Assignment 6 – Adding Functionality to a Website with JavaScript

Ran Cui
Andrew ID: rancui

Reflection
What challenges or bugs did you encounter? How did you overcome these challenges?

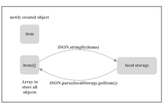
In assignment5, I finished removing, adding items function, and changing the amount of the items. However, I didn't save the operation I did in the local storage, which means, the items I added to the cart will be removed whenever I refresh the page.
In my design, there are different product pages. I also want to display the system status of the cart when users buy items on the product page. So, I definitely need to **keep the operation I did to the cart to local storage** and **sync the display of the cart.**

Solution:

Store item info in an object
In this function, whenever the user clicks the add to cart button, a new object "addedItem" will be saved as an object, and the addItemLocalStorage() function will save this object into local storage

```
function addToCartClicked(event){
  var button = event.target
  let shopItem = button.parentElement.parentElement
  var title = document.getElementsByClassName('title')[0].innerText
  var amount = document.getElementsByClassName('quantities')[0].value
  var price = document.getElementsByClassName('itemPrice')[0].innerText
  var flavours = document.getElementsByClassName('flavours')[0].value
  console.log(flavours)
  var img = document.querySelector('img').src
  var addedItem = {
      itemName: title,
      itemNo: amount,
      itemPrice: price,
      itemFlavours: flavours,
      itemImg: img
  };
  addItemLocalStorage(addedItem);
  updateCartTotal()
}
```

Save the object in an array
I use these two functions to store the newly created item object into local storage.
items[] here is an array store different item objects.
JSON.parse: The JSON.parse() method parses a JSON string, constructing the JavaScript value or object described by the string.




```
function addItemLocalStorage(item) {
  console.log(item);
  let items = getItemsFromStorage();
      console.log(items);
  for(var i = 0; i < items.length; i++) {
      console.log(items[i]);
    if(items[i].itemName == item.itemName && items[i].itemFlavours ==
      item.itemFlavours){
      alert("This item has already been added to the cart");
      return;
    }
  }
  items.push(item);
  localStorage.setItem('items', JSON.stringify(items));
  addItemToCart()
}
console.log(localStorage);
```

```
function getItemsFromStorage() {
  let items;
  const items$ = localStorage.getItem('items');
  // If no items in storage, return an empty array
  if(items$ === null) {
      items = [];
  } else {
      items = JSON.parse( items$ );
  }
  return items;
}
```

Change quantity
Changing quantity is also affected the localStorage. In other words, whenever the users change the quantity of the product they buy, they need to change the information in the local storage. The basic idea is
First, find the object, get the info of the object and store it to a new object. Second, change the quantity of the new object and replace the old object with the new one. How to find the object? I use the item name and the flavor(glazing) to locate a specific item, which means, you can only add the same product with the same glazing once in the cart.
I will also mention removing the object from local storage here. You can see from the code. I find the specific object through its name and glazing, when I find it, I use the splice function to remove it from local storage.

```
function removeFromLocalStorage(item, name, flavour){
  let items$ = getItemsFromStorage();
console.log(items$)
  for(var i = 0; i< items$.length; i++){
    console.log(items$[i].itemName + " " + items$[i].itemFlavours + " " + name +
     " " + flavour)
    if(items$[i].itemName === name && items$[i].itemFlavours === flavour){
      items$.splice(i, 1);
      console.log(items$[i])
    }
  }
  // Add the rest of the array
  localStorage.setItem('items', JSON.stringify(items$));
}
```

Assignment 6 – Adding Functionality to a Website with JavaScript

Ran Cui
Andrew ID: rancui

Separate display(HTML changes) from local storage changes

For javascript, the code should separate the function to add items on the page from add items to the local storage. They are totally different. For a complete shopping cart, it should have functions both for adding/ removing/ changing items for display and for local storage. For display, it is how you change the HTML, for example, you create new for the HTML content and appendChild to its parent node. The code on the right side shows how to change the display(HTML).

```
function addItemToCart(){
  var myItems = getItemsFromStorage();
var cartItemContainer = document.
    getElementsByClassName('cartItemContainer')[0]
for(var i = 0; i < myItems.length; i++) {
  console.log(myItems)
  }

    var thisItem = myItems.pop();
    var cartItem = document.createElement('div')
    var cartRowContent = `
    <div class = "cartItem">
    <p class = "cart-item-title">${thisItem.itemName}</p>
    <p class = "cart-item-flavour">${thisItem.itemFlavours}</p>

    <select class ="quantities2">
    <option value="${thisItem.itemNo}">${thisItem.itemNo}</option>
    <option value="1">1</option>
    <option value="2">2</option>
    <option value="3">3</option>
    <option value="4">4</option>
    <option value="12">12</option>
    </select>
    <p class = "priceElement">${thisItem.itemPrice}</p>
    <div>
    <button class="removeItem">Remove</button>
    </div>
    </div>

    cartItem.innerHTML = cartRowContent

    cartItem.getElementsByClassName("removeItem")[0]
      .addEventListener('click', removeCartItem)
    cartItem.getElementsByClassName('quantities2')[0]
      .addEventListener('change', quantityChanged)
    cartItemContainer.appendChild(cartItem);
    updateCartTotal();
}
```

Total Expense

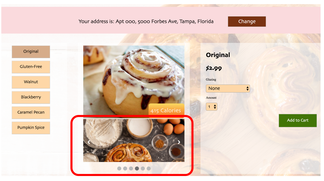
Another thing need to be set and get from local storage is the total expense. The code on the right side shows how I update the total expense and store it in the local storage.

```
function updateCartPage(){
  var cartItemContainer = document.getElementsByClassName("itemsInCart")[0];
  var cartItems = cartItemContainer.getElementsByClassName("item");
  var total = 0;

  for(var i=0;i<cartItems.length;i++){
    var item = cartItems[i];
    var priceElement = item.getElementsByClassName('cartItemPrice')[0];
    var quantityElement = item.getElementsByClassName('amountCartSelect')[0];
    quantity = quantityElement.value
    var euroPrice = parseFloat(priceElement.innerText.replace('$',''))+ quantity
    total = total + euroPrice;
  }

  total = Math.round(total*100)/100;
  document.getElementsByClassName('totalMoney')[0].innerHTML = '$'+total;
  localStorage.setItem('totalPrice', total)
}
```

Carousel

The basic idea of the carousel is when you click the next button, it shows the next slide; and when you click the prev button, it shows the previous slide. The most difficult point of the carousel is how to achieve the loop, it means if you click the next button when it has already gone to the last slide, it will be back to the first slide. The prev button is similar. The code attached here is showing how I achieve the loop.

```
function showSlides(n) {
  var i;
  var slides = document.getElementsByClassName("mySlides");
  var dots = document.getElementsByClassName("dot");
  if (n > slides.length ) {slideIndex = 1}
  if (n < 1) {slideIndex = slides.length}
  for (i = 0; i < slides.length; i++) {
      slides[i].style.display = "none";
  }
  for (i = 0; i < dots.length; i++) {
      dots[i].className = dots[i].className.replace(" active", "");
  }
  slides[slideIndex-1].style.display = "block";
  dots[slideIndex-1].className += " active";
}
```




```
// Next/previous controls
function plusSlides() {
  showSlides(++slideIndex );
}
function minusSlides() {
  showSlides(--slideIndex);
}
```

HTML call the plus/minus slide function

```
<!-- Next and previous buttons -->
<a class="prev" onclick="minusSlides()">&#10094;</a>
<a class="next" onclick="plusSlides()">&#10095;</a>
```

Assignment 6 – Adding Functionality to a Website with JavaScript

Ran Cui
Andrew ID: rancui

Programming Concepts

1. DOM - Document Object Model
The Document Object Model (DOM) is an application programming interface for valid HTML. It defines the logical structure of documents and the way a document is accessed and manipulated.

I use the document keyword a lot to find the specific part I need from the HTML file. Here on the right, you can see I got the info of the item I am going to add to the cart use the "document" keyword. And there are functions like getElementById(), getElementsByClassname(), querySelectorAll() to get the specific node.

```
function addToCartClicked(event){
  var button = event.target
  let shopItem = button.parentElement.parentElement
  var title = document.getElementsByClassName('title')[0].innerText
  var amount = document.getElementsByClassName('quantities')[0].value
  var price = document.getElementsByClassName('itemPrice')[0].innerText
  var flavours = document.getElementsByClassName('flavours')[0].value
  console.log(flavours)
  var img = document.querySelector('img').src
  var addedItem = {
      itemName: title,
      itemNo: amount,
      itemPrice: price,
      itemFlavours: flavours,
      itemImg: img
  };
  addItemLocalStorage(addedItem);
  updateCartTotal()
}
```

2. Event Listener
The EventListener interface represents an object that can handle an event dispatched by an EventTarget object. The code on the right shows how I add an event to a button, and when the button is clicked, the function will be called and operated.
For example, when you click "add to cart", the addToCartClicked function will be called.

```
function ready(){
  getFromLocalStorage()

  var removeButton = document.getElementsByClassName("removeItem");
  for(var i=0;i<removeButton.length;i++){
    var button = removeButton[i];
    button.addEventListener('click',removeCartItem);
  }

  var addToCartButton = document.getElementsByClassName("cartButton");
  for(var i=0;i<addToCartButton.length;i++){
    var button = addToCartButton[i];
    button.addEventListener('click',addToCartClicked);
  }

  var checkOutButton = document.getElementsByClassName("checkoutbutton")
  for(var i = 0; i < checkOutButton.length; i++){
    var button = checkOutButton[i]
    button.addEventListener('click', checkOutButtonClicked)
  }

}
```

3. Local Storage
The localStorage property allows you to access a Storage object for the Document's origin; the stored data is saved across browser sessions. The keys and the values in the local storage are always strings. A Storage object which can be used to access the current origin's local storage space.

```
function addItemLocalStorage(item) {
    console.log(item);
      let items = getItemsFromStorage();
        console.log(items);
      for(var i = 0; i < items.length; i++) {
          console.log(items[i]);
        if(items[i].itemName == item.itemName && items[i].itemFlavours ==
          item.itemFlavours){
          alert("This item has already been added to the cart");
          return;
        }
      }
    items.push(item);
    localStorage.setItem('items', JSON.stringify(items));
    addItemToCart()
  console.log(localStorage);
}
```

4. Window
Though here in the code on the right side is using localStorage directly, the Window interface concept is another concept that I want to mention. The Window interface represents a window containing a DOM document; the document property points to the DOM document loaded in that window. A window for a given document can be obtained using the document.defaultView property.

5. JSON
A common use of JSON is to exchange data to/from a web server. When receiving data from a web server, the data is always a string. Parse the data with JSON.parse(), and the data becomes a JavaScript object. Here is an illustration I created for the structure in my code, and how I store data in the local storage.