Jessica Xiao

Reflection

This assignment required me to intricately understand data storage. As a specific example, I created a custom object for products that were added to cart. These were stored in a cart array, and the cart array was stored in the session storage. At one point, I had to make a change to the data stored in a specific object (increase the quantity of an item), and persisting this change to the actual data stored in session storage required lots of trial and error. Unlike in some languages, elements in an array in JavaScript aren't stored as references. This meant that after modifying the object, I had to manually modify the array, then modify the session storage as well. Along with this issue, I had to carefully think about when session data was initialized, and I couldn't always assume that it was. My solution for this was to either check whether data was initialized, or logically conclude that it must be initialized.

I also struggled with effectively separating my code into logical chunks. There were many actions that were similar, and utilized the same code in some places, but required different logic in other places (for example, decrementing the quantity of an item in the cart versus removing the item altogether). When the HTML for the cart had to be updated, I could also either do a complete update (remove all children, then add the updated ones again), or make incremental changes. I ended up vaguely following a model-view-controller structure to my code, where specific functions were dedicated to either model, view, or controller. To reuse as much code as possible, I chose to perform a complete redraw of the cart whenever it was updated. For the purposes of this assignment where there are only a few products, I think completely redrawing is suitable. If we were to prioritize efficiency, I would choose to perform incremental changes instead.

Programming Concepts

1. MVC Functions

I familiarized myself with the MVC design pattern as the functionality of my website became more complex. Many of the functions are dedicated to controlling exactly the model, view, or controller. As a specific example, the minus sign for each product in the cart has an event listener attached to it. This listener acts as the controller; it takes in the user input, and proceeds to call the function that handles the model: decrementQuantity(). This function thus modifies the information stored in the cart, taking care of the data side of things. After the data has been updated, it calls displayCart(), which updates the view component of the cart. These three functions create the MVC cycle of decrementing the quantity of an item in cart.

2. Custom Objects

Since our cart has to store multiple products, but each product contains the same type of information, I decided to use a custom object. The constructor of this object takes in a size and color for the product. We don't take in a name because for the purposes of this assignment,

we're only adding different variations of one product into the cart. The object also has an associated quantity that stays up to date. Using a custom object helped to easily create many instances of the same type of object.

3. Representing Cart with Array

My cart needed to hold many products, which required the use of some collection data structure. An array was the most applicable here, as we want the cart items to stay in the same order. It would greatly mess with the user's mental model if their cart items were shuffled each time they visited their shopping cart. Using an array here helped to collect all of the items in a cart and maintain their order.

4. <template> Tag

Similar to the motivation of using a custom object, a <template> tag helped to simplify the JavaScript code to display multiple products that all had the same structure. Instead of having to create the HTML of a new item in the cart from scratch every single time, I was able to use this template tag and only modify product-specific components, which greatly simplified the code.

5. Event Listeners

Instead of calling functions within my HTML code, I utilized event listeners in JavaScript instead. Specifically, I used event listeners for decrementing, incrementing item quantities, and removing the item from the cart altogether. This helped to provide more appropriate separation between front- and back-end, since calling a function should remain in the backend.