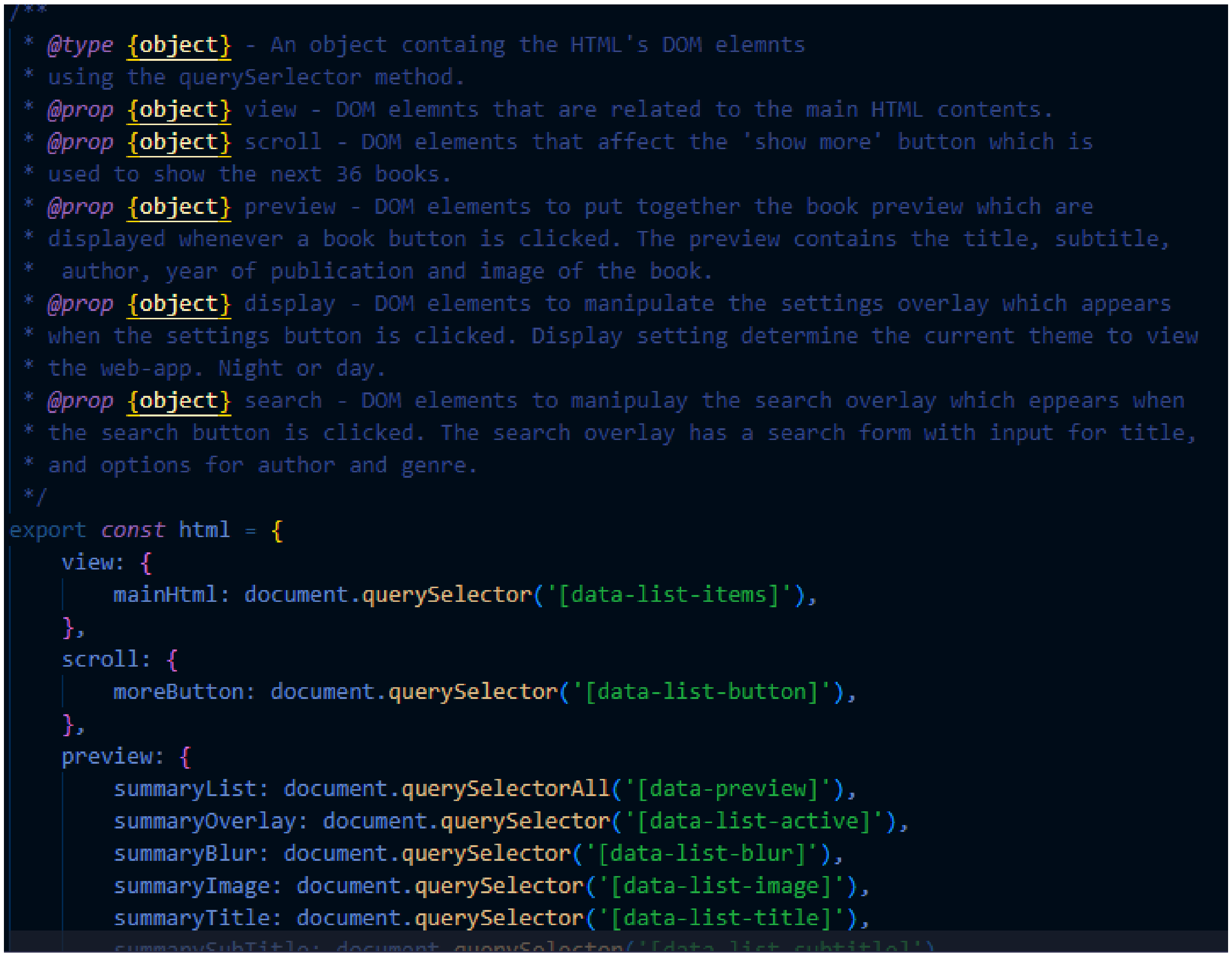
DWA_07.4 Knowledge Check_DWA7

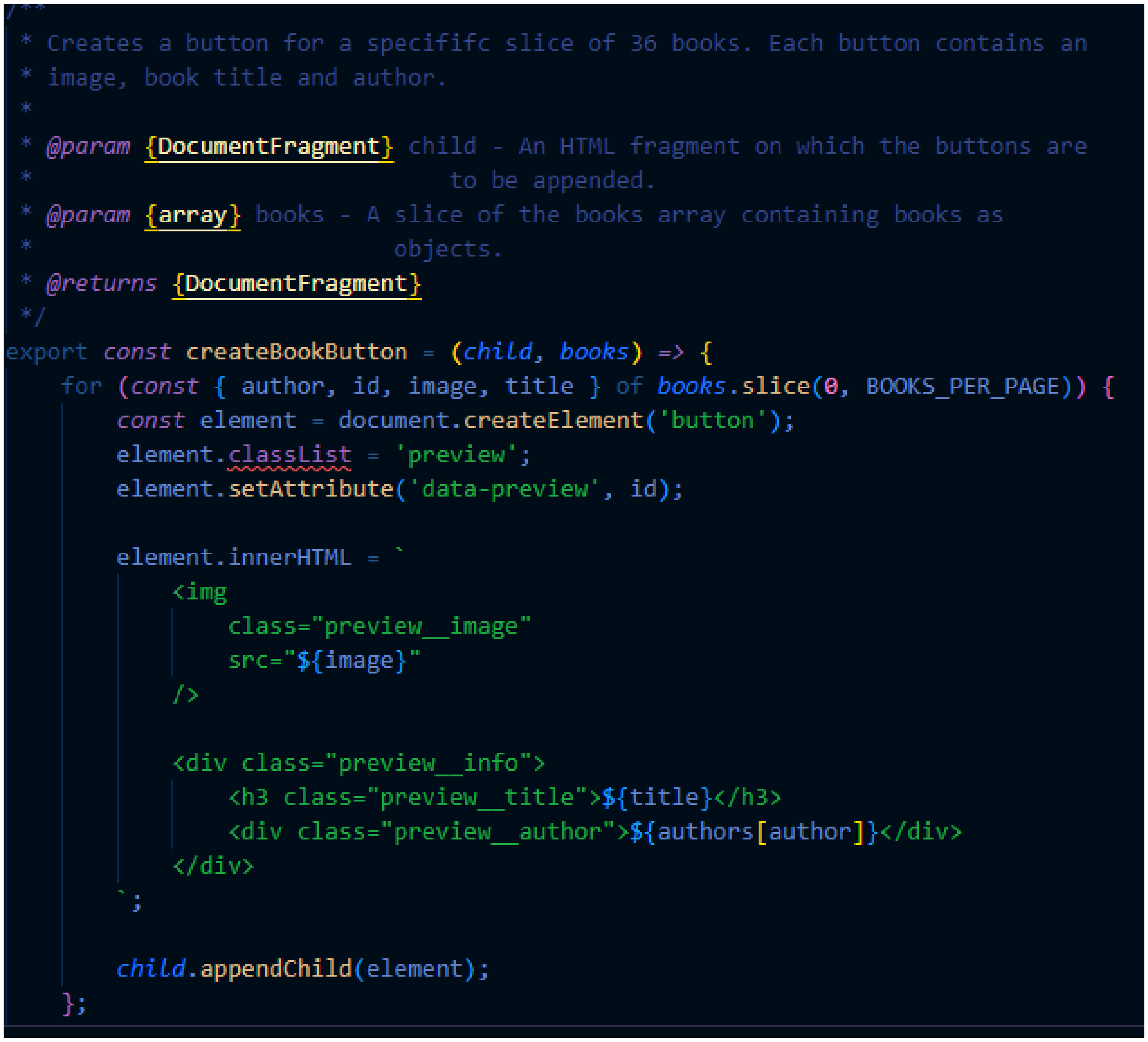
1. Which were the three best abstractions, and why?

Changes Applied

1. Use html object to query select DOM nodes.
2. Enabled 'show more' button. Removed unnecessary code. This button needs further debugging.
3. Used abstraction for the creation of the 'show more' button.
4. Used abstraction to create book buttons.
5. Added JSDoc.
6. Fixed ts-check errors via JSDoc. i.e. Adding correct types for variables.
7. Moved modules into a separate folder.
8. Added check if ‘books’ exists & is not empty. Will throw error otherwise.



```
/**
 * @type {object} - An object containg the HTML's DOM elemnts
 * using the querySerlector method.
 * @prop {object} view - DOM elemnts that are related to the main HTML contents.
 * @prop {object} scroll - DOM elements that affect the 'show more' button which is
 * used to show the next 36 books.
 * @prop {object} preview - DOM elements to put together the book preview which are
 * displayed whenever a book button is clicked. The preview contains the title, subtitle,
 *  author, year of publication and image of the book.
 * @prop {object} display - DOM elements to manipulate the settings overlay which appears
 * when the settings button is clicked. Display setting determine the current theme to view
 * the web-app. Night or day.
 * @prop {object} search - DOM elements to manipulay the search overlay which eppears when
 * the search button is clicked. The search overlay has a search form with input for title,
 * and options for author and genre.
 */
export const html = {
    view: {
        mainHtml: document.querySelector('[data-list-items]'),
    },
    scroll: {
        moreButton: document.querySelector('[data-list-button]'),
    },
    preview: {
        summaryList: document.querySelectorAll('[data-preview]'),
        summaryOverlay: document.querySelector('[data-list-active]'),
        summaryBlur: document.querySelector('[data-list-blur]'),
        summaryImage: document.querySelector('[data-list-image]'),
        summaryTitle: document.querySelector('[data-list-title]'),
        summarySubTitle: document.querySelector('[data-list-subtitle]'),
```

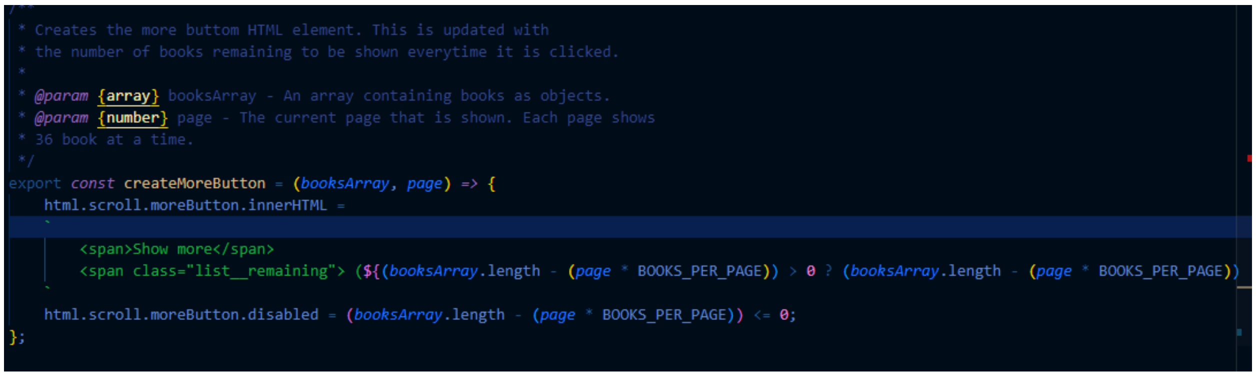


```
/**
 * Creates a button for a specififc slice of 36 books. Each button contains an
 * image, book title and author.
 *
 * @param {DocumentFragment} child - An HTML fragment on which the buttons are
 *                                 to be appended.
 * @param {array} books - A slice of the books array containing books as
 *                         objects.
 * @returns {DocumentFragment}
 */
export const createBookButton = (child, books) => {
    for (const { author, id, image, title } of books.slice(0, BOOKS_PER_PAGE)) {
        const element = document.createElement('button');
        element.classList = '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>
        `;

        child.appendChild(element);
    };
```



```
/**
 * Creates the more buttom HTML element. This is updated with
 * the number of books remaining to be shown everytime it is clicked.
 *
 * @param {array} booksArray - An array containing books as objects.
 * @param {number} page - The current page that is shown. Each page shows
 * 36 book at a time.
 */
export const createMoreButton = (booksArray, page) => {
    html.scroll.moreButton.innerHTML =
    `
        <span>Show more</span>
        <span class="list__remaining"> (${(booksArray.length - (page * BOOKS_PER_PAGE)) > 0 ? (booksArray.length - (page * BOOKS_PER_PAGE))
    `

    html.scroll.moreButton.disabled = (booksArray.length - (page * BOOKS_PER_PAGE)) <= 0;
};
```

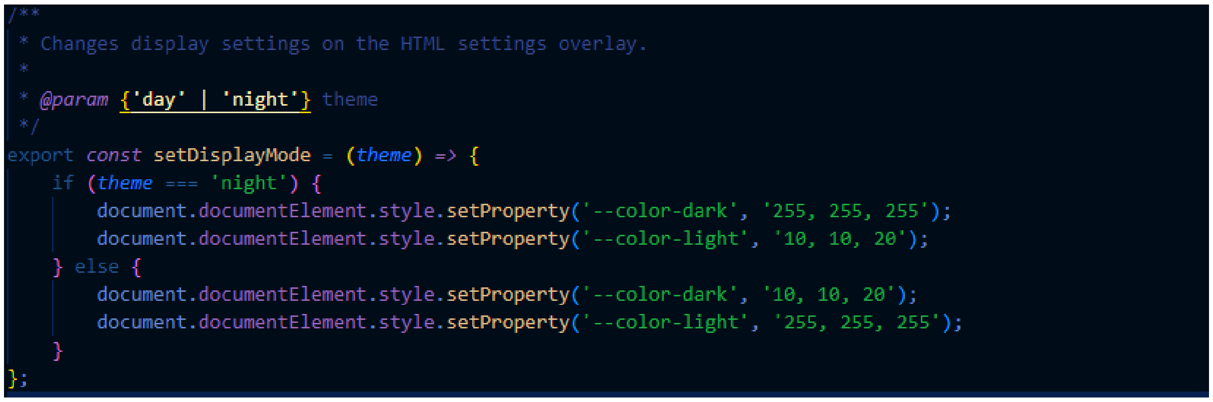
1. All three consist of code that is used multiple times. Helps to prevent mistakes when retyping.

2. Improves readability. Need to understand it once to know what it does everytime you see it.
3. Allows separation of the code into modules by function.

---

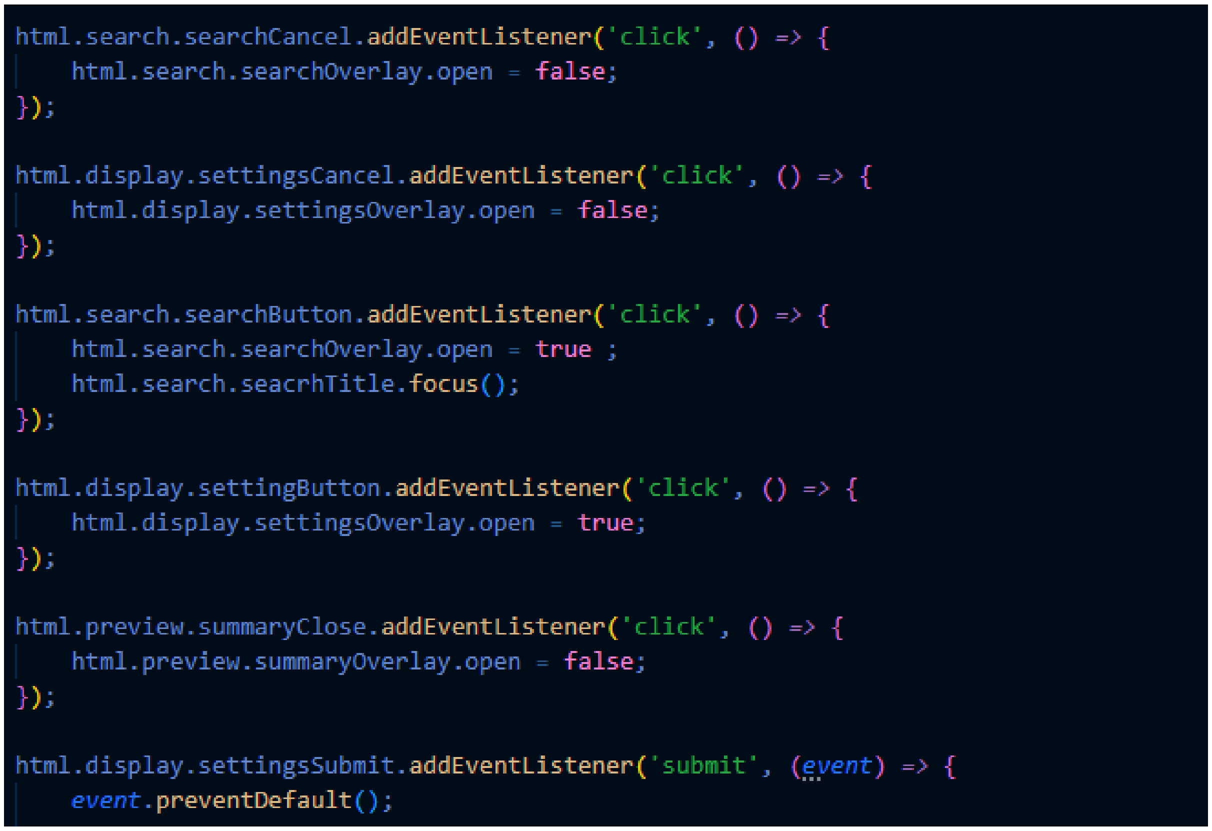
2. Which were the three worst abstractions, and why?

Reason: Could have created a function to handle all event listeners.
SOLID: 1. Will not handle a single responsibility.



```
/**
 * Changes display settings on the HTML settings overlay.
 *
 * @param {'day' | 'night'} theme
 */
export const setDisplayMode = (theme) => {
    if (theme === 'night') {
        document.documentElement.style.setProperty('--color-dark', '255, 255, 255');
        document.documentElement.style.setProperty('--color-light', '10, 10, 20');
    } else {
        document.documentElement.style.setProperty('--color-dark', '10, 10, 20');
        document.documentElement.style.setProperty('--color-light', '255, 255, 255');
    }
};
```

Reason: This is used twice. Although not as many times as Scalk suggested, in this case it can be used as an example.
SOLID: 1. Does not handle a single responsibility.
2. Does not adhere to the dependency inversion principle.


```
html.search.searchCancel.addEventListener('click', () => {
    html.search.searchOverlay.open = false;
});

html.display.settingsCancel.addEventListener('click', () => {
    html.display.settingsOverlay.open = false;
});

html.search.searchButton.addEventListener('click', () => {
    html.search.searchOverlay.open = true ;
    html.search.seacrhTitle.focus();
});

html.display.settingButton.addEventListener('click', () => {
    html.display.settingsOverlay.open = true;
});

html.preview.summaryClose.addEventListener('click', () => {
    html.preview.summaryOverlay.open = false;
});

html.display.settingsSubmit.addEventListener('submit', (event) => {
    event.preventDefault();
```

The app is relatively small, therefore very little code is used multiple times. Abstracting such code will result in over-abstraction. This will reduce readability and make it difficult to understand the function of all code. For this reason not many abstractions were made.

---

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

The first might be better converted into a function, object or class which will handle all event listeners. It could also be separated into modules that handle a single functionality. Search with all it’s event listeners and their handlers. Then another with the scroll functionality with all it’s event listeners and handlers.

The second could be broken up into 2 functions. Each handling only 1 responsibility. To change settings to night mode or to change settings to day mode.
