**Shopping list: Render the list**

**Objective:** By the end of this checkpoint, you can translate pseudocode into JavaScript.

In this checkpoint, we'll continue building out a well-architected shopping list app, specifically focusing on the first user story - rendering the list. We'll plan out and write the code to display the items stored in our list.

At the end of this checkpoint, you’ll complete 1 assignment:

1. For the remaining 3 user stories, plan each function in pseudocode.

**Key Terms**

* Render

**Rendering the shopping list**

In the last checkpoint, we left off with an implemented handleShoppingList function and stubs for our 4 functions for displaying and altering our shopping list (which you will find in the index.js file in the repl.it below).

If you have problems accessing the embedded repl.it below, please [**try opening it in a separate browser tab by clicking here**](https://repl.it/@thinkful/jquery-shopping-list-walkthrough-3).

In this checkpoint, we'll implement the renderShoppingList function. The responsibility of this function is to render the current state of the shopping list STORE to the DOM. We call renderShoppingList inside of handleShoppingList, which runs after page load, in order to initially render the shopping list. We'll also call renderShoppingList in our functions for altering the list. After adding, deleting, or toggling an item's checked property, we'll call renderShoppingList to update the DOM with the current state of the list.

**Describing renderShoppingList in plain language**

At the moment, our renderShoppingList function doesn't do much.

**function** **renderShoppingList**() {

*// render the shopping list in the DOM*

console.log('`renderShoppingList` ran');

}

We've got a very high-level description of what the function should do ("render the shopping list in the DOM"), but before jumping into coding, we should determine the steps this function will need to take in order to render the shopping list, stating them in plain language that will then guide us as we code.

Let's first ask *where* the shopping list should be rendered. Taking a look at index.html, below the form for adding new list items, we have:

<ul class="shopping-list js-shopping-list">

</ul>

This unordered list is where we'll render our list items. Our jQuery code will need to target the .js-shopping-list element, inserting <li>s inside.

Now let's think about the steps we'll need to take to display our list in this element. We already know that we're using the STORE array to house our data about the list, so that's our starting point. Ultimately we need STORE to get translated into an HTML string that can be inserted into the right place in the DOM. So the pseudocode version of renderShoppingList would be something like this:

* For each item in STORE, generate a string representing an <li> with:
  + the item name rendered as inner text
  + the item's index in the STORE set as a data attribute on the <li> (more on that in a moment)
  + the item's checked state (true or false) rendered as the presence or absence of a CSS class for indicating checked items (specifically, .shopping-item\_\_checked from index.css)
* Join together the individual item strings into one long string
* Insert the <li>s string inside the .js-shopping-list <ul> in the DOM.

As we begin to implement this function, there will be additional moments to consider, but this is about as far as we'll get with pseudocode.

**First things first: hardcoded list items**

Looking at our pseudocode, the easiest moment to implement is appending the string representing list items to the DOM. We can initially hardcode a value to be inserted in the DOM and focus on our implementation of the DOM traversal and insertion. Have a look at index.js in the repl.it below.

If you have problems accessing the embedded repl.it below, please [**try opening it in a separate browser tab by clicking here**](https://repl.it/@thinkful/jquery-shopping-list-walkthrough-4).

Inside of renderShoppingList we've hardcoded a string representing a single list item (shoppingListItemsString). In a moment, we'll replace that hardcoded value with one dynamically generated by a function whose purpose is to generate a string representing shopping list items.

renderShoppingList ends by targeting '.js-shopping-list' (the class of the shopping list <ul> in index.html), and setting its inner HTML to the value of shoppingListsItemsString. If you hit play on the repl.it, you can see that the hardcoded single list item gets rendered.

This means that we already know that renderShoppingList's end behavior is correctly wired. Line 17 ($('.js-shopping-list').html(shoppingListItemsString);) doesn't care *how* shoppingListItemsString gets generated, it just cares *that* it's generated.

**Generating the list item string**

The first step we'll take is to create a new function called generateShoppingItemsString. For now, it will return a hardcoded string of <li>s, but we'll call it from inside renderShoppingList.

If you have problems accessing the embedded repl.it below, please [**try opening it in a separate browser tab by clicking here**](https://repl.it/@thinkful/jquery-shopping-list-walkthrough-5).

At this point, even though we've got more work to do to generate a string representing our shopping list items, our renderShoppingList function is complete, in the sense that we won't need to touch that code again. It's wired up to render whatever string value gets returned by generateShoppingItemsString.

In order to generate this string, we'll need to iterate over each item in STORE and generate an <li> string with the right text and class set to reflect the properties of the item. Our first pass of this will be to map over the items in STORE, calling a new function (generateItemElement) on each one to generate the item string. We'll then join these individual item strings into one big string to be returned by generateShoppingItemsString.

If you have problems accessing the embedded repl.it below, please [**try opening it in a separate browser tab by clicking here**](https://repl.it/@thinkful/jquery-shopping-list-walkthrough-6).

We now pass STORE to generateShoppingItemsString as the shoppingList parameter. Since this function won't modify the shopping list (aka, there are no subtle side-effects to make explicit), we've chosen to pass it as a function parameter, rather than directly referencing the global STORE inside of generateShoppingItemsString.

In generateShoppingItemsString, we create a new array of item strings (items) by mapping over shoppingList and calling a new function, generateItemElement on each item. generateShoppingItemsString will return a single string that joins together the individual item strings (return items.join()).

In our first pass at generateItemElement, we create a string representing a list element that displays the item name but does not support adding, checking, unchecking, or deleting items. We'll tackle those features soon enough, but before doing that, click play on the repl.it above, and you'll see that we're now displaying the name of each item in STORE in the DOM.

**Generating individual list item strings**

The final thing to do in this assignment is fully implement our generateItemElement function. At the moment, it is only returning the name of each item.

To complete this function, we need to be generating list items that support checking and deletion. We won't fully implement checking and deletion until later, but we want our rendered shopping list items to contain buttons for checking and deleting, as well as a data attribute indicating their index in the STORE array.

Here's how that looks.

If you have problems accessing the embedded repl.it below, please [**try opening it in a separate browser tab by clicking here**](https://repl.it/@thinkful/jquery-shopping-list-walkthrough-v2-7).

Press play on this repl.it, and you'll see that we've now got a full implementation of our list rendering behavior. For items whose checked property is true in STORE, strikethrough styling is applied. If you inspect the list items using DevTools, you'll find that each one has a data-item-id attribute. This is a randomly generated unique identifier for each shopping list item, so that it allows us to create a link between the DOM and the actual object in our STORE. Each item also has a "check" and "delete" button.

***UNIQUE IDS ARE VERY COMMONLY USED TO TRACK DATA AS IT MOVES AROUND A SINGLE OR MULTIPLE SYSTEMS. WE'RE IMPORTING A SECOND LIBRARY (CUID) IN THE HTML TO GENERATE THE UNIQUE ID, WHICH USES AN ALGORITHM THAT MAKES THE LIKELIHOOD OF CREATING TWO IDENTICAL IDS ALMOST MATHEMATICALLY IMPOSSIBLE.***

Looking at index.js, here is what is new:

**function** **generateItemElement**(item) {

**return** `

<li class="js-item-index-element"

data-item-id="${item.id}">

<span class="shopping-item js-shopping-item ${item.checked ? 'shopping-item\_\_checked' : ''}">${item.name}</span>

<div class="shopping-item-controls">

<button class="shopping-item-toggle js-item-toggle">

<span class="button-label">check</span>

</button>

<button class="shopping-item-delete js-item-delete">

<span class="button-label">delete</span>

</button>

</div>

</li>`;

**function** **generateShoppingItemsString**(shoppingList) {

console.log("Generating shopping list element");

**const** items = shoppingList.map((item) => generateItemElement(item));

**return** items.join();

}

Inside the generateShoppingItemsString function, we target the element for cloning, and inside generateItemElement we set the name, the checked class, and the data attribute for the item's index in STORE.

Inside generateShoppingItemsString, for each item we map over, we return the result of calling generateItemElement(item). Looking at that function, we use ES6 template strings to generate a list item.

And with that, we've implemented the most complex feature of our app: rendering the shopping list. Next up, we'll implement the behavior for adding new shopping list items. Because we have created a reusable function for rendering the shopping list, adding list items will be relatively easy to implement.

**Assignment**

Create a document that you can share with your mentor (like a [**Google Doc**](https://www.google.com/docs/about/), [**DropBox Paper**](https://www.dropbox.com/paper), or [**Gist**](https://gist.github.com/)).

For the 3 remaining user stories (add items, check/uncheck items, and delete items), plan out each function in pseudocode (plain language).

Submit a link to your document below. The link will be shared with your mentor. Be sure to adjust any sharing permissions on the doc so that your mentor can view it.