Assignment 4 – Calorie App

Setup

Each week you will be asked to create a new folder under web-330 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 folder. All programs must be linked in the appropriate landing page. Projects will be linked under the "Projects" section of the index.html landing page.

The document title of all HTML files in this course must say "WEB 330 – Enterprise JavaScript II." And, all HTML and CSS files must be valid HTML/CSS, tested through the WC3 validator. The links were provided during WEB 200 and 231. As part of your submission, be sure to include screenshots of the results from the validation tests (HTML and CSS validators).

User interface styling and formatting requirements are located in the Web 330 HTML, CSS, and JavaScript Requirements document.

HTML: **<yourLastName>-calorie.html**
CSS: **<yourLastName>-calorie.css**
JS: **calorie-converter.js, food-model.js**

Grading Reminders

a. (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: Code is maintainable by others
   i. Code comments are present in all blocks of code, written as full sentences, free of grammatical errors, and function/class parameters and data types have been identified.
   ii. Code attribution is present in all files and authorship is clearly annotated.
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, JavaScript, and Google fonts
- food-model.js

  Additional JavaScript Requirements

  b. Create a class named FoodModel with a constructor.
  c. Give the constructor three parameters: id, name, and calories
  d. Export the class

- calorie-converter.js

  Additional JavaScript Requirements

  a. Add an import statement to the FoodModel
  b. Create a class named CalorieConverter with a static variable called data and populate it with 6 FoodModel objects. Supply the objects with the values shown below.

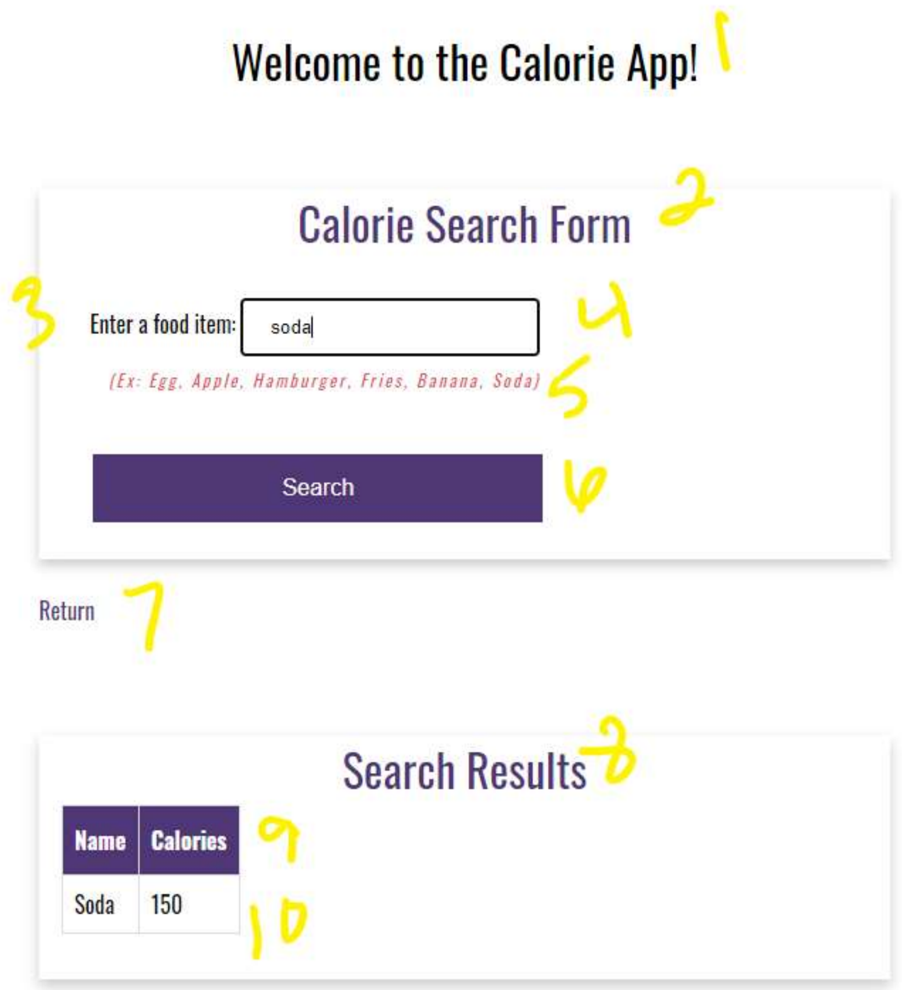
Required data

ID	Name	Calories
1007	Egg	78
1008	Apple	95
1009	Hamburger	354
1010	Fries	400
1011	Banana	105
1012	Soda	150

c. Create a static function named find with a single string parameter. Using JavaScript’s built-in filter function, return a new array of data with the objects matching the parameter string. Use the FoodModel name field to compare the parameter string value against.

d. Export the class.

Exhibit A. User Interface (final solution)



Name	Calories
Soda	150

1. h1 with a CSS class of app-header and a text value of "Welcome to the Calorie App!"
2. card-title with a text value of "Calorie Search Form"
3. form-field with a text value of "Enter a food item:"
4. input field with an id of txtFoodItem
5. <span> tag with an id of foodList

Additional Styling Requirements

a. Add a CSS entry of #foodList to <yourLastName>-calorie.css and float the div to the right, give it a text color of #df546a, font size of 12 pixels, and a letter spacing of two pixels.

6. form-field button with an id of btnSearch and a text value of Search
7. anchor tag with a link back to the index.html landing page
8. card-title with a text value of "Search Results"
9. table header (part of the second card-content with an id of searchResults)
10. table body (part of the second card-content with an id of searchResults)

theme.css

Additional Styling Requirements

a. Add the following variables to the light-theme class:
   1) --table-header-background-color and give it a value of #4F3674
   2) --table-header-color and give it a value of #fff
   3) --table-border-color and give it a value of #ddd
   4) --table-hover-background-color and give it a value of D6A800
   5) --table-hover-color and give it a value of 4F3674

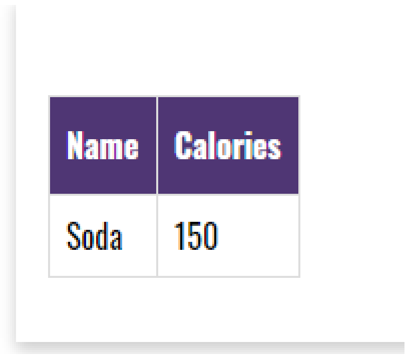
b. Add the following variables to the dark-theme class
   1) --table-header-background-color and give it a value of D176E1
   2) --table-header-color and give it a value of fff
   3) --table-border-color and give it a value of 303030
   4) --table-hover-background-color and give it a value of 7D4787
   5) --table-hover-color and give it a value of fff

c. Add the following CSS classes from WEB 231
   1) .table
   2) .table th
   3) .table td, th

d. Set the CSS **.table th** background color to the variable --table-header-background-color and the text color to the variable --table-header-color

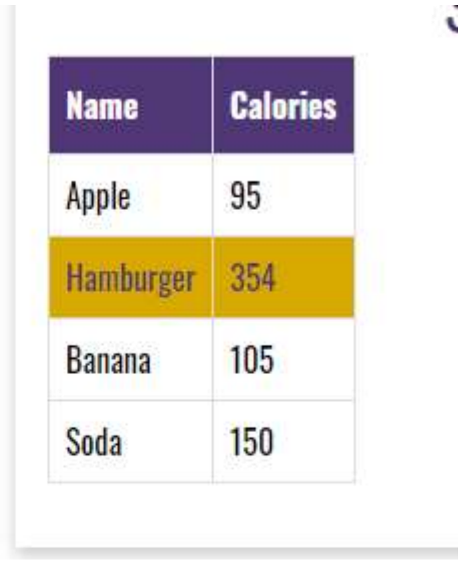
e. Set the table data and header's border to the variable --table-border-color
f. Create a CSS class called table-hover and set the pseudo class tr:hover background color to the variable --table-hover-background-color and the text color to the variable --table-hover-color. Set the cursor to a pointer. See Exhibit C-F for assistance.

Exhibit C. Light mode table



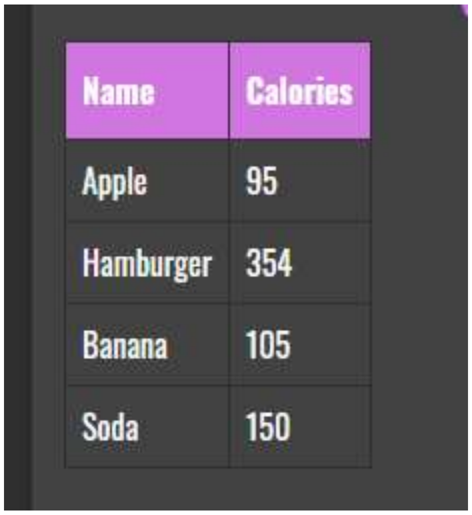
Name	Calories
Soda	150

Exhibit D. Light mode table on hover



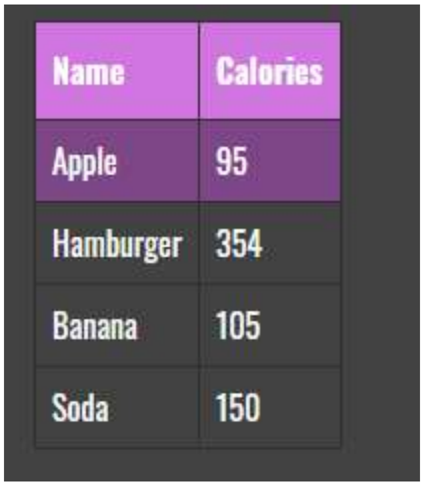
Name	Calories
Apple	95
Hamburger	354
Banana	105
Soda	150

Exhibit E. Dark mode table



Name	Calories
Apple	95
Hamburger	354
Banana	105
Soda	150

Exhibit F. Dark mode table on hover



Name	Calories
Apple	95
Hamburger	354
Banana	105
Soda	150

<yourLastName>-calorie.html

a. Add an import statement for the CalorieConverter class

b. Create a variable named txtFoodItemEl and assign it the txtFoodItem input field.

c. Register an onclick event for the btnSearch field using the document.getElementById function.

Additional JavaScript Requirements (body of the click event)

1) Create a variable named txtFoodItem and assign it the inputted value

2) Create a variable named **foods** and call the static find function from the CalorieConverter class. Make sure you use JavaScript's built-in toLowerCase() to lower case the passed-in inputted value.
3) Create a variable named tableData and build a string to represent an HTML table. Using a for…of loop, iterate over the foods variable and populate the HTML table with table rows and table data. Outside of the tableData string, close the opening table tag and bind the results to the searchResults div.

d. Register an event listener for **keyup**

Additional JavaScript Requirements (body of the keyup event listener)

1) Add an if statement that checks if the "Enter" key is pressed. If pressed, find the btnSearch field and invoke the click() function. The expected functionality is for the user to enter text into the input field and press "Enter." This should invoke the code we added in btnSearch onclick event. In other words, we want the event to trigger when users use their mouse to click the search button or when they press "Enter" on the keyboard.

e. Create a variable named foodList and call the static data field off of the CalorieConverter class.

f. Create a variable named foodListData and build an HTML <i> string for the list of available food items. Using a for…of statement, append the food.name to the variable string. Outside of the for…of statement bind the results to the foodList div's innerHTML. See Exhibit A, item 5 for the expected output.