JavaScript Practice 1

Introduction: JavaScript is a client-side scripting program that allows you to modify your Web page based on events and other methods. While JavaScript is a programming language, we will just learn enough about how it works to implement some simple features and make use of existing JavaScript code in our Web pages.

Some reference sources for this JavaScript practice include:
http://www.w3schools.com/js/
http://www.javascriptkit.com/script/script2/transitionshow.shtml

You may use the text editor of your choice for this practice.

Step 1. Download the JavaScript1.zip file and extract the JavaScript1 folder.
The JavaScript1 folder is the site folder. It contains some example Web pages where we will experiment with JavaScript. This client-side scripting language allows us to add behaviors to the page content and presentation.

Step 2. Open the **Alertbox.htm** page in source view. JavaScript can be placed is several places within the Web page. Usually functions are placed in the <head> of the page, outputs from the scripts are usually displayed in the <body>, and after the </body> scripts can see all the elements of the page, and finally, within HTML tags, JavaScript event handlers are used to call functions.

The **<script>** tag is used to surround your JavaScript code. Use the **type="text/javascript"** to make sure your page correctly validates your JavaScript code.

Step 3. JavaScript comments can be used to explain the code, and make the code more readable. Multi- line comments use /* and */ and single-line comments use //.

Statements in JavaScript are a command to the browser to do something. If you want to have more than one statement on a line, you should end each statement with a semicolon (**;**). The **alert** pop up box can be used to send a message to the user. Notice this statement ends with a semicolon. The user must close it with an "OK". Run the Alertbox.htm page inthe browser. If you want you can change the string that is displayed in the alert.

Step 4. JavaScript can also be used to perform calculations. Open the **Calculations.htm** page in source view. Javascript variables are "containers" for storing data and are usually declared with the **var** keyword. You may have global variables which can be seen on the entire page and local variables which are only visible within the function in which they are declared.

There are many types of operators available in JavaScript, such as the **arithmetic operators** *, /, + and -. You can assign the results of a calculation to a variable with the **assignment operator** =. Within the <head> script, the **writeln** function can be used to send output to a page. The document object can make use of this built-in function as in document.writeln(secs_per_year);. Characters within **" "** are considered strings which are printed to the page. A string can be set apart with either ' ' or " ". Just try to be consistent in what you use. Characters not within " " are variables and their value is printed. Numbers are also evaluated and printed to the page. View this page in the browser to see the result of the calculations and the document.writeln( ) statements.

Step 5. We will add some additional variables and statements to the Calculations.htm page in source

view. The prompt is another pop-up box that can be used to get information from the user. Place the instructions on what should be entered as a parameter string with " " and store the result of what the user enters in a variable.

When you add two strings or string variables together it is called *concatenation*. Follow the comments in the source code and create a short madlib sentence using variables, prompts and document.writeln( ) statements. Show the Calculations.htm page again in the browser and add any spaces to your strings if needed between words. Makes me want to play Clue again.

Step 6. Next we will look at **Event Handlers**. Open the **EventHandler.htm** page in source view. An event handler is a way to trigger a function call based on an event that happens in the browser. One commonly used-event is the click event, known as the **onClick** event. You can write a function that will be executed when this event happens to an HTML element on the Web page.

In this page code, you can see in the <head> a function has been created called MsgBox that is expecting a parameter variable to be passed that will be referred to as text string within the function. A form with two HTML controls for a textbox and a button have been placed in the body. We will learn more about these form controls later in the semester. The user can enter some text in the textbox. When the user then clicks on the Show Me button, this causes the onClick event to be **"raised",** which then triggers the MsgBox function to **"handle"** the event. The value entered in the form's textbox is passed into the MsgBox function and shown in an alert pop-up box. When the user closes the alert box, the value in the textbox is cleared. View the EventHandler.htm page in the browser, enter some text and then click on the Show Me button.

Another event that can be used in JavaScript is the **onMouseOver** event. Change the <input> for the button onClick to now be an onMouseOver. View the page again in the browser. This time your text will be passed to the event handler function when the mouse moves over the button.

Step 7. The next example will show how to combine the use of an Event handler, and also demonstrate how to set CSS properties of HTML elements in JavaScript. Open the **SetCSS.htm** file in source view.
In the <body> tag you will see an <h1> heading with an **id** property set. We also have a form again with some <input> buttons. Notice these tags make use of the **onClick** event and the event handler is the changecolor event. In the <body> tag itself, you will see another event called **onLoad** will be raised then the browser loads the page. This will call the same event handler, the changecolor event.

Notice in the changecolor function, the **getElementbyId( )** function is used to set the myHeading variable with the element information for the <h1> heading because it had the id="Top". Next, the document object is used to access the body element and then set the backgroundColor style to be the value passed as a parameter into the function. View the page in the browser. When you click on each button, it should change the body's background-color CSS property.

Step 8. The **if . . .else** is a conditional statement that can be used to test for one or more conditions and then perform some action if the condition is true. Follow the English-like description (pseudo code) of what your logic should be for the if-else if-else statement. View your page is the browser to make sure the <h1> CSS style change when the buttons are clicked.

Step 9. Some of the most interesting uses of JavaScript involve the creative use of images and links. View the **SwapImage.htm** page in the browser. There are 4 small images (thumbnails) on the page and one larger image. The larger image has a caption or title also. If you mouse over the thumbnail, the associated larger image will be swapped in to replace the existing large image. If you look in the Images folder, you will see a FullSize folder that contains the larger images that have all be sized the same with a resolution of 72 px/inch(most used monitor resolution for the Web). In the Thumbnails folder, are the smaller images again sized the same with the resolution of 72 px/inch. Let's look in the source code now to see how JavaScript was used to create these image swaps.

Step 10. Looking first in the <body> notice an onLoad function called **preloadImages( ).** This causes the large images to be loaded into memory when the page loads, so when you swap them in, there is not a "flashing" effect because of a delay when the page loaded. The thumbnail images are places in a with the id="thumbs". A CSS id selector was created in the embedded style sheet for this area of the page. Each <img> tag for the thumbnails makes use of an **onMouseOver** event to trigger the swapImage function. Notice numbers from 0-3 are passed as a parameter into the event handler function. We will explain this indexing in a moment.

Another with the id="theImage" is used to show the first large image on the page. Its title is placed in the <h3> tag and has an id="bottomText ". There is an embedded style sheet where the id selectors for the two tags on the page are styled. Also a class style rule is created for the instructions on the page.

Step 11. In the JavaScript in the <head> two arrays are created. The first imgArray holds a comma separated list of the large image filenames (without the path info). The second titleArray holds a list of the titles for each of the larger images, again separated by commas. A variable to hold the image path was created and will be concatenated onto the filename later.

The most interesting function, is the swapImage function which takes as a parameter the numbers 0-3 which will be used to index into both arrays when we swap. First the **getElementByID()** is used to get the <img> element that displays the large image and also

the <h3> element that displays the title. Then the idex passed into the function is used to get the corresponding image and then title out of each of the arrays and assign it to a variable. The new image filename is then concatenated onto the imagePath to give the new src for the large image on the page. The **innerHTML** property of the <h3> element is then set to be the corresponding title. Remember the text node of the <h3> element is not the value, but the innerHTML property.

Step 12. Now that we have looked at the JavaScript for the SwapImage, let's see if we can successfully add another thumbnail, fullsize image, and title to this script. Look in the Image/Thumbnail folder again. We will add a **new <img>** for the **HamsterSm.jpg** image in the id="thumbs". Make sure the swapImage parameter is the next index, **4**. Provide alt text for the image.

In the <head> JavaScript, add the name of the **HamsterLg.jpg** to the **imgArray**, remember to separate with a comma. Then in the **titleArray**, add a **good title** for the image at the end of the array, again, separate with a comma. Believe it or not, your SwapImage should now work for the new hamster thumbnail and fullsize image.
View the SwapImage.htm file in the browser. Mouse over the thumbnails to see the large images swap.

Step 13. While the **SwapImage** script is getting a bit more complicated, you can find many even more complicated JavaScripts that are freely licensed, mostly for non-commerical use. One that I found that I would like you to try is for an **image slideshow** with some cool transitions if viewed in IE. Otherwise the transitions are just fades. The user of the script can also add optional links and window targets to the images. The bulk of the JavaScript has been placed in an **external JavaScript file** (just a text file) with a **.js** extension. View the **Slideshow.htm** page in **IE** to see the seasons slideshow. Refresh the page to see it change the type of transitions. You can also view it in Firefox to see the more simple fade transitions.

Step 14. So how is this done? The author of this JavaScript kit placed the custom parts of the script in the Web page and put the complex functions in the external .js file. Let's first look at the **Slideshow.htm** file in source view to see what we would have to change to get our own images in a slideshow.

Looking in the <body> of the page, there is a <script> section that the flashyslideshow function passes some very complicated parameters: there is a wrapperid, a wrapperclass, an imagearray, a pause and a transduration. The only values we need to modify in these parameters are the items in the **imagearray**. The first element is the full path of each image, the second optional element is the link or "" if there is not one, the third element is the target value is there is one or else " ", and the last element is a string that holds the text associated with the image. You could also change the pause time between slide changes or the duration of the effect.

Modify the **imagearray** to show the **4 thumbnail images** that are the same shape. If you want to add a link and target to some of these you can add these element or leave them "". View the **Slideshow.htm** page in the browser to see the new images and text in the slideshow.

Step 15. The CSS **.flashclass** class for the image slideshow container has a width that is probably too wide for our images. You could adjust the width of this property in the CSS style sheet. You could also add a **text-align:center** if you wanted to align your image within this container.

Step 15. This was not too painful to get this Slideshow working with our images. If you want to see how complicated the JavaScript that was used to achieve this feature, open the **transitionshow.js** file using your text editor. First, notice there is **NO <script></script>** tag in the external .js file. At the top is an array that holds a string for each of the IE transitions used. Next, the flashyslideshow function takes the settings we set in the parameter and sets them in the slideshow object. The images are preloaded, and the pause and transduration values are used. Other functions deal with the target, the opacity, looping through the imagearray, and using filters and timers. Fun, fun, fun. Notice in the Slideshow.htm page, how the <script> tag uses the **src** attribute to give the path to the external Javascript file.

Conclusion: Trying to become familiar with the key features of the JavaScript language will allow us to make use of some key behaviors in our Web pages. When we really want to implement a complicated feature in client-side script we can look for freely licensed JavaScript code that will achieve the behavior we want or need. If you are so inclined, you can explore JavaScript to a deeper level and maybe we will be downloading your cool scripts in future semesters. In our last project, we will make use of JavaScript for validating controls on a Web form. This is one of the most commonly uses of JavaScript.

Hope you enjoyed this exploration.