 – Experimenting with the F12 Developer Tools

1. Open Internet Explorer.
2. Click File, Open and Browse.
3. Navigate to your lab folder and the Module1 folder.  
    Most likely: C:\JavaScript\Module1 (Check with your instructor.)
4. Double-click “default.html” (or “default” if the file extensions are not displayed)

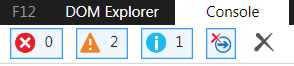
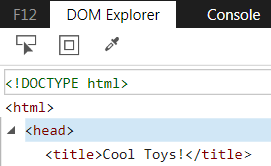
View the source of the page.

1. In the browser’s menu (tap Alt if not displayed) click View and Source.  
    
2. Browse the source and note:
   1. This is an HTML5 document. (<!DOCTYPE html>)
   2. There is a linked CSS file.
   3. There is style block. <style>…</style>

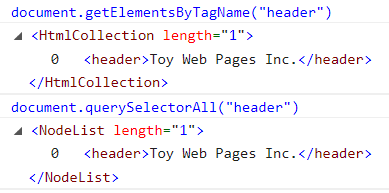
View the source of the page using F12.

1. In the browser, press F12.
2. You can leave the F12 as a pane, or you can click the Unpin icon () to float the F12 window.  
   
3. Click the DOM Explorer tab.
4. Expand the <head> section and compare the <head> element and its children to the HTML in the Original Source window you opened earlier.  
   
   1. Notes:
      1. The Original Source is a text file.
      2. The DOM is a collection of Element objects created from the source text file.
      3. The DOM and the source will rarely match in formatting.
      4. The DOM and the source may vary if JavaScript code has changed the DOM.
      5. Changes to the DOM in the F12 tools will not change the source file.

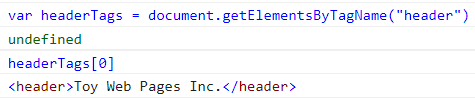
##### Write JavaScript to Interact with the DOM

1. Click the Console tab.
2. Click the “X” to clear any messages.  
    
3. In the Console Input area (at the bottom of the F12 window) enter the following to display the site title. Press Enter.  
    
   1. Notes:
      1. document is the JavaScript / browser object that represents the loaded document.
      2. title is the content of the <title>…</title> tag.
4. Note that the command was echoed and the result displayed.  
    
5. Change the document’s title.  
    
6. Verify the change n the browser’s tab.  
    
7. Verify the change in the DOM Explorer:
   1. Click the DOM Explorer tab.
   2. Expand the <head> section.
   3. Note the title contents.  
       
8. Close the Original Source window.
9. In the browser’s menu (tap Alt if not displayed) click View and Source and note that the <title> has not changed in the source code.  
     
   The DOM is created from the source, but is not the source!

##### Find and Change the Header Element

1. Click the Console tab. Type either of the following commands to find the H1 Element:
   1. 
   2.   
        
      

Note: .getElementsByTagName returns an HtmlCollection object while the .querySelectAll returns a NodeList object. For our purposes here, they can be used interchangeably. If you are curious about the differences, then see this article: https://www.nczonline.net/blog/2010/09/28/why-is-getelementsbytagname-faster-that-queryselectorall

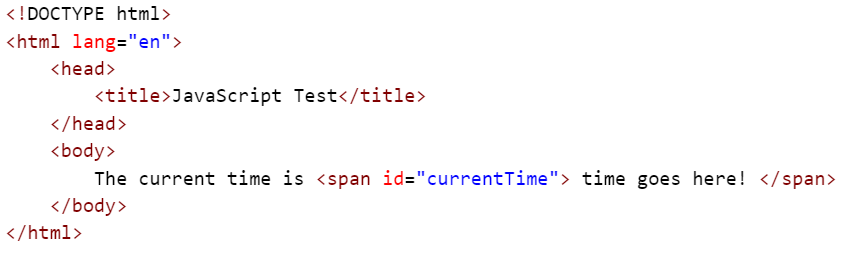
1. Both of the above methods retrieved a collection of <header> elements. In our example this was a collection of one! To work with a single item from a collection we will need to use an indexer.
   1. Example 1:   
      
   2. Example 2:  
      
2. To change the content of the <header> tag we can use the .textContent property if the content is all text, or .innerHTML if the new content will include HTML tags.
   1. Test 1: Use textContent:  
        
      Confirm that the site’s main heading has changed!
   2. Test 2: use innerHTML:  
       Confirm that the site’s main heading has changed, and includes italics!
   3. What would have happened if you had used .textContent with the text that included the <em> tags? Try it and see…

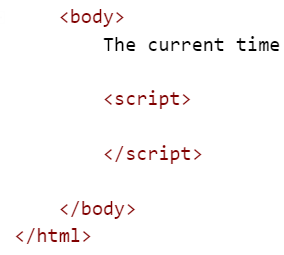
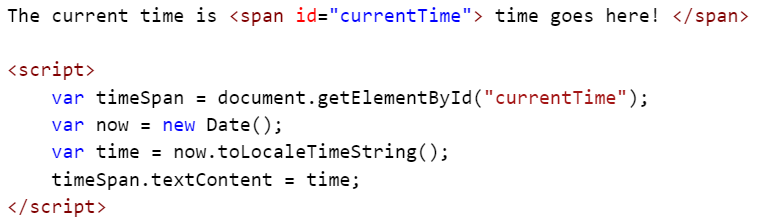
##### Does this site use tables, divs or hyperlinks?

1. Use document.getElementsByTagName to find all of the “table” tags. How many did you find? \_\_\_\_
2. How many “div” tags are there? \_\_\_\_\_
3. How many “a” tags are there? \_\_\_\_\_  
     
   Tip: The document object has a property named links that returns a collection of hyperlinks. Do both of the following produce the same number of links?
   1. document.getElementsByTagName(“a")
   2. document.links

#### Exercise 2 – Update the time on the page by using JavaScript.

1. Create the following web page using your editor of choice. Save it as TheTime.html.



1. Just above the </body> tag insert a few blank lines and a script block:  
   
2. Then in the script block add a line of JavaScript to retrieve the span element and store it in a variable. Note that the span could be found using .getElementsByTagName("span"), but that is not reliable as there could be multiple spans in the page. Instead use .getElementById("currentTime").  
    
3. Set the textContent property of the span to the current time.
   1. The current time can be retrieved from the Date object. While there are several ways to build the time string, the easiest is the .toLocaleTimeString() method.
   2. Add the following lines as the next two lines of your script.  
       
4. Update the span’s textContent property to use the time string value.  
    
5. The result should look like this:  
    
6. Save the file and open it in a browser.  
    

The script above could have been written as one line.



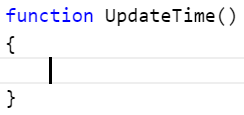
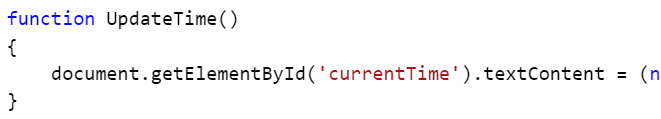
Both work. One adds a little overhead by creating variables that will only be used once, but may be clearer to the next developer who needs to understand it. The other is “concise”!

#### Exercise 3 – Add JavaScript to respond to an event.

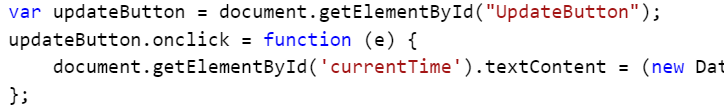
##### Add a button to update the time.

1. Just above the <script> block you added earlier, add a button tab with some text.  
    
2. Add an onclick event handler (an attribute) to the button.  
    
3. In that event handler add JavaScript between the quotes to update the time. Use the single line version of our previous script.  
   
   1. Note that our onclick attribute used double quotes, so the JavaScript code must use single quotes. You can reverse this if you like.
4. The result should look like this:  
    
5. Save the page and test.  
    
6. While this solution works, there are at least two better solutions:
   1. Take our code, wrap it in a function, and then call the function from the onclick event.
   2. Leave all of the JavaScript code in the <script> block and let JavaScript update the onclick event dynamically.

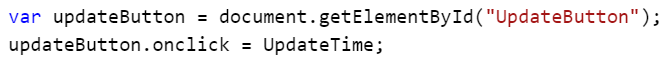
##### Add a function, and another button.

1. In the existing script block add a function named “UpdateTime”.  
    
2. Copy either the multiple line or the single line solution into the new function.  
    
3. Add a new button with an onclick that calls our function.  
   
4. Save the page and test.

##### Add a button and JavaScript add the onclick.

1. While the button we just added is cleaner and its use is now more obvious, we still have JavaScript embedded in our tags. An even cleaner solution would be to keep the HTML and JavaScript completely separated.
2. Add a new button and give it an ID of “UpdateButton”.   
   
3. At the end of the existing script block add this script:  
   
4. Save the page and test.

An alternate solution… As we already have a function to update the time, we could just assign it to the onclick event.



## Module 1 Lab – Answers

### Questions

❑ Which of the following are names for the browser’s scripting language?

* LiveScript
* VisualScript
* **JavaScript**
* **ECMAScript**
* **JScript**

Jscript is Microsoft’s name for JavaScript.

❑ Which of the following are names for JavaScript routines to deal with older browsers?

* Slices
* **Shims**
* Wedges
* **Shives**
* **Pollyfills**

❑ Which of the following is the most commonly used shortcut key for launching the browser’s debugging tools?

* F1
* F2
* F5
* F10
* **F12**

❑ A JavaScript variable is formally created using:

* Dim
* **var**
* **let**
* new

var is the most common way of declaring a variable. ECMAScript 2015 adds the let keyword to create block scoped variables. (More on “let”: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/let)

❑ Which of the following are valid JavaScript comments?

* **// this is a comment**
* \\ this is a comment
* !- this is a comment
* <!-- this is a comment -->
* **/\* this is a comment \*/**
* **/\*  
   this is a comment  
  \*/**
* REM this is a comment

❑ Which of the following are used to compare two variables?

* =
* **==**
* **===**
* **>**
* **!=**
* <>

A single “=” is used to assign a value.

“<>” is used in other languages as “not equals”. JavaScript uses “!=” for “not equals”.

### Hands on Lab

Solution:

<!DOCTYPE html>

<html lang="en">

<head>

<title>JavaScript Test</title>

</head>

<body>

The current time is <span id="currentTime"> time goes here! </span>

<button onclick="document.getElementById('currentTime').textContent = (new Date()).toLocaleTimeString();" >Update the time</button>

<button onclick="UpdateTime()">Update time with a function</button>

<button id="UpdateButton">A clean update time button</button>

<script>

var timeSpan = document.getElementById("currentTime");

var now = new Date();

var time = now.toLocaleTimeString();

timeSpan.textContent = time;

function UpdateTime()

{

document.getElementById('currentTime').textContent = (new Date()).toLocaleTimeString();

}

var updateButton = document.getElementById("UpdateButton");

updateButton.onclick = UpdateTime;

// alternate solution

//updateButton.onclick = function (e) {

// document.getElementById('currentTime').textContent = (new Date()).toLocaleTimeString();

//};

</script>

</body>

</html>

## Module 2 Lab - The Document Object Model

### Questions

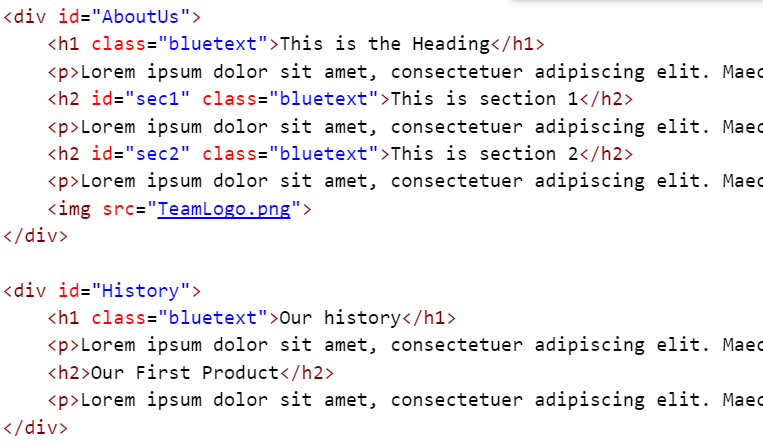
❑ The Document Object Model is:

* Part of core JavaScript
* Created from the downloaded HTML file.
* Past of the browser, and is different for each brand.

❑ Which of the following would retrieve the <head> section of the DOM?

* window.head
* window.document.head
* navigator.head
* document.head
* document.getElementByTagName("head")
* document.getElementsByTagName("head")[0]

The next five questions are based on the following HTML. It represents the full contents of the page’s body tag.



❑ Which of the following will retrieve all of the paragraph elements (<p>)?

* var paragraphs = document.getElementById("p")
* var paragraphs = document.getElementTagName("p")
* var paragraphs = document.getElementsByTagName("p")
* var paragraphs = document.getElementsByTagName("<p>")
* var paragraphs = document.getElementsByClassName("p")

❑ Which of the following will retrieve the image element (<img>)?

* var myImage = document.getElementById("src")
* var myImage = document.getElementsByTagName("img")
* var myImage = document.getElementsByTagName("img")[0]
* var myImage = document.getElementsByTagName("img")[1]

❑ Which of the following will retrieve just the first heading element (<h1>)?

* var firstH1 = document.getElementById("AboutUs").getElementsByTagName("h1")[0]
* var firstH1 = document.getElementsByTagName("h1")
* var firstH1 = document.getElementsByTagName("h1")[0]
* var firstH1 = document.getElementsByClassName("bluetext")
* var allH1s = document.getElementsByTagName("h1")  
  var firstH1 = allH1s[0]
* var allH1s = document.getElementsByTagName("h1")  
  var firstH1;  
  for (var i=0; i<allH1s.length; i++) {   
   if (allH1s[i].textContent == "This is the Heading") {   
   firstH1 = allH1s[i];   
   break;   
   }  
  }

❑ Considering only the correct answers from the question above, which of the following is the most reliable? I.e. would survive edits to the content.

* var firstH1 = document.getElementById("h1")
* var firstH1 = document.getElementById("AboutUs").getElementsByTagName("h1")[0]
* var firstH1 = document.getElementsByTagName("h1")
* var firstH1 = document.getElementsByTagName("h1")[0]
* var firstH1 = document.getElementsByClassName("bluetext")
* var allH1s = document.getElementsByTagName("h1")  
  var firstH1 = allH1s[0]
* var allH1s = document.getElementsByTagName("h1")  
  var firstH1;  
  for (var i=0; i<allH1s.length; i++) {   
   if (allH1s[i].textContent == "This is the Heading") {   
   firstH1 = allH1s[i];   
   break;   
   }  
  }

❑ Which of the following will retrieve all of the H2 elements in the first DIV?

* var H2tags = document.getElementsByTagName("h2")
* var H2tags = document.getElementsByTagName("div")[0].getElementsByTagName("h2")
* var H2tags = document.getElementById("AboutUs").getElementsByTagName("h2")
* var H2tags = document.querySelector ("h2")
* var H2tags = document.querySelector ("AboutUs h2")
* var H2tags = document.querySelector ("#AboutUs h2")
* var H2tags = document.querySelector (".AboutUs h2")
* var H2tags = document.querySelectorAll ("AboutUs h2")
* var H2tags = document.querySelectorAll ("#AboutUs h2")
* var H2tags = document.querySelectorAll (".AboutUs h2")

❑ Considering only the correct answers from the question above, which of the following is the most reliable? I.e. would survive edits to the content.

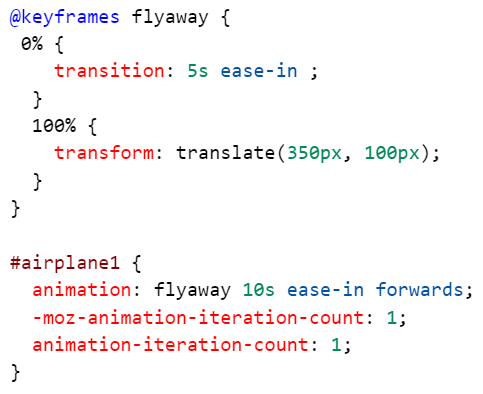
* var H2tags = document.getElementsByTagName("h2")
* var H2tags = document.getElementsByTagName("div")[0].getElementsByTagName("h2")
* var H2tags = document.getElementById("AboutUs").getElementsByTagName("h2")
* var H2tags = document.querySelector("h2")
* var H2tags = document.querySelector("AboutUs h2")
* var H2tags = document.querySelector("#AboutUs h2")
* var H2tags = document.querySelector(".AboutUs h2")
* var H2tags = document.querySelectorAll("AboutUs h2")
* var H2tags = document.querySelectorAll("#AboutUs h2")
* var H2tags = document.querySelectorAll(".AboutUs h2")

### Hands on Lab

You have been asked to add new functionality to a site with the least modification to the existing HTML as possible. You are allowed to link to a JavaScript file.

#### Exercise 1 – Add an animation to the home page.

The UI designers have come up with a CSS animation that needs to be added to the page. While it’s usually better to add this CSS to a CSS file, the “powers that be” have said that the only allowed modification to the site is the addition of a JavaScript code block or linked file. (Odd, but let’s us experiment with a few JavaScript features!)

1. Open Internet Explorer.
2. Click File, Open and Browse.
3. Navigate to your lab folder and the Module2\Lab2 folder.  
    Most likely: C:\JavaScript\Module2\Lab2 (Check with your instructor.)
4. Double-click “default.html” (or “default” if the file extensions are not displayed)
5. Note the location of the airplane image.
6. Open the default.html page in your HTML editor.
7. The graphics designers have given us the CSS needed to animate the image.   
   
8. When they tested this, it worked ok except that the airplane jumped back to the starting location at the end and the text around the airplane did not expand to fill the now empty space. We will fix this by running some JavaScript at the end of the animation.

Add JavaScript to add the CSS to the page.

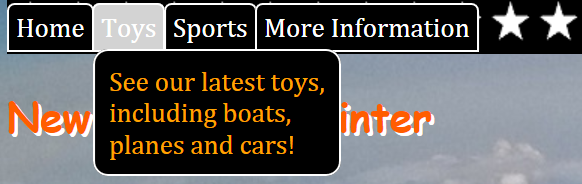
1. For initial testing, we will add a script block at the end of the page.
   1. Just before the </body> tag add a script block.  
       <script>  
       </script>
   2. Using Notepad open the CSSanimation.txt file and copy all of the text to the clipboard.
   3. As this text has many lines, this sample CSS uses the “new line” escape character. The is a backslash immediately followed by a carriage return.
   4. (This is covered in Module 5 – Diving in Deeper.) Between the script tags add these lines: (One line starting with “var”, one blank line and one line with only a quote.)  
        
       var styletext = "  
         
       "
   5. Paste the copied CSS between the quotes.
2. Now we need to add a style block in the <head> section of the page using JavaScript.
   1. Add the following after the last back tick.  
       var newStyleBlock = document.createElement("style");  
       newStyleBlock.textContent = styletext;  
       document.head.appendChild(newStyleBlock);
   2. This code will create a new style element object in memory, add the CSS as text between the start and end tags, and add the new <style> block at the end of the <head> section.
3. Save your file.
4. Return to the browser and refresh the home page.
5. Did the airplane fly off of the page? If not check your code!
6. Did the airplane jump back to the starting position?
7. We need to hide the airplane image after the CSS animation has completed. CSS animations raise events at the start and end of each animation. We need to add and event listener to hide the image at the end of the animation.
8. At the end of the JavaScript code add a line to get the “airplane1” image element.  
    var airplane = document.getElementById("airplane1");
9. Below that line add a line that will set an anonymous function to the animationend event that will hide the image.  
    airplane.addEventListener("animationend",   
    function() { airplane.style.display="none" }, false);
10. Save your file.
11. Return to the browser and refresh the home page.
12. Did the airplane fly off of the page followed by the text rewrapping to fill the now empty space?

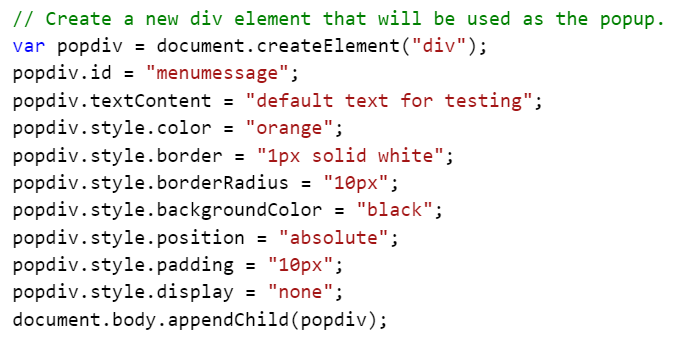
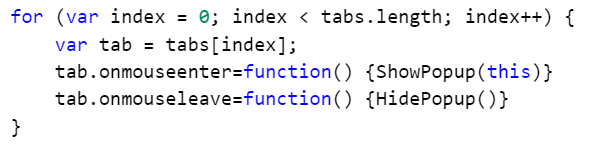
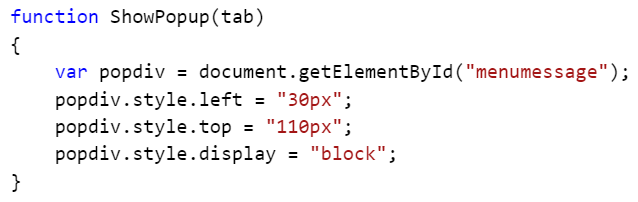
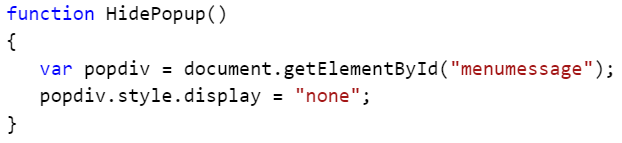
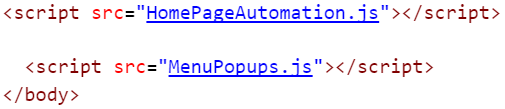
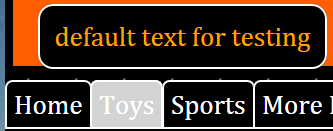
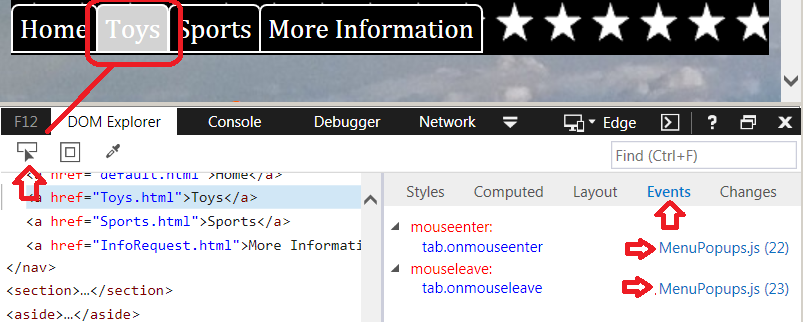
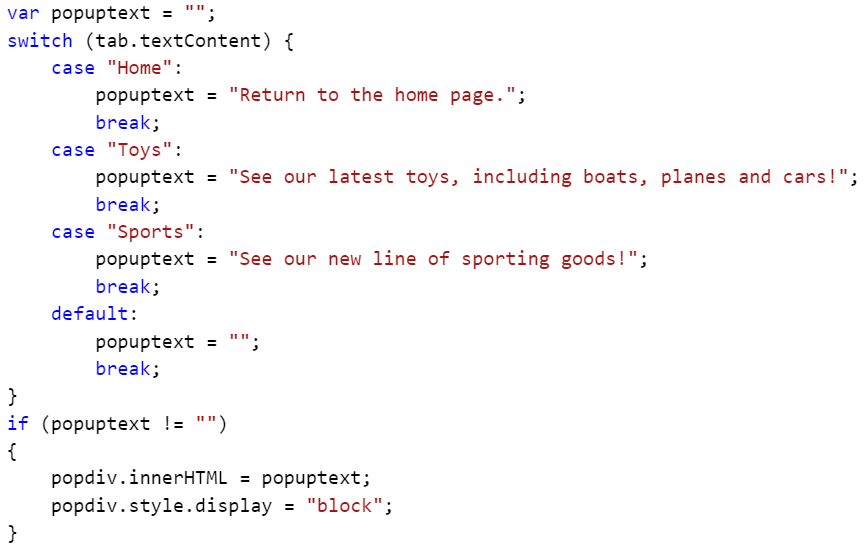
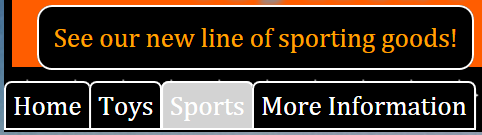
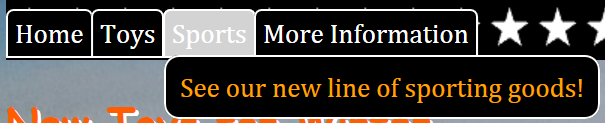
#### Exercise 2 – Move the JavaScript to a File

1. Open the default.html page in your HTML editor.
2. Select and Cut all of the code between the <script> and </script> tags.
3. Save the default.html file.
4. Open a new file in your editor. If your editor asks for a name when creating the file, name it “HomePageAutomation.js” and save it in the same directory as the home.html page.
5. Paste the code into the new file.
6. Save the file. If prompted, save the file as “HomePageAutomation.js” and save it in the same directory as the home.html page.
7. Return to the default.html file in the editor.
8. Modify the <script> tag by adding a src attribute with the name of the file you just created.  
    <script src="HomePageAutomation.js"></script>
9. Return to the browser and refresh the page. The plane should still fly off of the page.
10. Return to the editor for the “HomePageAutomation.js” file.
11. Add a JavaScript comment block and add your name as the author and a few comments about the purpose of the script. Something like this:  
     /\*   
     Author: Mike  
     Date: 3/14/17  
     Used by: default.html  
     Notes:  
     Creates a new <style> block to slowly move the image off of the page.  
     Add an event handler to hide the image after the animation.  
     \*/

#### Exercise 3 – Add JavaScript to Supply MouseOver Tips to the Menu Tabs

We have been asked to create popouts that are displayed when the user moves their mouse over the navigation tabs. It might look something like this:



1. Create a new text file named “MenuPopups.js” and save it in the same location as web site.
2. We need to create a DIV to contain the popup message. Its exact starting location is not too critical as it will default to being hidden. The basic JavaScript to create the DIV and add it to the end of the page looks like this:  
    var popdiv = document.createElement("div");  
    // more JS here to style the div  
    document.body.appendChild(popdiv);
3. You may want to apply your graphics design skills and fancy this up a bit… but for your first test use add this JavaScript to your file:  
    
4. We now need to attach some code to the navigation links. These are stored in a <nav> tag with an ID of “topNav”. Add a blank line to your JavaScript file and then add a line to retrieve the <a> tags that are inside of the <nav> tag.  
      
   Note that you could also have written this as:  
    
5. Now that we have a collection of the navigation links (named “tabs”) we need to add event handlers to the onmouseover and onmouseleave events for each tab (<a>). We will write a FOR loop to iterate though the tabs collection. Note that we have not written the ShowPopup and HidePopup functions yet.  
    
6. For a first test to see it the popup will appear and disappear as it should, let’s just display the default box just above the navigation area. The “for testing” ShowPopup might look like this:  
    
7. Just below the ShowPopup function add the HidePopup function.  
    
8. Save your JavaScript file.
9. Open the default.html page in your HTML editor. (This is our home page.)
10. At the bottom of the file, just before the </body> tag, add a script block to load our JavaScript file.  
     
11. Save the page.
12. Open our site (default.html) in a browser.
13. Move your mouse over one of the tabs and verify the popup is displayed. If not, debug fix your code!  
     
14. If you are testing with Internet Explorer, press F12 to open the Developer Tools page. Click the DOM Explorer tab, click the Select Element button (box with an arrow) and click one of the navigation tabs. In the right pane click the Events tab. Notice that two events are attached to that <a> element.  
     
15. Return to your JavaScript file so we can add some text to the popup.
16. The text will be different for each tab, so we need to find a way to identify a tab. As the original designer of the page did not add IDs to each <a> tag, we could work from their order (tab[0], etc.) or maybe from the text in the tab. This example will use the text. Write code to select text based on the textContent property of each <a> element. While there are multiple ways to do this, let’s use a switch statement for now.  
    
17. Save the page.
18. Open our site (default.html) in a browser.
19. Move your mouse over one of the tabs and verify the popup is displayed. If not, debug fix your code!  
     
20. After presenting this to your client, they ask if you can position the popup directly below the tab instead of in a fixed location above the tabs. Something like this:  
     
21. To do this we need to find the location of the tab. As they are not individually styled using CSS absolution positions, we will have to read their current locations. The DOM includes .offsetLeft and .offsetTop for this. Sadly ☹ we cannot just do this:  
     popdiv.offsetLeft = tab.offsetLeft;   
    Instead we have to set the .style.left from a string that includes the “px” unit. So ….  
     popdiv.style.left = tab.offsetLeft + "px";
22. Find these lines in your JavaScript file, and delete or comment them.  
     
23. Add these lines to position our popup:  
       
    This will position our popup DIV to be aligned with the left edge of the tab, and down 30px from the tab. (Feel free to tweak this positioning!)
24. Save the page.
25. Open our site (default.html) in a browser.
26. Move your mouse over one of the tabs and verify the popup is displayed. If not, debug fix your code!

Just for fun… add these two lines where we were setting all of the other style options for the popup:



And then test…  
 

Ok, that’s an overkill…

1. If this were a real project, you would add the following script block to all other pages in the site. (AboutUs.html, Contact.html, InfoRequest.html, Priavcy.html and Toys.html)  
    

Extra Credit!

We’ve been asked to add HTML tags to the text for bold, italics, and line breaks. If you just add the tags they currently will be displayed in the output. To support HTML in the text:

Find:  
 popdiv.textContent = popuptext;

Change it to:  
 popdiv.innerHTML = popuptext;

Test by changing the “Toys” message to something like:  
 "See our latest toys, **<br>**including boats, **<br>**planes and cars!"

## Module 2 Lab - Answers

### Questions

❑ The Document Object Model is:

* Part of core JavaScript
* **Created from the downloaded HTML file.**
* Past of the browser, and is different for each brand.

❑ Which of the following would retrieve the <head> section of the DOM?

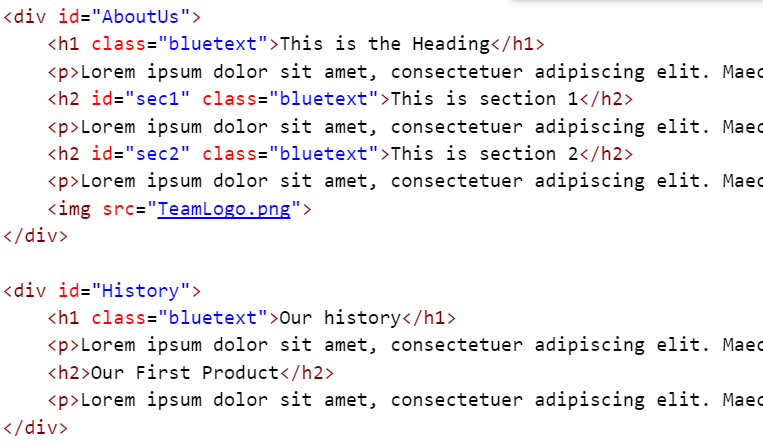
* window.head
* **window.document.head**
* navigator.head
* **document.head**
* document.getElementByTagName("head")
* **document.getElementsByTagName("head")[0]**

document.head is the easiest to use. window.document.head is also correct as document is a child of the window object.

document.getElementByTagName("head") is close, but not 100% correct as it returns a collection of <head> elements with a count of 1.

document.getElementByTagName("head")[0] is correct as it returns just the one object we asked for.

The next five questions are based on the following HTML. It represents the full contents of the page’s body tag.



❑ Which of the following will retrieve all of the paragraph elements (<p>)?

* var paragraphs = document.getElementById("p")
* var paragraphs = document.getElementByTagName("p")
* **var paragraphs = document.getElementsByTagName("p")**
* var paragraphs = document.getElementsByTagName("<p>")
* var paragraphs = document.getElementsByClassName("p")

document.getByElementTagName("p") is not correct as there is not a method by that name. (missing “s”)

An element/tag is not an Id or ClassName.

The tag “name” is “p”, not “<p>”.

❑ Which of the following will retrieve the image element (<img>)?

* var myImage = document.getElementById("src")
* var myImage = document.getElementsByTagName("img")
* **var myImage = document.getElementsByTagName("img")[0]**
* var myImage = document.getElementsByTagName("img")[1]

The tag name is “img”, not “src”.

document.getElementsByTagName("img") as it returns a collection of elements, even if it is a collection of only zero or one items.

Tip: If at a later time any new images are added above this image, none of the choice above will return the image we wanted. For best results, the image should have an ID. <img **id="logo"** src="TeamLogo.png">

❑ Which of the following will retrieve just the first heading element (<h1>)?

* **var firstH1 = document.getElementById("AboutUs").getElementsByTagName("h1")[0]**
* var firstH1 = document.getElementsByTagName("h1")
* **var firstH1 = document.getElementsByTagName("h1")[0]**
* var firstH1 = document.getElementsByClassName("bluetext")
* var allH1s = document.getElementsByTagName("h1")  
  var firstH1 = allH1s[0]
* **var allH1s = document.getElementsByTagName("h1")  
  var firstH1;  
  for (var i=0; i<allH1s.length; i++) {   
   if (allH1s[i].textContent == "This is the Heading") {   
   firstH1 = allH1s[i];   
   break;   
   }  
  }**

You probably don’t want to use the for-loop version!

❑ Considering only the correct answers from the question above, which of the following is the most reliable? I.e. would survive edits to the content.

* var firstH1 = document.getElementById("h1")
* **var firstH1 = document.getElementById("AboutUs").getElementsByTagName("h1")[0]**
* var firstH1 = document.getElementsByTagName("h1")
* var firstH1 = document.getElementsByTagName("h1")[0]
* var firstH1 = document.getElementsByClassName("bluetext")
* var allH1s = document.getElementsByTagName("h1")  
  var firstH1 = allH1s[0]
* var allH1s = document.getElementsByTagName("h1")  
  var firstH1;  
  for (var i=0; i<allH1s.length; i++) {   
   if (allH1s[i].textContent == "This is the Heading") {   
   firstH1 = allH1s[i];   
   break;   
   }  
  }

Best choice: document.getElementById("AboutUs").getElementsByTagName("h1")[0] as this will only return H1s in the DIV with ID of AboutUs.

document.getElementsByTagName("h1")[0] would return the wrong H1 if a new section/DIV was added or the order of the DIVs was changed.

The for-loop would fail if the text in the here was ever changed.

❑ Which of the following will retrieve all of the H2 elements in the first DIV?

* var H2tags = document.getElementsByTagName("h2")
* **var H2tags = document.getElementsByTagName("div")[0].getElementsByTagName("h2")**
* **var H2tags = document.getElementById("AboutUs").getElementsByTagName("h2")**
* var H2tags = document.querySelector ("h2")
* var H2tags = document.querySelector ("AboutUs h2")
* var H2tags = document.querySelector ("#AboutUs h2")
* var H2tags = document.querySelector (".AboutUs h2")
* var H2tags = document.querySelectorAll ("AboutUs h2")
* **var H2tags = document.querySelectorAll ("#AboutUs h2")**
* var H2tags = document.querySelectorAll (".AboutUs h2")

The .getElementById uniquely selects a DIV and will still work if new DIVs are added to the page or the existing DIVs get rearranged. The document.getElementById("AboutUs") returns a single DIV element and the .getElementsByTagName selects from that’s DIV’s child elements.

The “.querySelector” options only return a single item, so will not return all of the H2 elements.

The .querySelectorAll ("#AboutUs h2") works as the CSS selector for IDs uses “#” and the “All” version returns all of the matching elements.

❑ Considering only the correct answers from the question above, which of the following is the most reliable? I.e. would survive edits to the content.

* var H2tags = document.getElementsByTagName("h2")
* var H2tags = document.getElementsByTagName("div")[0].getElementsByTagName("h2")
* **var H2tags = document.getElementById("AboutUs").getElementsByTagName("h2")**
* var H2tags = document.querySelector("h2")
* var H2tags = document.querySelector("AboutUs h2")
* var H2tags = document.querySelector("#AboutUs h2")
* var H2tags = document.querySelector(".AboutUs h2")
* var H2tags = document.querySelectorAll("AboutUs h2")
* **var H2tags = document.querySelectorAll("#AboutUs h2")**
* var H2tags = document.querySelectorAll(".AboutUs h2")

When available, selecting by ID is always the most precise way to select elements.

### Hands on Lab

#### Exercise 1 Code:

/\*

Author: Mike

Date: 3/14/17

Used by: home.html

Notes:

Creates a new <style> block to slowly move the image off of the page.

Add an event handler to hide the image after the animation.

\*/

var styletext = "\

@keyframes flyaway { \

0% { \

transition: 5s ease-in; \

} \

100% { \

transform: translate(350px, 100px); \

} \

} \

\

#airplane1 { \

animation: flyaway 10s ease-in forwards; \

-moz-animation-iteration-count: 1; \

animation-iteration-count: 1; \

} \

"

var newStyleBlock = document.createElement("style");

newStyleBlock.textContent = styletext;

document.head.appendChild(newStyleBlock);

var airplane = document.getElementById("airplane1");

airplane.addEventListener("animationend", function() { airplane.style.display="none" }, false);

#### Exercise 3 Code:

// Create a new div element that will be used as the popup.

var popdiv = document.createElement("div");

popdiv.id = "menumessage";

popdiv.textContent = "default text for testing";

popdiv.style.color = "orange";

popdiv.style.border = "1px solid white";

popdiv.style.borderRadius = "10px";

popdiv.style.backgroundColor = "black";

popdiv.style.position = "absolute";

popdiv.style.padding = "10px";

popdiv.style.display = "none";

// popdiv.style.transformOrigin ="0 0";

// popdiv.style.transform = "rotate(15deg)";

document.body.appendChild(popdiv);

//var topnav = document.getElementById("topNav");

//var tabs = topnav.getElementsByTagName("a");

// alternate to the previous two lines

var tabs = document.querySelectorAll("#topNav a");

for (var index = 0; index < tabs.length; index++) {

var tab = tabs[index];

tab.onmouseenter=function() {ShowPopup(this)}

tab.onmouseleave=function() {HidePopup()}

}

function ShowPopup(tab)

{

// get the popup div and position it

var popdiv = document.getElementById("menumessage");

popdiv.style.left = "30px";

popdiv.style.top = "110px";

popdiv.style.left = tab.offsetLeft + "px";

popdiv.style.top = (tab.offsetTop + 30) + "px";

// select the text to display

var popuptext = "";

switch (tab.textContent) {

case "Home":

popuptext = "Return to the home page.";

break;

case "Toys":

popuptext = "See our latest toys, including boats, planes and cars!";

break;

case "Sports":

popuptext = "See our new line of sporting goods!";

break;

default:

popuptext = "";

break;

}

// set the text and display the popup

if (popuptext != "")

{

popdiv.innerHTML = popuptext;

popdiv.style.display = "block";

}

}

function HidePopup()

{

// hide the popup

var popdiv = document.getElementById("menumessage");

popdiv.style.display = "none";

}

## Module 3 Lab – jQuery

### Questions

❑ jQuery:

* is built into modern browsers
* is built into Firefox, but must be downloaded for all other browsers
* may be downloaded to a server and then linked by the page
* may be linked from public CDNs by the page

❑ jQuery can be accessed by using:

* the jQuery function name ( jQuery("div") )
* the “$” alias ( $("div") )
* any alias you choose ( x("div") )

❑ jQuery is written in:

* C#
* Java
* JavaScript

❑ To select elements jQuery uses:

* CSS selectors, but limited to the version of CSS supported by the browser
* CSS-like selectors independent of the browser version
* A proprietary query language

❑ jQuery can do things that cannot be done with JavaScript:

* True
* False

❑ jQuery “queries” return: (Example: $("table") )

* Null if no items found, a single item, or a collection of items.
* Null if no items found or a collection of items.
* A collection of zero or more items.
* A single item or a collection of items.

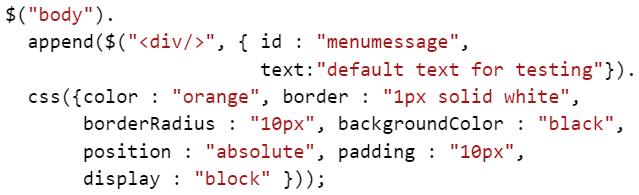
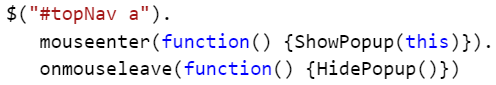
### Hands on Lab

In this lab we will update the Module 2 lab to use jQuery to simplify the JavaScript routines.

#### Exercise 1 – Add the jQuery Library to the Project

1. Open MenuPopups.js in your HTML editor. If using a folder oriented editor, open the C:\JavaScript\Module3\Lab3 folder, then open the MenuPopups.js file.
2. Note the JavaScript used to create the <div> used for the popup. Using the editor’s shortcuts, if available, comment the entire block of code used to create the <div>. (Starting with “var …” and ending with “document…”.)
3. Before we can use jQuery we need to load the jQuery library.
   1. Open the default.html page.
   2. In the blank line in the <head> section add code to load the jQuery library. This example will load the library from the jQuery CDN site.  
      
4. Save the default.html file and return to the MenuPopups.js file.

#### Exercise 2 – Create New Elements using jQuery

1. jQuery provides a number of ways of creating a new element and setting it’s properties. Choose one of the following and add it below the code you just commented out. (“b” is the most “self documenting”)
   1. A one line version that builds the <div> from a string. (ugly!)  
      var popdiv = $("body").append('<div id = "menumessage" style="color:orange; border:1px solid white; border-radius:10px; background-color:black; position:absolute; padding:10px; display:none;">default text for testing</div>');
   2. A version that uses $(<div/>, object) to create a new <div> and a JavaScript object to create a style “object”. This version is very object oriented and each property, such as “orange”, can be replaced with a variable.  
      See: http://api.jquery.com/jQuery/#jQuery2  
       
   3. A one line version of “b.”  
       
2. Find the line that starts with “var tabs” and comment out that line and FOR loop that follows.
3. Write jQuery that selects all of the <a> elements in the <nav> element (ID=topNav). (This can be written as a single line!)  
    
4. Save the file and test the page.

## Module 3 Lab – Answers

### Questions

❑ jQuery:

* is built into modern browsers
* is built into Firefox, but must be downloaded for all other browsers
* **may be downloaded to a server and then linked by the page**
* **may be linked from public CDNs by the page**

❑ jQuery can be accessed by using:

* **the jQuery function name ( jQuery("div") )**
* **the “$” alias ( $("div") )**
* **any alias you choose ( x("div") )**

❑ jQuery is written in:

* C#
* Java
* **JavaScript**

❑ To select elements jQuery uses:

* CSS selectors, but limited to the version of CSS supported by the browser
* **CSS-like selectors independent of the browser version**
* A proprietary query language

❑ jQuery can do things that cannot be done with JavaScript:

* True
* **False** *(jQuery is written in JavaScript!)*

❑ jQuery “queries” return: (Example: $("table") )

* Null if no items found, a single item, or a collection of items.
* Null if no items found or a collection of items.
* **A collection of zero or more items.**
* A single item or a collection of items.

## Module 4 Lab - Ajax and Web Services

### Questions

❑ Ajax is available:

* Only in Internet Explorer 5.0 and later.
* Only as part of jQuery.
* With most modern browsers.

❑ Common formats for web service data include:

* Binary
* Text
* XML
* HTML
* Comma Separated Values (CSV)
* JSON

❑ Which of the following are HTTP verbs:

* ACT
* GET
* POST
* SEND
* PUT
* DELETE

❑ The key feature of REST is:

* Returns both XML and JSON
* The URL points to a resource, not a page
* Supports queries
* Is the most common web service for free services

### Hands on Lab

#### Exercise 1 – Exploring Data Files

This exercise is just a brief review of data formats.

1. Open Notepad.
2. Open the sample CSV file in the C:\JavaScript\Module 4\Demos\Bikes.csv

Although CSV files are a common way to exchange data, they are not typically used with web services. They are convenient in that they are supported by a large number of tools like Excel. They are not useful when you have multiple tiers, or a hierarchy of data.

1. From Notepad, open C:\JavaScript\Module 4\Demos\Bikes.xml
2. Note the multiple tiers of data. The first Bike object has child details with inventory information. This could not be cleanly represented in a CSV file. It also better represents the hierarchical design of relational databases.
3. Browsers are often a handy tool for browsing XML files.
   1. Open Internet Explorer or Firefox and navigate to C:\JavaScript\Module 4\Demos\Bikes.xml either from the File, Open menu or by directly typing in the browser’s address bar.
   2. Click the dash (“-“) beside on of the elements and note that you can expand and collapse sections.
4. There are numerous online and offline xml viewers and explorers.
   1. Return to Notepad and copy all of the XML content.
   2. Open a browser and navigate to http://www.xmlviewer.org/ (or do a web search for “online XML viewer” or “online XML explorer”. The steps that follow are using www.xmlviewer.org.)
   3. Paste the XML into the XML panel.
   4. Click the Tree View button and explore the data.
   5. Click the Minify button to compact the XML by removing all excess spaces and carriage returns.

Today, JSON is becoming the preferred data format for web services. It’s lighter weight and very easy to use in a JavaScript application.

1. From Notepad, open C:\JavaScript\Module 4\Demos\Bikes.json
2. Note that in place of the start and end tags found in XML files, JSON uses curly brackets to define objects and square brackets for collections of objects. Also note that JSON also can represent hierarchical data.
3. While Internet Explorer 11 does not have a built-in JSON browser, other browsers like Firefox do.
   1. Open Firefox and navigate to C:\JavaScript\Module 4\Demos\Bikes.json either from the File, Open File menu or by directly typing in the browser’s address bar.
   2. Click the dropdown arrows beside the items and note that you can expand and collapse sections.

#### Exercise 2 – Exploring a REST Web Service

You have been provided with the URL to a REST web service so you can access data stored on a remote system. You have not been supplied with any details about the data.

Ask your instructor for the URL to the REST web service. It may look like one of the following:

* http://localhost:6307/odata
* http://ClassroomServer/odata
* http://restdemo.azurewebsites.net/odata

The steps below will reference a fictitious server with this URL: http://demoserver/odata/Bikes

**Ask the service for details**

1. Some REST web services can tell you about the services available at a URL.
   1. Open a web browser and enter the URL to the service and add “/$metadata”.  
       http://demoserver/odata/$metadata   
      Note: The text after “/odata/” is case sensitive!



1. Note that there are two schemas defined: Bike and Bikes. (there may be more in the example site…)
   1. Bike defines a single object while Bikes simply defines the collection of objects.
   2. Note that the Bike object has two string and three numeric properties and that the key for object lookup is “id”.
2. GET requests can be tested from any browser by just typing the URL into the address bar. Some browsers will display the results while others will offer to download the results. Try typing the following URLs into the browser’s address bar. Note: *The text after “/odata/” is case sensitive!*
   1. http://demoserver/odata/Bikes
   2. http://demoserver/odata/Bikes(800) (*select the bike with ID 800*)
   3. http://demoserver/odata/Bikes?$filter=color eq 'Red'
3. In the browser’s address bar enter this path: (May be different on your PC. Check with your instructor.)  
    C:\JavaScript\WebServiceTester.html   
   Note: If there are issues using this page, you can also use the same page on the demo web site:  
    http://demoserver/webservicetester.html (Remember to use your REST server’s URL!)
   1. This web page uses jQuery and Ajax to send requests to a web server.
   2. For GET and DELETE queries you will only need to enter a URL and set the HTTP Verb.
   3. For POST, PUT and PATCH you will also need to enter Data (JSON for our examples) to send with the request.
4. Enter the URL for the Bikes web service, set the dropdown to GET and click SEND. You should get back JSON code describing nearly 100 bikes.  
    http://demoserver/odata/Bikes
5. REST with ODATA lets you query data using:
   1. $select – specifies the properties to return
   2. $filter – specifies the objects to return
   3. $orderby – specifies the sort order
   4. $top – sets the number of items to return
   5. $skip – skips the first “n” records (used with $top for paging)
6. Create a query that returns only the bike id and price.  
    \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
7. Create a query that returns only bikes that cost less than $1,000.  
    \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
8. Create a query that returns only blue mountain bikes. (search text is also case sensitive!0  
    \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

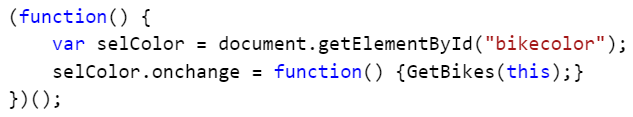
#### Exercise 3 – Load Data from a Web Service

Notes:

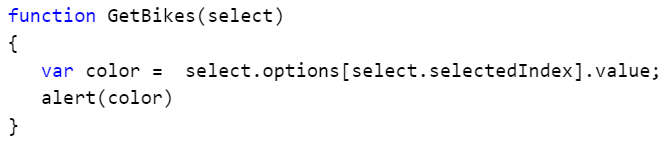
* As you are running these lab files from a local disk instead of a web server, you may find that Firefox and Chrome will report errors accessing a “cross domain” remote server. Use Internet Explorer, or use the “If your lab environment does not have access to the internet” option in the steps below.

1. Open your HTML editor.
2. If your editor is project or folder oriented, open the C:\JavaScript\Module4\Lab4 folder.
3. Open the Sports.html file from the C:\JavaScript\Module4\Lab4 folder.
4. Find the following in the page:
   1. The <script> block that loads the jQuery library. (This will be used for the Ajax web service call.)
   2. The <table> tag with the ID of “products”.
   3. The <select> dropdown list.
   4. The empty (comments only) <script> block.
   5. The empty (comments only) <style> block.

**Add an event handler to the <select> element.**

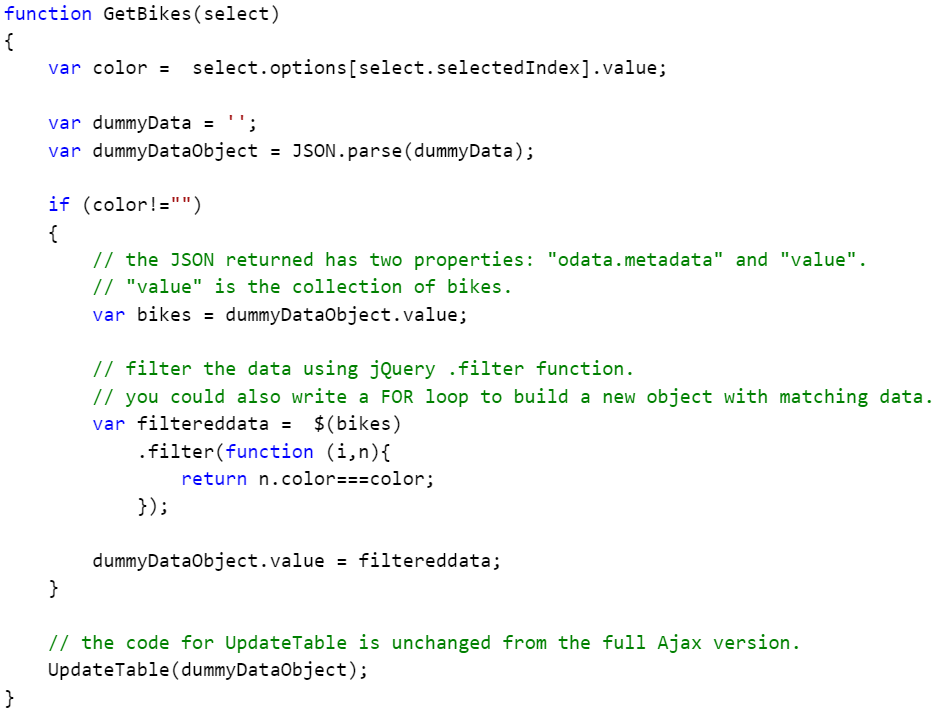
1. There are several ways to add an event handler to the <select> element:
   1. Add the code directly to the <select> tag:  
       <select id="bikecolor" onchange="GetBikes(this);">
   2. Add the code dynamically setting the element’s onchange property:  
       var selColor = document.getElementById("bikecolor");  
       selColor.onchange = function() {GetBikes(this);}
   3. Add the code dynamically setting the element using addEventListener:  
       var selColor = document.getElementById("bikecolor");  
       selColor. addEventListener("onchange", function() { GetBikes(this) }, false);
2. In all of the above examples, the “this” keyword refers to the current element, in this case the <select> element.
3. For this lab, let’s use option “b”. You may want to wrap the code in an anonymous, self-calling, function to avoid the creation of a Global variable. Add the code just below the first comment in the <script> block.  
   

**Write the event handler function (GetBikes).**

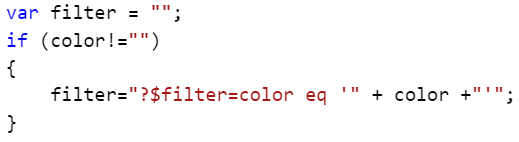
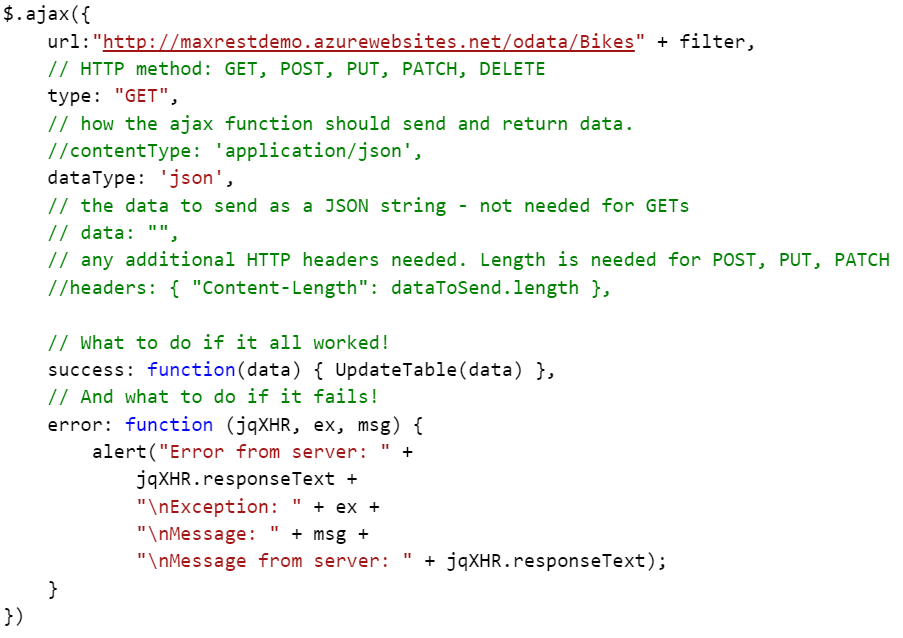
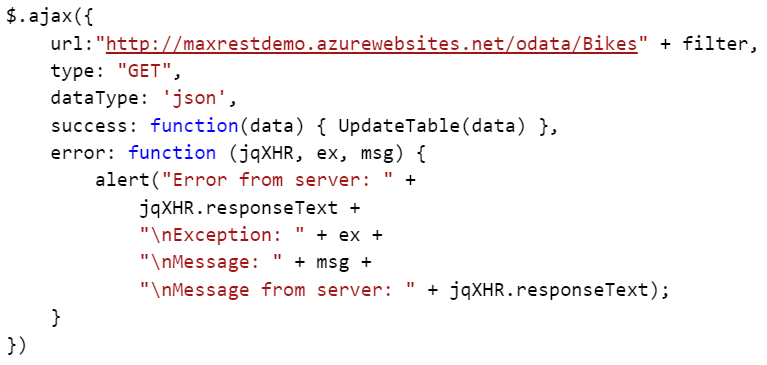
1. Start by writing a function that retrieves the <select> element’s current value. Something like this:  
    
2. Open the page in a browser, change the select dropdown and verify the correct value has been selected.
3. Delete the “alert(color)” line.

**If your lab environment does not have access to the internet, or to a local web service with lab data**, you can emulate the Ajax call using the following replacement for the GetBikes function.

***Alternate steps for the GetBikes() function:***

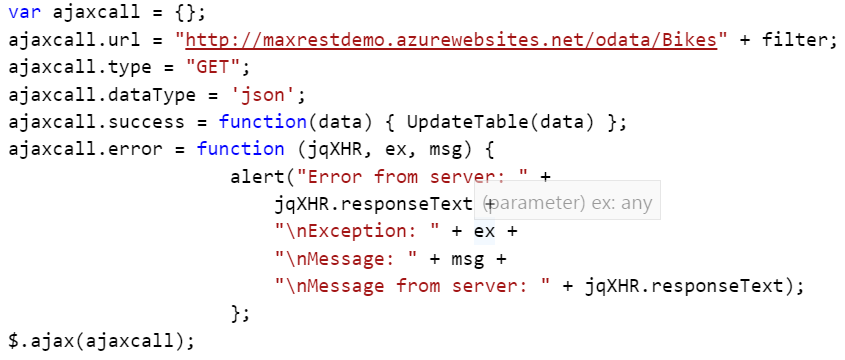
1. Write the following code:  
   
2. Open the LabData.json file using Notepad or other text editor.
3. Copy all of the text. (One long line!)
4. Paste the JSON text between the “ var dummyData = '' ” quotes:  
    
5. Skip the following section and continue with “Write the function to process the data back from the web service” steps. (Step 13)

**Build the query string for the search.**

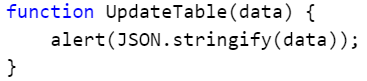
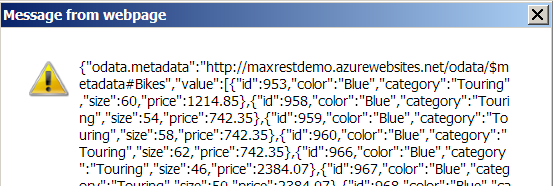
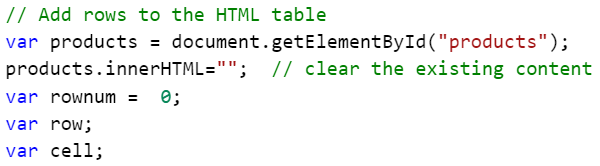
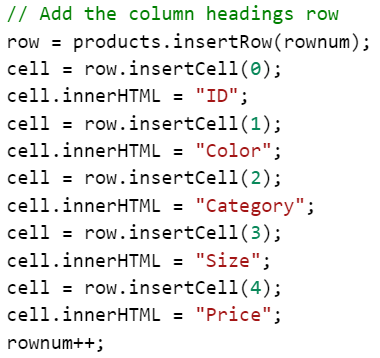
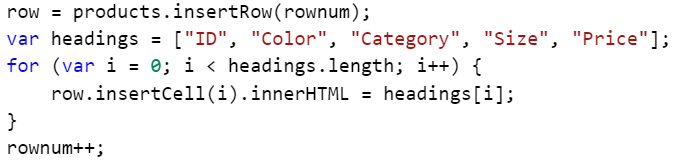
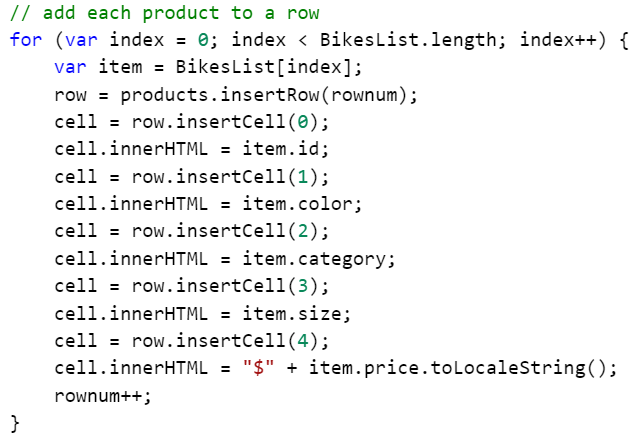
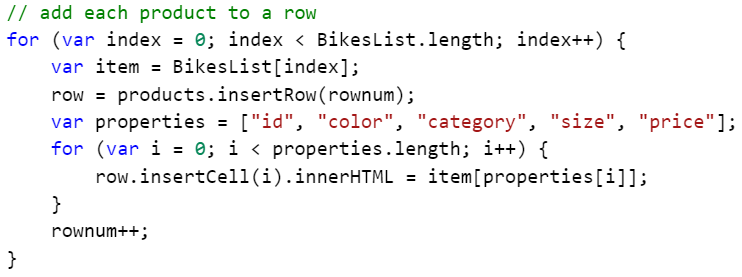
1. Use the returned color from the <select> to build an odata query string. The result for the filter will look something like this:  
    $filter=color eq 'Red'  
   Add the “?” for the query string and write the code to build the string:  
    
2. Now to write the Ajax call using jQuery’s $.ajax() function. (Remember to use your REST server’s URL!)  
      
     
   Or without the comments: (Remember to use your REST server’s URL!)  
    

Note:

As the curly brackets define a JavaScript object, the above could also have been written as:  
 

Or as:  
 

**Write the function to process the data back from the web service**

1. In order to quickly test the Ajax code above, we will write a simple function to just display the returned data.  
    
2. Save the file and test the page in a browser. You should see something like this:  
    
3. Take a look at the returned JSON. The object returned has two properties: “odata.metadata” (which we don’t care about) and “value” which is a collection ( [ … ] ) of bikes. You may also want to note the bike properties available: id, color, category, size and price.
4. Delete, or comment out, the “alert()” line from the function.
5. Retrieve the collection of bikes from the “value” property of the data object returned from the web service call.  
    
6. As you saw earlier, there’s an empty <table> object in the page. Retrieve the <table> element, remove any existing contents, and create variables that will be used to add content to the table:  
    
7. There are numerous ways of adding rows to a table. In this example we will use the table object and it’s row and cell properties. Let’s start by adding a heading row. (Feel free to change the heading text, but let’s wait and use CSS to style the heading row.)  
   You could write the code below…  
      
   But this will be cleaner and easier to update in the future.  
    
8. Following a similar pattern, you can add the data rows while looking through the collection of Bikes.   
   You can either add the columns manually:  
      
   or you can write a loop to add the columns:  
    
9. Save your work and test the page. Test each color. (There are no Green bikes!)

## Module 4 Lab – Answers

### Questions

❑ Ajax is available:

* Only in Internet Explorer 5.0 and later.
* Only as part of jQuery.
* **With most modern browsers.**

❑ Common formats for web service data include:

* Binary
* Text
* **XML**
* HTML
* Comma Separated Values (CSV)
* **JSON**

While web services can return any kind of data, XML and JSON are the two most common formats.

❑ Which of the following are HTTP verbs:

* ACT
* **GET**
* **POST**
* SEND
* **PUT**
* **DELETE**

❑ The key feature of REST is:

* Returns both XML and JSON
* **The URL points to a resource, not a page**
* Supports queries
* Is the most common web service for free services

While REST often supports queries, queries are usually part of REST + odata.

## Module 5 Lab – Diving in Deeper

### Questions

❑ JavaScript is compiled:

* Never
* Just in time (JIT) as the code is loaded
* Manually using the JavaScript.exe compiler.
* Automatically using the JavaScript.exe compiler.

❑ Which of the following will create an array?

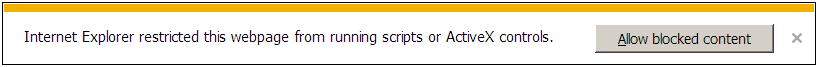
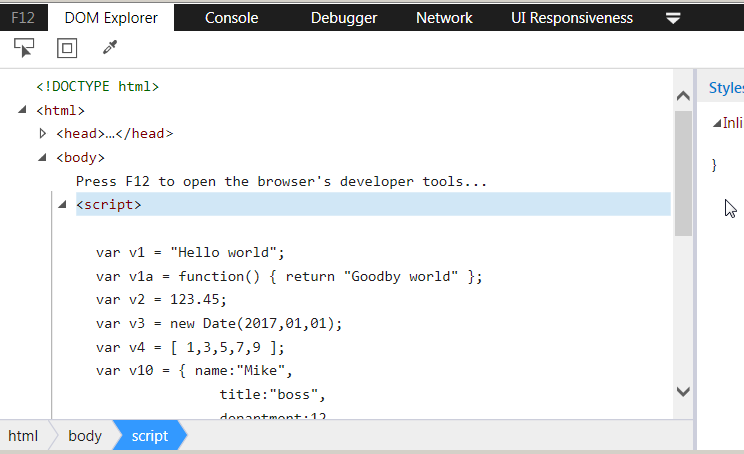
* var x = "abc", "def", "xyz"
* var x = "red, green, blue".split(",")
* var x = ["abc", "def", "xyz"]
* var x = []; a.push("Monday"), a.push("Tuesday")
* var x = new Array("Mon", "Tue", "Wed")
* var x = []
* var x = new Array()
* var x = document.getElementById("abc")
* var x = document.getElementsByTagName("div")

❑ You have a string like: "Sam said hello to Susan".  
Which of the following lets you embed double quotes around “hello”?

* "Sam said ""hello"" to Susan"
* "Sam said \"hello\" to Susan"
* "Sam said "hello" to Susan"
* 'Sam said "hello" to Susan'
* "Sam said 'hello' to Susan"
* "Sam said \qhello\q to Susan"

### Hands on Lab

#### Exercise 1 – Exploring Types and Objects

1. Open a browser and navigate to C:\JavaScript\Module 5\Lab1\objects.html. If you get a warning about running JavaScript, click Allow blocked content.  
    
2. Press F12 to open the developer’s tools.
3. Expand the script block and review the variables being created in the script.  
    
4. Click the Console tab of the F12 tools pane.

**Explore data types:**

1. At the console prompt, explore the data types of each of the variables. Type these commands and record the types:
   1. typeof v0 \_\_\_\_\_\_\_\_\_\_\_\_\_\_ ( var v0; )
   2. typeof v1 \_\_\_\_\_\_\_\_\_\_\_\_\_\_ ( var v1 = "Hello world"; )
   3. typeof v2 \_\_\_\_\_\_\_\_\_\_\_\_\_\_ ( var v2 = "123.45"; )
   4. typeof v3 \_\_\_\_\_\_\_\_\_\_\_\_\_\_ ( var v3 = 123.45; )
   5. typeof v4 \_\_\_\_\_\_\_\_\_\_\_\_\_\_ ( var v4 = v2 + 5; )
   6. typeof v6 \_\_\_\_\_\_\_\_\_\_\_\_\_\_ ( var v6 = new Date(2017,01,01); )
   7. typeof v7 \_\_\_\_\_\_\_\_\_\_\_\_\_\_ ( var v7 = [ 1,3,5,7,9 ] )
   8. typeof v8 \_\_\_\_\_\_\_\_\_\_\_\_\_\_ ( var v8 = function() { return "Goodby world" }; )
   9. typeof v9 \_\_\_\_\_\_\_\_\_\_\_\_\_\_ ( var v9 = null; )

Note that both v6 and v7 report a type of “object”. How could you tell if v6 is a date? (See step 7 below!)

**Explore the Boolean type:**

You often write logical tests using “if” and other statements. Exactly what is a logical “true” may not always be obvious.

Examples of logical tests:

var x = 5;

if ( x > 0 ) { … }

while ( x < 10 ) { … }

var y = ( x > 10 ) ? "x is big" : "x is small"

Examples of a few logical tests that are valid, but may not produce the expected result:

var x = 5;

if ( x ) { … }

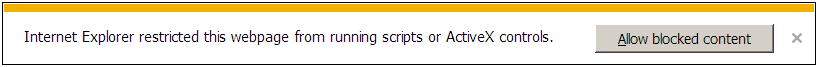
if ( window.alert ) { … }

The Boolean() function returns true if a value “exists”. Any object of which the value is not undefined or null, will return true. Logical tests like “if (window.alert)” test to see if the function “alert” exists on the “window” object. You can think of logical tests with only one expression, like “if ( 5 )”, to be the equivalent of “if ( Boolean(5) )”.

1. Try these experiments at the Console prompt:
   1. Boolean( v0 ) \_\_\_\_\_\_\_\_\_\_\_\_\_\_ ( v0 is undefined )
   2. Boolean( v3 ) \_\_\_\_\_\_\_\_\_\_\_\_\_\_ ( v3 is a number )
   3. Boolean( v20 ) \_\_\_\_\_\_\_\_\_\_\_\_\_\_ ( v20 is a divide by zero… i.e. Infinity)
   4. Boolean ( v10 ) \_\_\_\_\_\_\_\_\_\_\_\_\_\_ ( v11 is Boolean set to true )
   5. Boolean ( v11 ) \_\_\_\_\_\_\_\_\_\_\_\_\_\_ ( v11 is Boolean set to false )
   6. Boolean ( DoubleIt ) \_\_\_\_\_\_\_\_\_\_\_\_\_\_ ( DoubleIt is a function )
   7. Boolean ( DoubleIt(2) ) \_\_\_\_\_\_\_\_\_\_\_\_\_\_ ( DoubleIt(2) is a call to a function )
   8. Boolean ( DoNothing ) \_\_\_\_\_\_\_\_\_\_\_\_\_\_ ( DoNothing is a function )
   9. Boolean ( DoNothing(2) ) \_\_\_\_\_\_\_\_\_\_\_\_\_\_ ( DoNothing(2) is a call to a function )
   10. What’s different about “DoNothing(2)”? \_\_\_\_\_\_\_\_\_\_\_\_\_
2. Use Boolean (or and “if” statement) to see if the v6 variable is a date. Tip: date objects have a “GetDate” function.
   1. Solution: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
3. Use the same solution to see if v7 is a date.
   1. Solution: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
4. Which property could you test to see if v7 is an array? Tip: In the Console type “v7” followed by a dot and look to see if there is a property or function in the popup list that might be unique to an array.
5. Confirm that the “window” object has an “alert” function.
   1. Solution: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
6. Does the “window” object have an “popup” function?
   1. Yes or no? \_\_\_\_\_
7. Which of the following will test to see if the “v100” variable has a “title” property?
   1. if (v100.title) { alert("Title is " + v100.title) } else { alert("no title!") }
   2. if ( Boolean(v100.title) ) { alert("Title is " + v100.title) } else { alert("no title!") }
   3. if ( v100.title != undefined ) { alert("Title is " + v100.title) } else { alert("no title!") }
   4. if ( (typeof v100.title) == "string" ) { alert("Title is " + v100.title) } else { alert("no title!") }

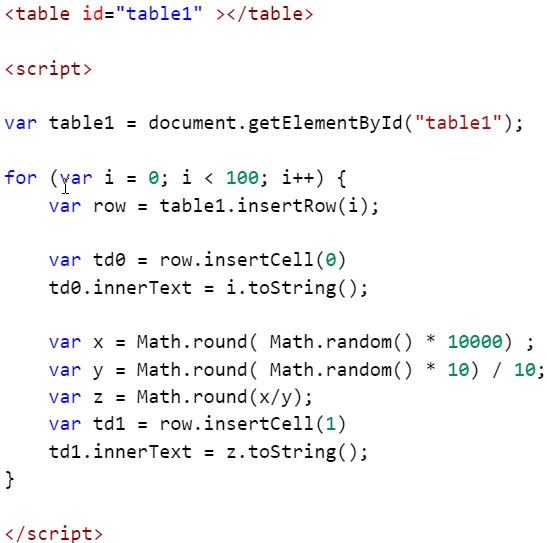
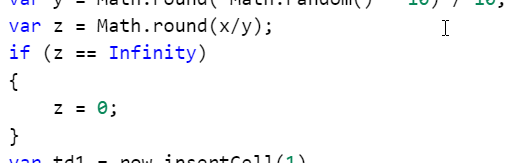
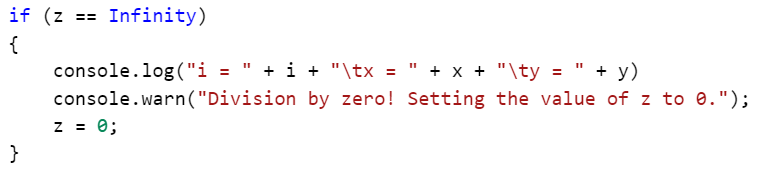
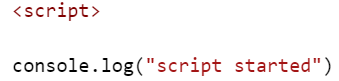
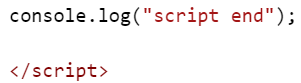
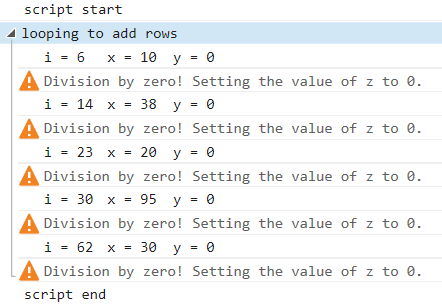
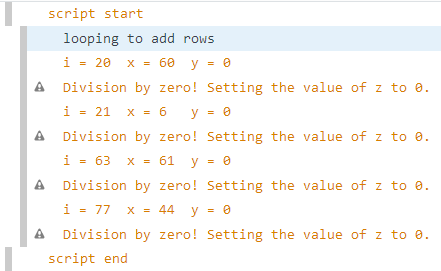
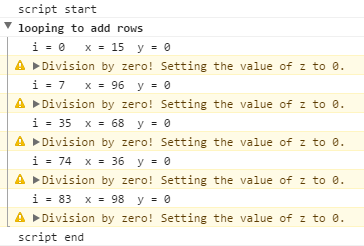
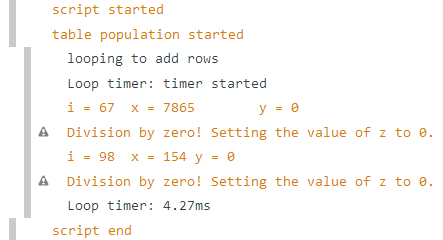
#### Exercise 2 - Exploring Default Objects

JavaScript includes a number of “out of the box” objects like “window” and “Math”. Some of these objects are more dynamic than you might expect.

1. If the “objects.html” file is not still open:
   1. Open a browser and navigate to C:\JavaScript\Module 5\Lab1\objects.html. If you get a warning about running JavaScript, click Allow blocked content.  
       
   2. Press F12 to open the developer’s tools.
2. At the Console prompt type the following commands and note their results.
   1. alert("hello") - displays a popup message
   2. window.alert("hello") - displays a popup message (“alert” is a property of “window” and as “window” is the default object, you don’t have to type it.
   3. window.alert - displays a message that shows that “alert” is a function.
3. At the Console type:
   1. var alert = 5;
4. Now try the same three examples in step 13 above. (You just broke the browser!)
   1. Actually, you have just replaced the top level function named “alert” with a new top level variable named “alert”.
5. How can you get the original definition of “alert” back?
   1. Tip: Refresh the browser. Then, test it by typing “alert("hello").
   2. Big Tip: Never use any of the default properties of the window object in your code!
   3. Bigger Tip: Avoid creating, directly or through hoisting, top level variables when possible.
6. Another developer, no longer at the company, created a library with hundreds of alert popups! You would like to fix this, but don’t have to time to edit the entire library. You can replace the built-in alert function with one of your own. Let’s test this at the console:
   1. At the Console prompt, save the current definition of “alert” to a variable.   
      var oldalert = window.alert (or for clarity: window.oldalert = window.alert )
   2. Define the new way of doing alerts:
      1. Display the message in the status bar:  
         window.alert = function(text) { window.status = text }
      2. Or as often done by accident:  
         var alert = function(text) { window.status = text }
      3. or maybe just do nothing:  
         window.alert = function() {}
   3. Test the new alert: (*you may need to turn on the browser’s status bar*)  
      alert("this is the new alert")
   4. Restore the default alert:  
      window.alert = window.oldalert (or just alert = oldalert)
   5. Check to see if the annoying old alert still works:  
      alert("this is the old alert") (you should see the popup again)
7. And just for fun: A black hole has just passed near the Earth and has temporarily distorted space and time resulting in a new value for PI. Your code now uses Math.PI in a lot of locations.
   1. Test the default value of PI at the Console prompt:  
      Math.PI
   2. Change the value of PI:  
      Math.PI = 3
   3. Test the new value of PI:  
      Math.PI
   4. Turns out that the Math object is not as dynamic as the window object!

#### Exercise 3 – Debugging using the “console” Object

You can debug JavaScript using an editor’s debugger or the browser’s debugger features. You can add breakpoints and single step through code. Here we will look at outputting messages to the browser’s F12 tools Console. This can be useful when you want to be able see status and debugging messages without the debugging tools. As an example, you could ask an end user to visit a page they are having problems with and simply press F12, click the Console tab and read back any messages.

1. Open ConsoleObject.html in your HTML editor. If using a folder oriented editor, open the C:\JavaScript\Module5\Lab3 folder, then open the ConsoleObject.html file.
2. Review the code… it generates a table from random numbers.  
    
3. Open the file in a browser: C:\JavaScript\Module5\Lab3\ConsoleObject.html
4. Review the generated data.
5. While the generation of a value that causes a divide by zero is a normal, but somewhat rare, occurrence in this business process, the users don’t like to see the “Infinity” text being displayed. They would rather see a zero instead.
   1. Add an “if” block to replace “Infinity” with a zero. Maybe something like this:  
       
6. Save your work and refresh the page a few times. You should no longer see “Infinity” in the results.
7. Occasionally someone questions the output. You can write messages to the Console for those rare times when you need to know if the zero was just a zero, or the alternate way they requested to display “Infinity”.
   1. Inside of the “if” statement, add a “console.warn” to report the replacement of “Infinity”. The “warn” method will display an icon next to the output. Maybe something like:  
       
   2. Note: the “\t” inserts a tab into the output text.
8. Save your work and refresh the page. Scroll through the results to see if there are any zeros. Count the number of zeros.
9. Press F12, if the F12 tools window is not open, and click the Console tab.
   1. Any messages about Division by Zero? \_\_\_\_\_
   2. Do any of the zeros in the results not display Console messages? \_\_\_\_\_\_
10. As you may have a number of scripts that output messages to the Console, you may want to write a few additional notes to make the Console output easier to understand.
    1. Just after the starting <script> tag, write a message to the Console:  
        
    2. Just before the ending <script> tag, write another message to the Console:  
        
    3. Note that “console.log” only outputs text. No special icons are displayed.
11. Save your work and refresh the page. Note the new messages in the Console.
12. If you have many sections of code that output messages, it might be nice if they are grouped in some way.
    1. Just before the “for” loop, add a “console.group()”. While optional, the group can have a name to better identify the group.  
        
    2. Just after the closing bracket of the “for” loop, add a “console.groupEnd()”.  
        
13. Save your work and refresh the page. Note the new message structure in the Console. Each browser will format the Console differently. Chrome and Internet Explorer add triangles to expand and collapse groups.  
    Internet Explorer:  
       
       
    Firefox:  
       
       
    Chrome:  
     
14. Performance testing of code is easily done by adding console.time() and console.timeEnd() around the code to test. We need to determine if the creation of the table is a performance bottleneck in our project.
    1. Just before the “for” loop add a “console.time()” with a name of “Loop timer”.  
        
    2. Just after the closing bracket of the “for” loop, add a “console.timeEnd()” with a name of “Loop timer”.  
        
15. Save your work and refresh the page.  
     
16. Refresh the page a few times and note that the times will vary due to other processes and programs running at the same time.
17. Open this page in each browser on your lab computer and see if any one browser is much slower, or much faster, than the other browsers.

#### Exercise 4 - Working with Scope

1. Open Scopes.html in your HTML editor. If using a folder oriented editor, open the C:\JavaScript\Module5\Lab3 folder, then open the Scopes.html file.
2. Review the code… we will be uncommenting one demo at a time, and then fixing any issues found.
   1. Tips for commenting and uncommenting blocks:
      1. Visual Code: Ctrl + / (adds and removes comments)
      2. Visual Studio:
         1. Comment: Ctrl K followed by Ctrl C
         2. Uncomment: Ctrl K followed by Ctrl U
         3. Or use the buttons in the ribbon. 
3. Open ConsoleObject.html in your browser. Press F12 and click the Console tab. You should just see the console.log() output for the demo headings.
4. Uncomment the code for “demo 1” and save the file. Refresh the browser and note the messages. The author of the code expected the final value of the counter variable to be 12. Which of the following changes will ensure that the value of the counter variable will be 12 at the end of the code? You may want to test each change…
   1. [\_\_] Add a “var” in the for loop to keep that counter variable to the loop.  
       
   2. [\_\_] Add a “let” in the for loop to keep that counter variable to the loop.  
       
   3. [\_\_] Rename the variable in the for loop.  
       
5. Uncomment the code for “demo 2” and save the file. Refresh the browser and note the messages. The author of the code expected the final values x and y to not change. Which of the following changes will ensure that the value of the counter variable will be 12 at the end of the code? You may want to test each change…
   1. [\_\_] Add a “var” in front of the first use in the function of the y variable.  
       
   2. [\_\_] Add a “let” in front of the first use in the function of the y variable.  
       
   3. [\_\_] Rename the y variable in the function.  
       
6. Uncomment the code for “demo 3” and save the file. Refresh the browser and note the messages. The author of the code expected the code in the inner for loop to execute 25 times and therefore call TestFunction02 25 times. This code has multiple issues. Correct the code so the TotalLoops variable equals 25 at the end and the TotalValue variable equals 250.
   1. List your changes here:

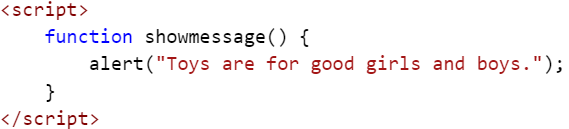
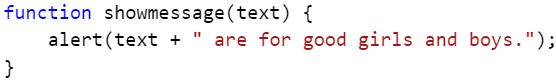
#### Exercise 5 – Functions and Event Handlers

In this exercise, we will be exploring event handlers. The sample page has a collection of buttons that need to respond to user clicks.

**Add inline code:**

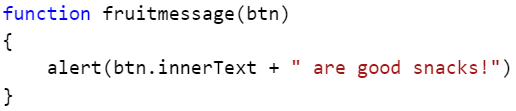
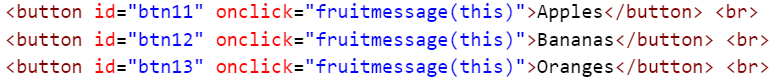
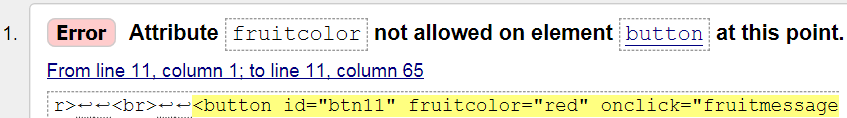
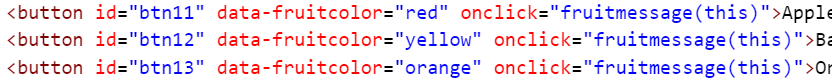
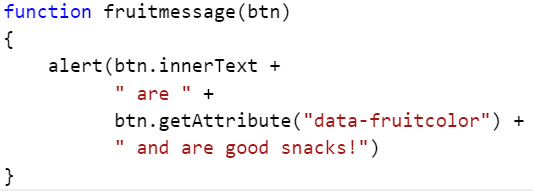
1. Open Events.html in your HTML editor. If using a folder oriented editor, open the C:\JavaScript\Module5\Lab3 folder, then open the Events.html file.
2. Review the HTML.
3. Open the page in a browser.
   1. Click the buttons and note that they have a visual response to the click event. As no event listener code has been written, nothing else happens.
4. Edit btn1 and add an onclick attribute and an alert with a message. Maybe something like this:  
   
   1. Note that you are typing the code as text inside of quotes.
5. Save the file and refresh the page in the browser. Click the button and verify that you get a popup message.

**Call a function:**

1. While inline code works, you will have code scattered everywhere. A better solution is call a function.
2. Return to your editor.
3. In the script block write a function that displays a message.  
    
4. In btn2 add an onclick attribute with the text to call the function.  
    
5. Save the file and refresh the page in the browser. Click the button and verify that you get a popup message.
6. We need a similar function for btn3, but with the word “kittens” instead of “toys”. Instead of writing another function, modify the existing one to accept a parameter.  
    
7. Update the btn2 onclick attribute to pass the word “Toys” as a parameter to the function.  
    
8. In btn3 add an onclick attribute with the text to call the same function.  
    
9. Save the file and refresh the page in the browser. Click the buttons and verify that you get a popup with the correct messages.

**Use the “this” keyword:**

As you start writing more reusable code you will minimize the amount of customized code that is used inline to very little, or even none. The “this” keyword refers to the owner of the function where the keyword is used, or if not in a function, then it refers to the window object. In this example, “this” refers to the button that was clicked.

1. Return to your editor and write a function that:
   1. Accepts a parameter. (the button object)
   2. Displays a message that uses the innerText property of the button.  
       
2. Edit btn11, btn12 and btn13 to call the new function and pass “this” as the parameter. I.e. pass the caller object, or the button. Note that all of the buttons now have identical onclick attributes, making the addition of more buttons easier.
   1. Add onclick attributes to btn11, btn12 and btn13:  
      
3. Save the file and refresh the page in the browser. Click the buttons and verify that you get a popup with the correct messages.
4. We often need access to custom data that is not represented by a built-in HTML attribute. In the examples above we used the text displayed in the button. If you need additional data to process the event, you can store it in variables, or better, in a custom attribute. In older versions of HTML developers often invented new attributes as needed, often causing conflicts with new attributes added in later versions of HTML. These “invented” attributes will not validate using HTML validators like http://validator.w3.org/.   
      
      
      
   Starting with HTML 5, you can create valid custom attributes if you prefix them with “data-”.  
      
    
5. Add “data-fruitcolor” attributes, with appropriate values, to btn11, btn12 and btn13.  
    
6. Modify the “fruitmessage” function to access the new attributes.   
      
   Note that the alert statement can also be written on one line.
7. Save the file and refresh the page in the browser. Click the buttons and verify that you get a popup with the correct messages.

**Dynamically added event handlers**

One design practice is to completely separate design from code… HTML from JavaScript. You can then give the HTML to a graphics designer to edit any way they like. The only requirement is that they don’t change element IDs and attributes. They can’t damage the JavaScript code, because there’s none in the HTML. To support this separation, you will need to write JavaScript to dynamically add the event handlers.

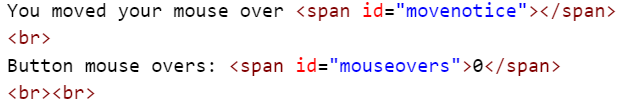
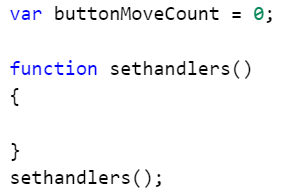
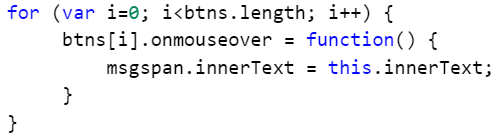
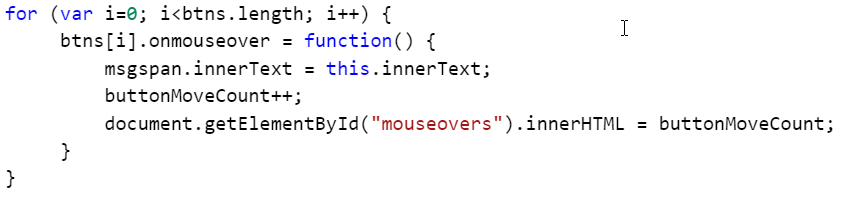
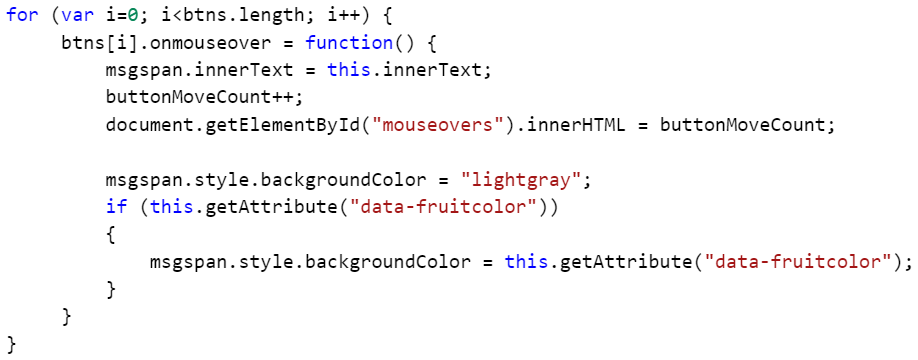
Without design and code separation:



With design and code separation:



In this demo we will leave the existing Events.html file’s HTML and JavaScript as is and add some new functionally dynamically. We will add “onmouseover” event handlers to every Button object on the page to report the button the mouse is over and count the number of times the mouse moved over a button.

1. If not currently open, open Events.html in your HTML editor. If using a folder oriented editor, open the C:\JavaScript\Module5\Lab3 folder, then open the Events.html file.
2. Add the following text at the top of the file:  
    
3. In the script block, add the following variable, the function structure and the call to the function.  
    
   1. Note that we are adding two globally scoped objects “buttonMoveCount” and “sethandlers”. We will clean this up in the next exercise.
4. Inside of the sethandlers function create and initialize two variables to hold the buttons collection and the span for the message area.  
    
5. Below those two lines add a for loop to process each button and add an onmouseover event handler.   
    
   1. Note the use of “this” in the code. “this” refers to the “owner” of the code, which is the button the event handler was added to.
6. Save the file and refresh the page in the browser. Move your mouse over the buttons and verify that you the button name is displayed at the top of the page.  
    
7. Inside of the for loop, add code to increment the buttonMoveCount variable and to display the current value.  
    
8. Save the file and refresh the page in the browser. Move your mouse over the buttons and verify that the button name is displayed at the top of the page and the count is being updated.  
    
9. Just for fun, lets change the color of the text in the top line to match the color of the fruit. Add the five lines starting with “msgspan” from the example below to your code.  
    
10. Save the file and refresh the page in the browser. Move your mouse over the buttons and verify the background color of the button name span changes when moving the mouse over a “fruit” button.  
     

#### Exercise 6 - Closure

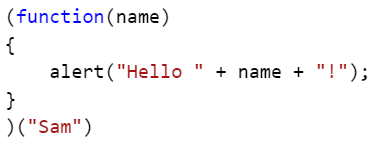
In the example created in Exercise 5 we used a globally scoped variable (buttonMoveCount) and a globally scoped function. Other developers working on our project could accidentally use the same names in their code and cause a conflict.

**Encapsulate the buttonMoveCount variable**

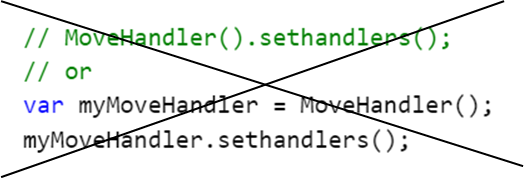
1. Delete the line that calls our existing function. (sethandlers();)
2. Wrap the existing code used to dynamically add the event handlers, including the variable declaration, in an outer function:  
    
3. Add a return statement that returns an object with a single property/function. It will look something like this:  
    
   1. The “return” defines the data being returned from the outer function.
   2. The “{…}” following the return defines a new object.
   3. The “sethandlers : ” is the name of a property or method of the new object.
   4. “function() { }” is an anonymous function being assigned to the “sethandlers” property.
   5. The final code looks like this:  
      
4. We now need to call the outer function to get the object with the function we need to use. Just after the function listed above, add this call:  
      
   or this shorter version:  
    
5. Save the file and refresh the page in the browser. Move your mouse over the buttons and verify the background color of the button name span changes as before.
6. All of that work to remove one variable from the Global scope! But we never have to be concerned about that variable being overwritten by other code. But… that still leaves the outer function’s name on the Global scope.

**Convert the function to be anonymous and self-calling.**

The code to add the event handlers only needs to run once on this page. It will never be called by any other code. The following is a simple example of what’s often called an Immediately Invoked Function Expression (IIFE).



Now let’s update our code…

1. Change the “function MoveHandler()” line to remove the function name and add a parentheses.  
    
2. Move to the end of the function and add a closing parentheses for the function and a pair of parentheses to call the function.  
    
3. The above will now call the outer function, but we need to call the enclosed function. Just add a dot and call that function.  
    
4. Remove the lines that called the original function.  
    
5. Save the file and refresh the page in the browser. Move your mouse over the buttons and verify the background color of the button name span changes as before.
6. The function to dynamically add event handlers to all of the buttons on the page now:
   1. Runs once and only once.
   2. Does not add anything to the Global scope.
   3. Maintains and internal, and otherwise hidden, counter variable.

## Module 5 Lab – Answers

### Questions

❑ JavaScript is compiled:

* Never
* Just in time (JIT) as the code is loaded
* Manually using the JavaScript.exe compiler.
* Automatically using the JavaScript.exe compiler.

❑ Which of the following will create an array?

* var x = "abc", "def", "xyz"
* **var x = "red, green, blue".split(",")**
* **var x = ["abc", "def", "xyz"]**
* **var x = []; a.push("Monday"), a.push("Tuesday")**
* **var x = new Array("Mon", "Tue", "Wed")**
* **var x = []**
* **var x = new Array()**
* var x = document.getElementById("abc")
* **var x = document.getElementsByTagName("div")**

“document.getElementById” always returns a single item, not an array.

Technically, “getElementsByTagName” returns an HtmlCollection object that be generally processed as an array.

❑ You have a string like: "Sam said hello to Susan".  
Which of the following lets you embed double quotes around “hello”?

* "Sam said ""hello"" to Susan"
* **"Sam said \"hello\" to Susan**"
* "Sam said "hello" to Susan"
* **'Sam said "hello" to Susan'**
* "Sam said 'hello' to Susan"
* "Sam said \qhello\q to Susan"

### Hands on Lab

#### Exercise 1

Step 5: At the console prompt explore the data types of each of the variables. Type these commands and record the types:

* 1. typeof v0 \_\_\_\_\_\_\_**undefined**\_\_\_\_\_\_\_ ( var v0; )
  2. typeof v1 \_\_\_\_\_\_\_**string** \_\_\_\_\_\_\_ ( var v1 = "Hello world"; )
  3. typeof v2 \_\_\_\_\_\_\_**string** \_\_\_\_\_\_\_ ( var v2 = "123.45"; )
  4. typeof v3 \_\_\_\_\_\_\_**number** \_\_\_\_\_\_\_ ( var v3 = 123.45; )
  5. typeof v4 \_\_\_\_\_\_\_**string** \_\_\_\_\_\_\_ ( var v4 = v2 + 5; )
  6. typeof v6 \_\_\_\_\_\_\_**object** \_\_\_\_\_\_\_ ( var v6 = new Date(2017,01,01); )
  7. typeof v7 \_\_\_\_\_\_\_**object** \_\_\_\_\_\_\_ ( var v7 = [ 1,3,5,7,9 ] )
  8. typeof v8 \_\_\_\_\_\_\_**function** \_\_\_\_\_\_\_ ( var v8 = function() { return "Goodby world" }; )
  9. typeof v9 \_\_\_\_\_\_\_**object** \_\_\_\_\_\_\_ ( var v9 = null; )

Step 6: Try these experiments at the Console prompt:

1. Boolean( v0 ) \_\_\_\_\_\_**false**\_\_\_\_\_\_\_\_ ( v0 is undefined )
2. Boolean( v3 ) \_\_\_\_\_\_**true**\_\_\_\_\_\_\_\_ ( v3 is a number )
3. Boolean( v20 ) \_\_\_\_\_**true**\_\_\_\_\_\_\_\_\_ ( v20 is a divide by zero… i.e. Infinity)
4. Boolean ( v10 ) \_\_\_\_\_**true**\_\_\_\_\_\_\_\_\_ ( v11 is Boolean set to true )
5. Boolean ( v11 ) \_\_\_\_\_**false**\_\_\_\_\_\_\_\_\_ ( v11 is Boolean set to false )
6. Boolean ( DoubleIt ) \_\_\_\_\_**true**\_\_\_\_\_\_\_\_\_ ( DoubleIt is a function )
7. Boolean ( DoubleIt(2) ) \_\_\_\_\_\_\_**true**\_\_\_\_\_\_\_ ( DoubleIt(2) is a call to a function )
8. Boolean ( DoNothing ) \_\_\_\_\_\_\_**true**\_\_\_\_\_\_\_ ( DoNothing is a function )
9. Boolean ( DoNothing(2) ) \_\_\_\_\_**false**\_\_\_\_\_\_\_\_\_ ( DoNothing(2) is a call to a function )
10. What’s different about “DoNothing(2)”? \_**it returns a null, and Boolean(null) is “false”\_**

Step 7: Use Boolean (or and “if” statement) to see if the v6 variable is a date. Tip: date objects have a “GetDate” function.

Solution: **Boolean(v6.getDate)** or **if (v6.getDate) { }**

Step 8: Use the same solution to see if v7 is a date.

Solution: **v7 is not a date.** I.e. Boolean(v6.getDate) returns false.

Step 9: Which property could you test to see if v7 is an array? Tip: In the Console type “v7” followed by a dot and look to see if there is a property or function in the popup list that might be unique to an array.

Arrays have several unique properties to test. Examples: **push and pop**

Step 10: Confirm that the “window” object has an “alert” function.

* 1. Solution: \_**Boolean(window.alert)**\_

Step 11: Does the “window” object have an “popup” function?

1. Yes or no? \_\_**No, but you could add one!**\_\_\_

Step 12: Which of the following will test to see if the “v100” variable has a “title” property?

1. if (v100.title) { alert("Title is " + v100.title) } else { alert("no title!") }
2. if ( Boolean(v100.title) ) { alert("Title is " + v100.title) } else { alert("no title!") }
3. if ( v100.title != undefined ) { alert("Title is " + v100.title) } else { alert("no title!") }
4. if ( (typeof v100.title) == "string" ) { alert("Title is " + v100.title) } else { alert("no title!") }  
     
   **All of the above!**

#### Exercise 3 – Debugging using the “console” Object

See the ConsoleObject Solution.html file.

#### Exercise 4 - Working with Scope

Step 4: Uncomment the code for “demo 1” and save the file. Refresh the browser and note the messages. The author of the code expected the final value of the counter variable to be 12. Which of the following changes will ensure that the value of the counter variable will be 12 at the end of the code? You may want to test each change…

* 1. [\_\_] Add a “var” in the for loop to keep that counter variable to the loop.  
      
  2. [\_**X**\_] Add a “let” in the for loop to keep that counter variable to the loop.  
      
  3. [\_**X**\_] Rename the variable in the for loop.  
      

Note: “b.” is true only browsers that support ECMA Script 2015 (ECMA Script Version 6) “let” statement.

Step 5: Uncomment the code for “demo 2” and save the file. Refresh the browser and note the messages. The author of the code expected the final values x and y to not change. Which of the following changes will ensure that the value of the counter variable will be 12 at the end of the code? You may want to test each change…

1. [\_**X**\_] Add a “var” in front of the first use in the function of the y variable.  
    
2. [\_**X**\_] Add a “let” in front of the first use in the function of the y variable.  
    
3. [\_**X**\_] Rename the y variable in the function.  
    

Note: “b.” is true only browsers that support ECMA Script 2015 (ECMA Script Version 6) “let” statement.

#### Exercise 5 – Functions and Event Handlers

See the Events Solution.html file.

#### Exercise 6 - Closure

See the Closure Solution.html file.