Assignment 4: jQuery DOM Traversal and Manipulation

- I have provided HTML and CSS code for this assignment. Do not modify this code unless specifically directed to in the instructions!
- I recommend using Chrome to test your assignment, since it has keyboard accessibility is set up by default.

Part A: Accessible Menu

Task 1:

- I have provided the HTML (including ARIA) and CSS for a menu and menu button. Your task is to use jQuery to make this menu functional according to W3C ARIA best practices, and according to the demo video I provided.
- Save your jQuery code in a file called **aria-menu.js** and link it to your document.
- **Guidelines:**
    - The code should be generalizable to any menu on the page. No new JS or CSS code should have to be written to add additional menus. Do not change the HTML mark up for the menu (or any other part of the page, for that matter.) Remember the advanced CSS selectors and jQuery traversal methods.
    - Try to minimize the number times you create jQuery objects; store jQuery objects in variables to avoid additional queries. In addition to being more efficient, this also improves readability of code.
    - The functionality should match that recommended by W3C ARIA specifications:

      Menu Button Pattern: https://www.w3.org/TR/wai-aria-practices/#menubutton
      Menu or Menubar Pattern: https://www.w3.org/TR/wai-aria-practices/#menu
      Menu Button Example: https://www.w3.org/TR/wai-aria-practices/examples/menu-button/menu-button-links.html

        - You can use the above example, provided as part of the specification document, for guidance. Note that for mouse users, we will set up the menu to open when the button is clicked instead of hovered upon.
        - Remember that **an ARIA role is a promise**. Specifying certain roles means we must implement certain functionality.
    - Use jQuery traversal methods to move the focus throughout the menu and apply the necessary CSS classes.
        - Some methods that may be helpful: **children, find, first, parent, closest, next, prev**. These are methods that I used in my solution, but there is more than one way to solve this, and you may find other methods helpful as well.
    - Use the CSS that I've written to style the menu. (if you want to make adjustments, check with me.)

        - Particularly, notice that I've written styles for a class called "**expanded**", and figure out how to apply this class using jQuery.
    - Make sure the menu properly handles application of the **aria-expanded** attribute. Read the specification and study to the example to see how this should be implemented.
- **Hints:**
    - Use the **on** method with the **keydown** event to listen for key presses. Use **event.which** to find out which key was pressed. Key codes: https://css-tricks.com/snippets/javascript/javascript-keycodes/
    - Remember that you can add listeners for more than one event to an element by *chaining* them!
    - Remember that you may need to prevent the default behavior for certain keys.
    - Remember that if you find yourself copying and pasting code, you can write a function and call it instead.

Hand in

- Create a zip archive with all necessary files called **a4_partA** and hand it in to the Lab 4 folder.

Checklist

- [6 marks] keyboard functionality properly implemented
- [2 marks] aria-expanded properly handled
- [2 marks] mouse functionality properly implemented

Total: 10

Part B: Random Images

Task 1:

- The **main** element of the page contains a section called "**image-gallery**". Write a jQuery script that populates this section with ten randomized images from the **images** folder, provided with the starter files. (Don't worry if some of the images are repeated.) The images used and their order should be randomly selected every time the user reloads.
    - To do this, you will need to create and insert DOM elements using jQuery. Some jQuery methods that may be useful are: **append, prepend, after, before.**
- Save your jQuery code in a file called **gallery.js.**

Task 2:

- I have provided the HTML and CSS for a random image generator that uses the **Unsplash Source** service, which allows for randomized Unsplash images to be embedded. Your task is to use jQuery (and Unsplash Source) to implement the functionality demonstrated in the demo video. The user should be able to type a search term into the text box, click the "generate" button and see the new image appear in the widget. While the image is loading, a spinning "loading" icon should display inside the image box. I've set up the CSS for this; all you have to do is use Icomoon to add the icon to the element with the class "image-container". (see the style.css file to see how this is implemented.) When the user clicks the "add to gallery" button, a copy of the image should be added to the image gallery in the main element.
    - Use the **load** event to detect when an image has loaded (incidentally, this event can have unpredictable behavior depending on the browser, so watch out.)
    - I have written the CSS for a class called "**loaded**". Figure out how to use this class in JQuery for the desired loading behavior.

- Save your jQuery code in a file called **generator.js**.

  Study the Unsplash Source syntax: https://source.unsplash.com/

- Note that there is an image generator near the top of the page and another one near the bottom. Your jQuery code should be generalized to work with any generator widget on the page.

Hand in

- Create a zip archive with all necessary files called **a4_partB** and hand it in to the Lab 4 folder.

Checklist

- [2 marks] Gallery populated with random assortment of images
- [1 marks] CSS layout of gallery
- [2 marks] Random image with search terms generated on submit
- [2 marks] Button for adding image to gallery
- [1 mark] "Loading" message or icon displayed properly

Total: 8

Additional Resources

- A very useful jQuery tutorial: http://try.jquery.com/