Link to code:

https://github.com/JennyShuyuLiu/JennyShuyuLiu

Link to the live page:

https://jennyshuyuliu.github.io/JennyShuyuLiu/homework_6b/

Difficulty encountered #1:

I faced some difficulties when trying to implement the function of **deleting items**. When removing a product from the cart, I will need to remove the item from the product array. Yet, I don't know how to **identify the item that shall be deleted**.

Solution: I figured out when loading the cart page, the order that products occur in the cart would be the same as their order in the product array. Hence, the question became retrieving the too-be-deleted item's index (as in the cart list). I did a bit of research and learned a method on StackOverflow: https://stackoverflow.com/questions/5913927/get-child-node-index/42692428

```
var i = 0; // hypothesize that this is the first child

while( (child = child.previousSibling) != null )

     i++; // check if there are any sibling nodes before the
child node =, if yes then increase index.At the end i will
contain the index.
```

Difficulty encountered #2:

I realized the situation where **users removed all items from the cart** is quite special. For example, the product count on the cart icon has to be removed, and the cart page has to display some special text to inform the user that the cart is empty. All these can only be achieved by retrieving various elements in the DOM and altering them. To be more specific, one bug I encountered was: when the last product was removed, Chrome always gave me this error message "*Cannot read properties of null (reading 'removeChild')*" at line *pdiv.parentNode.removeChild(pdiv);*

Solution: I found the solution after going through many trials using Chrome's developer tool and reading through the log. When there are no products in the cart, I set parentNode to display the text "You have chosen nothing... ". Therefore, under such circumstances, there are no child elements under the parent node. The final step is to change the order of js code for a bit: Delete the to-be-removed product's div, and then determine whether to reset the html content of the parent node.

Highlighting few key things I learned:

- **Learned how to get child node's index.**This is very handy since the DOM is essentially a tree structure with nodes on various levels.
- **Logging** & reading the log to see what really happened, rather than making assumptions. Wrong assumptions can be misleading and make the debugging process very time-consuming and confusing. Spending some extra time to study the log is actually saving time. This will definitely be part of my workflow going forward.
- **Using Chrome's developer tool** to test small things first and then gradually build onto a larger system. Although we mentioned this in the lecture, this assignment really illustrated how powerful Chrome's developer tool could be. However, I do find it is **easy to make mistakes while copying and pasting content** between programs while testing multiple approaches, causing code to crash.

Five Programming Concepts

1. **Local storage** could be used to store data over sessions. The data remains even if the tab is closed. This can be used to store products & corresponding features that users picked and left in the cart. For example:

```
localStorage.setItem("product", JSON.stringify(products));
// convert product array to json, saving to localStorage

var pro = localStorage.getItem("product");
// read product's data from localStorage
```

However, the one remaining issue is that we rely on the users not to clear the cache. Otherwise, the stored data would be completely gone. Maybe in future assignments, we can get chances to experience some server-side storage.

2. **Js array** can be used to store a group of elements. It can be used to manage (creating, adding, accessing, and deleting) the elements that it holds. For example, I used the following in my code:

```
var products = [product]; // creating: when user add items to the cart for the first time, create a new array.

var product = products[i];// accessing elements: used to retrieve a product, then display on the cart page.
```

```
old_products.push(product); // Adding new element: used every time when adding a new product to the cart (other than the first time).
products.splice(product_index, 1); // Deleting: when users delete an item from the cart, the corresponding element is also deleted from the array.
```

3. **getElementsByTagName(tag)** can return a collection of elements with a specified tag. This concept is useful when trying to access multiple elements with similar properties. This concept is useful when trying to access multiple elements with similar properties. In contrast, retrieving one by one is difficult to manage and inflexible. An example:

```
var selects = shop_div.getElementsByTagName("select"); // make it easy to access all choices that users make, such as type of glazing and flavor
```

4. **js vs. JSON:** I used to be confused about the difference between these two. Through doing this assignment, I feel I have a much better mental model of how js and JSON are related and work together. JSON is the format for storing and transmitting data. JS objects can be converted to JSON. Similarly, Jason format can be converted back to JS objects. For example:

```
JSON.stringify(products) //JavaScript change to JSON

JSON.parse(pro) //JSON convert back to JavaScript object
```

5. **For loop in js:** can be used to efficiently accomplish repetitive actions, for example:

```
for(var i = 0; i < products.length; i++){...} // stepping through all elements in the array and performing desired task until loop is broken. Used to display all products that a user has selected on the cart page.
```