**Assignment 8 – Alphabet Game**

**Setup**

Each week you will be asked to create a new folder under web-231 following a naming convention of “week-<number>.” If we are on week two, the folder name should be “week-2.” All files associated with the weekly assignment will be added to the appropriate weekly folder. All programs must be linked in the index.html landing page under the “Weekly Assignments” section. Projects will be linked under the “Projects” section of the index.html landing page. To be clear, **all** of the JavaScript, HTML, images, and CSS files associated with a weekly assignment must be placed under the appropriate weekly folder. The page title for all HTML files in this course must say “WEB 231 – Enterprise JavaScript I.” And, all HTML and CSS files must be valid HTML/CSS, tested through the WC3 validator. The links were provided during WEB 200 and were added to the index.html landing page. Also, the blue border around the provided images is to show they are images and should not be included in your submission. In other words, do not add a blue border around your work, unless the instructions explicitly ask for it.

**User interface styling and formatting requirements are located in the HTML, CSS, and JavaScript Requirements document.**

HTML: **<yourLastName>-alphabet-game.html**

CSS: **<yourLastName>-alphabet-game.css**

**Grading Reminders**

1. (50%-points) All code sources (.html, .css, .js) must be cited in the opening programmers’ comments, following the format specified in the code attribution document.
2. (25%-points) All code sources (.html, .css, .js) must show evidence of code comments. This means each section of the program (.html, .css, .js) must include code comments that explains what the block of codes purpose is, what the required parameters are (data type, if any), and what the expected output is.
3. (rubric) All code sources (.html, .css, .js) are measured against
   1. Code functionality: Does it work? Does it meet requirements?
   2. Adherence to standards and conventions. Are you using the appropriate data types, including proper indention, are variables named appropriate (variable x is an example of poor naming conventions), is there an appropriate use of whitespace, is the code organized, and are semicolons being used to terminate code sentences?
   3. Efficiency: Use of language features. Are you practicing DRY (Don’t-Repeat-Yourself?), are you leveraging built-in language features where appropriate, and are you using classes/functions to reduce code clutter?
   4. Documentation: Self-documenting, naming conventions, code is maintainable by others. Is the code your write easy to read and maintainable by others?
   5. Error trapping/handling. Are there errors in the program? Is there evidence of coding best practices to reduce user errors?
   6. Assignment Specific Compliance. Does the delivered solution follow the instructions, as they are written? Does the output match what was provided in the screenshots (including spaces, styling, etc.)?

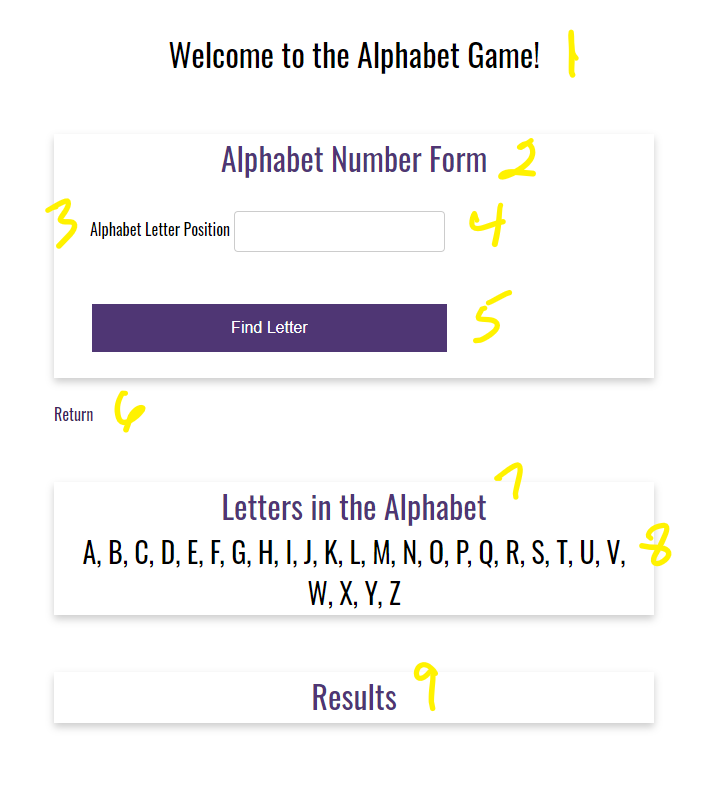
**Required Modifications**

* Cite any sources in your opening programmer’s comment
* Link the appropriate CSS files and Google fonts
* onload…

**Additional JavaScript Requirements**

1. Create a character array of all the letters in English alphabet and assign the array to variable named **alphabet**.
2. Create a string variable named **alphabetOutput** and give it a default value of an empty string.
3. Using JavaScript’s for/in loop, iterate over the **alphabet** array and build a string of the letters in the alphabet with each letter separated by a comma. Use the **alphabetOutput** variable to build the string.
4. Bind the **alphabetOutput** string to the alphabet div. Use JavaScript’s slice() operation to remove the trailing comma from the string (if, any).

**Exhibit A. User Interface (Initial view)**

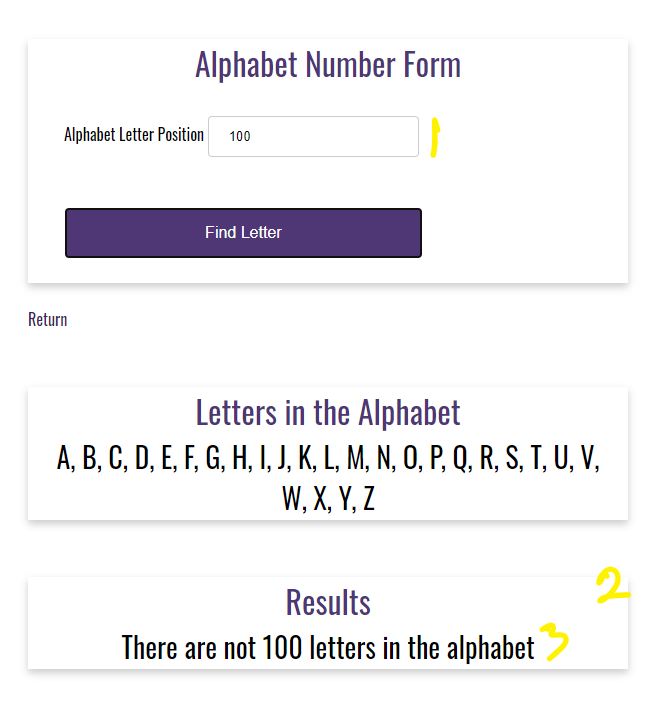
****

1. h1 with a CSS class of app-header and a text value of “Welcome to the Alphabet Game!”
2. card-title with a text value of “Alphabet Number Form.”
3. form-field label with a text value of “Alphabet Letter Position.”
4. HTML input field with a CSS class of input and an id of **txtPosition**.
5. HTML button with an id of **btnFindLetter** and a text value of “Find Letter.”

**Additional JavaScript Requirements**

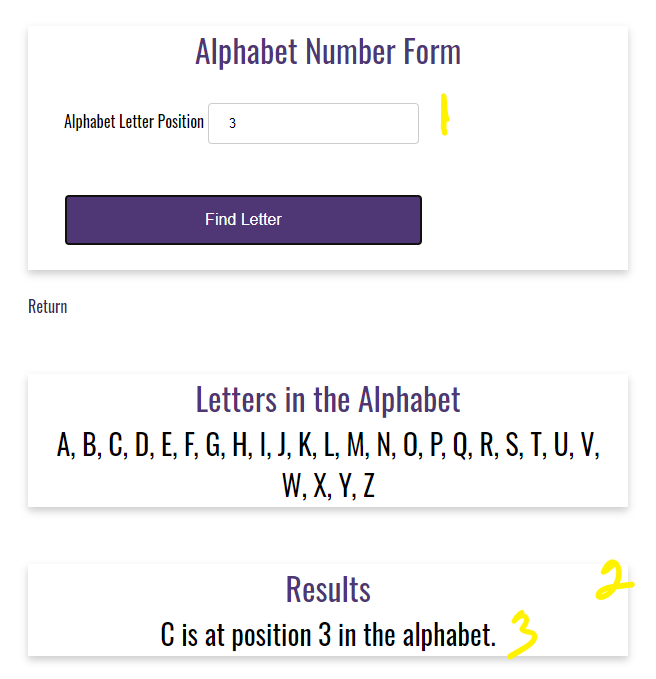
1. Register an onclick event for **btnFindLetter.**
2. Capture the entered value and bind it to a variable named **pos**.
3. Find the alphabet letter in the array that matches the entered position and assign it to a variable named **letter**.
4. Using an if…else statement, compare **letter** against the JavaScript keyword **undefined**
5. if **undefined** (Exhibit B, #1), bind the string value “There are not <pos> letters in the alphabet” to the alphabet-letter div (Exhibit B, #2-3).

**Exhibit B. There are not <pos> letters in the alphabet**

****

1. else (Exhibit C, #1), bind the string value “<letter> is at position <pos> in the alphabet” to the alphabet-letter div (Exhibit C, #2-3).

**Exhibit C. <letter> is at position <pos> in the alphabet**

****

1. anchor link with the CSS class return-home and a link back to the index.html landing page.

**Additional Assistance**

1. Notice the anchor tags placement…
2. card-title with a text value of “Letters in the Alphabet.”
3. card-content where you will bind the **alphabetOutput** string to.

**Additional Assistance**

1. The alphabet div will have a nested card, card title, and card content.
2. The string value you build during the for/in loop will be bound to the card-content section.

**Additional Styling Requirements**

1. As observed in Exhibit B and C, #2-3, the text is centered with a font size of 28 pixels.
2. card-title with a text value of “Results.” The results of the onclick event will be displayed here.