jQuery and Review

This lab gets you to practice many of the concepts seen throughout the semester in preparation for the final exam.

Start by taking a quick look at the included screenshots to see what your finished result might look like.

Part 0: setup

Choose root folder for the web app you'll be creating, copy in the provided files. Your source dir structure should be something like:

- (root dir)
  - jquery (empty for now)
  - review_jQuery.html
  - review_jQuery.js
  - review.html
  - review.js
  - style.css

Part 1: Practice jQuery and jQuery UI

In this part, you will be editing `review_jQuery.js` and `review_jQuery.html`.

Include jQuery UI assets:

- Go to jqeuryui.com/download and download a custom zip file:
  - choose the latest stable version
  - uncheck everything except Accordion. (When you check Accordion, other components it depends on will become checked.)
  - choose the Le Frog theme
  - everything else can be left blank/unchanged
  - Once you download and extract the zip file, copy the following to the empty jQuery dir in your app root dir: `external`, `images`, `AUTHORS.txt`, `jquery-ui.css`, `jquery-ui.js`, `LICENSE.txt`. Note that the zip file includes a `jquery.js` file under `external`.
  - Add the js and css dependencies you just downloaded to `review_jQuery.html` in correct order: the JS you write depends on jQuery UI, and jQuery UI depends on jQuery.

Turn the second section of your html file into a jquery-ui accordion, which is a group of elements that can collapse and expand on click:

- Remember that your code should only run after the DOM is loaded. Set that up using jQuery.
- To create an accordion:

```
$("selector for parent el").accordion(
  // you will fill in some options
  {option1: value, option2: value}
);
```

- Use the div with id "testAccordion" as the parent element for the accordion.
- By default, accordion uses h3 elements as headers but we don't want that: change the header option to a class name of your choice, like "subsection"; then make each of the paragraphs in the testAccordion div belong to that class. You should have 3 headers: First description, Second description, Third description (see screenshot)
- Make the last header expanded by default by passing in the active option with value 2.

Populate the last section with images (use jQuery for all these steps):

- Use the `map` Array method on `g.imageData` to create an Array of jQuery-wrapped image elements. Set the `src` attribute of each image to the `thumbnailUrl` property, and `alt` to `title`.
- Append a div with id "imageWrapper" to the last section element. Set the width CSS property to 50%. (See screenshot.)
- Append all the image elements you just created to that div.
- Add a click event handler: clicking on an image should hide that image (use the jQuery hide method).

Part 2: No jQuery

Edit the code provided in `review.js` and `review.html`.

This time, write all your functions in the `g` global namespace like:

```
g = {
  imageData: [...],

  showNext: function () {
    //...
  },

  displayStuff: function () {
    // ...
  }
};
```

Set up an image slide show in the second section:

- Preload an Array of image elements based on `g.imageData`. (Again, use the Array `map` method.)
- Clicking on the `showBtn` should update the image element in the `wrapper` div every two seconds and change the button text to "Pause". Use replaceChild to update the image.
- Clicking the `showBtn` again should stop the slide show and change the button text back to "Start".
- The show should start with image 0, and wrap around to the beginning after it reaches the last image.

In the `#welcome a` element, a user should be able to set their name, and their name should be remembered next time they visit.

- Rather than using the cookie.js utility library provided previously in class, practice writing the cookie-related code yourself from scratch.
  - Write a helper function called `updateCookieObj` that populates a variable in your namespace (e.g. `g.allCookies`) with an object literal where each property is a cookie name and each value the corresponding cookie value from `document.cookie`. For

example, if `document.cookie` contains `a=b;c=d`, `g.allCookies` should be `{a: "b", c: "d"}`.

- There's an `invisible` class in style.css: use it to switch the visibility of `#welcome a` element and the `#welcome input` element. When you click on `#welcome a`, it should disappear and show `#welcome input` instead. Use `preventDefault` so that no navigation happens when you click on the link.
- When `#welcome input` loses focus or the user hits the "Enter" key, validate the text they entered in the input (use regex) and save it in a cookie called "username" that expires in a week. (Use `Date`, `getDate`, `toGMTString`.) Display the name in `#welcome a`.
- If the text is invalid, show a red border around the input. The border should disappear automatically after 2 seconds.
- When the DOM loads, you should check the cookie value to display the username in `#welcome a` if available. Use the helper you wrote before: `updateCookieObj`. Remember to validate the cookie value.

Grab some remote data using Ajax and display it in a list.

- The result of your Ajax request should be to display the `email` properties from the json result in a unordered list at the end of the Query Results section. Or it should display an error message in that location instead.
- Clicking the "Good Query" sends a request to http://jsonplaceholder.typicode.com/comments?postId=4.
- Clicking the "Bad Query" sends a request to http://jsonplaceholder.typicode.com/unicorns?age=200, and you should handle the resulting error.
- Write a helper function called `displayEmails` that accepts a string in json format, parses it, and displays `email` values from the json object in `<li>` elements in the Query Results section. Remember to clear any previously-displayed results first (remove the children).
- Write a helper function called `displayError` that accepts an XMLHttpRequest object and and displays the request status in an `<li>` element. Previous results should be cleared (remove ul children).
- Write a function called `getRequestHandler` that accepts a url and returns a function that sends an request and defines an onreadystatechange handler that uses the `displayEmails` and `displayError` callbacks.
    - When adding event listeners for clicking the query buttons, `getRequestHandler` should be called (the resulting handler is what you're attaching to the click event).

Source: _javascript_423/exercises/15_jQuery_and_review/15_jquery_and_review.md