Reflection

Major challenge: use localStorage to store items users select and display the correct number of items in the shopping cart.

Challenge 1: store array

One challenge that I've encountered was to store the information of items that the user selected and display them on other pages. For example, after the user adds one item to the shopping cart, the shopping cart page should display the corresponding item. However, although I used localStorage, the information was still lost when I go to the new page.

I checked this post on StackOverflow:
https://stackoverflow.com/questions/3357553/how-do-i-store-an-array-in-localstorage
and realized that the issue was localStorage.setitem can only take string as input, not list or array. If I want to store an array, I should use json.stringify when using setitem. Correspondingly, I should also use json.parse for getitem. I did so and the problem was solved.

Challenge 2: correct items number

The number of items displayed in the circle at the top right corner of the page should be consistent throughout pages and indicates the correct number of items in the shopping cart. It should also be updated appropriately in terms of 'add to cart' feature in the product detail page and 'remove' and quantity change features on the shopping cart page. It's hard to implement all of these features at once, so I divided them into individual tasks and then implement them one by one. Once I solve one task, I move on to the next one. Iteratively, I managed to complete all the shopping cart features.

Other challenges: general implementation

There are many cases where I am not confident about how to implement certain features using Javascript. For example, using javascript to create dom tree, and formatting strings, etc. Whenever I encountered these issues, I googled with keywords immediately and found solutions online. I referenced StackOverflow and W3School a lot.

Programming Concepts

1. Top-down design: top-down design is a common practice in programming, where a big problem is divided into smaller problems, which is represented as the main function that calls other helper functions.
    a. Main function:
        i.
```
function removeItem(removeButton)
{
  var index = removeButton.getAttribute("value");
  var items = JSON.parse(localStorage.getItem("itemsInCart"));
  items.splice(index,1);
  localStorage.setItem("itemsInCart", JSON.stringify(items));
  /* Remove row from DOM and recalc cart total */
  var productRow = $(removeButton).parent().parent();
  productRow.slideUp(300, function() {
    productRow.remove();
    recalculateCart();
  });
  updateIndex();
  storeCartNumber();
}
```
    b. Helper function:
        i.
```
/* Recalculate cart */
function recalculateCart()
{
  var subtotal = 0;

  /* Sum up row totals */
  $('.product').each(function () {
    subtotal += parseFloat($(this).children('.product-line-price').text());
  });

  /* Calculate totals */
  var tax = subtotal *  0.05;
  var shipping = (subtotal > 0 ? 15.00 : 0);
  var total = subtotal + tax + shipping;

  /* Update totals display */
```
        ii.
```
function storeCartNumber() {
  var items = JSON.parse(localStorage.getItem('itemsInCart'));
  var counter = 0;
  for (var i=0; i<items.length; i++) {
    counter += parseInt(items[i].quantity);
  }
  var cart_n = document.getElementsByClassName('count');
  cart_n[0].innerHTML = counter;
}

function updateIndex() {
  var x = document.getElementsByClassName('remove-product');
  for (var i=0; i<x.length; i++) {
    x[i].setAttribute('value', i);
  }
}
```
        iii. In the example above, I used top-down design for updating the shopping cart page. The main function removeItem updates the shopping cart page by removing items, and three helper functions (recalculateCart, undateIndex, and storeCartNumber) do the smaller tasks of updating the shopping cart page separately: 1. recalculate the total cost, 2. update the index of the rest items in the shopping cart, 3. and update the number of

items in the shopping cart displayed in the circle at the top right corner of the page.

2. Global Scope v.s. Local Scope: variables in the glocal scope can be accessed outside of functions, while variables in the local scope can only be accessed within the functions that declare them.
    a. Global:
    ```
    // two global variables
    var colors = ["strawberry", "crazyberry", "blackberry", "fire orange"];
    var sizes = ["tiny", "small", "medium", "large"];
    ```
    ```
    size.className = "product-size";
    size.innerHTML = sizes[items[i].size];
    var color = document.createElement("div");
    color.className = "product-color";
    color.innerHTML = colors[items[i].color];
    ```
    Local:
    ```
    function recalculateCart()
    {
      var subtotal = 0;

      /* Sum up row totals */
      $('.product').each(function () {
        subtotal += parseFloat($(this).children('.product-line-price').text());
      });

      /* Calculate totals */
      var tax = subtotal *  0.05;
      var shipping = (subtotal > 0 ? 15.00 : 0);
      var total = subtotal + tax + shipping;
    ```
    b. In the example above, colors and sizes are global variables, while tax, shipping, and total are local variables.
3. Prototype: prototype is a feature in javascript programming where multiple objects can be declared.
    a.
    ```
    function Item(name, price, size, color, image, alt, quantity) {
      this.item_name = name;
      this.price = price.substring(1, );
      this.size = size;
      this.color = color;
      this.image = image;
      this.alt = alt;
      this.quantity = quantity;
    }
    ```
    b.
    ```
    var itemObject = new Item(item_name, price, size_item, color_item, image, alt, quantity);
    ```
    c.
    ```
    var img = document.createElement("img");
    img.src = items[i].image;
    img.alt = items[i].alt;
    ```
    d. In the example above, Item is the object constructor and itemObject is an instance of the Item object.
4. Dom creation and modification: use javascript to modify HTML content to create dynamic web pages that can change based on user interaction.
    a. create(partial):

b.

```
// create a product element
function createProductElement(i, items) {
      var product = document.createElement("div");
      product.className = "product";

      // image
      var image = document.createElement("div");
      image.className = "product-image";
      var img = document.createElement("img");
      img.src = items[i].image;
      img.alt = items[i].alt;
      image.appendChild(img);
      product.appendChild(image);

      // details
      var details = document.createElement("div");
      details.className="product-details";
      var title = document.createElement("div");
      title.className = "product-title";
      title.innerHTML = items[i].item_name;
      var size = document.createElement("div");
      size.className = "product-size";
      size.innerHTML = sizes[items[i].size];
      var color = document.createElement("div");
      color.className = "product-color";
      color.innerHTML = colors[items[i].color];
      details.appendChild(title);
      details.appendChild(size);
      details.appendChild(color);
      product.appendChild(details);

      // price
      var price = document.createElement("div");
      price.className = "product-price";
      price.innerHTML = items[i].price;
```

c. In the example above, I use javascript to create a product for the shopping cart page.

d. Modify

e.

```
function storeCartNumber() {
  var items = JSON.parse(localStorage.getItem('itemsInCart'));
  var counter = 0;
  for (var i=0; i<items.length; i++) {
    counter += parseInt(items[i].quantity);
  }
  var cart_n = document.getElementsByClassName('count');
  cart_n[0].innerHTML = counter;
}
```

f. In the example above, I use javascript to get a class element from the HTML and modify its inner HTML text.

5. Events and Event Handling: most if not all code run by a web page is triggered by an event of some type.
    a. Example (partial)

b.

```
$('.cart-btn').on('click', function() {
  var cartBtn = this;
  var cartCountPosition = $(cartCount).offset();
  var btnPosition = $(this).offset();
  var leftPos =
    cartCountPosition.left < btnPosition.left ? btnPosition.left - (btnPosition.left - cartCountPosition.left)
      : cartCountPosition.left;
  var topPos = cartCountPosition.top < btnPosition.top  ? cartCountPosition.top : cartCountPosition.top;

  var quantity = parseInt($('.product-quantity input').val());
  $(cartBtn)
```

c. In the example above, I create an event handler that creates the animation of adding the item to the shopping cart for the even ‘onclick’ for the button ‘cart-btn’.