1. localStorage allows you to save values in a web browser so that even if you move to a different page, this data is not lost. I love how easy it is to use as all you have to do is call it, make whatever change you want (in this example, I added a new item by pushing it into an array), and update it to reflect the change on HTML.
    a.
```
arrayOfItems = JSON.parse(localStorage.getItem("globalCart")); //get global
arrayOfItems.push(newOriginal); //add new item
localStorage.setItem("globalCart", JSON.stringify(arrayOfItems)); //update global
```

2. You can use ++ or -- to increase or decrease a value that you set, respectively. This was also easy to use as all I had to due to declare a variable and increase or decrease it using ++ or --.
    a.
```
$('#addtocart').click(function(){
    var count = localStorage.getItem("cartNumber"); //get global
    count++; //update count
    localStorage.setItem("cartNumber",count); //update global
    updateCartIndicator();
function minusCart(){
    var cart = localStorage.getItem("cartNumber"); //get global
    cart--; //update cart
    localStorage.setItem("cartNumber",cart); //update global
    updateCartIndicator();
```

3. I learned how to use splice for the first time while doing this assignment. Using splice made it much easier to use arrays as I could designate how many items to replace at which index. For this example, since the index number was not fixed, I use ‘i’ instead of an actual number.
    a.
```
function removeElementByIndex(i){
    arrayOfItems = JSON.parse(localStorage.getItem("globalCart")); //get global
    arrayOfItems.splice(i,1); //remove item
    localStorage.setItem("globalCart", JSON.stringify(arrayOfItems)); //update global
    showCart();
    minusCart();
}
```

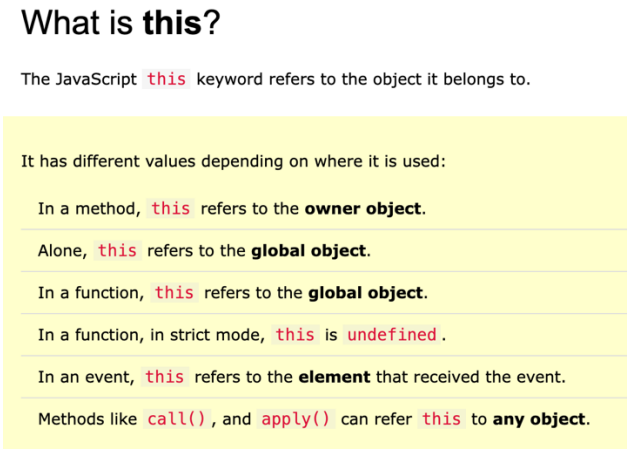
4. jQuery makes changing HTML attributes a lot easier as all you have to do is decide the target, action, which type of change (attr and text in my case), and how you wish to pursue the change. This seems a lot more straightforward and cleaner than using getElementById and changing attributes that way.

a.

```
$('#none').click(function(){
    $('.indiimage').attr('src','./images/original.png')
});
$('#sugar').click(function(){
    $('.indiimage').attr('src','./images/sugar.png')
});
$('#vanilla').click(function(){
    $('.indiimage').attr('src','./images/vanilla.png')
});
$('#chocolate').click(function(){
    $('.indiimage').attr('src','./images/chocolate.png')
});

$('#amt1').click(function(){
    $('.indiprice').text("$4.50");
});
$('#amt3').click(function(){
    $('.indiprice').text("$13.50");
});
$('#amt6').click(function(){
    $('.indiprice').text("$27.00");
});
$('#amt12').click(function(){
    $('.indiprice').text("$54.00");
});
```

5. The concept of 'this' was also extremely confusing at first because it seemed to have so many different use cases.



However, I soon realized that this is just simply referring to the object itself so that we can use it in multiple places across the code.

a.

```
const item = class {
    constructor (itemname, itemglaze, itemcount, itemprice, itemphoto) {
        this.itemname = itemname;
        this.itemglaze = itemglaze;
        this.itemcount = itemcount;
        this.itemprice = itemprice;
        this.itemphoto = itemphoto;
    }
}
```