Creative Project 2 - Reflection

Reflection Questions

Motivation (1-2 sentences): *How did you choose your topic? Did you find any inspiration from lectures, peer discussions, or other websites you've interacted with?*

I chose to continue building on the "Exploring Edinburgh" website from CP1 because starting with an existing base allowed me to focus more on enhancing interactivity and design rather than building from scratch. I see these Creative Projects as a great way to track my progress with one website, allowing me to apply new skills and techniques learned in each CP and to see how much I can improve with each iteration. It will also be a way to see my progress.

This is designed to make sure you understand the concepts of asynchronous JS, events, the DOM, and time management, as well as help you be aware of common bugs (we recommend not doing it last). You are allowed to ask questions about these on Discord #hw2, or utilize weekend OH with El to go over answers. Grading will be based on demonstration of understanding; if you're unsure, ask! These are designed to help identify things students miss that will ultimately save time in the long run. Particularly strong answers/extensions are eligible for points lost somewhere else in HW 2A/B.

Bug List: *Add any bugs that come up, how you identified them, and anything you learned from the process of debugging.*

1. Flip Card Not Working on Initial Click: When testing the attractions page, I clicked on a card expecting it to flip, but nothing happened. To debug, I added console.log statements to check if the click event was firing. I found that I had applied the event listener to the wrong element. I learned to always double-check which elements the event listeners are bound to, especially when working with nested structures.
2. Carousel Images Not Loading: On the homepage, I noticed that the image slider wasn't displaying any images. To debug, I checked the console for errors and found a null error for carouselImages.scrollWidth. This happened because the DOM hadn't fully loaded. Adding defer in the script tag ensured carouselImages was fully available by the time the JS code executed. This was an easy thing to miss but it was important.
3. Form Confirmation Message Not Displaying: After submitting the contact form, the form would clear, but the confirmation message wouldn't appear. To debug, first I added multiple console messages. Then, I checked the CSS and realized the '.hidden' class was still being applied. Updating '.hidden' correctly in JS resolved this. Debugging CSS visibility issues often involves checking both CSS and JS interactions, especially with dynamic class toggling.
4. Filter Not Updating Count After Selection: On the cafes page, changing the cuisine filter did not update the visible item count as I expected it to. To debug, I inspected the function and realized that displayFilterFeedback was not correctly updating the count in the DOM. Adding a

breakpoint and testing displayFilterFeedback individually helped pinpoint the issue. I learnt to break down functions and test them independently to quickly locate errors in specific functionalities.

5. Form Refreshing After Submit: Initially, when submitting the contact and suggestion forms, the page would refresh instead of displaying the confirmation message. My professor recommended using event.preventDefault() within the submit event function to prevent this default refresh behavior. Implementing event.preventDefault() in the form's submit handler resolved the issue, allowing the confirmation message to display as expected.
6. CSS Conflicts Across Pages: When multiple pages shared styles, I noticed some styling conflicts due to overlapping class names, causing unexpected design elements to appear on the wrong pages. To resolve this, I used 'Inspect Element' to see which properties were taking precedence and affecting the element. This helped a lot.

The DOM and Page Lifecycle: *What is the purpose of defer in the script tag? Why don't we use it for utils.js?*

Based on El's explanation on Discord and the lecture, I understood that using the `defer` attribute in a script tag delays the execution of that script until the entire DOM is fully loaded. This ensures that any DOM elements referenced within the script are accessible, helping prevent errors, like null pointer exceptions, which occur when a script tries to access an element that hasn't been loaded yet. That happened to me at first. For example, when using code like `qs("#demo-btn").addEventListener("click", someCallback);`, having `defer` ensures that `#demo-btn` exists in the DOM when the script runs. Without `defer`, the script would execute as soon as it's encountered, potentially before the `<body>` elements are available, leading to errors.
However, we don't need `defer` for `utils.js` because it only defines utility functions and doesn't immediately interact with any DOM elements. This makes it safe to load `utils.js` at any time, even before the DOM is fully constructed. Using `defer` on scripts that manipulate the DOM ensures that they load in the correct order and only when all required elements are accessible, reducing the chance of runtime errors.

Anonymous Functions with function() { } vs. Arrow Functions with () => { }: *El introduced the module pattern with anonymous functions. Later, we learned about the `this` keyword and how to use it to simplify callback functions and event listeners. Arrow functions are popular "syntactic sugar" for anonymous functions in ES6, and you'll see them quite often in modern JS tutorials (they are popular for short functions, especially those passed to higher-order functions like map, reduce, filter, and forEach). We will continue to use these as we start working with APIs, especially when learning Node.js/Express. However, the two forms are not exactly the same. Subtle bugs come up when using arrow functions as callbacks with `this`; take a look at the slides, and briefly identify the key difference between the two types of anonymous functions.*

The main difference between regular anonymous functions (function() { }) and arrow functions (() => { }) is how they handle this. In regular functions, this refers to the object that called the function, like the DOM element in an event listener. But in arrow functions, this doesn't change and instead refers to where the function was originally written. This can be helpful because it keeps this consistent, which is useful when we want to keep the same context. However, it can also cause problems: if you use an arrow function in an event listener, this won't point to the element that triggered the event, which might not be what you want.

Design, Implement, Test, Iterate: *Show someone your webpage (a friend, Discord, etc.) and ask for feedback (don't worry about how it looks/etc. this is your first project :)). Think content, responsiveness on different screen sizes, design (a word that refers to any of aesthetics, usability, readability, etc.), creativity, etc. What do they like? What are 2 suggestions they'd have to change/add? How could you implement those suggestions given what you know so far?*

I asked a friend from Sweden, who has visited Edinburgh before and is currently studying at CalTech with me, to review my website. He found it visually appealing and informative, mentioning that the flip card feature added a fun interactive element, and the filtering options made it easy to explore different places in the city. He appreciated the overall usability, readability, and aesthetics, and even discovered some new places he hadn't known about before.
He suggested some additions: expanding the filter options to include criteria like price range for restaurants and adding a review feature where users could leave feedback on cafes, restaurants, or attractions. To implement these, I could enhance the filter by adding a price range dropdown in HTML and updating the JavaScript to filter based on both cuisine and price. For the review feature, I could add a form allowing users to submit comments that appear as static feedback, which would give the site a more interactive feel without requiring a full database. However, I am still not sure of how I would implement the reviews. Lastly, I had considered adding a favorites feature where users could "bookmark" places they're interested in. This could probably be achieved with a favorite button on each card and JavaScript's localStorage to save and display a user's selected favorites even upon returning to the site but I am not sure of the implementation so I did not do it this time. His feedback helped me see the value of personalization features to enhance the user experience and make the site even more engaging. The filter by budget and cuisine and possibly even locations could make the website more user-friendly and helpful.