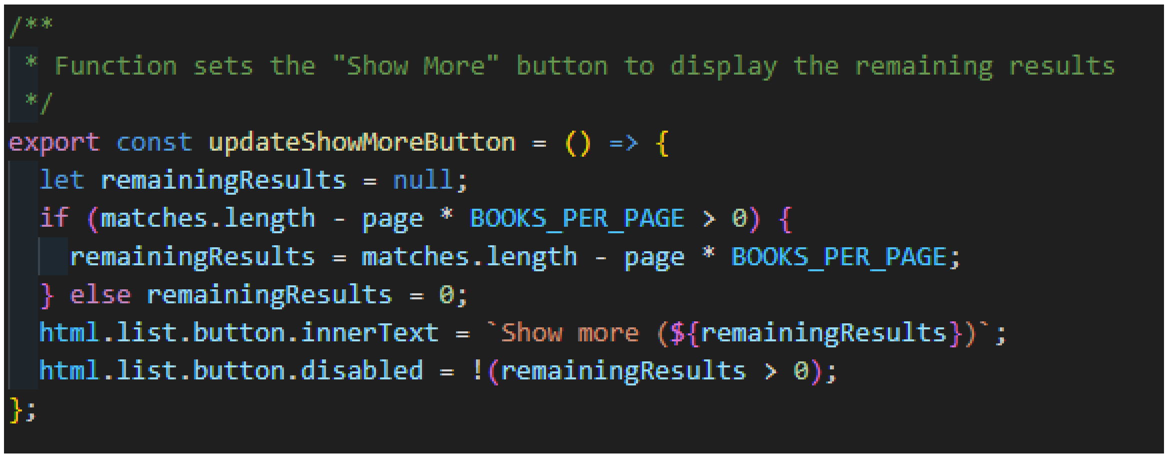
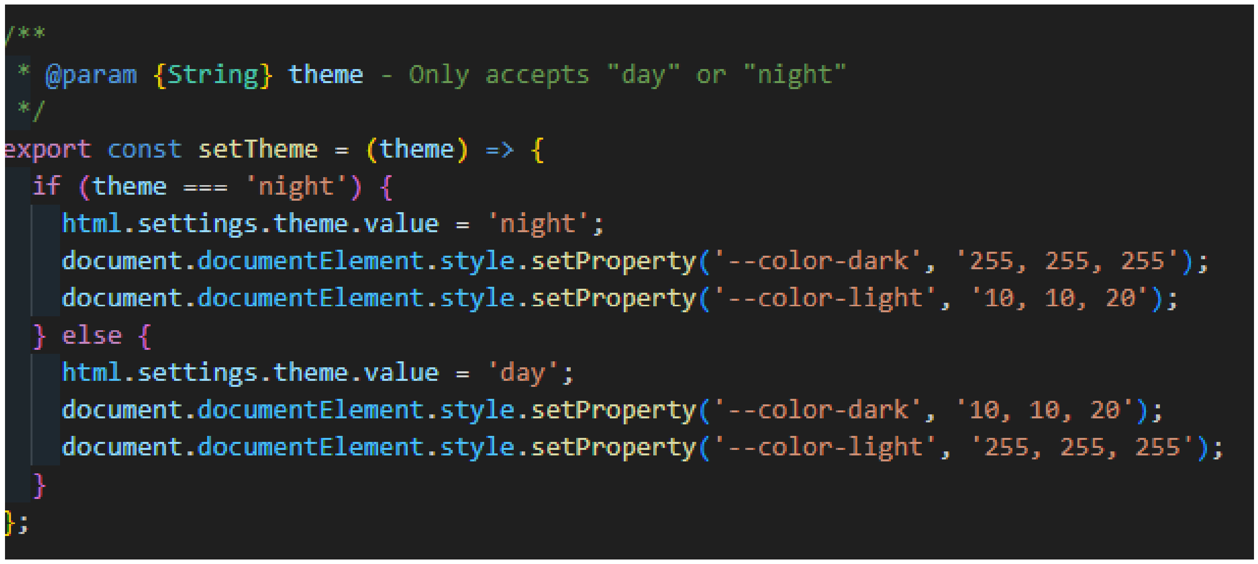
DWA_07.4 Knowledge Check_DWA7

1. Which were the three best abstractions, and why?



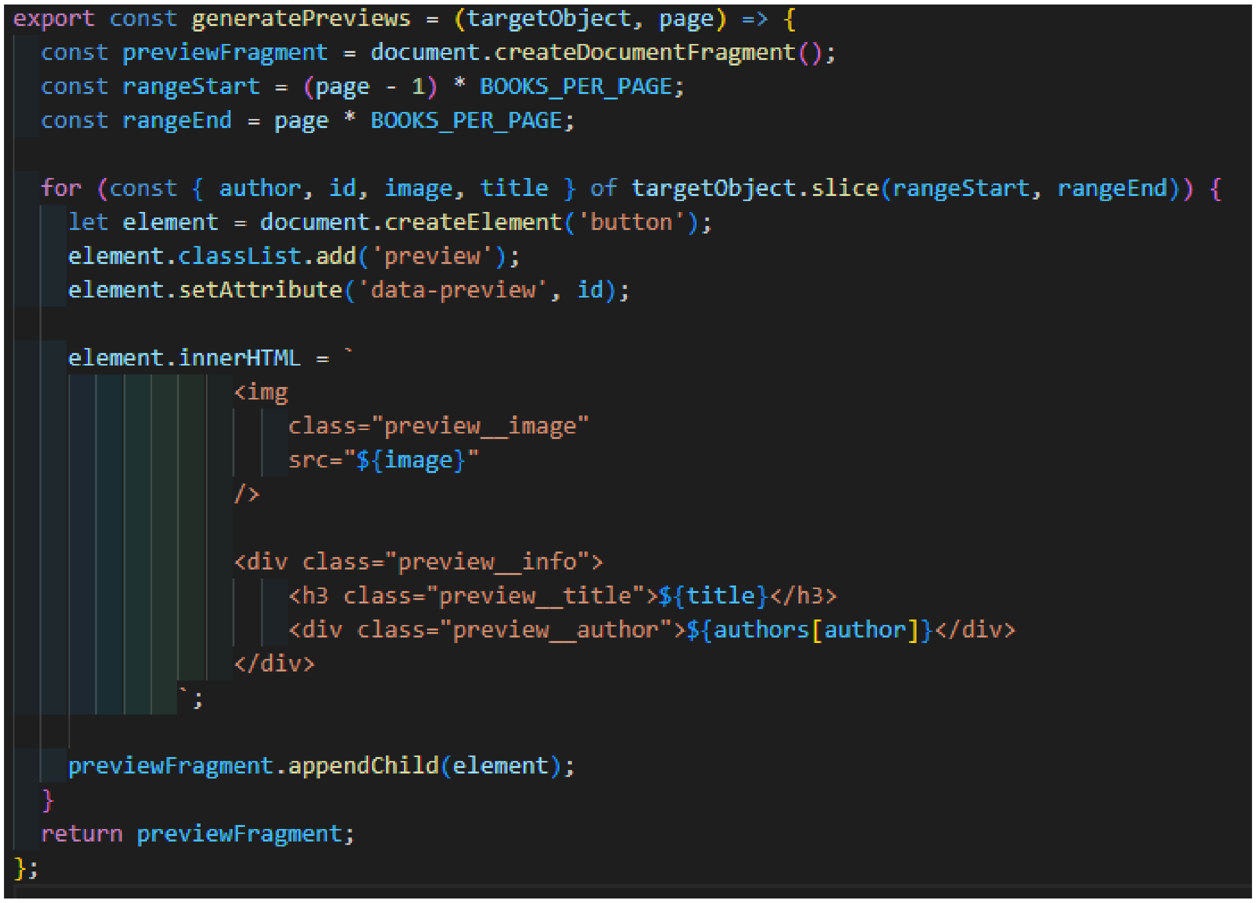
```javascript
/**
 * Function sets the "Show More" button to display the remaining results
 */
export const updateShowMoreButton = () => {
  let remainingResults = null;
  if (matches.length - page * BOOKS_PER_PAGE > 0) {
    remainingResults = matches.length - page * BOOKS_PER_PAGE;
  } else remainingResults = 0;
  html.list.button.innerText = `Show more (${remainingResults})`;
  html.list.button.disabled = !(remainingResults > 0);
};
```

I can call this function upon initialisation, whenever a new search is submitted or when the “Show more” button is clicked without having to modify it.



```javascript
/**
 * @param {String} theme - Only accepts "day" or "night"
 */
export const setTheme = (theme) => {
  if (theme === 'night') {
    html.settings.theme.value = 'night';
    document.documentElement.style.setProperty('--color-dark', '255, 255, 255');
    document.documentElement.style.setProperty('--color-light', '10, 10, 20');
  } else {
    html.settings.theme.value = 'day';
    document.documentElement.style.setProperty('--color-dark', '10, 10, 20');
    document.documentElement.style.setProperty('--color-light', '255, 255, 255');
  }
};
```

Allows for easy toggling between themes, and can easily be extended to include more themes.

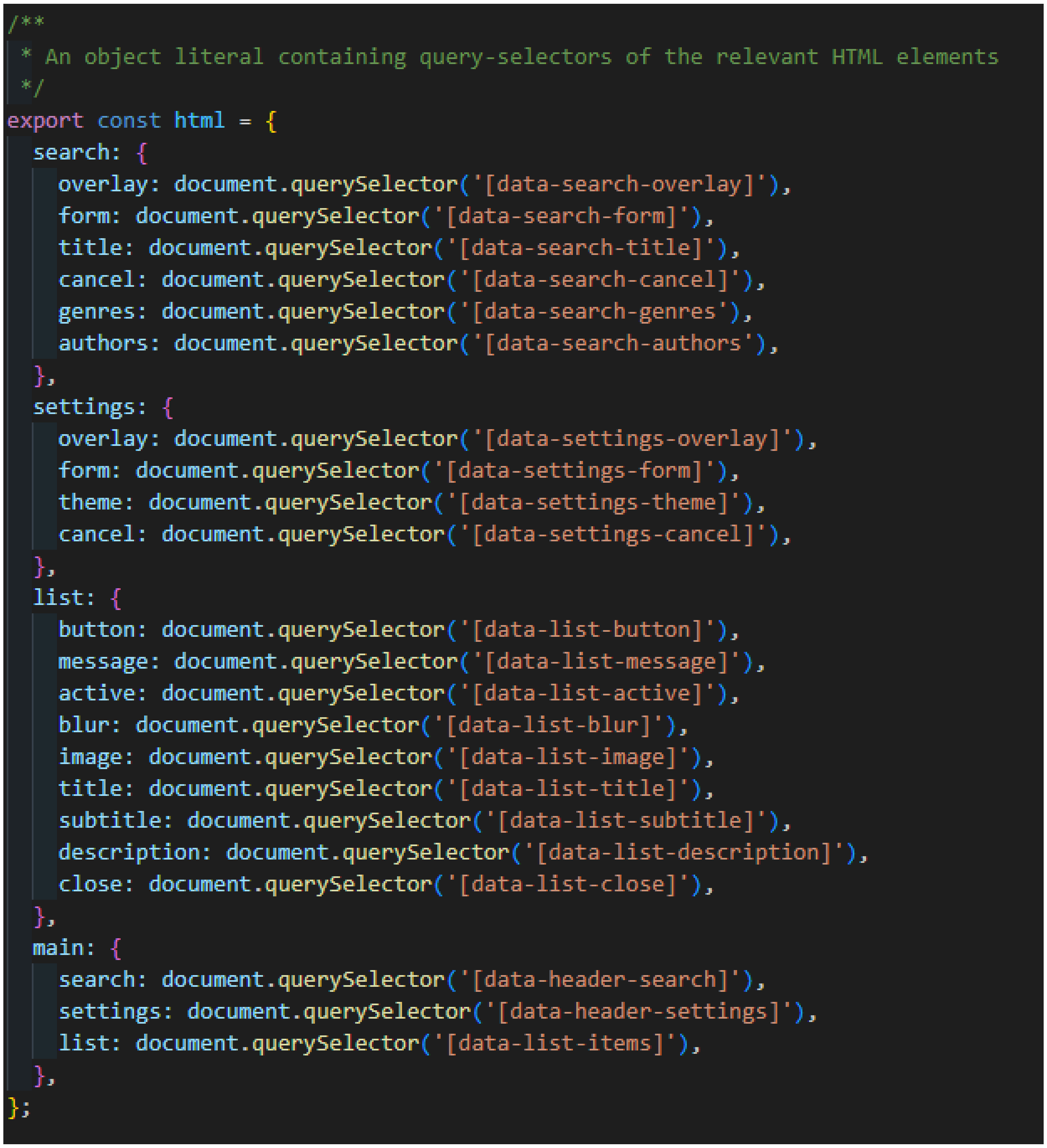


```
export const generatePreviews = (targetObject, page) => {
  const previewFragment = document.createDocumentFragment();
  const rangeStart = (page - 1) * BOOKS_PER_PAGE;
  const rangeEnd = page * BOOKS_PER_PAGE;

  for (const { author, id, image, title } of targetObject.slice(rangeStart, rangeEnd)) {
    let element = document.createElement('button');
    element.classList.add('preview');
    element.setAttribute('data-preview', id);

    element.innerHTML = `
            <img
                class="preview__image"
                src="${image}"
            />

            <div class="preview__info">
                <h3 class="preview__title">${title}</h3>
                <div class="preview__author">${authors[author]}</div>
            </div>
          `;

    previewFragment.appendChild(element);
  }
  return previewFragment;
};
```

I can reuse this function whenever I need to generate more previews to be appended. It's used upon initialisation, when a new search is submitted, and when I click "Show More".

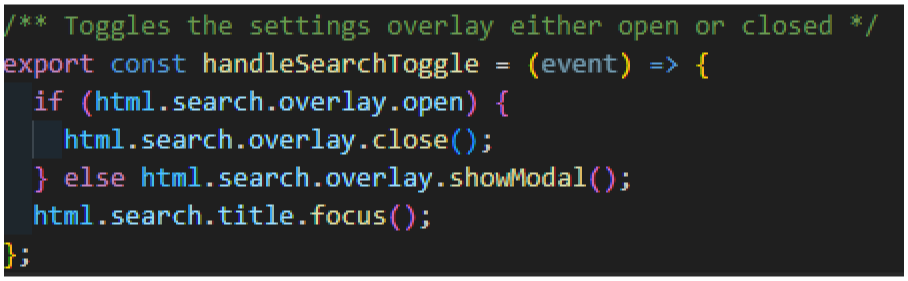
---

2. Which were the three worst abstractions, and why?

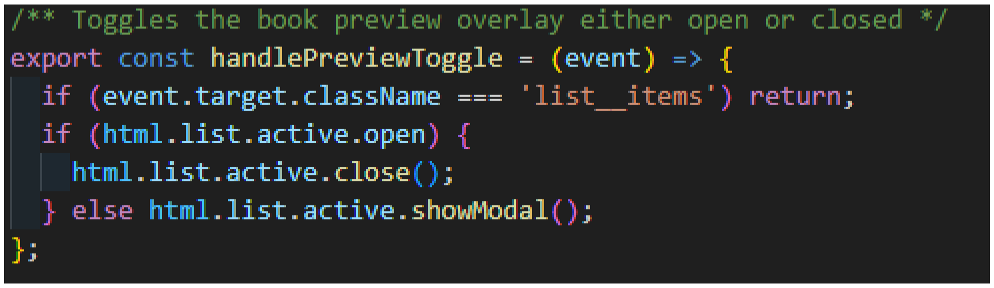


```
/**
 * An object literal containing query-selectors of the relevant HTML elements
 */
export const html = {
  search: {
    overlay: document.querySelector('[data-search-overlay]'),
    form: document.querySelector('[data-search-form]'),
    title: document.querySelector('[data-search-title]'),
    cancel: document.querySelector('[data-search-cancel]'),
    genres: document.querySelector('[data-search-genres'),
    authors: document.querySelector('[data-search-authors'),
  },
  settings: {
    overlay: document.querySelector('[data-settings-overlay]'),
    form: document.querySelector('[data-settings-form]'),
    theme: document.querySelector('[data-settings-theme]'),
    cancel: document.querySelector('[data-settings-cancel]'),
  },
  list: {
    button: document.querySelector('[data-list-button]'),
    message: document.querySelector('[data-list-message]'),
    active: document.querySelector('[data-list-active]'),
    blur: document.querySelector('[data-list-blur]'),
    image: document.querySelector('[data-list-image]'),
    title: document.querySelector('[data-list-title]'),
    subtitle: document.querySelector('[data-list-subtitle]'),
    description: document.querySelector('[data-list-description]'),
    close: document.querySelector('[data-list-close]'),
  },
  main: {
    search: document.querySelector('[data-header-search]'),
    settings: document.querySelector('[data-header-settings]'),
    list: document.querySelector('[data-list-items]'),
  },
};
```

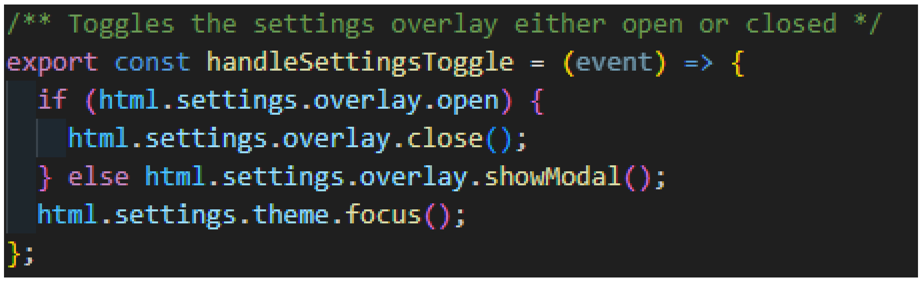
An object literal like this is very neat, but can result in the creation of values that are never used. In hindsight, using a “getHTML” function that finds elements by datasets would be cleaner, and I would avoid creating Queryselectors for elements I don’t actually use.



```
/** Toggles the settings overlay either open or closed */
export const handleSearchToggle = (event) => {
  if (html.search.overlay.open) {
    html.search.overlay.close();
  } else html.search.overlay.showModal();
  html.search.title.focus();
};
```

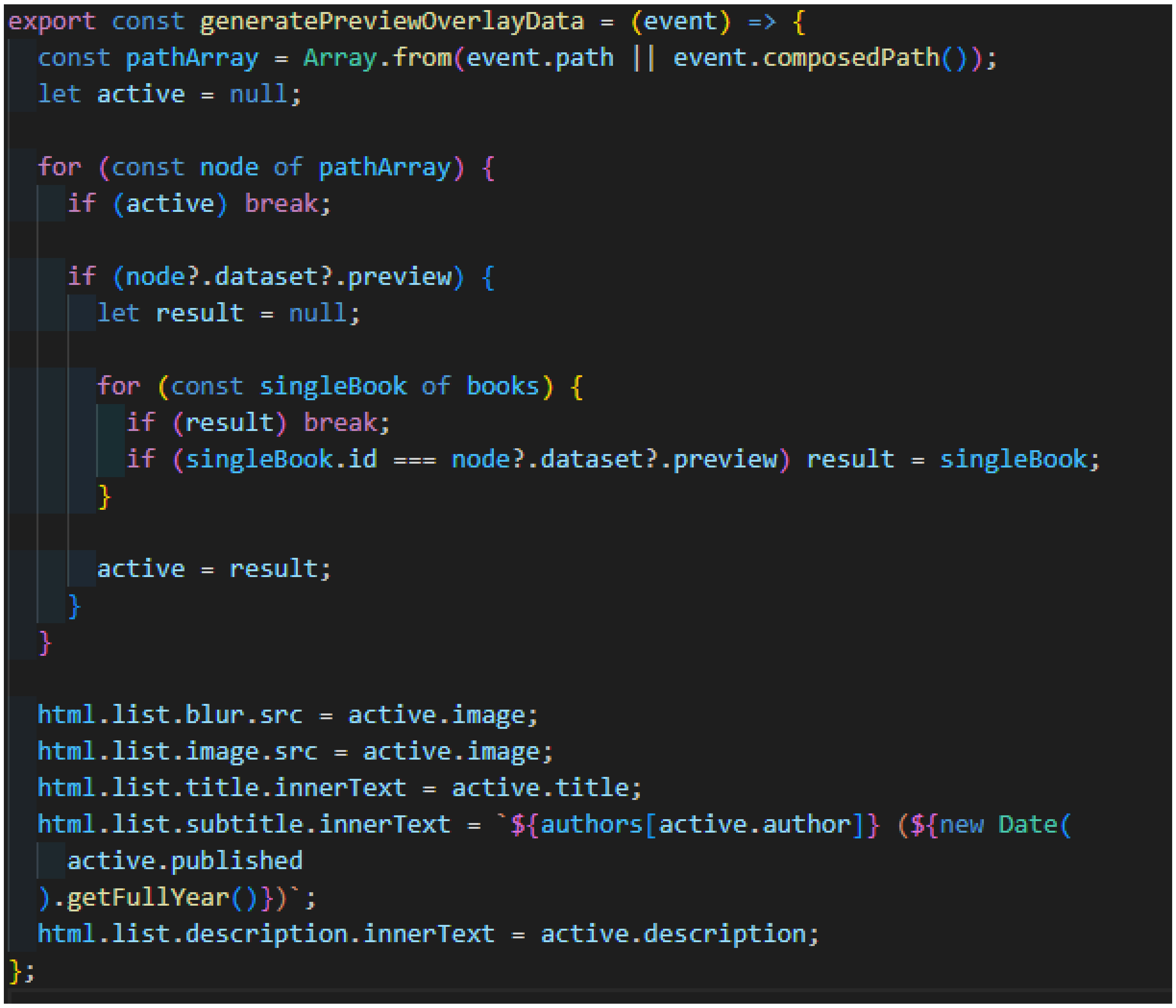


```
/** Toggles the book preview overlay either open or closed */
export const handlePreviewToggle = (event) => {
  if (event.target.className === 'list__items') return;
  if (html.list.active.open) {
    html.list.active.close();
  } else html.list.active.showModal();
};
```



```
/** Toggles the settings overlay either open or closed */
export const handleSettingsToggle = (event) => {
  if (html.settings.overlay.open) {
    html.settings.overlay.close();
  } else html.settings.overlay.showModal();
  html.settings.theme.focus();
};
```

I have multiple “toggle” handlers in my code with similar functionality. Ideally one could create a single function (i.e. handleModalToggle) that automatically finds the relevant modal to open/close, which can be reused across all eventListeners



```
export const generatePreviewOverlayData = (event) => {
  const pathArray = Array.from(event.path || event.composedPath());
  let active = null;

  for (const node of pathArray) {
    if (active) break;

    if (node?.dataset?.preview) {
      let result = null;

      for (const singleBook of books) {
        if (result) break;
        if (singleBook.id === node?.dataset?.preview) result = singleBook;
      }

      active = result;
    }
  }

  html.list.blur.src = active.image;
  html.list.image.src = active.image;
  html.list.title.innerText = active.title;
  html.list.subtitle.innerText = `${authors[active.author]} (${new Date(
    active.published
  ).getFullYear()})`;
  html.list.description.innerText = active.description;
};
```

This function is perhaps a bit too procedural, and logic to find the active node, as well as the logic that assigns the data to the element could both be abstracted into two separate functions to improve readability. Additionally the function does more than the name implies, as it finds the correct node, grabs the data matching the node's id, and then populates the element with the data.

---

3. How can The three worst abstractions be improved via SOLID principles.

1. Not sure how SOLID can be applied here
2. Not sure how SOLID can be applied here

3. I believe that this abstraction doesn't fully adhere to the Single-Responsibility Principle, as it's responsible for 3 different actions. Further abstraction for each responsibility would improve this function.