CSCI 3342/6312 – Tutorial/Assignment #2 **Deadline: 3:00 pm on June 5, 2020**

– In-Class Project/Quiz #2 (Separate *Container*) **Deadline: 3:00 pm on June 5, 2020**

Completion: Required **Submission: Required**

Last Name:_____ First Name:_____

This *tutorial+assignment* is to help you to learn basic Java Script code interacting with HTML/CSS code to render ***behavior*** to the components of a webpage.

1) How and what to submit?

Submit the following (upload in Blackboard to the available container) in "one" PDF document (not in docx or any other format):

i) The certification page (see next page) should be the first page, followed by
ii) your solution to the problems given on this assignment.

One way is to copy the certification page and your solution to the problems into a Word document and then save the Word document as PDF, and upload the PDF version (not docx version). Only the PDF version will be graded.

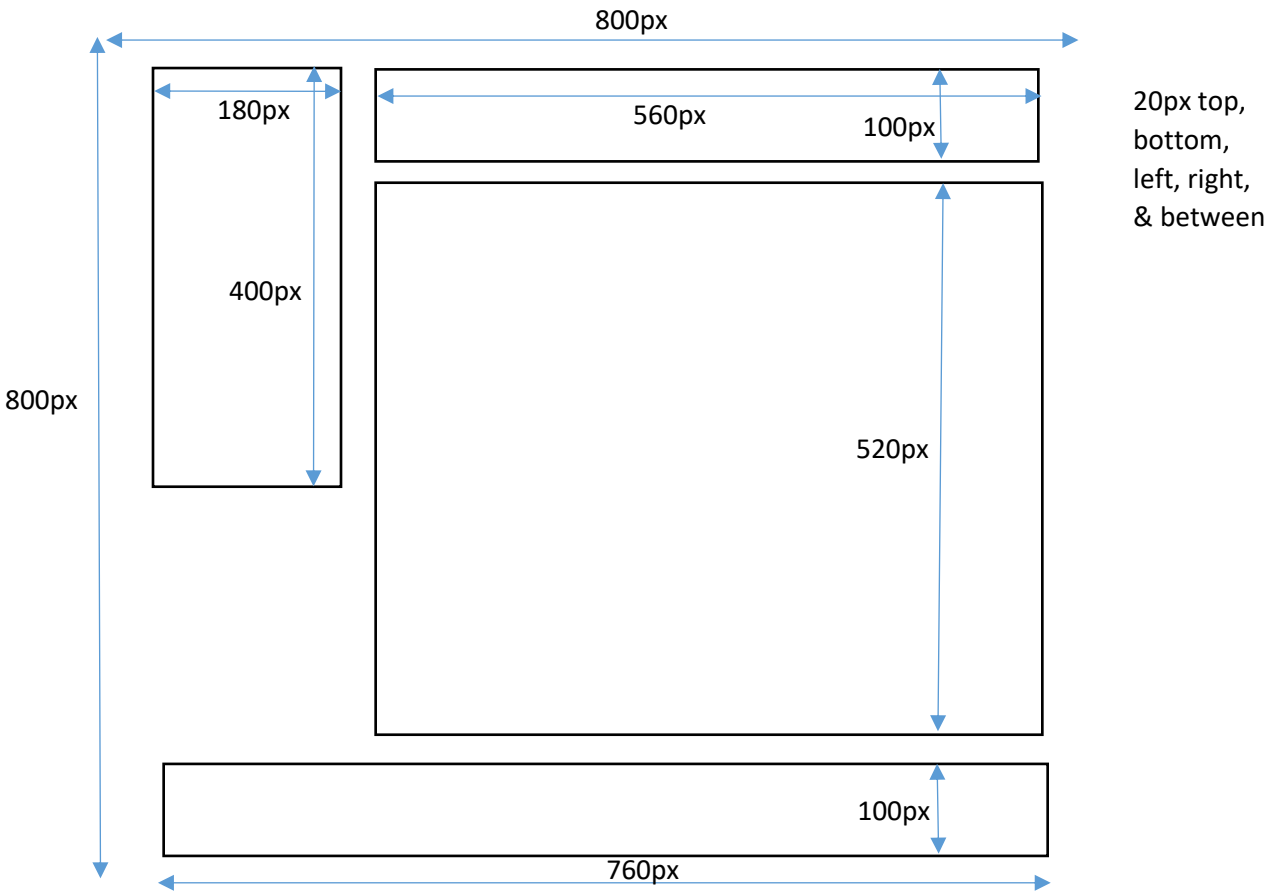
2) Only ONE upload attempt is allowed: Before submitting a document through Blackboard, you should review the document being uploaded to make sure that you are uploading the correct document (e.g. do not upload the assignment belonging to another course). To help you prevent uploading wrong documents, notes (titled "HelpOnSubmissionThroughBlackboard" on how to save & review drafts before final submission have been uploaded under Reference Material folder.

Certification Page

This page must be the first page of your uploaded document.
Your assignment will not be graded without this page (completed with your full name in the area provided) as the first page of your uploaded document.

I, _____, certify that the work I am uploading represents my own efforts, and is not copied from anyone else or any other resource (such as Internet). *Furthermore, I certify that I have not let anyone copy from my work.*

Tutorial Portion



Some notes!

JavaScript (JS) is a Client-Side Scripting Language (different from Java language) rendering ***behavior*** to a webpage, for example, for implementing:

- Form Validation
- Effects on a webpage
- Interactive content

PascalCase vs. ***camelCase*** notation for naming variables (JS uses camelCase) ➔ In camelCase notation, a variable starts with a lower case while subsequent words start with upper case. In PascalCase notation, every word starts with uppercase.

JS is ***case sensitive***.

JS is considered ***dynamic*** (rather than static) language based on it allowing the type of variable to be changed in the same environment.

(1) Copy the previously created html file to *a2.html* in your RPi's /home/pi/ directory with the following changes to get started on learning JavaScript:

```
<!DOCTYPE html>
<html>
	<head>
		<title> CSCI 3342 Assignment #2 </title>
		<link rel="stylesheet" type="text/css" href="e2.css" />
	</head>
	<body>
		<div id="canvas">
			<div id="guide">
				<h3>Guide</h3>
				<hr>
				<ul>
					<li><a class="thispage" href=" ">Intro</a></li>
					<li><a href=" ">Exams</a></li>
					<li><a href=" ">Quizzes</a></li>
					<li><a href=" ">Assignments</a></li>
				</ul>
				<img src="mascot.png" width=30px height=30px alt="Our Mascot" />
			</div>
			<div id="header">
				<h1>CSCI 3342 Assignment #2</h1>
			</div>
			<div id="body">
				<h2>Learning JavaScript</h2>
				<p>
					<script>
						document.write("My first JavaScript Line!!!");
					</script>
				</p>
				<p>Topics</p>
				<p>	<ol>
						<li>Form Validation</li>
						<li>Webpage Effects</li>
						<li>Interactive Content</li>
					</ol>
				</p>
			</div>
			<div id="footer">
				<h6>Edinburg-Harlingen-Brownsville</h6>
			</div>
		</div>
	</body>
</html>
```

Also, copy the previously created CSS file to *e2.css* to specify the styles for the new webpage whose structure is defined in *a2.html*:

```
/* This is the CSS file
  for Assignment/Tutorial #2
*/
#canvas {
        background-color: grey;
        color: white;
        width: 800px;
        margin-left: auto;
        margin-right: auto;
        padding-top: 20px;
        padding-bottom: 20px;
}
#guide {
        background-color: blue;
        color: yellow;
        width:160px;
        margin-left: 20px;
        float: left;
        padding:10px;
}
#header {
        background-color: yellow;
        color: blue;
        width 560px;
        margin-left: 220px;
        margin-right: 20px;
        text-align: center;
}
#body {
        background-color: cyan;
        color: black;
        width 560px;
        margin-left: 220px;
        margin-right: 20px;
        margin-top: 20px;
}
#footer {
        background-color: black;
        color: white;
        margin-left: 20px;
        margin-right: 20px;
        text-align: center;
}

#guide .thispage {
```

```
        font-weight: bold;
}

a {
        text-decoration: none;
        color: yellow;
}

#guide ul {
        list-style-type: none;
        padding: 0px;
        margin: 0,0;
}
h1, h2, h3, h4, h5, h6 {
        margin: 0px;
}
```

Run file:///home/pi/a2.html in RPi's browser to make sure that a2.html & e2.css provide the intended webpage.

(2) One can declare and initialize variables, operate on numbers and concatenate strings as well. JS automatically converts numbers to strings. Modify *a2.html* as follows and refresh the browser:

```
        <script>
                let v1 =13, v2 =7.5, v3 = "Click Here"; //use let rather than var; use const for constant
                v1 = v1 - v2 + 1.5;
                document.write("Concatenating this with " + v1 + ", " + v2 + " & " + v3);
        </script>
```

(3) Properties, such as length, of objects (like strings) can be accessed as done in other languages. Similarly, methods are available for objects, such as extracting substring of a string. Modify *a2.html* as follows and refresh the browser:

```
<script>
        let v1 =13, let v2 =7.5, v3 = "Click Here";
        v1 = v1 - v2 + 1.5;
        document.write("Length of v3 string is " + v3.length + " and a substring is " + v3.substring(6,10));
</script>
```

(4) One can declare *arrays and dictionaries*, and use a *for* loop to display the contents. Since JS has ***dynamic type***, an array can have mixed values. Also, we will see later that *arrays and dictionaries* may be treated in the same ways since index of an array is same as the key. Furthermore, one can ***dynamically*** change the size of an array by adding elements:

```
<script>
        let v4 = ["zero","one","two",3, 4.5,"six"];  //let v4 = new Array ["zero","one",…"six"]
        for(i=0;i<6;i++){
          document.write("Item #" + i + ": " + v4[i] + "<br/>");
        }
        let v5 = {ssn:1234,fname:"joe",lname:"garcia",city:"bville"};
        for (i in v5) {
          document.write(i + ":" + v5[i] + " ");
        }
</script>
```

Note that one can use ***dot*** notation or ***bracket*** notation to access values from a dictionary since dictionary is really an object:

v5['fname'] or v5[1] is same as v5.fname and all three will return *joe* as the value.

(5) One can define functions as well. These can be defined within the <body> tag or as part of the <head> tag. Latter is preferred to keep these out of the scripting code within <body>, hence causing less clutter in the main code. Let's write functions, print1 and print2, within the <head> tag to print array and dictionary values:

```
<head>
        <title> CSCI 3342 Assignment #2 </title>
        <link rel="stylesheet" type="text/css" href="e2.css" />
        <script>
          function print1(v) {
            for(i=0;i<v.length;i++){
                  document.write("Item #" + i + ":: " + v[i] + "<br/>");
            }
          }
          function print2(v) {
            for (i in v) {
              document.write(i + "::" + v[i] + "; ");
            }
            document.write("<br/>");
          }
        </script>
</head>
```

Modify the code within the <body> tag to call the defined functions to print values of the array and dictionary declared earlier. Refresh the browser to see the effect:

```
<p>
        <script>
                let v4 = ["zero","one","two",3, 4.5,"six"];  //let v4 = new Array ...
                print1(v4);
                let v5 = {ssn:1234,fname:"joe",lname:"garcia",city:"bville"};
                print2(v5);
        </script>
</p>
```

(6) One could have used the same function to print values of both array and dictionary, since the index of the array is interpreted as the key (array is an object just like a dictionary):

```
<head>
        <title> CSCI 3342 Assignment #2 </title>
        <link rel="stylesheet" type="text/css" href="e2.css" />
        <script>
          function print(v) {
            for (i in v) {
              document.write(i + ":" + v[i] + " ");
            }
            document.write("<br/>");
          }
        </script>
</head>
```

Modify the code within the <body> tag to call the defined functions to print values. Refresh the browser to see the effect:

```
<p>
        <script>
                let v4 = ["zero","one","two",3, 4.5,"six"];  //let v4 = new Array ...
                print(v4);
                let v5 = {ssn:1234,fname:"joe",lname:"garcia",city:"bville"};
                print(v5);
        </script>
</p>
```

One can also use a ***return*** statement to return a value to the caller.

(7) One can pop dialog boxes using ***alert()*** function. For example, let's show a dialog box and display a message whether the person in the dictionary is "garcia" or "smith":

```
<script>
        let v5 = {ssn:1234,fname:"joe",lname:"garcia",city:"bville"};
        if (v5['lname']=="garcia") {
          alert("Person is " + v5['fname'] + " garcia");
        }
        else {
          alert("Person is not garcia");
        }
        print(v5);
</script>
```

Refresh the browser to see the effect.

(8) It is better to reduce the clutter and define your *Java Script* in a separate file (same approach as having a separate CSS file). Create the *script2.js* file with Java Script needed for the web page:

```
function print(v) {
    for (i in v) {
      document.write(i + ":" + v[i] + " ");
    }
    document.write("<br/>");
  }
let v4 = ["zero","one","two",3, 4.5,"six"];  //let v4 = new Array ...
print(v4);
let v5 = {ssn:1234,fname:"joe",lname:"garcia",city:"bville"};
print(v5);
```

Remove the Java Script from the <head> and <body> tags. Refer to *script2.js* in *a2.html* file within the <body> tag where one needs to use the script:

```
<p>
        <script src="script2.js"> </script>
</p>
```

Refresh the browser to see the effect. *Note* that we could have made ***two Java Script*** files; one consisting of functions to be referred from within the <head> tag, and another consisting of the remaining code to be referred from within the <body> tag.

(9) One of the reasons for using Java Script is to *dynamically* change the webpage. Let's say that we would like to change the heading of "*body*" DIV to whatever we click on under topics. To be more specific, recall that there are three topics in the ordered list created earlier in "body" DIV:

1. Form Validation
2. Webpage Effects
3. Interactive Content

So, we would like the heading *<h2> Learning JavaScript </h2>* to be changed to the element that we click on in the ordered list above.

In order to point to the ordered list and the heading to be manipulated, we need to assign IDs to these elements. Change the a2.html as follows *(note that script needs to be referred to after the <ol> tag so that it knows about the IDs declared in the code)*:

```
<h2 id="head1">Learning JavaScript</h2>
. . .
<p>     <ol id="list1">
                <li>Form Validation</li>
                <li>Webpage Effects</li>
                <li>Interactive Content</li>
        </ol>
</p>
<p>
        <script src="script2.js"> </script>
</p>
```

In Java Script one can then access the heading or the elements of the ordered list as follows:

```
document.getElementById("head1");     //Heading

document.getElementById("list1").getElementsByTagName("li")[0];       //Element 0 of the list
document.getElementById("list1").getElementsByTagName("li")[2];       //Element 2 of the list
```

In order to ***sense*** certain actions happening to an element, addEventListener() function is available. For example, if we want to sense a mouse "click" on the second element in the ordered list, the following JS statement may be written:

```
document.getElementById("list1").getElementsByTagName("li")[1].addEventListener("click",f1);
```

where f1 is the action that needs to be taken when the second element is clicked. In our example, function f1 should replace the heading with the element clicked on. This is accomplished as follows:

```
function f1 () {
  document.getElementNyId("head1").innerHTML = this.innerHTML
}
```

where innerHTML points to the HTML content of the element and "this" points to the recent relevant element which in our case is the element that is clicked on. Replace *script2.js* file with the following content to incorporate the above functionality:

```
//Below is the Java Script for the webpage with "click" detection and respective action
function f1() {
  document.getElementById("head1").innerHTML = this.innerHTML;
}

for(i=0;i<3;i++) {
  document.getElementById("list1").getElementsByTagName("li")[i].addEventListener("click",f1);
}
```

Refresh the browser to see the effect. Replace "click" with "mouseover" in the above code if you want the same action if mouse is moved over an element.
In order to make the code work for any number of elements, simply replace "3" with the length of the ordered list:

```
//Below is the Java Script for the webpage with "click" detection and respective action
function f1() {
  document.getElementById("head1").innerHTML = this.innerHTML;
}

for(i=0;i<document.getElementById("list1").getElementsByTagName("li").length;i++) {
  document.getElementById("list1").getElementsByTagName("li")[i].addEventListener("click",f1);
}
```

To make the code more readable, one can assign variables. For example, the code can be rewritten as:

```
//Below is the Java Script for the webpage with "mouseover" detection and respective action
head1 = document.getElementById("head1");
list1 = document.getElementById("list1").getElementsByTagName("li");
function f1() {
  head1.innerHTML = this.innerHTML;
}

for(i=0;i<list1.length;i++) {
  list1[i].addEventListener("mouseover",f1);
}
```

Refresh the browser to see the effect.

(10) In this section we will learn how JS can be used to validate content submitted through a form. Let's create two input fields in "body" DIV to accept a username and age of the user. The form itself can be designed with HTML and CSS. Add the following to a2.html *(note that the form and each input field has an ID/class for CSS to give **style** to. Also, function **f2** is called when submit button is pressed which validates the values entered into the fields. If **f2** returns "true" a2b.html is called, otherwise, action is not taken. For now, you can create a simple a2b.html file for testing.)*:

```
<div id="body">
    <h2 id="head1">Learning JavaScript</h2>
    <form id="form1" action="a2b.html" method="post" onSubmit="return f2();">
        <p>
            <div class="row">
                <label for="user">user:</label>
                <input type="text" id="user" name=""> </br>
            </div>
            <div class="row">
                <label for="age">age:</label>
                <input type="text" id="age" name=""> </br>
            </div>
    </p>
    <input type="submit" value="submit">
    </form>
    <p>Topics</p>
    . . .
</div>
```

Add the following to e2.css to give "form1" a table format with each "row" as *row* of the table and *label* and *input* portions of "row" specified as *cells* for alignment:

```
#form1 {
    display: table;
    color: yellow;
    background-color: grey;
    padding: 10px;
    border: solid 3px blue;
}
.row {
display: table-row;
}
.row label {
display: table-cell;
}
.row input {
display: table-cell;
}
```

Refresh the browser to see the effect of the changes. Clicking on the submit button does not do much right now. Let's add function ***f2()*** to *script2.js* as follows to disallow empty fields and also only allow age to be between 18 and 100:

```
function f2() {
  let v1 = document.getElementById("user");
```

```
 let v2 = document.getElementById("age");
 if(v1.value == "") {
        alert("Error: Field may not be left empty.");
        return false;
 }
if(v2.value == "") {
        alert("Error: Field may not be left empty.");
        return false;
 }
 if ((parseInt(v2.value) < 18) || (parseInt(v2.value) > 100)) {
        alert("Error: Age is supposed to be between 18 and 100.");
        return false;
 }
 else {alert("Ok ... Values are acceptable and we will process the form."); return true;}
}
```

Refresh the browser to see the effect of the changes. Submit the forms with different values to make sure that the form is ***behaving*** as it should according to the JS code above.

(11) Let's add some code to ***f2()*** function in *script2.js* as follows to indicate which filed has an error:

```
function f2() {
 let v1 = document.getElementById("user");
 let v2 = document.getElementById("age");
 if(v1.value == "") {
        alert("Error: Field may not be left empty.");
        v1.style.border = "solid 2px red";
        return false;
 }
else v1.style.border = "solid 1px grey";
if(v2.value == "") {
        alert("Error: Field may not be left empty.");
        v2.style.border = "solid 2px red";
        return false;
 }
 else v2.style.border = "solid 1px grey";
 if ((parseInt(v2.value) < 18) || (parseInt(v2.value) > 100)) {
        alert("Error: Age is supposed to be between 18 and 100.");
        v2.style.border = "solid 2px red";
        return false;
 }
 else {alert("Ok ... Values are acceptable and we will process the form."); return true;}
}
```

Refresh the browser to see the effect of the changes. Submit the forms with different values to make sure that the form is ***behaving*** as it should according to the JS code above.

(12) One can also get input from a user through the ***prompt()*** function interactively. Let's say that we would like the user to suggest another name to replace the default heading of "Learning JavaScript." In order to do this, add the following code to *script2.js* at the end after the existing *for* statement:

```
. . .
for(i=0;i<list1.length;i++) {
 list1[i].addEventListener("mouseover",f1);
}
document.getElementById("head1").innerHTML = prompt("Suggest another heading");
```