**DOM Programming Exercise**

In this exercise, we will practice working with HTML, images, URLs, the DOM, events, and JavaScript to create an interactive web page.

1. Create a **folder** called cats on your computer
2. Create a **file** inside the cats folder named index.html
3. Open a **terminal** to your cats folder (i.e., cd cats)
4. In your **terminal**, start a web server by running the following command: npx http-server (alternatively, you can use the command: npx lite-server, refer week 5 notes)
5. Open the cats folder in Visual Studio Code
6. Edit the index.html file so it contains a [basic HTML5 web page](https://web222.ca/weeks/week05/#basic-html5-document), including a <head>, <body>, etc. Try to do it from memory first, then look up what you’ve missed.
7. Save index.html and try loading it in your browser by visiting your local web server at http://localhost:8080/index.html
8. In your editor, modify the body of your index.html file to contain the text of the poem in [cats.txt](https://web222.ca/weeks/week07/cats.txt). Use HTML tags to markup the poem for the web. Your page should have a proper heading for the title, each line should break at the correct position, and the poet’s name should be bold.
9. Add an image of a cat to the page below the text. You can use <https://upload.wikimedia.org/wikipedia/commons/c/c1/Six_weeks_old_cat_%28aka%29.jpg>.
10. Adjust the width of your image so it fits nicely on your page. What happens if you adjust the width and height?
11. Create a new file in your cats folder called script.js. Add the following line of JavaScript:
12. console.log('cats!');
13. Add a script element to the bottom of your body (i.e., right before the closing </body> tag). Set its src to a file called script.js:
14. <script src="script.js"></script>
15. </body>
16. Refresh your web page in the browser, and open your browser’s Dev Tools, and Web Console. Make sure you can see the cats!message printed in the log.
17. Try changing cats! in script.js to some other message, save your script.js file, and refresh your browser. Make sure your console updates with the new message.
18. Modify index.html and update your <img> tag: add an attribute id="cat-picture" and remove the src="...":
19. *<!-- NOTE: there is no longer a src attribute in our HTML, we'll do it JavaScript below -->*
20. <img id="cat-picture">
21. Modify your script.js file to add the following code:
22. window.onload **=** **function**() {
23. **let** img **=** document.getElementById('cat-picture');
24. img.src **=** 'https://upload.wikimedia.org/wikipedia/commons/c/c1/Six\_weeks\_old\_cat\_%28aka%29.jpg';
25. };
26. Save your script.js file and reload your browser. Do you still see a cat? If not, check your web console for any errors.
27. Modify your script.js and change your cat URL used by img.src to use <https://cataas.com/cat>. The [cataas.com](https://cataas.com/#/) site provides cat pictures as a service via URL parameters. Save script.js and reload your page a few times. Do you see a different cat each time?
28. Modify your script.js file to move your image code to a separate function. Make sure it still works the same way when you’re done (save and test in your browser):
29. **function** loadCatPicture() {
30. **let** img **=** document.getElementById('cat-picture');
31. img.src **=** 'https://cataas.com/cat';
32. }
33. window.onload **=** loadCatPicture;
34. Rewrite script.js to update the picture after 5 seconds:
35. **function** loadCatPicture() {
36. **let** img **=** document.getElementById('cat-picture');
37. img.src **=** 'https://cataas.com/cat';
38. }
39. window.onload **=** **function**() {
40. loadCatPicture();
41. *// Call the loadCatPicture function again in 5s*
42. setTimeout(loadCatPicture, 5 **\*** 1\_000 */\* 5s = 5000ms \*/*);
43. };
44. Rewrite script.js to update the picture every 15 seconds, forever:
45. **function** loadCatPicture() {
46. **let** img **=** document.getElementById('cat-picture');
47. img.src **=** 'https://cataas.com/cat';
48. }
49. window.onload **=** **function**() {
50. loadCatPicture();
51. *// Call the loadCatPicture function every 15000ms*
52. setInterval(loadCatPicture, 15 **\*** 1\_000 */\* 15s = 15000ms \*/*);
53. };
54. Rewrite script.js to update the picture only when the user clicks somewhere in the window:
55. **function** loadCatPicture() {
56. **let** img **=** document.getElementById('cat-picture');
57. img.src **=** 'https://cataas.com/cat';
58. }
59. window.onload **=** **function**() {
60. loadCatPicture();
61. *// Call the loadCatPicture function when the user clicks in the window*
62. window.onclick **=** loadCatPicture;
63. };
64. Modify index.html and put a <div>...</div> around all the text of the poem. Give your div an id="poem-text" attribute:
65. <div id="poem-text">
66. <p>Cats sleep anywhere, any table, any chair....</p>
67. ...
68. </div>
69. Rewrite script.js to load the picture only when the user clicks on the text of the poem:
70. **function** loadCatPicture() {
71. **let** img **=** document.getElementById('cat-picture');
72. img.src **=** 'https://cataas.com/cat';
73. }
74. **let** poemText **=** document.getElementById('poem-text');
75. poemText.onclick **=** loadCatPicture;
76. Rewrite script.js to also load the picture only when the user presses a key on the keyboard:
77. **function** loadCatPicture() {
78. **let** img **=** document.getElementById('cat-picture');
79. img.src **=** 'https://cataas.com/cat';
80. }
81. **let** poemText **=** document.getElementById('poem-text');
82. poemText.onclick **=** loadCatPicture;
83. window.onkeypress **=** **function**(event) {
84. **let** keyName **=** event.key;
85. console.log('Key Press event', keyName);
86. loadCatPicture();
87. };
88. Rewrite script.js to also load the picture only when the user presses a key on the keyboard, but only one of b, m, s, n, p, x:
89. **function** loadCatPicture() {
90. **let** img **=** document.getElementById('cat-picture');
91. img.src **=** 'https://cataas.com/cat';
92. }
93. **let** poemText **=** document.getElementById('poem-text');
94. poemText.onclick **=** loadCatPicture;
95. window.onkeypress **=** **function**(event) {
96. **let** keyName **=** event.key;
97. console.log('Key Press event', keyName);
98. **switch**(keyName) {
99. **case** 'b':
100. **case** 'm':
101. **case** 's':
102. **case** 'n':
103. **case** 'p':
104. **case** 'x':
105. loadCatPicture();
106. **break**;
107. **default**:
108. console.log('Ignoring key press event');
109. }
110. };
111. Rewrite script.js to also load the picture only when the user presses a key on the keyboard, but only one of b, m, s, n, p, x, and load the picture with one of the supported [cataas filters](https://cataas.com/#/):
112. **function** loadCatPicture(filter) {
113. **let** url **=** 'https://cataas.com/cat';
114. **let** img **=** document.getElementById('cat-picture');
115. *// If the function is called with a filter argument, add that to URL*
116. **if** (filter) {
117. console.log('Using cat picture filter', filter);
118. url **+=** `?filter=${filter}`
119. }
120. img.src **=** url;
121. }
122. **let** poemText **=** document.getElementById('poem-text');
123. poemText.onclick **=** **function**() {
124. loadCatPicture();
125. };
126. window.onkeypress **=** **function**(event) {
127. **let** keyName **=** event.key;
128. console.log('Key Press event', keyName);
129. **switch**(keyName) {
130. **case** 'b':
131. **return** loadCatPicture('blur');
132. **case** 'm':
133. **return** loadCatPicture('mono');
134. **case** 's':
135. **return** loadCatPicture('sepia');
136. **case** 'n':
137. **return** loadCatPicture('negative');
138. **case** 'p':
139. **return** loadCatPicture('paint');
140. **case** 'x':
141. **return** loadCatPicture('pixel');
142. **default**:
143. console.log('Ignoring key press event');
144. }
145. };
146. Rewrite script.js so that we only load a new cat picture when the old picture is finished loading (don’t send too many requests to the server). Also, add some [cache busting](https://www.keycdn.com/support/what-is-cache-busting):
147. *// Demonstrate using a closure, and use an immediately executing function to hide*
148. *// an `isLoading` variable (i.e., not global), which will keep track of whether*
149. *// or not an image is being loaded, so we can ignore repeated requests.*
150. **let** loadCatPicture **=** (**function**() {
151. **let** isLoading **=** **false**;
152. *// This is the function that will be bound to loadCatPicture in the end.*
153. **return** **function**(filter) {
154. **if**(isLoading) {
155. console.log('Skipping load, already in progress');
156. **return**;
157. }
158. **let** img **=** document.getElementById('cat-picture');
160. **function** finishedLoading() {
161. isLoading **=** **false**;
162. *// Remove unneeded event handlers so `img` can be garbage collected.*
163. img.onload **=** **null**;
164. img.onerror **=** **null**;
165. img **=** **null**;
166. }
167. img.onload **=** finishedLoading;
168. img.onerror **=** finishedLoading;
169. *// If the function is called with a filter argument, add that to URL*
170. **let** url **=** 'https://cataas.com/cat';
171. *// Add something unique (and meaningless) to the query string, so the browser*
172. *// won't cache this URL, but always load it again*
173. url **+=** '?nocache=' **+** Date.now();
174. **if** (filter) {
175. console.log('Using cat picture filter', filter);
176. url **+=** '&filter=' **+** filter
177. }
178. *// Finally, set isLoading to true, and begin loading image*
179. isLoading **=** **true**;
180. img.src **=** url;
181. };
182. })();
183. **let** poemText **=** document.getElementById('poem-text');
184. poemText.onclick **=** **function**() {
185. loadCatPicture();
186. };
187. window.onkeypress **=** **function**(event) {
188. **switch**(event.key) {
189. **case** 'b':
190. **return** loadCatPicture('blur');
191. **case** 'm':
192. **return** loadCatPicture('mono');
193. **case** 's':
194. **return** loadCatPicture('sepia');
195. **case** 'n':
196. **return** loadCatPicture('negative');
197. **case** 'p':
198. **return** loadCatPicture('paint');
199. **case** 'x':
200. **return** loadCatPicture('pixel');
201. **default**:
202. console.log('Ignoring key press event');
203. **break**;
204. }

};

**1. Setting Up the Environment (Questions 1-6)**

**Big Picture**: Establish a working development environment to start building and testing your project.

* **Key Details**:
  1. Create a cats folder and add an index.html file.
  2. Use **JavaScript** to print a simple message (console.log) to the browser console and link the script.js file in your HTML.
  3. Start a local server (npx http-server or lite-server) to test your web app in a real browser environment.
  4. Add an image (<img>) to your HTML file and dynamically load it using JavaScript.
  5. Test DevTools for debugging (console.log) to ensure proper setup.

**Why It Matters**: This sets up a foundation for working on modern web applications and ensures a feedback loop for quick testing.

**2. Introducing Interactivity with JavaScript (Questions 7-10)**

**Big Picture**: Add basic interactivity using DOM manipulation and event listeners.

* **Key Details**:
  1. Use window.onload to load an image dynamically when the page is ready.
  2. Move your code into functions for reusability (e.g., loadCatPicture()).
  3. Experiment with timers like setTimeout to refresh the image after 5 seconds.
  4. Use setInterval to refresh the image continuously (every 15 seconds).

**Why It Matters**: Learning how to manipulate the DOM and handle page load events is critical for creating dynamic and responsive web applications.

**3. User Interaction via Events (Questions 11-16)**

**Big Picture**: Make your application interactive based on user input (clicks or keypresses).

* **Key Details**:
  1. Attach a click event to the window to load a new image when the user clicks anywhere.
  2. Attach a click event specifically to the poem's text (#poem-text) to restrict interaction.
  3. Trigger updates based on specific keypresses (b, m, s, n, p, x).
  4. Enhance interactivity by introducing filters (e.g., blur, sepia) that modify the cat images dynamically.

**Why It Matters**: Event-driven programming is essential for modern web apps, enabling real-time user interactions like clicking, typing, or scrolling.

**4. Advanced Event Handling and Optimization (Questions 17-22)**

**Big Picture**: Introduce conditional and optimized image loading to handle performance and avoid redundant server requests.

* **Key Details**:
  1. Only load a new image after the previous image has fully loaded (using onload and onerror handlers).
  2. Add a **cache-busting query string** (e.g., ?nocache=timestamp) to ensure the browser fetches a fresh image each time.
  3. Use closures to manage state (e.g., isLoading) for better encapsulation and to avoid repeated requests.

**Why It Matters**: Handling performance issues and optimizing server requests are vital for creating efficient and scalable web applications.

**5. Combining Logic for Real-World Scenarios (Questions 23-28)**

**Big Picture**: Combine all features—dynamic image loading, event handling, and user-defined filters—into a cohesive, interactive application.

* **Key Details**:
  1. Modularize your code (e.g., loadCatPicture() handles loading, while keypress and click handlers manage events).
  2. Use a switch-case to define specific actions for keypresses and ensure only supported keys trigger actions.
  3. Apply filters to the image dynamically via query strings (e.g., https://cataas.com/cat?filter=blur).
  4. Finalize the app to work efficiently by avoiding redundant requests and dynamically updating the UI based on user input.

**Why It Matters**: Combining multiple features and optimizing interactions teaches you how to build real-world, scalable web apps with clean, reusable code.

**The Big Picture as an SWE**

1. **Foundation**:
   * Learn to set up a simple project with proper file structures, local servers, and debugging tools.
   * Understand how JavaScript interacts with HTML to create dynamic, interactive pages.
2. **Interactivity**:
   * Dive into DOM manipulation and user interaction through events (clicks, keypresses).
   * Start thinking about user experience (UX) by adding timers, interactions, and filters.
3. **Performance**:
   * Address potential bottlenecks, such as redundant server requests and inefficient event handling.
   * Learn to optimize web performance using closures, onload handlers, and caching techniques.
4. **Modularity**:
   * Move from basic functions to modular, reusable code.
   * Start thinking about **separation of concerns** (e.g., loading logic vs. event handling).
5. **Scalability**:
   * Combine all features into a single application while maintaining clean and efficient code.
   * Learn how small components (like dynamic images or filters) can fit into larger, complex projects.

**Why This Flow is Important**

As an SWE, these tasks gradually expose you to **key front-end concepts**:

1. **Basic Setup**: Setting up the project and environment.
2. **DOM Manipulation**: Dynamically interacting with and modifying the DOM.
3. **Event-Driven Programming**: Responding to user actions in real-time.
4. **Performance Optimization**: Avoiding unnecessary operations and handling complex logic.
5. **Modularity and Reusability**: Writing clean, structured, and reusable code.

**Outcome**

At the end of this exercise, you will have:

* Built an interactive web app that dynamically loads content.
* Learned to optimize user interactions and server requests.
* Practiced writing modular, maintainable code for scalability.