PUI Homework 6B – Muddy Paws Adventure Gear – Javascript

Philip Gase

Link to Website Home Page:
https://pjgase.github.io/PUI-FALL-2020-GitHubRepo/Homework%206/home.html

Link to HW-6B Branch in My PUI GitHub Repository:
https://github.com/pjgase/PUI-FALL-2020-GitHubRepo/tree/HW-6B/Homework%206

Reflection - Bugs and Issues Encountered

While updating my Javascript from Homework 6A to Homework 6B, I expected to encounter issues driven by the new functionality of displaying the cart and allowing the user to delete items in it. At the same time, I also discovered bugs and issues with the legacy code from 6A, and so I also had to fix certain aspects of Homework 6A, including the display of the cart quantity in the navigation bar.

The first bug I encountered and fixed was an error in which local storage of the list of items in the cart was being overwritten every time a new item was added to the cart, leading to a miscount of the number of items. This was due to a high-level logical error where the send to local storage functionality was embedded within the add to cart function. This was an easy fix for the most part, as I simply decided to separate the functions, so that they were only called when needed. For example, the script was changed to only store the list of items when the page “unloads”. Following this thought, I also split up the functionality for retrieving from local storage, which occurs when the page “onloads”. With these changes, I needed somewhere to store the cart list when various functions (adding to or deleting from the cart) are called on the individual HTML pages, and so I decided to declare the cart_list as a global variable. In this manner, the cart list is always up to date with the latest information regardless of page.

When implementing my cart counting function which tallies up the total number of items in the cart, I ran into a problem with how the quantity was stored in the instance of the object class. This was a pretty easy fix to use parseInt() to convert the string value taken from the input element with type attribute “number”, but it did cause some headache with troubleshooting as I assumed that grabbing the value property from the element would return a number, not a string.

Another small bug I discovered was the ability for a user to select the “dog” option for a product that is only available for a “cat”, i.e. the “cat backpack”. This bug was missed in my code submission for Homework 6A, so it was fixed for the next push by automatically selecting this property for the item class and removing onclick functionality when on the cat backpack page.

One of the last issues I discovered was a problem with how the items in the cart were loaded from local storage when going to a new page. I implemented a function to retrieve the cart when every HTML page in my site loads and return that array. Unfortunately, if the cart is empty when the data is retrieved from local storage, the function returns a null value. My code makes use of for loops and the length property of the cart list quite often, and I ran into the problem of trying to access the length of a null object. I noticed this while looking at the console in the developer tools of Chrome, but interestingly not in Edge. It was an easy fix to add an “or” statement with a blank array to the global variable that stores the return of retrieve function (in case it returns null). I suppose this is one of many reasons why you should double check your code in different browsers.

Although not a bug, one final thought on a general issue was the difficulty of rendering new HTML of nested elements with varying attributes when displaying the cart on the shopping cart page. This struggle was partly due to starting off the website in Homework 5 with plain HTML and CSS, so I would definitely consider utilizing a javascript library in the future to convert the entire ensemble of HTML, CSS, and JS code to something a little more friendly.

Programming Concepts (5 with examples)

1. Creating an Object Class with Multiple Methods

The first Javascript concept that I utilized in this assignment was the creation of an object class “Product” that could be used to store the properties and options for each item that a user could choose. The class constructor method created five properties: name, animal, size, color, and quantity. Each of these properties were able to be set based on user selections in the webpage. This was achieved by creating methods for the product class that could be called “onclick” and set the property when it’s respective HTML element was chosen in the webpage. In addition, these methods were used to alternate styling so that users could clearly see which options they have selected. Here is a portion of the code in my script showing the declaration of the object class:

```
JS main.js > ...
1   // Create Product Class
2   class Product {
3
4       constructor(name, animal, size, color, quantity){
5           this.name = name;
6           this.animal = animal;
7           this.size = size;
8           this.color = color;
9           this.quantity = quantity;
10      }
11
12      load_name(item_name){
13          this.name = item_name;
14      }
15
16      choose_animal(animal_element){
17          this.animal = animal_element.getAttribute('data-animal-type');
18          //change element styling when selected
19          let animal_list = document.getElementsByClassName('animal_option');
20          for (let i=0; i<animal_list.length; i++){
21
22              // Reset Animal Options if Selected Option is Clicked Again
23              if (this.animal === animal_list[i].getAttribute('data-animal-type') && (animal_list[i].classList.contains('selected'))){
24                  this.animal = '';
25                  for (let j=0; j<animal_list.length; j++){
26                      animal_list[j].classList.remove('selected');
27                      animal_list[j].classList.remove('unselected');
28                      animal_list[j].classList.add('start');
29                  }
30                  break;
31              }
```

2. Using the “data-*” Attribute

While researching ways to retrieve information from elements on w3schools, I came across the “data-*” attribute, which essentially allows you to create your own attribute for an HTML element. The “*” in this attribute can be replaced by how you want to define the name for that attribute, and you can set the value for the element just as you would any other element attribute in HTML. aThroughout my javascript file for Muddy Paws Adventure Gear, I made use of this attribute in the different cart functions as well as in the product class methods to pull the values for all of the different option properties in the class. Here is an example of using the “data-*” attribute in HTML as well as how I used that attribute to pull the data into my javascript variables:

HTML

```html
<tr>
    <td id='tiny_option' class='size_option start' onclick='item.choose_size(this)' data-size-type='tiny'>Tiny</td>
```

Javascript

```
49      choose_size(size_element){
50          this.size = size_element.getAttribute('data-size-type');
```

3. Storing and Retrieving Data from Local Storage Using Stringify and Parse

I used local storage for the browser to save the cart should the user change pages or leave momentarily. Since I'd never use it before, I was not certain all of the outputs and how this would store my data, but after checking on variables in the console and logging to it, I managed to come up with a plan for storing the user's cart items. I used Javascript Object Notation (JSON) and stringify on the cart list to store the array of objects as a string, and then I used JSON and parse when retrieving that string to convert back to an array of objects. Since I used a global variable for the cart list and updated for each page, I returned the parsed array whenever calling the retrieve function. Here are the functions that I wrote for storing and retrieving the array:

```
247   // Store Cart Items (onunload for all HTML pages)
248   function store_cart(){
249       localStorage.setItem('cart_list', JSON.stringify(cart_list));
250   }
251
252   // Retrieve Cart Items (when main.js file runs, cart_list stored in global variable)
253   function retrieve_cart(){
254       cart_list = JSON.parse(localStorage.getItem('cart_list'));
255       return cart_list;
256   }
```

4. Calling Functions Onunload and Onload

In conjunction with the local storage concepts mentioned above, I used "onunload" and "onload" event listeners to work with local storage in order to save the cart list variable over sessions and different pages. When the body was loaded (i.e. when the user navigated to a new page), the script would call the retrieve function and set the cart list to the stored array of items. When the body is unloaded (i.e. when the user navigates away from the current page), the script would call the store function to save the current state of all the objects in the cart list on that page. When "onloading" the body, I also used a get_name() function to retrieve the name of the item featured on that particular page (if it applied) as well as a display_cart() function to show the current contents of the cart on the shopping cart page. Here is an example of setting the onload and onunload attributes in the shopping cart HTML page:

```
11      <body onload='get_name(); display_cart(); count_cart()' onunload='store_cart()'>
```

5. Creating, Appending, and Setting Attributes for Elements in Javascript

This may have been the most complicated concept I learned during this assignment and not from a syntax point of view. In regard to keeping track of all the nested elements that I created, it was very hard to picture the output and required multiple iterations with the developer tools in hand. I used vanilla javascript to create elements and append children elements to construct a tree structure of all the cart elements within a blank unordered list in the HTML. The sheer number of lines in the code for the "display cart" function makes it difficult to understand the structure, so using some libraries in a future conversion would definitely help condense this and make it more succinct. In addition to creating this large tree of nested elements, I also learned concepts for setting attributes to those elements in javascript. One particularly useful concept was

using javascript to set an onclick attribute in the generated HTML elements, so that the generated element could be deleted by calling the “delete item” function. Here is example code for the display cart function:

HTML for the Shopping Cart Page

```
<div class='page-subcontent'>
    <div class='feature'>
        <div class ='cart-area'>
            <ul id='user-cart'>
                <!-- add cart items here -->
            </ul>
        </div>
    </div>
    <div class='cart-buttons-area'>
        <button class='cart-button' id='checkout' type='button'>Checkout</button>
        <button class='cart-button' id='keep-shopping' type='button' onclick="location.href='./products.html'">
            Keep Shopping
        </button>
        <button class='cart-button' id='edit-cart' type='button'>Edit Cart</button>
    </div>

</div>
```

Javascript Creating the Child List Element

```
// Display Cart Function (Shopping Cart HTML Page)
function display_cart(){
    // Get Parent List Element for Cart
    let list_parent = document.getElementById('user-cart');
    for (let i=0; i<cart_list.length; i++){
        // Create New List Item
        let list_child = document.createElement('li');
```

Javascript Setting the Onclick Attribute for the Delete Item Function

```
  // DELETE BOX
  let item_delete = document.createElement('figure');
  let item_delete_icon = document.createElement('img');
  let item_delete_name = document.createElement('figcaption');
  item_delete_icon.src = './images/trash-icon.svg';
  item_delete_name.innerHTML = 'Remove';
  item_delete.appendChild(item_delete_icon);
  item_delete.appendChild(item_delete_name);
  item_delete.setAttribute('onclick','delete_item(this)');
  item_delete.classList.add('item-delete');
```

Future Changes for the Prototype

If I were to continue on with the Muddy Paws Adventure Gear website prototype, I would first add some functionality for editing the cart on the shopping cart page followed by implementing the checkout process on the site. Past this, I would consider adding other user features, like a wish list.