Project Instructions

Open the project files in your favorite text editor and preview them in the browser. Also, it's a smart idea to test your project in multiple browsers!

Download the project files.

- You are provided with several starter files:
    - An **index.html** file contains the initial HTML markup. ** *Do not make any changes to this file*.**
    - Two CSS files, **design.css** and **reset.css** inside the css folder. These files contain specific styling required to make the pagination features and list look good.
    - A **script.js** file in the js folder. You'll add code to this file. We've provided code comments to get you started.
    - Example HTML files in the examples folder:
        - The example-exceeds.html and example-meets.html files demonstrate what a meets and exceeds project will look like in the browser.
        - The 44students.html and 64 students.html files allow you to test your solution against lists of varying length.
- It can be a lot easier to write a program, if you divide it into smaller, more manageable parts that you program one at a time. The instructions below divide the project into individual sections, which are further broken up into even smaller steps. Follow along, one step at a time.

No snippets, plugins or libraries, including jQuery.

- 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.

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.
    - **Pro Tip**: Log out the variable storing the list to ensure it equals the list of li items and not the container of the li elements.
- 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.

Pro Tip: If you call this function in the early stages of building the function, you'll be able to use log statements and the Chrome dev tools console to test and check that variables, values and indexes are what you think they are.

<u>**Add Pagination links.**</u>

- 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.
        - **Pro 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.
        - **Pro 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. You can test that your solution works for any size list by opening the examples/44students.html and examples/64 students.html files, and linking your JS file.

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.

Cross-Browser Consistency

- Google Chrome has become the default development browser for most developers. With such a selection of browsers for users to choose from, it's a good idea to get in the habit of testing your projects in all modern browsers.

EXTRA CREDIT

Add search component

- Dynamically create and append a search bar. Avoid making any changes in the index.html file. You can reference the examples/example-exceeds.html file, lines 16-19, to see an example of the markup you'll create.

Add functionality to the search component

- When the "Search" button is clicked, the list is filtered by student name for those that include the search value. For example, if the name Phillip is typed into the box, list all items with a name that includes Phillip. If the letter S is typed in, all items with an S in the name will show.
- **Pro Tip**: To improve the functionality and add to the user experience, consider adding a keyup event listener to the search input so that the list filters in real time as the user types. This would be in addition to making the search button clickable since pasting text into the search bar might not trigger the keyup event.

Paginate search results

- Display pagination links based on how many search results are returned. For example: if 10 or fewer results are returned, 0 or 1 pagination links are displayed. If 22 search results are returned, 3 pagination links are displayed.

- **Pro Tip**: To paginate the search results, try storing the search results in an array that can act as a list, on which you can call your functions to show a page and append pagination links.

Handle no results returned

- If no matches are found by the search, include a message in the HTML to tell the user there are no matches.
- Note Don't use the built in alert() method here. The "No results" message must be printed to the page.