Lily Huang

Assignment 6B Reflection

Bugs/Issues

Indexes

One bug I encountered had to do with HTML guidelines. On the cart page, the select tag needs an ID, but an ID must be unique. This caused an issue because each product card has its own select tag, and therefore needs a unique ID. In order to generate the cards for the cart page, I used a map function, which also meant I had to iteratively generate a unique ID. I thought to just make the ID an integer and add 1 every iteration, but the map function did not work like a for loop. The integer did not change with every iteration, so all the IDs ended up being the same. After some testing, I recalled from another programming class that map functions usually track the index as well. I searched it up and sure enough, I could use the index to help generate a unique ID. Now I know that the map function has that ability and can use it for future code.

Below is part of the code to make the cards.

```
 Object.values(bagItems).map((item, index) => {
                      <select name="amount${index}" id="amount${index}"
onchange="updateAmount(${index}, '${item.type}')">
 })
```

Selector Tag

I spent a lot of time trying to figure out how to have the selector tag option set to the actual amount selected. What I mean by this is that when you, for example, select 3 original rolls and go to the cart page, the select tag should show 3. By default, it showed 1, which was the first option of the select tag. To address this issue, the first thing I did was search up how to set an option in a select tag. To do that, all I needed was to write selected=”selected” in the corresponding option tag. Then, I had to figure out how to set the right option to be “selected.”

In my first attempt, I tried using querySelector (the code is shown below).

```
document.querySelector(`option[value="${item.amount}"]`)
```

This returned the correct option tag, but I couldn’t figure out how to actually set it to “selected.”

I changed my approach and tried creating a JSON object with the key being the amount and the value being the HTML code. I’d then key into the object and change its value to be the HTML code with “selected.”

```
var options = {"1":"<option value='1'>1</option>",
               "3": "<option value='3'>3</option>",
               "6":"<option value='6'>6</option>",
              "12": "<option value='12'>12</option>"}
let amount = item.amount
options[amount] = `<option value='${amount}
'selected="selected">${amount}</option>`
```

Now I had the right HTML code but I couldn't figure out an efficient way to insert the code into the HTML page itself. I could map into this object, but then I'd have a map inside a map, which is inefficient. I already didn't like how messy this code was, so I abandoned this approach.

My third approach, which was successful, was to use selectedIndex, which is a property that finds the selected option within a select tag. I found this property after doing more Googling on how the select tag works. In this approach, I inserted the select tag normally (there was no selected="selected"). Then, for each card, I'd grab the select tag, loop through its options, and set the option that equaled the correct amount using the option.selectedIndex property. This was a pretty clean solution and worked well.

```
// sets select tag to correct amount
      Object.values(bagItems).map((item, index) => {
          let sel = document.getElementById(`amount${index}`)
          let opts = sel.options
          for (var opt, i = 0; opt = opts[i]; i++) {
              if (opt.value == item.amount) {
                  opts.selectedIndex = i;
              }
          }
      })
```

From trying to solve this bug, I learned the value in trying different approaches. Also, liberal use of console.log really helps with debugging.

Programming Concepts

1. LocalStorage

I learned how to use localStorage for this assignment, including getItem, setItem, and removeItem. Since I wanted to keep each addition to the cart separate, I had to make sure each item added to local storage had a unique name, otherwise they would override each other. The "total" variable gives each added item a unique ID. I use localStorage throughout the code to make sure changes to the bag are reflected correctly.

```
// add selected item to bag in local storage
function addItemToBag(item) {
    let bagItems = JSON.parse(localStorage.getItem('itemsInBag'))

    if(bagItems != null) {
        let total = Object.keys(bagItems).length
        itemName = item.type + total
        bagItems = {
            ...bagItems,
            [itemName]: item
        }
```

```
    } else {
        let itemName = item.type + "0"
        bagItems = { [itemName]: item }
    }
    localStorage.setItem("itemsInBag", JSON.stringify(bagItems))
}
```

2. == vs ===

We were told to use === rather than ==. However, since everything grabbed from localStorage was a string, I thought to use == rather than === as I was often comparing a string to an int. Furthermore, I was confident that if I was ever comparing two values, I wouldn't be considering its type (i.e. 1 == "1" should be true for my uses). I am aware of the benefits of both == and ===, but for my project approach, I thought == was enough.

```
if (opt.value == item.amount) {
    opts.selectedIndex = i;
}
```

3. JSON

I had to convert the items in localStorage using JSON.parse and JSON.stringify because I was storing JSON objects and localStorage converts everything to a string. In order to be able to use my objects and localStorage like I expect to, I must convert them to the proper form.

```
    let bagItems = JSON.parse(localStorage.getItem('itemsInBag'))
    localStorage.setItem('itemsInBag', JSON.stringify(bagItems))
```

4. Objects

Since I used JSON.parse, I was able to iterate through its values or keys like a normal object. I used Object.values and Object.keys depending on what my intentions are.

```
// enables x button to delete item
        Object.keys(bagItems).map((itemName, index) => {
            removeItem(itemName, index, bagItems)
        })
```

5. Map

The map method is used for arrays, not objects, but since I used the programming concept of Object.values, which returns an array, I was able to map through the values of my localStorage object. The ability to track the index within a map method also helped me create unique ID tags for my select tags.

```
// inserts item cards into HTML
        Object.values(bagItems).map((item, index) => {
            // make sure img name is matches assets/image name
            let imgName = item.type.toLowerCase()
            orderContainer.innerHTML += `
```

```
            <div class="cart-card-container">
                <img src="assets/${imgName}.png" alt="${item.type}" />

                 <div class="cart-card-content">
                    <h3>${item.type}, ${item.glaze}</h3>
                    <p id="cost${index}">$${item.cost*item.amount}</p>

                    <label for="amount${index}"></label>

                    <select name="amount${index}" id="amount${index}"
onchange="updateAmount(${index}, '${item.type}')">
                        <option value="1">1</option>
                        <option value="3">3</option>
                        <option value="6">6</option>
                        <option value="12">12</option>
                    </select>
                    <br />
                </div>
             <span class="delete">&times;</span>
            </div>
            `
        })
```