

## Assignment 5 – Bob’s Automotive Repair Shop

### 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>-bobs-auto-repair.html**

CSS: **<yourLastName>-bobs-auto-repair.css**

JS: **cart-component.js, product.js, shopping-cart.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
- product.js

### Additional JavaScript Requirements

- a. Create a class named Product with a constructor and two parameters: name and price
- b. In the body of the constructor set the class properties to the parameters and add a new property for id. Generate a new random value and assign it to the id field (see Exhibit A)
- c. Export the Product class.

### Exhibit A. Random id

```
this.id = Math.random().toString( radix: 16 ).slice(2);
```

- cart-component.js

**Additional JavaScript Requirements**

- Create a new class named CartComponent that extends JavaScripts HTMLElement object and add it to the cart-component.js file.
- Give the class a constructor and call super() function for the HTMLElement parent object.
- Add a function named connectedCallback() and set the innerHTML to a string for the font-awesome shopping-cart icon. See Exhibit B.
- Outside of the class declaration, call the customElements.define() function and pass-in the string value cart-component and the CartComponent class.
- If you run into issues, the courses GitHub repository has an example project that illustrates how components are created (look for a folder named portfolio). The concept of web components is a requirement in the personal portfolio project.

**Exhibit B. Font awesome string**

```
this.innerHTML = `
  <i id="cartIcon" class="fa fa-shopping-cart"></i> (<span id="cart-count"></span>)
`;
```

- shopping-cart.js


**Additional JavaScript Requirements**

- Create a class named ShoppingCart and give it a constructor.
- In the body of the class's constructor create a property for products and assign it an empty array.
- Create a function named count and return the length of the products array.
- Create a function named add with a single parameter for a product object. In the body of the function, add the product object to the products array.
- Using JavaScript Iterators, create an iterator and in the body use a for...of statement to loop over the products array. Yield return each iterated product object.
- Export the ShoppingCart class.

**Exhibit C. User Interface (final solution)**

# Welcome to Bob's Automotive Repair Shop!

## Product Listing

3  (2)

4 Choose a product:

*To view your shopping cart click on the cart icon*

5

6

7

Return

## Shopping Cart

ID	Name	Price
3706dc8f77b72	Oil Change	19.99
150929ca9701e	Filter Replacement	18.44

1. h1 with a CSS class of app-header with a text value of “Welcome to Bob’s Automotive Repair Shop!”
2. card-title with a text value of “Product Listing”
3. cart-component
4. form-field with a text value of “Choose a product:”
5. form-field select with an id of productList and a CSS class of drop-down-menu

### Additional HTML Requirements

HTML `<option>`: text/values

Value	Text
Select	--Select--
19.99	Oil Change
18.44	Filter Replacement
9.99	Wiper Fluid Refill

6. `<i>` with an id of cart-message

### Additional Styling Requirements

- Add an entry to `<yourLastName>-bobs-auto-repair.css` for `#cart-message` and float the div to the right, give it a text color of `df546a`, a font size of 12 pixels, and a letter spacing of 2 pixels.
  - Using the pseudo hover class, set the hover of the HTML `<i>` element to a cursor with a pointer.
7. form-field button with an id of btnAddProduct
8. anchor tag with a link back to the index.html landing page
9. card-title with a text value of “Shopping Cart”
10. table header placed inside the second card’s content area with an id of shoppingCart
11. table data placed inside the second card’s content area with an id of shoppingCart

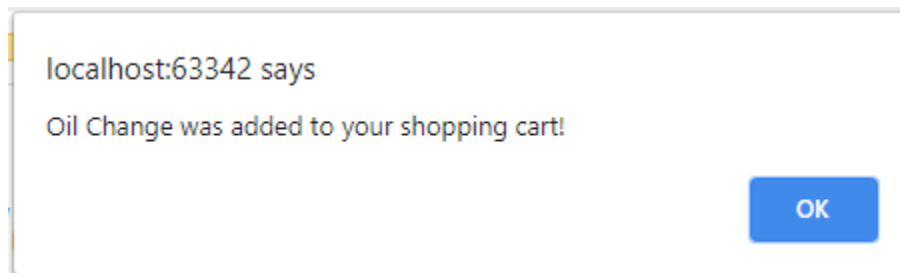
### `<yourLastName>-bobs-auto-repair.html`

- Add import statements for the Product and ShoppingCart classes.
- Instantiate a new ShoppingCart object and assign it to a variable named shoppingCart.
- Call the `setCartCount()` function (see item f.)
- Register an onclick event for the btnAddProduct element (see Exhibit C, item 7).

### Additional JavaScript Requirements (body of the click event)

- Create a variable named productList and assign it the productList div (see Exhibit C, item 5).

- 2) Create a variable named `product` and assign it the value `productList.options[productList.selectedIndex].text`
- 3) Create a variable named `productValue` and assign it the value `productList.options[productList.selectedIndex].value`
- 4) Using an if statement, compare the product variable against the string value “—Select--”
  - i. If they do not match, call the `ShoppingCart.add` function and pass-in a new `Product` object using the `product` and `productValue` variables and arguments.
  - ii. Call the `setCartCount()` function.
  - iii. Alert the message “<product> was added to your shopping cart!”
  - iv. Find the `productList` div and set option value to “select”
  - v. Hint: The expected behavior is for a user to select an option from the HTML select menu, click the “Add to Cart” button, and receive an alert box that indicated the item was added to their shopping cart. See Exhibit D and E.

**Exhibit D. Alert message format****Exhibit E. Updated cart count**

- e. Register an onclick event for the `cartIcon` element (see Exhibit C, item 3).

**Additional JavaScript Requirements (body of the click event)**

- 1) Create a variable named `cartDisplayTable` and assign it the string value with the header of an HTML table (see Exhibit C item 10).
  - 2) Using a `for...of` statement, iterate over the `ShoppingCart` object using the Iterator you created in the `shopping-cart.js` file.
  - 3) In the body of the `for...of` statement append the table with the properties from the iterated product object (see Exhibit C, item 11).
  - 4) Outside of the `for...of` statement close the HTML string table and bind the string to the `shoppingCart` div's `innerHTML` (see Exhibit C, item 10 and 11).
- f. Create a function named `setCartCount()`. In the body of the function find the `cart-count` div (see Exhibit C, item 3; the id was added in the `cart-component.js` file), call the `count()` function from the `ShoppingCart` object and bind the results to the div's `innerHTML`.