ASSIGNMENT 6 WRITE UP

ANTHONY TUNG

Repo Link: https://github.com/atung2/homework_6b
Website Link: https://atung2.github.io/homework_6b/index.html

REFLECTION

SQUASHING BUGS

One of the most frustrating bugs I encountered was when I attempted to update the cart page to reflect newly added items. Whenever I would click the "Add to Cart" button on my product page, the previously-existing objects in localStorage would disappear -- meaning that I could, for some reason, only have one object at a time in the cart saved in my localStorage.

I realized that this was because I was constantly uploading a new cart to localStorage everytime I clicked the button instead of updating the already-existing cart; specifically, I wasn't making use of the *localStorage.getItem* command to retrieve my existing cart before pushing new items into my cart. Once I included the line *localStorage.getItem('savedCart')* to retrieve my pre-existing cart, I was able to update my cart upon clicking "Add to Cart".

```
var savedCart = localStorage.getItem('savedCart');
```

Or so I thought. The next bug I encountered is an extension of the first. The idea behind the "Add to Cart" button is to: 1) retrieve my saved cart from localStorage 2) edit the cart by adding a new item 3) return the edited cart back to localStorage. My code in Javascript for retrieving the cart from localStorage was *localStorage.getItem('savedCart')* (notice the lack of *JSON.parse()*). When I tried to push my new item to the savedCart, I get receiving a type error: *cart.push is not a function.*

```
Uncaught TypeError: cart.push is not a function
    at addToCart (js.js:116)
```

After a few frustrating hours, I realized that I was retrieving the saved cart from localStorage as a string and not as an array -- as such, the *push* command was invalid because it's not possible to push an item into a string. Adding *JSON.parse()* immediately fixed this issue. It was an extremely easy fix, which made the whole debuggin process feel extremely frustrating.

```
var savedCart = JSON.parse(localStorage.getItem('savedCart'));
```

Of course, I would argue that these issues with adding items to the cart can be attributed to my lack of experience with localStorage; coming from a non-technical background, I definitely struggled with implementing all the concepts discussed in lab and with getting the syntax down. I believe that avoiding these bugs comes with further refinement/mastery of using localStorage; mitigating these bugs in the future is something that comes from greater experience.

PROGAMMING CONCEPTS

1 - LOCAL STORAGE

In this assignment, I heavily depended on localStorage to store items added to the cart. Local storage provides a space to store session-independent client-side data. I pushed my cart to localStorage and would retrieve/edit it whenever a new item was added. The general pseudocode is as follows:

1. Retrieve cart from localStorage.
2. Push new item into cart.
3. Set newly edited cart into localStorage.

For example, if I wanted to add object "X" to my cart, I would first have to retrieve my cart from the local storage (*JSON.parse(localStorage.getItem('savedCart))*), push object X into the retrieved car (*cart.push(X)*), and reupload the edited cart to localStorage (*localStorage.setItem('savedCart', JSON.stringify(cart))*). Because the cart is stored in localStorage, the items are stored in there regardless of if I refresh the page, close the page, or navigate to another page. This allows my cart page to remain consistent across all the different pages of my website.

```
function addToCart(productName, productColor, productMaterial, productQuantity, productImage, productPrice){
    const newItem = new Product (productName, productColor, productMaterial, productQuantity, productImage, productPrice);
    var cart;
    if(!localStorage.getItem('savedCart')){
        cart = [];
        cart.push(newItem);
        localStorage.setItem('savedCart', JSON.stringify(cart));
    }
    else{
        cart = JSON.parse(localStorage.getItem('savedCart'));
        cart.push(newItem);
        localStorage.setItem('savedCart', JSON.stringify(cart));
    }
```

2 - CONSTRUCTORS AND OBJECTS

I wanted each cart item to have the same parameters to store in localStorage; as such, to accomplish this easily, I created a constructor called Product. This constructor would allow me to create objects with name, color, material, quantity, image, and price parameters; each object represents a single item in the shopping cart. Upon clicking "Add to Cart", a new object is created, and a few programs work together to define each parameter for the object before sending it into localStorage.

For example, if I wanted to add two red Bombay Pillows made of Duck Down, I would create a *new Product* (to call upon my constructor) and run my code so that each of the parameters described above are defined.

```
function Product(productName, productColor, productMaterial, productQuantity, productPrice, productImage){
    this.name = productName;
    this.color = productColor;
    this.material = productMaterial;
    this.quantity = productQuantity;
    this.price = productPrice;
    this.image = productImage;
}
```

```
function addToCart(productName, productColor, productMaterial, productQuantity, productImage, productPrice){
    const newItem = new Product (productName, productColor, productMaterial, productQuantity, productImage, productPrice);
```

PROGAMMING CONCEPTS

3 - FOR LOOP

I needed a way to update the cart page so that it displayed each of the objects saved to my cart in localStorage. Because the cart in localStorage is an array of objects, I decided to use a for loop to run through each object and create corresponding HTML code on the cart page. For each object in my cart, the program would create a new item-card to display on the cart page.

For example, let's say I have objects x, y, and z in my cart. Because the cart is an array (*[{x.name = 'X'...}, {y.name = 'Y'...}, {z.name = 'Z'...}]*), I can loop through x, y, and z, use their individual parameters to create customized item cards (e.g. an item card with the title *'X'*, and item card with the title *'Y'*, and an item card with the title *'Z'*), and add the HTML code of each item card to the cart page.

```
function newCartItem() {
    var savedCart = JSON.parse(localStorage.getItem('savedCart'));
    if (savedCart.length > 0) {
        let count = 0;
        document.getElementById('empty-cart-text').innerHTML= 'Your Cart';
        console.log("saved cart length: " + savedCart.length);
        var totalPrice = 0;
        for(let x of savedCart) {
        totalPrice = totalPrice + x.price;
        var htmlCode = `
          <div class='product-details-cart' id='cartItem${count}'>
```

4 - DOCUMENT.GET

Document.get allows me to retrieve and/or change specific elements from my HTML. For instance, if I wanted to set the innerHTML of a * hello *, I would use *document.getElementById('hello').innerHTML = 'changed text'*. Of course, the use of document.get isn't just limited to changing innerHTML -- in my website, I've also used it to change styles (*document.getElementById('hello').style*), add HTML code (*document.getElementById('hero-container-cart-main').insertAdjacentHTML("beforeend", cartItemList)*), and remove items (*document.getElementById(`cartItem${count} `).remove()*).

```
document.getElementById('hero-container-cart-main').insertAdjacentHTML("beforeend", cartItemList);
```

```
document.getElementById('empty-cart-text').innerHTML= 'Your Cart is Empty';
```

```
document.getElementById(`cartItem${count}`).remove();
```

5 - IF/ELSE STATEMENT

If/else statements allow me to run programs if a given condition is met. For instance, *if* I had items in my cart, I would want my cart page to display the items in the cart; *else*, if I didn't have items in my cart, I would want my cart page to display "Your Cart is Empty". An if/else statement allows my code to respond to user inputs and actions, allowing for a dynamic web page.

```
if (savedCart.length > 0) {
    let count = 0;
    document.getElementById('empty-cart-text').innerHTML= 'Your Cart';
    console.log("saved cart length: " + savedCart.length);
    var totalPrice = 0;
    for(let x of savedCart) {...37 lines }
}
else {
    document.getElementById('empty-cart-text').innerHTML= 'Your Cart is Empty';
}
```