Link to Prototype

Link to live version: https://janellewen.github.io/homework_6/index.html
Link to repository: https://github.com/janellewen/janellewen.github.io/tree/main/homework_6

Reflection

I will start by saying this was an incredibly difficult assignment. I have never built something this complicated, which made me respect the process even more. On top of figuring out the logic, I spent a lot of time watching tutorials, asking peers questions, and debugging.

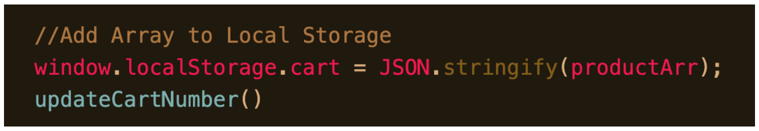
It took me a bit to debug counting the quantities in the individual arrays created when an item was added to the cart, so that the cart quantity would reflect the total number of buns, rather than the number of product arrays. I think I could have mitigated this problem by drawing out the logic first rather than initially basing the product array quantity on the number of buns the customer would add from the dropdown. This messed up how many divs would populate in my cart. For example, if a customer ordered 6 buns of "The Original," the cart icon would show 6, but the cart page would show 6 divs of "The Original" with a quantity of 6 buns each. I had to go back and create a for loop to go through the indices of the cart, and add the quantities together `sum += parseInt(currentCart[I].quantity)`. This way, the product array would have the correct amount of orders, and the cart icon would show the correct quantity of items.

I also learned that sometimes, no matter how much you try to debug, you need another pair of eyes to glance at your code. There were times when the console did not throw any errors, despite something being broken. One example of this was when I was unaware about a concept that I didn't know I needed to use. A peer pointed out that I had was not using `outerHTML`, which was causing my quantities and subtotals to not appear. For example, from using the line `spanQuantity.innerHTML` I was not including the outer div that needed to update.

I experienced this in the the first part of Assignment 6, but I need to *really* figure out a way to organize. At first, I tried creating multiple JS files so I could separate my functions from the first part of the assignment with the second, however I found it difficult to toggle back and forth to refer to pieces of code in another file. I think this will come with practice, though. I do recognize that I was better at naming variables and functions to be more descriptive of its purpose, as well as commented each section with a description. This helped me as the file grew!

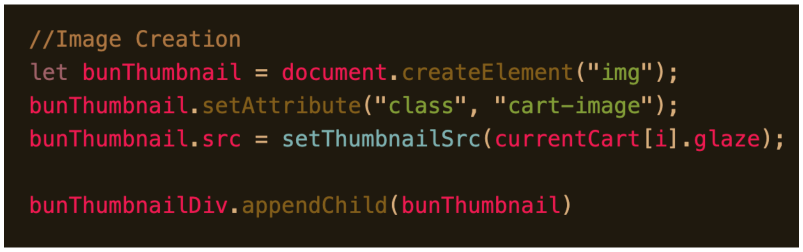
Programming Concepts

1. **Local Storage:** Setting up local storage was the first new concept that I learned. Something interesting that I learned that I did not expect was the translation of the code from JSON string to JS object when extracting from local storage and JS object to JSON string when going into local storage. Below are the lines of code for extracting my array of items from local storage, converting it into a string, then updating the cart name. I also utilized `localStorage.clear()` in the page console to reset my local storage when trying to debug and find edge cases.



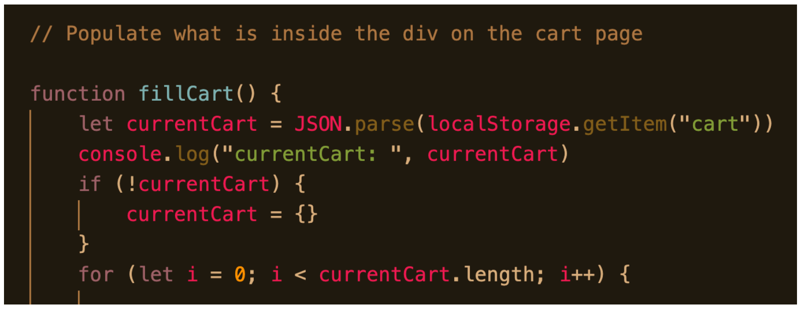
```
//Add Array to Local Storage
window.localStorage.cart = JSON.stringify(productArr);
updateCartNumber()
```

2. **JS HTML DOM Elements**: Creating DOM elements through JavaScript was way more complicated than I thought it would be. w3schools helped me a lot in this. It was tedious to create each element, then essentially stack the new elements together by appending. After getting the hang of it, it made more sense to me. It was helpful to copy my HTML from my cart page into my JS file and comment it out for reference. I cannot imagine how long it would take to build something more complex. Below is a snippet from creating my bun thumbnail image for cart items and appending it to the thumbnail div.

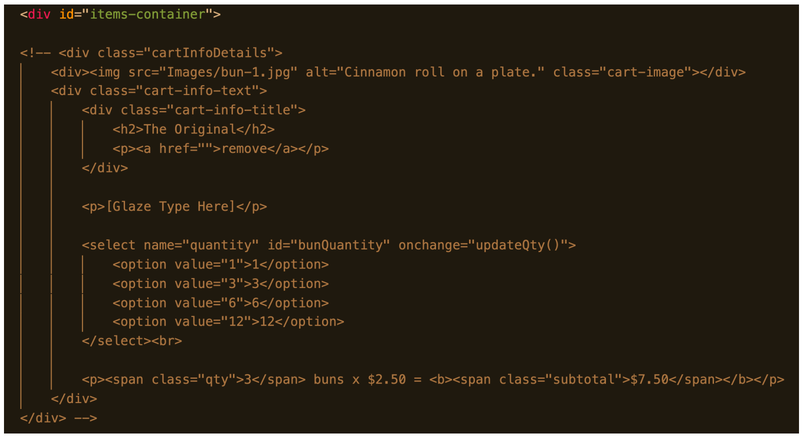


```
//Image Creation
let bunThumbnail = document.createElement("img");
bunThumbnail.setAttribute("class", "cart-image");
bunThumbnail.src = setThumbnailSrc(currentCart[i].glaze);

bunThumbnailDiv.appendChild(bunThumbnail)
```

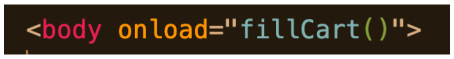
3. **Populating page with new DOM Elements:** With constructing the DOM elements from the HTML code that I built, I needed to create and implant the function that would populate the page with the div when initiated. (1) The function `fillCart()` first checks the length of the cart. (2) For every index in the cart, the set of HTML elements would be created in the `items-container.` I commented out the code that was originally there so I could be reminded that this section would be populated when items would be added. (3) In the HTML, I had to add the code `onload` to initiate the function when the page loads, and then a div to hold the newly populated divs.



```
// Populate what is inside the div on the cart page

function fillCart() {
    let currentCart = JSON.parse(localStorage.getItem("cart"))
    console.log("currentCart: ", currentCart)
    if (!currentCart) {
        currentCart = {}
    }
    for (let i = 0; i < currentCart.length; i++) {
```

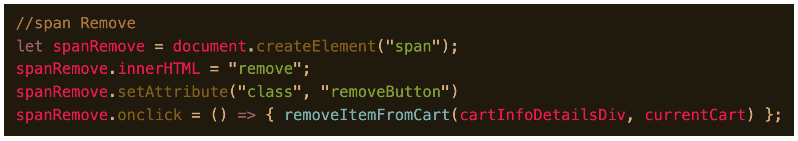


```
<div id="items-container">

<!-- <div class="cartInfoDetails">
    <div><img src="Images/bun-1.jpg" alt="Cinnamon roll on a plate." class="cart-image"></div>
    <div class="cart-info-text">
        <div class="cart-info-title">
            <h2>The Original</h2>
            <p><a href="">remove</a></p>
        </div>

        <p>[Glaze Type Here]</p>

        <select name="quantity" id="bunQuantity" onchange="updateQty()">
            <option value="1">1</option>
            <option value="3">3</option>
            <option value="6">6</option>
            <option value="12">12</option>
        </select><br>

        <p><span class="qty">3</span> buns x $2.50 = <b><span class="subtotal">$7.50</span></b></p>
    </div>
</div> -->
```



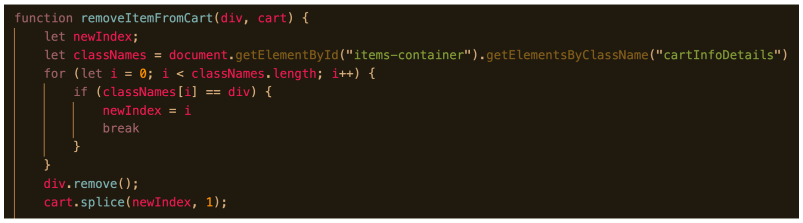
```
<body onload="fillCart()">
```

4. **Arrow Function:** I had difficulty adding the `onclick` action to the "remove" link in the cart page. In the process of figuring this out, I made the on click action with shorter function syntax, which I found on w3schools. This was pretty neat to use, as it was contained in one line. The arrow function here was initiated when the `spanRemove` element was clicked. The item then would be removed from cart. I also found out in this process that the `{}` would prevent the function from running until the `onclick` action was performed. Without this, the function ran.



```
//span Remove
let spanRemove = document.createElement("span");
spanRemove.innerHTML = "remove";
spanRemove.setAttribute("class", "removeButton")
spanRemove.onclick = () => { removeItemFromCart(cartInfoDetailsDiv, currentCart) };
```

5. **Removing Index from Array:** The process of removing from cart was a huge hurdle. The step-by-step logic took me a bit to think through. From this process I utilized a for loop to go through the array of the `cartInfoDetails` divs displayed in the cart under `items-container`, match the index number with the div, remove the div, then reconfigure the indices with one less.



```
function removeItemFromCart(div, cart) {
    let newIndex;
    let classNames = document.getElementById("items-container").getElementsByClassName("cartInfoDetails")
    for (let i = 0; i < classNames.length; i++) {
        if (classNames[i] == div) {
            newIndex = i
            break
        }
    }
    div.remove();
    cart.splice(newIndex, 1);
```