Jailyn Zabala

Github Repo Link: https://github.com/jzabala1/PUI/tree/main/homework6B
Github.io Link: https://jzabala1.github.io/

Reflection

One challenge I had while I was coding was making sure that once I initialized a variable I did not overwrite it. A problem I had was I tried to use global variables and then I switched to using local storage but since they were initialized in either a page loading or a button clicked they would always reset to what they were in the beginning. I ended up fixing this about learning about try and catch and break. I used to try and catch to initialize my empty cart and this was effective in fixing the issue because since it would only throw an error the first time since it was undefined then it would go through to the catch where it would initialize it. Every other time it wouldn't have an error since it was not undefined and so it wouldn't go to the catch so this was an effective way to initialize variables that I didn't want to overwrite. This worked really well with local storage elements especially with my cart array since it would through an error if I tried to access the length. However, I came to another issue when trying to do this for numbers like total because you can do arithmetic with null so it wouldn't throw an error. But I used the same technique from the try and catch and just made a checking function that just had a simple if statement which ended up working to initialize those numbers into local storage. Another thing I struggled with was making sure that things were taken from storage are in the right form to be manipulated and just figuring out the correct order of operations. I overcame these challenges by having a lot of print statements and also writing things out on a whiteboard or saying things out loud just to make sure I had a clear plan of what I think the order of functions to be and then also making sure things are working like I think they are and if they are not seeing where they are not.

Programming Concepts

1. Local Storage

   I used local storage to store the cart, total quantity, and the total amount of items that the user has added to the cart. When using local storage I had to make sure I was reinitializing it and when I took things out of the local storage to change them that I was setting the item after I was done manipulating it.

```
let overallQuant = parseInt(localStorage.getItem("totalItems"), 10);
let productquant = parseInt(product.quantity, 10);
let subTotal = parseInt(localStorage.getItem("total"), 10);
let priceTrimmed = price.replace("$", "");
let total = parseInt(productquant, 10) * parseInt(priceTrimmed, 10);
if (cart.length > 0) {
  for (let i = 0; i < cart.length; i++) {
    if (product.name === cart[i].name && product.color === cart[i].color && product.filling === cart[i].filling) {
      let cartquantity = parseInt(cart[i].quantity, 10);
      cartquantity += productquant;
      cart[i].quantity = JSON.stringify(cartquantity);
      localStorage.setItem("cart",JSON.stringify(cart));
      overallQuant += productquant;
      localStorage.setItem("totalItems", JSON.stringify(overallQuant));
      subTotal += total;
      localStorage.setItem("total", JSON.stringify(subTotal));
    }
```

2. Objects

   My products were custom objects that I made that have the attributes of the name, quantity, color, filling, price, image, and alt text. This way I could easily access these attributes when I got the items from my cart.

```
function Product(name, quantity, image, color, price, altText, filling) {
  //creating a product object
  this.name = name;
  this.quantity = quantity;
  this.image = image;
  this.color = color;
  this.price = price;
  this.alt = altText;
  this.filling = filling;
  this.type = "Product";
}
```

   In the image above I initialize the product and the bottom snippet you can see me use the product attributes to create the elements in my html.

```
if (i === 0) {
  let cartRow = document.createElement('div');
  cartRow.classList.add("row");
  let productContainer = document.getElementById("product-container");
  //this also makes it so that when the bag loads the products load in the right format for the first object
  let productDetails = `
    <img class = "productimage" src="${cart[i].image}"
    alt="${cart[i].altText}">
    <ul class = "productbagdetails">
      <li class ="productli">${cart[i].name}</li>
      <li class ="productli">Color: ${colorTrimmed}</li>
      <li class ="productli">Filling: ${cart[i].filling}</li>
    </ul>
    <button class="bagbutton editbutton">Edit</button>
    <button class="bagbutton removebutton">Remove</button>
    <ul class = "productnumbers">
      <li class = "productnumbersli" id="pricebag">${cart[i].price}</li>
      <li class = "productnumbersli" id="quantbag">${cart[i].quantity}</li>
      <li class = "productnumbersli" id="totalbag">${totalPrice}</li>
    </ul>`;
  cartRow.innerHTML = productDetails;
  productContainer.append(cartRow);
```

3. Event Listeners

   Event Listeners were very important for when I removed an object from the cart. Instead of adding an inline event in the html I instead looped through all the remove buttons on the page and then added an event listener that on click removed the parent element. This made it very easy to remove individual elements even though they shared a class.

```
let removeButtons = document.getElementsByClassName("removebutton");
//got some of this from Web Dev Simplified on youtube (the button parent element remove)
//this goes through all of the remove buttons and checks if they have been clicked
//if they have been clicked it deletes the parent object, updates total, totalItems, and the cart
for (let i=0; i < removeButtons.length; i++) {
  let button = removeButtons[i];
  button.addEventListener('click', function(event) {
    let buttonClicked = event.target;
    buttonClicked.parentElement.remove();
```

4. Removing and Finding Objects in an array

I used an array to be my cart and in order to find and delete items I used indexing in for loops and splicing.

```
if (cart.length > 0) {
  let subT = parseInt((localStorage.getItem("total")), 10);
  document.getElementById("subtotal").innerHTML = `Subtotal: $${subT}`;
  document.getElementById("total").innerHTML = `Total: $${subT}` ;
  //this manipulates the parts of the product object so that they can be prese
  for (let i = 0; i < cart.length; i++) {
    let priceTrimmed = cart[i].price.replace("$", "");
    //Ther rounding with math.round is from stackoverflow user drudge
    let total = parseInt(cart[i].quantity, 10) * parseInt(priceTrimmed, 10);
    let math = (Math.round(total * 100) / 100).toFixed(2);
    let totalPrice = "$" + math;
    let colorTrimmed = cart[i].color.replace("circle", "") ;
```

In the above snippet you can see how I indexed through the cart to get to specific products and in the bottom snippet you can see how I used similar indexing to remove the item from the array.

```
let cart = JSON.parse(localStorage.getItem("cart"));
let overallQuant = parseInt(localStorage.getItem("totalItems"), 10);
let price = cart[i].price;
let priceTrimmed = price.replace("$", "");
let subTotal = parseInt(localStorage.getItem("total"), 10);
let total = parseInt(cart[i].quantity, 10) * parseInt(priceTrimmed, 10);
subTotal -= total;
localStorage.setItem("total", JSON.stringify(subTotal));
overallQuant -= parseInt(cart[i].quantity, 10);
localStorage.setItem("totalItems", JSON.stringify(overallQuant));
cart.splice(i, 1);
localStorage.setItem("cart", JSON.stringify(cart));
```

5. Manipulating variable types
   I had to make sure that variables were the right type to manipulate so I could do arithmetic and string addition properly.

```
if (cart.length > 0) {
  let subT = parseInt((localStorage.getItem("total")), 10);
  document.getElementById("subtotal").innerHTML = `Subtotal: $${subT}`;
  document.getElementById("total").innerHTML = `Total: $${subT}` ;
  //this manipulates the parts of the product object so that they can be presented in a certain way (changing the co
  for (let i = 0; i < cart.length; i++) {
    let priceTrimmed = cart[i].price.replace("$", "");
    //Ther rounding with math.round is from stackoverflow user drudge
    let total = parseInt(cart[i].quantity, 10) * parseInt(priceTrimmed, 10);
    let math = (Math.round(total * 100) / 100).toFixed(2);
    let totalPrice = "$" + math;
    let colorTrimmed = cart[i].color.replace("circle", "") ;
    if (colorTrimmed === "rainyday") {
      colorTrimmed = "Rainy Day";
    }
    else if (colorTrimmed === "morninghaze") {
      colorTrimmed = "Morning Haze";
    }
    else if (colorTrimmed === "cozydenim") {
      colorTrimmed = "Cozy Denim";
    }
    else{
      colorTrimmed = "AfterSchool Special";
    }
    if (i === 0) {
      let cartRow = document.createElement('div');
      cartRow.classList.add("row");
      let productContainer = document.getElementById("product-container");
      //this also makes it so that when the bag loads the products load in the right format for the first object
      let productDetails = `
        <img class = "productimage" src="${cart[i].image}"
        alt="${cart[i].altText}">
        <ul class = "productbagdetails">
          <li class ="productli">${cart[i].name}</li>
          <li class ="productli">Color: ${colorTrimmed}</li>
          <li class ="productli">Filling: ${cart[i].filling}</li>
        </ul>
        <button class="bagbutton editbutton">Edit</button>
        <button class="bagbutton removebutton">Remove</button>
        <ul class = "productnumbers">
          <li class = "productnumbersli" id="pricebag">${cart[i].price}</li>
          <li class = "productnumbersli" id="quantbag">${cart[i].quantity}</li>
          <li class = "productnumbersli" id="totalbag">${totalPrice}</li>
        </ul>`;
      cartRow.innerHTML = productDetails;
```