Software Developer Course Assessment

Quantitative Assessment Practice

Course Name: Programming with JavaScript

Current Week: (2023/05/29)

## Introduction:

The purpose of this assessment is to help us understand how the class is doing in terms of the review material that we have covered during the previous couple of weeks. The only purpose of this assessment is for us to improve our approach to review and ensure that what we’re currently doing is an effective strategy. Completion of this assessment is mandatory - if you don’t submit a solution, it will be marked as incomplete. If you do submit a solution, it will be marked as complete, as you will receive full marks.

Again, the goal here is to help you all in the best way that we can, so please do be honest when answering the questions related to how long it took, which resources you used, etc. And please ensure that you do your own work – don't just copy off a friend to get it done, earnestly do your best with it. If you can’t get it completely working, give us what you have. While it will be graded, the grade will not count against you, it’s just a way for us to see where everybody is, and to know which concepts, if any, we, as a class, may be struggling with.

Deadline: You will have until the end of the day on **Wednesday June 7 ,2023 (4:00pm)** to submit your assessment solutions. Please ensure you answer all the questions outlined in the instructions portion of this document as well in your submission.

Instructions: Your name: \_Evan Davies\_\_\_\_\_\_\_\_\_

You are allowed to complete the assessment problems below in whatever way you can but please answer the following questions/points as part of your submission:

1. How many hours did it take you to complete this assessment? (Please keep try to keep track of how many hours you have spent working on each individual part of this assessment as best you can - an estimation is fine; we just want a rough idea.)

Answer: 7-8 hours total. 2 hours making the initial lists of products and categories, around 4 and a half hours working on the html and javascript, and an additional hour making extra products.

1. What online resources you have used? (My lectures, YouTube, Stack overflow etc.)

Answer: Mostly used a combination of your lectures, previous programs that we’ve made during your lectures, and a whole lot of Google. Specifically the website I found most useful was w3schools.com.

1. Did you need to ask any of your friends in solving the problems. (If yes, please mention name of the friend. They must be amongst your class fellows.)

Answer: Yes. Although, a lot of the time it was hard to really get advice that worked, since the people trying to help did things quite a bit differently than I did. Brandon and Phil were the ones who tried to help most often, including some others, although I only know some by usernames.

1. Did you need to ask questions to any of your instructors? If so, how many questions did you ask (or how many help sessions did you require)?

Answer: I considered it, but the questions I had I was able to solve myself in time.

1. Rate (subjectively) the difficulty of Making this all! from your own perspective, and whether you feel confident that you can solve a similar but different problem requiring some of the same techniques in the future now that you’ve completed this one.

Answer: It was not easy, but it could have been a lot harder. It was a challenge, but not so challenging that it felt impossible. After what I’ve learned while doing this QAP, I do believe I could do something similar in the future, in less time.

# Overview

This qap is designed to have you practice working with HTML and the DOM in order to create both *static* and *dynamic* web content.

You are asked to prototype a fictional online store. Your store will sell several different product categories, and many products in those categories. Because a store’s products and categories will change frequently, we often separate our data from its UI representation. This allows us to quickly make changes and have the store’s web site always use the most current inventory information.

*NOTE: in a real e-commerce web store, our data would be stored in a database. We will simulate working with a database by using JavaScript Objects and Arrays.*

## Pick Your Store and Product Inventory

You need to decide on the following details for your store:

* **Name**: what is your store called? Pick something unique and relevant to your products.
* **Slogan or Description**: what is your store’s slogan or what is a short description of what you sell? This will help a user to determine if your store’s site is worth reading.
* **Products**: what does your store sell? Baked goods? Ferraris? Cosmetics? Candles? Sneakers? It’s up to you! Pick something that no one else is going to choose. No two students can use the same store products. Your store must have a **minimum of 20 items, and at least 2 of these should be Discontinued (see below)**. You are free to make things up. Be creative.
* **Product Categories**: your products will fit into one or more categories. For example, if you are selling Winter Gloves, you might have the following categories: “Men’s Gloves”, “Women’s Gloves”, and “Children’s Gloves” or maybe “Active”, “Formal”, “Decorative”. Your store should have a **minimum of 4 categories**. Each product must belong to one or more of these categories.

## Modelling your Store Data

### Categories

Each category needs two things:

* **id**: a unique **String** that identifies this category. For example: “c1” or “category-01” or “V1StGXR8”. It doesn’t matter what format you choose as long as each category has its own unique value.
* **description**: a human-readable **String** meant for display. While the id is a unique key for the data used by programs, the description is meant to be shown to a user. For example: “Men’s Shoes” or “Pickup Trucks” or “Skydiving Tours.”

### Products

Each product needs the following things:

* **id**: a unique **String** that identifies this product. For example: “p1” or “product-01” or “V1StGXR8”. It doesn’t matter what format you choose as long as each product has its own, unique value. Also, make sure the product id and category id are different.
* **name**: a short **String** that names the product (e.g., "Gingerbread Cookie")
* **description**: a longer **String** that defines the product
* **price**: a **Number** of whole cents (i.e., an Integer value) for the product’s unit price. When we store currency data, we often do so as integers vs. floats, and convert it for display (e.g., 100 = $1.00, 5379 = $53.79)
* **discontinued**: a **Boolean** indicating whether or not the product has been discontinued. If this property is absent, your system should assume that it is NOT discontinued.
* **categories**: an **Array** that includes one or more category ids. Each product belongs to one or more categories (e.g., ["c1"] or ["c1", "c2"]). Make sure you match the category id to your format above.

Your category and product data will go in `src/categories.js` and `src/products.js` respectively. See these files for technical details about how to code your data.

Take some time now to enter all of your store’s data.

## Store Web Site HTML

Your store’s HTML file is located in `src/index.html`. A brief HTML skeleton has been created, and you are asked to fill in the rest using your information above.

Some of your site will be static (i.e., coded in HTML directly in index.html) and never change. Other parts of the site will be dynamic (i.e., created using DOM API calls at run-time) and will update in response to various events and user actions.

Here is a basic wireframe of what your site needs to include, and which parts are static or dynamic. NOTE: don’t worry too much about how it looks. Focus on the structure and functionality.

**Header: Always the same**

**Menu Navigation: Clickable Category Names**

**Dynamically Generated**

**Store Name**

*Store Slogan/Description…*

**Category1** Category2 Category3 Category4

**Category1 Name**

|  |  |  |
| --- | --- | --- |
| **Name** | **Description** | **Price** |
| Item 1 | Item 1 description… | $3.99 |
| Item 2 | Item 2 description… | $53.00 |
| Item 3 | Item 3 description… | $0.75 |
| … | … | … |

**Table Header: Always the Same**

**Selected Category Name:**

**Dynamically Set When Category is Clicked**

**Table Body: Dynamic, All Non-Discontinued Inventory Items for Chosen Category**

**Clicking Anywhere On A Row should console.log() the item**

**Price formatted to Dollars and Cents**

## Dynamic Content

All of your store’s dynamic content will be written in JavaScript in the `src/app.js` file. Here is a list of the tasks you need to complete:

1. Create an event handler to run when the page is loaded. Your function should do the following:
   1. Create all of the buttons for your store’s Categories
      1. Loop through all of your category objects and create a <button> element for each, adding it to the <nav id=”menu”>…</nav>
      2. Use each Category’s description for the button’s text
      3. When the button is clicked, show that category’s products in your <tbody>…</tbody>. See below for more details.
   2. Show a list of products in the <tbody>…</tbody> of your Table. By default, you should use your first category. See below for more details
2. Write a function that will show a product list in the <tbody>…</tbody> based on a category Object:
   1. Update the text of the Selected Category Title above your table with the category’s descripton
   2. Clear the current <tr>…</tr> rows from the <tbody>…</tbody>. HINT: innerHTML = “”
   3. Filter your products Array (i.e., use Array.prototype.filter()) to get:
      1. All products in the chosen category. HINT: use Array.prototype.includes()
      2. All products that are NOT discontinued
   4. Loop (use Array.prototype.forEach()) over your filtered product list and add them to the table’s body:
      1. Create a <tr> element
         1. Add a click handler to your <tr> that will console.log() the product’s full details whenever the user clicks it
      2. Create <td> elements for the product’s title, description, and price
         1. Convert the price in cents to dollars and cents
         2. Format it as Canadian Currency (HINT: research Intl.NumberFormat() and currency formatting)
      3. Append these <td> elements to the <tr>
      4. Append this <tr> to the <tbody>

In your solution, you will likely require all of the following:

* console.log() and NOTE that you can log Objects like so: console.log({ object })
* document.querySelector() to find elements in the DOM
* document.createElement() to create new DOM elements
* node.appendChild() to add an element as a child of a DOM node
* element.innerHTML to modify the HTML content of an element. Don’t overuse this!

You are encouraged to use what you learned in our sessions as well as previously in JS course. Write proper functions and semantic HTML.

## Submission:

When you are done with the solutions, make one folder “alan\_qap1” (in case your first name is Alan 😊 ), and put this qap1.js along with this document after answering all the questions given above(use the green color for your answers as given example)