Capstone Project - List Pagination & Filtering Study Guide

The two main parts of this project that you need to complete require you to add buttons to the bottom of the page and show different sets of student information when each button is clicked. The project start files are located at the folder “Student Project Pagination/Example Mockup”.

Go to folder “Student Project Pagination/Example Mockup” and check example.html to see how the project should look like. In this project you will be given a list of some students and you need to write a program to break all of the list into multiple pages of 10 profiles, so you can go back and fort to any of the pages.

For this project, avoid using the popular jQuery library. All code for this project must be your own, and must be plain JavaScript, often referred to as "vanilla" JavaScript.

Avoid using any other libraries, plugins or code snippets for any aspect of this project, including the pagination.

For this project the list of all students are in the html file. There is no requirements to move them into a separate database or JSON file. However, doing so will earn you 3 bonus marks.

The steps below are meant to guide you. You are free to code on your approach. However, if you find the following tips helpful, you can use them as you write your code.

Create your global variables

This project can be completed with just two global variables. Create a variable to store the student list item elements in the student list.

Create a variable to store the number of items to show on each “page”, which for this project, is 10.

Display a “page”

- Create a function to hide all the students except for the ten you want displayed on a given page.
- This function should have two parameters:
  - The list parameter to represent the actual list of students that you’ll pass in as an argument later when you call this function.
  - The page parameter to represent the page number that you’ll pass in as an argument later when you call this function.
- Inside the function:
  - Create two variables to store the start index and the end index of the list items to be displayed on the given page. To make this function dynamic and work with a list of any length, a bit of basic math can be used to determine these values.
    - Start Index = (page parameter * items per page) - items per page

        - End Index = page parameter * items per page
    - Loop over the list parameter.
        - Inside the loop, display any list item with an index that is greater than or equal to the start index variable and less than the end index variable.

Add pagination links

- Create a function that creates and appends functioning pagination links.
- This function should accept a single list parameter to represent the actual list of students that you'll pass in as an argument later when you call this function.
- Inside the function:
    - Create and append the DOM elements for the pagination links.
        - Use lines 119-137 in the examples/example-meets.html file as your template.
        - Pay close attention to how elements are nested, any necessary class names or other element attributes, and where your additions should be appended.

    - You should end up with:
        - A container DIV element with a class name of "pagination", and appended to the div element with the class name of page.
        - A nested UL element containing one LI element for every ten students in the list.
        - Tip: You can divide list.length by the max number of items per page to figure out how many pages are needed, and you can use a loop that iterates that many times to create the correct number of LI elements.
        - Each LI element should contain an A element with an href attribute of #, and text set to the page number each link will show. First link is 1. Second link is 2. And so on.
        - Tip: The loop index can be helpful in setting the text of the pagination links.

    - Add the active class name to the first pagination link initially.
    - Add a "click" event listener to each A element. A loop can be helpful here.
    - When a pagination link is clicked:
        - The active class name should be removed from all pagination links. A loop can be helpful for this step.
        - The active class name should be added to the link that was just clicked. The target property of the event object should be useful here.
        - The function to show a page should be called, passing in as arguments, the global variable for the list items, and the page number that should be shown. The text content of the A element that was just clicked can be helpful here.
- Note: Your program needs to work for any number of list items, so your solution needs to be dynamic.

Call your functions

- Call the show page function, passing in as arguments, the global variable for the list items, and the number 1 for the first page, which should be shown initially.
- Call the append page links function, passing in as an argument, the global variable for the list items.

Add code comments

- Replace the code comments in the file with your own code comments.
- The key to creating good code comments is to keep them brief, but clear, so that your fellow developers can get an idea of what's going on in your code at a glance, without having to review every line of your code.

NOTE: What you submit is what will get reviewed

Some more Detailed Notes:

The showPage Function

To hide all students and show only a particular set of ten, we could create a function that takes in parameters for the list (all of the students) it's supposed to work on, and the "page" that's supposed to be shown. A "page" here just means a list of 10 students: so the first 1-10 students would be "page 1", students 11-20 appear on "page 2", and so on.

```
const showPage = ( list, page ) => {
   /*
   Loop over items in the list parameter
   -- If the index of a list item is >= the index of the first
   item that should be shown on the page
   -- && the list item index is <= the index of the last item
   that should be shown on the page, show it
   */
   }
```

The appendPageLinks Function

Then we could use a function that creates all the pagination buttons, adds them to the DOM, and adds their functionality. So you would see a button with the number 1 which, when clicked, would show the first ten students (the first "page"). When each link is clicked, the showPage function displays the corresponding page (set of ten students), and highlights that page's link. For example, clicking the link to page 2 will display students 11 through 20 and highlight button 2.

```
const appendPageLinks = (list) => {
  /*
      1. Determine how many pages are needed for the list by dividing the
         total number of list items by the max number of items per page
      2. Create a div, give it the “pagination” class, and append it to the
         .page div
      3. Add a ul to the “pagination” div to store the pagination links
      4. for every page, add li and a tags with the page number text
      5. Add an event listener to each a tag. When they are clicked
         call the showPage function to display the appropriate page
      6. Loop over pagination links to remove active class from all links
      7. Add the active class to the link that was just clicked. You can
         identify that clicked link using event.target
      */
};
```