

Parker Nussbaum  
Oct. 22nd 2019

## PUI Assignment 6B

Files can be found at

6A write up can be found at the bottom

The relationship that exists between Javascript, HTML, and CSS is a strange and fascinating example of awful design. The different languages, syntax, and tight coupling - create a walking disaster for any uneducated coder (like myself). Yet - even with this challenging environment I feel as though assignment 6 was a huge success for me. Not only in getting my hands dirty with basic website coding - but also in understanding this relationship better and how to utilize these systems for HCI.

Project 6B proved in many ways to be less challenging than 6A. This might be because I redesigned a large majority of my CSS in 6A - but also because something about this process began to click for this assignment. At some point I was able just to understand the syntax and apply my knowledge of Python to help me progress in the coding. I also think to that CSS - for some reason or another - is more challenging than Javascript.

4) My process began with starting over from 6A ( pictured above. I knew that my cart functionality in 6A was flawed and would not translate well to be scaled. I also knew that my total cost (on the product page) was not functioning on loading. So I scrapped the majority of it and started at the very top.

```
1
2 // This code updates total cart quantity product//
3
4 var count1 = 0;
5 function myFunction() {
6     var x = document.getElementById("tony").selectedIndex;
7     var y = document.getElementsByTagName("option")[x].value;
8     count1 = parseInt(y) + count1;
9
10    document.getElementById("cartquantity").innerHTML = count1;
11    alert ("Added"+" " + y + " " + "buns to cart!");
12 }
13
14 // This code updates total cost on product//
15 var count2 = 0.00;
16 function myFunction2() {
17     var k = document.getElementById("tony").selectedIndex;
18     var z = document.getElementsByTagName("option")[k].value;
19     count2 = ((0.25)*parseInt(z))
20
21    document.getElementById("totalCost").innerHTML = "Total:" + " " + "$" + count2;
22 }
23
```

My first step was creating array of all the product information I needed per specific add to cart interaction using the onClick function. Essentially - when you click the the add to cart button an array is created with [product name, glaze, quantity, cost] - this information then would be appended to a total list of the cart. From there I was able to then loop all of the cart quantity by adding totalProduct[i][3] for the length of product list. After that I sent both the total quantity and the totalProduct array to be converted into a JSON string - then into Session storage.

I tested this out and it appeared to work just fine - until I realized it was not loading back properly . So doing a bit of research and realized I needed to do JSON.parse after sessionStorage.get. I also ran into an error that if it was a fresh load - the cart would read as null. I created an if statement that if the sessionStorage brought back a null value for cart or the total product array - It would then equal cart to 0 and total product array to =[]. In the future I will definitely remember this trick.

Double checking that everything was working error free- I confirmed that cart was consistent across pages and that the total product array loaded by using developers tools .

```
var totalProduct=[]
var carty = sessionStorage.getItem( "cart" ); //gets cart data back
var cart = JSON.parse( carty ); //makes cart data readable
if (cart != null){
    totalProduct=cart;
}
var cost = 0; //sets cost variable as 0

var totalquantit= 0 //checks if there is quantity already in shopping cart
var quantity = sessionStorage.getItem( "totalquantit" ); //gets quantity data back
var quant1 = JSON.parse( quantity ); //makes quantity data readable
if (quant1 != 0){
    totalquantit=quant1;
}

var cartquant= 0;
function drawCartValue(){
    for (var i=0; i< totalProduct.length; i++){
        cartquant = parseInt(totalProduct[i][2]) + cartquant;
    }
    document.getElementById("cartquantity").innerHTML = cartquant; //setting value of words
    cartquant= 0;
}
```

```
function addProduct(){
    var pro = document.getElementById("loriginalbun").innerHTML
    var quant = document.getElementById("tony").selectedIndex;
    var quantit = document.getElementsByTagName("option")[quant].value;
    var glaze = document.getElementById("glaze").selectedIndex;

    if (glaze == 0){
        cost= 0.75;
    }
    else{
        cost = 1.00;
    }

    if (glaze == 0){
        glaze = "Non-glazed"; }
    else if (glaze == 1 ){
        glaze = "Sugar + milk";
    }
    else if (glaze == 2 ){
        glaze = "Vanilla + Milk";
    }
    else {
        glaze = "Double choc.";
    }

    cost = cost*quantit;
    var product = [pro,glaze,quantit,cost]; //creating individual product array
    totalProduct.push(product); //adding it to master list of all products in cart
    var jsonStr = JSON.stringify( totalProduct ); //sending that information to JSON
    sessionStorage.setItem( "cart", jsonStr ); //sets item labeled "cart"

    totalquantit = parseInt(quantit) + totalquantit; // getting cart
    alert ("Added"+" " + quantit + " " + "buns to cart!"); //alerts user it is added
    var jsonStr2 = JSON.stringify( totalquantit ); //sending that quantity information to JSON
    sessionStorage.setItem( "totalquantit", jsonStr2 ); //sets item labeled "totalquantit"
    //alert(totalProduct)
    drawCartValue();
}
```

From there was the hardest portion: building the table of the cart with the product information. Luckily - there was lots of recourses on the commands to add table row and cells. So I decided my strategy should be to use a loop to read the array of total products individually and build the rows one by one. This worked fairly well - but it took some time to understand how to append an image and button to the list. AS well as name it a class to relate to CSS. Not to mention - if I kept reloading the page the list would keep growing the indexes of the product array. I managed to fix that error after some time and did gain more insight on using `appendChild(obj)` so I was finally able to fully build the table out.

The last step - and by far most annoying - was the delete button. I really struggled with this for a few hours. My strategy was to get the delete button to recognize the index of the product it was deleting- and slice it. The issue is that while you are creating a button and can give it a class plus even a function - how can you get the button to recognize the index? I struggled with this for a long time until I realized you could give the button an "id" of "i" in a loop. This would then name the buttons the index of each product. I also used the `this.id`; function to send the variable to the function every time the button was clicked. I feel like in the future I will definitely remember the value of creating HTML obj with functions built in within JS.

From there I created the function that actually deleted by slicing the product based on index and then updating total product array. The function then concluded with reloading the cart to rebuild it. Overall not an easy task but some valuable lessons learned.

I do feel that looking forward I gained lots of insights on how to utilize javascript. I do wish in hindsight I used obj instead of arrays - if only for the learning purpose. I cannot say that I felt one was more appropriate for this project but it does seem the norm in JS to use obj. I also felt that my implementation across the website felt sloppy. Not just JS - but from the ground up. I hope that in the future I can utilize my pains in this process to build a better - more efficient - web page.

```
var totalQuantityCart = 0;
var totalcost=0;
function createCart(){
    //for length of totalProduct darws all the elements
    for (var i=0; i< totalProduct.length; i++){
        var table = document.getElementById("bunbox");
        var row = table.insertRow(i);
        // Insert new cells (<td> elements) at the 1st and 2nd position of the "new" <tr> element:
        var cell1 = row.insertCell(0);
        cell1.classList.add("productimage");
        var cell2 = row.insertCell(1);
        var cell3 = row.insertCell(2);
        var cell4 = row.insertCell(3);
        var cell5 = row.insertCell(4);
        var cell6 = row.insertCell(5);

        var img = new Image();
        if ( totalProduct[i][0] == " Original Bun Bun GF"){ //checking to see which image will use for product
            img.src = 'bun2.png';
        }
        else if (totalProduct[i][0] == " Walnut Bun Bun"){
            img.src = 'BUN8.png'
        }
        else{
            img.src = 'BUN3.png';
        }
        cell1.appendChild(img);
        cell2.innerHTML = (totalProduct[i][0]); //appending product image to cell
        cell3.innerHTML = (totalProduct[i][1]); // setting product name
        cell4.innerHTML = (totalProduct[i][2]) + " " + "qty"; //setting product glaze name
        cell5.innerHTML = "$ " + (totalProduct[i][3]); //setting total quantity
        totalcost = totalcost + (totalProduct[i][3]); //setting cost associatedf with product
        totalQuantityCart = totalQuantityCart + parseInt(totalProduct[i][2]); //setting sub total and cost calculations
        totalQuantityCart = totalQuantityCart + parseInt(totalProduct[i][2]); //setting cart total

        var button = document.createElement("button"); //creating button
        button.classList.add("addtocart"); // naming class of button
        button.setAttribute("id", i);
        button.innerHTML = "Delete";
        var k = 0;
        button.onclick = function(){ //creates button onclick function!
            k = this.id;
            //sends the id of the button clicked to delete funcrtion - the id represnrts the index in the product list
            deletBoi(k); //naming the function that we are calling and sending it an index based on i value to delete
        }
        cell6.appendChild(button);
    }
}
```

5) The first big concept that I took away was using session storage and local storage for saving between page loads. This concept is very powerful for carrying interaction between pages and data. I also think realizing that you are just sending a string - is quite an interesting concept. I assumed that it would be an array so this definitely broke my initial understanding. Definitely though - I would love to know if you can just append directly to localStorage vs creating a list of objects/ lists and constantly re-updating.

The second big concept was how to actually write for loops in javascript. While I am bit embarrassed to say this - I really only understood how to use while loops for the longest time because I didn't understand the notation. But - alas - I understand it thanks to our TA's breaking down the syntax. This is the loop I implemented when generating the table -  
for (var i=0; i< totalProduct.length; i++)

The third big concept was HTML object creation in JS. Using the document.createElement function is surprisingly very simple and makes perfect sense. What becomes more complicated but useful - is the extras. Such as naming class, ID, and adding a functionality like on click. I utilized all of these codes when creating my delete button in the table generation of the cart.

```
var button = document.createElement("button"); //creating button
    button.classList.add("addtocart"); // naming class of button
    button.setAttribute("id", i);
    button.innerHTML = "Delete";
button.onclick = function(){
    k = this.id;
    deletBoi(k);
};
```

The fourth big concept was the legitimate value of having both class and ID in css, javascript and HTML. When I first started this assignment, I didn't really understand why you would need both and why class was not callable by JS. But I realize it is really that way to separate CSS and JS identification from overlapping too much. This was a powerful realization. As an example with my delete buttons - each has an independent ID but the same class - so they all look the same. Not until this specific example did I really get it.

The last big concept was the onLoad="" function that you can slide into tags like <body> in html. Even though it is a simple command and idea- I assumed that JS could not be initialized with out a human interaction! Yet I was proven wrong upon a quick search of the web. I suppose the overarching point here is that there are numerous ways of activating JS with HTML and that these can be leveraged to create functional interaction.

Parker Nussbaum  
Oct. 22nd 2019

## PUI Assignment 6A

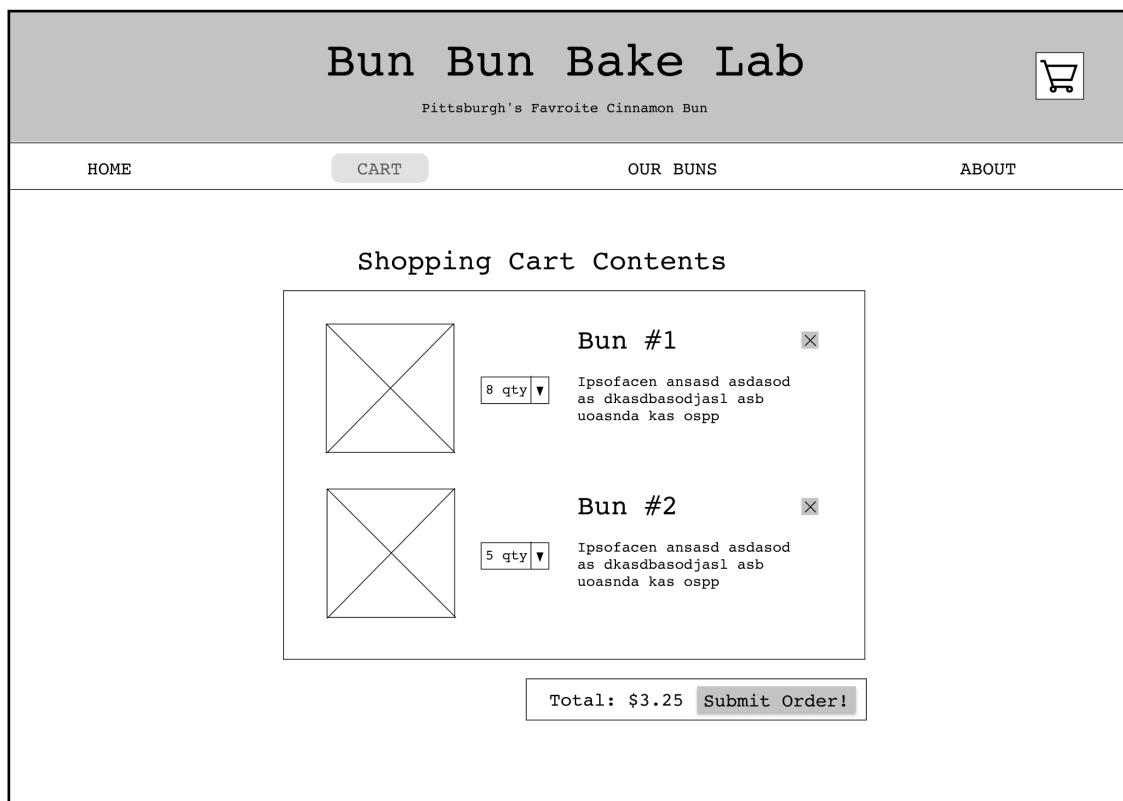
Files can be found at my GitHub - [https://github.com/parkernussbaum/parkernussbaum.github.io/tree/master/Homework\\_6](https://github.com/parkernussbaum/parkernussbaum.github.io/tree/master/Homework_6)  
Website link - [parkernussbaum.github.io/Homework\\_6/index.html](https://parkernussbaum.github.io/Homework_6/index.html)

Tackling assignment 6A was at times challenging but overall a good experience. As a preface - I decided to relook at the aesthetics of the overall website and did a major overhaul from the last iteration trying to approve the appearance. I did this to make myself more proud of the assignment and to give nod to the comments I received on the last assignment. The cart page is accessible through the redesigned new page. I will speak to this more in this paper

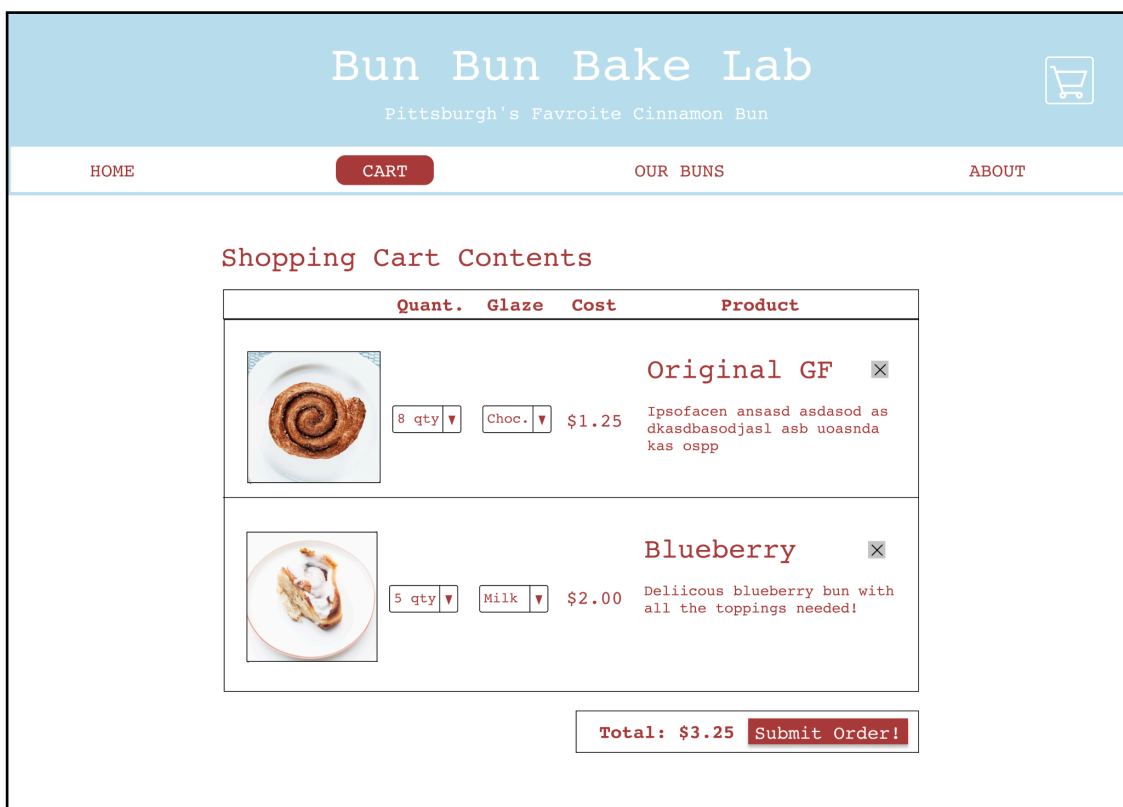
### 1) Low Fi prototype

The image shows a low-fidelity wireframe of a checkout page for a bakery named "Bun Bun Bake Lab". The page has a blue header with the site name and a hamburger menu icon. Below the header, the title "Cart: Finalize order" is displayed. On the left, there is a blue button labeled "Back to Our Buns!". The main content area is divided into two columns. The left column contains a large empty box for the cart items and a "Total: \$0.00" at the bottom. The right column contains the payment and billing information. It starts with a "Payment option: Please select" section with three radio buttons: "Credit Card", "Debit Card", and "Bit Coin". Below this is a "Billing Address" section with a text input field. Then, there are two input fields for "Zip" and "State". This is followed by "Card Holder Name\*" and "Card Number\*" input fields. At the bottom of the right column, there are two input fields for "Exp Date\*" and "CVV\*", with a note "All fields required" below them. A red "PLACE ORDER" button is located at the bottom right of the form. At the very bottom of the page, there is a navigation bar with five blue buttons: "OUR BUNS", "ORDER", "ABOUT", "CART", and "CONTACT".

I decided since I was retackling my aesthetics - that I would redo the low fidelity cart model. The original InVision model (above) needed some updates so I started with a general wireframe based on common designs I had seen around the web.



Moving form the basic wire frame - I noticed that I had not accounted for the glaze, I had not labeled the columns, and the overall design needed some more feedforward indications. So I decided to tackle this issues on the high fidelity figma model



After from the high fidelity mockup - I created a cart page on my html file incorporating a very close design for the cart. Though there are some slight differences and modifications (for legibility and functionality) the principle was the same. Overall - creating the cart page was not a huge challenge considering redoing the website aesthetics proved to be more challenging.

I did find that when creating a cart - both a table and list are viable solutions. I ended up using a table but noticed that there could be some distinct advantages for doing a list if you had more volumes of text. I started to lay the id work as well for assignment 2B.



Bun Bun Bake Lab

Pittsburgh's Favorite Cinnamon Bun

0

[Home](#)
[Our Buns](#)
[About](#)
[Cart](#)

Shopping Cart

Bun Bun	Bun Selection	Glaze	Quantity	Cost	Click to update
	Pumpkin Spice	<div>Non-glazed</div>	<div>1</div>	\$3.00	<div>Update</div>
	Original GF	<div>Non-glazed</div>	<div>1</div>	\$3.00	<div>Update</div>
	Walnut	<div>Non-glazed</div>	<div>1</div>	\$3.00	<div>Update</div>

Total Quantity	Tax	Final Price	Submit
\$6.00	\$0.60	\$6.60	<div>Finalize Order</div>

Copyright Bun Bun Bake Lab 2019

[https://parkernussbaum.github.io/Homework\\_6/bunbunproduct.html](https://parkernussbaum.github.io/Homework_6/bunbunproduct.html)

## 2) Javascript Functionality

Adding the javascript functionality proved to be more challenging than I had originally considered. While I consider myself to be a decent programmer in python - I found that HTML configuration with JS a hard concept to wrap my brain around. I think my lack of GUI type programming also contributed to my problem. Nonetheless - I accomplished 2 big features of functionality.

The first functionality was that when you select the quantity of the buns - it adds the number accordingly to the cart (so 6qty will show 6 in the cart). Because the cart was not visible from the confirm cart add button - I decided to add an alert code to show the user that it was added (also to slow them down from over clicking). The reason this task was challenging was because it took me a long time to realize that Javascript was calling a string value from the select html function. This was leading to the cart showing values like "1261126", instead of 37. It also took awhile to understand how to manipulate inside of a <p> element

The next functionality I added was updating the total cost of the buns on the product page ( this was so that if you selected 12 buns - you knew the cost before adding it to the cost). This proved to be a much simpler task because the ground work was mostly laid out in the last code. It

did take awhile though to realize that onChange was the appropriate function to use for the selection update vs. on-click. This made it so the cost update as a select function was selected vs clicking on the box which caused errors.

### 3) Aesthetic updates and fixes

As stated at the beginning I did tackle some of the aesthetic concerns that were brought up in the previous project. These proved challenging to implement but overall I did. I also condensed all the CSS files to one and created a proper index page as mentioned in the comments. I also created larger buttons to help with Fitts law related errors.

I also fixed the CSS file indentions (as noted) and tackled most of the errors in the HTML code. Some though proved to be too challenging to understand. Such as that a <p> element cannot be under a <span>. I am not sure how one fixes this but I would love to understand an alternate solution!