JavaScript Intro Lab

Overview:

In this lab, you shall create a JavaScript "framework" in a file named "03_jsIntro/framework.js" and you will create a HTML page "03_jsIntro/index.html" that uses the framework. This is a lab about learning JavaScript, so ***you MAY NOT use jQuery***. You cannot make a counter framework – I am just using this as an example. Note: your next lab will extend this lab with "loose requirements", so feel free to go beyond the requirements for this lab if you want to get a head start on your next JavaScript lab.

Basic Requirements:

1. The JavaScript file shall have the "use strict" directive as its first line of code (see below). By having this directive, you won't be able inadvertently declare a new variable by misspelling a variable name. It seems odd, but if you forget to use the keyword "var" when first using a variable, a global variable is created, regardless of where the variable was first used. So it is very important that you always use the keyword "var" when declaring variables/objects. Note that you will actually have the double quotes in your JavaScript code when you "use strict". Test that you have this right by trying to use a variable that was never declared with the "var" keyword.

   "use strict";

2. The JavaScript file shall have a "**make framework**" function (like "*MakeCounterFramework*") that returns a framework object. So, the HTML file would have some code like this:

```
var counterFramework = MakeCounterFramework ( ); // "counterFramework" acts like a namespace
                                                 // encapsulating variable and function names.
```

3. The HTML page shall declare **at least two component objects** (e.g., counterDiv1 and counterDiv2, below).

```
<div id="counterDiv1"></div>
<div id="counterDiv2"></div>
<script>
    var counterObj1 = counterFramework.makeCounter( {  ele: "counterDiv1",  initVal: 5, color: "blue"}  );
    var counterObj2 = counterFramework.makeCounter(  { ele: "counterDiv2",       color: "green"}  );
</script>
```

5. The framework object shall have a "**make component**" function (like "makeCounter") that creates and returns your component object (e.g., a "*counter*" object).
6. The "make function's" input parameter will be an object with properties (instead of an arbitrary, ordered list of named input parameters). One property of the input parameter shall reference the id of an HTML element. This is a kind of **dependency injection** (telling the framework about elements that the framework can reference).

7. The "make function" shall **set default values** wherever the input parameter object does not provide values. In the example above, the first call to makeCounter provided a "initVal" property (specified value 5) ut the second call did not, so the "make function" will set that to a default value (see example below):

```
function makeCounter (params) {
        // if any of the param properties do not exist, set them to default values.
        params.initVal = params.initVal || 0;    // will be set to 0, if initVal not supplied
```

8. The component returned by your "make function" shall have **at least 2 private properties** using **closure**. Any time you declare a variable within an object (using the "var" keyword), that variable is private to the object. In the example code, in function MakeCounter, there are several private variables: eleRef, counter, name, and errorCallBackFn. (lines 24-27 of counterFramework.js).
9. The component shall have **at least 2 modifier methods** that change the properties such that the change is visually noticeable. In the example code, there is an "increment" method and a "setName" method.
10. The HTML page shall enable the user, via **javascript events** (such as onclick, keypress, onchange, onmouseover, onmouseout) to invoke each of the methods you incorporated into your framework. Google "w3schools javascript events to get a list of possible events). For example, your HTML page might have code like this:

```
<type="button" value="Add One" onclick="counterObj1.increment ( );" />
<type="button" value="Make Red" onclick="counterObj2.setColor ( 'red' );" />
```

11. The HTML page shall **provide instructions** that tell us how we can test your component objects - which user event we should do to invoke a method of your component object, how we can see (visually in the browser viewing area) the change made by invoking that method.

Originality and Other Requirements:

1. **Originality.** Come up with something very original. You can create virtually any object and then give it visual aspects. For example, if you decided to implement an employee object as your component, your employee object could have an image and some text (like name and salary) and your component could visually show both. Your component code can change almost anything that you can set with CSS, e.g., size, position, color, font, background image, text. You can use setInterval to do something repeatedly. For example, if the users clicks on a ball object, the ball "bounces" a certain number of times (based on a "numBounces" property of the ball object). While you can combine techniques presented in the JavaScript tutorial page, your HTML/JavaScript cannot be overly similar to any one example. You shall not use the counter example or validation examples described in the object oriented JavaScript tutorial page on the 3344 web site.
2. **Professionalism.** To get full credit, you have to create a component that is professional - visually appealing and/or useful (for example, would help the user purchase an item).
3. **Look and Feel.** You can use the look and feel either of your initial layout lab or your advanced layout lab (or you can create a new or modified look and feel that meets the requirements of either lab 1 or lab2).
4. **Blog.** For each lab, you will add a blog to your labs page and this lab is no exception. In your blog, link to the work you did ("03_jsIntro/index.html") and describe your experience doing this lab. What aspects were easy? Which were hard? What were the most important things you learned in this lab?
5. **Good Programming Style.** At the top of the 3344 labs page, there is an entry entitled "Requirements For All Labs". Make sure you adhere to the "good programming and design practices" listed in there.

Grading and Submission:

1. After completing all the requirements, test locally (and syntax check), then publish and test what you published.
2. Then submit into blackboard a zip file of your whole web site (including all the work you have done in all the labs so far). Make sure to include all the necessary files and folders in your zip file.

Here is How We Will Evaluate Your Lab:

1. We will visit your published web site, link to your labs page then check the blog for this lab.
2. We will click on the blog link to get to "03_jsIntro/index.html", where we should find (visually) at least two components on the page and instructions that tell us how to test your page.
3. Your instructions should have us perform one event to invoke one of your modifier methods and another event to invoke another modifer method. Each time we invoke a modifier method, we should see (visually in the browser viewing area) the effect of the change to a component/object.
    - Minimal requirements: one event invokes one method on one component and a second event invokes a different method on the second component.
4. We will view source on your HTML page and your JavaScript file"03_jsIntro/framework.js" and check for the program style requirements (listed in the "Requirements For All Labs" section at the top of the 3344 labs page). We will also check that you have followed the basic requirements listed for this lab:
    - The js file has the "use strict;" directive (so you can't inadvertently create a new variable by misspelling).
    - The js file has a "make framework" function that returns a framework object (namespace).
    - The framework object has a “make component” function that returns a component object.
    - The function that creates your component:
        - takes an input parameter object (not a named list of input parameters) and one of the properties of this object is an id that ties the component to an HTML element on the page (dependency injection).
        - sets default values for any property that was not specified in the parameter object
    - The component has at least two private properties (closure).
    - The component has at least two modifier methods (which not only modify private properties of your component/object, but also changes aspects of the component that are noticeable visually).
    - Your HTML page instantiates the framework object, then (using that framework object) instantiates at least two components that the framework knows how to make.
    - At least two user events (on the HTML page) invoke at least two different modifier methods on at least two components/objects. (Minimal requirement: one event invokes one method on one component and a second event invokes another method on another component.)
5. We will make a subjective evaluation of how appealing and/or useful your components are. While we recognize that the requirements of this lab are a bit "mechanical", try to create something simple that blends in with your web site topic and looks somewhat professional.

Suggested Approach

1. Carefuly study the links that were provided (on the 3344 labs page under this lab).
2. Remember that this lab is about learning ***JavaScript,*** so you ***may not use jQuery*** for this lab.
3. As you work on this lab (or any lab that involves JavaScript), work in Chrome, press F12, and watch the console tab - this is the only place that you'll see JavaScript error messages. To debug, add console.log() statements to print information to the console. As things begin to work, you might want to comment out your debug statements instead of deleting them (might be useful again).
4. You can start out with your JavaScript code in your HTML page, then move it later to the external file later.
5. As with ANY programming project (and especially if you are new to JavaScript), code just a few lines of code between testing. If you think you might not be running your latest changes, ***Control-Refresh*** your browser.