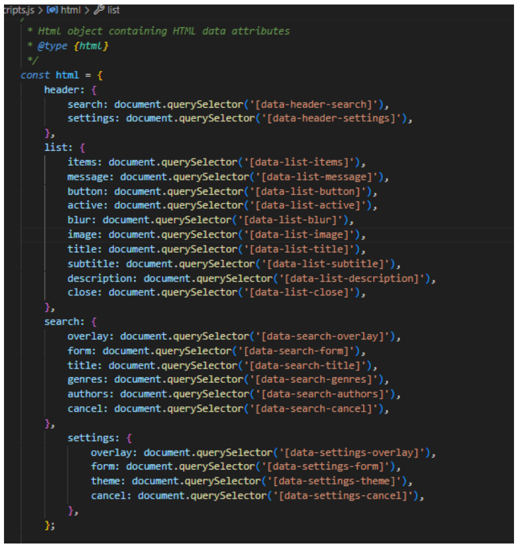
DWA_07.4 Challenge 1

Which were the three best abstractions, and why?

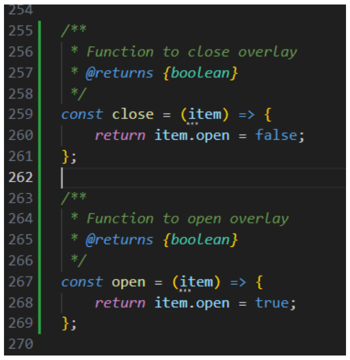
1. Creating DOM elements object



```
ripts.js > html > list
 * Html object containing HTML data attributes
 * @type {html}
 */
const html = {
    header: {
        search: document.querySelector('[data-header-search]'),
        settings: document.querySelector('[data-header-settings]'),
    },
    list: {
        items: document.querySelector('[data-list-items]'),
        message: document.querySelector('[data-list-message]'),
        button: document.querySelector('[data-list-button]'),
        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]'),
    },
    search: {
        overlay: document.querySelector('[data-search-overlay]'),
        form: document.querySelector('[data-search-form]'),
        title: document.querySelector('[data-search-title]'),
        genres: document.querySelector('[data-search-genres]'),
        authors: document.querySelector('[data-search-authors]'),
        cancel: document.querySelector('[data-search-cancel]'),
    },
        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]'),
        },
    };
```

This abstraction collects the HTML DOM elements from the html object which includes ‘header’, ‘list’, ‘search’ and ‘settings’, each with their own data attributes. This allows for easy access to HTML DOM elements.

2. Function to open and close overlay



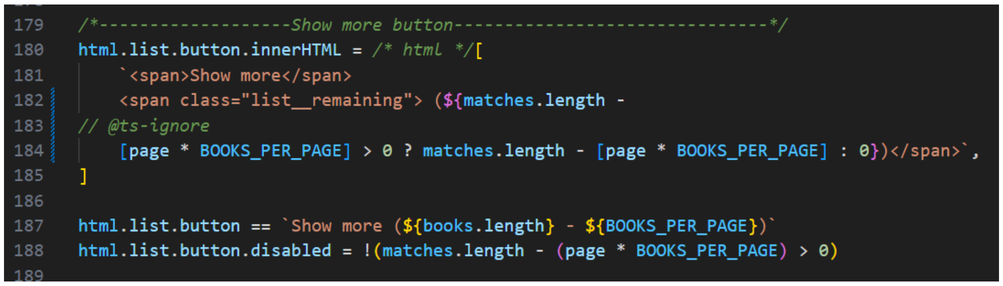
```
/**
 * Function to close overlay
 * @returns {boolean}
 */
const close = (item) => {
    return item.open = false;
};

/**
 * Function to open overlay
 * @returns {boolean}
 */
const open = (item) => {
    return item.open = true;
};
```

This code represents 2 functions. This function was created to open or close multiple overlays.

Which were the three worst abstractions, and why?

1. HTML list having the same output

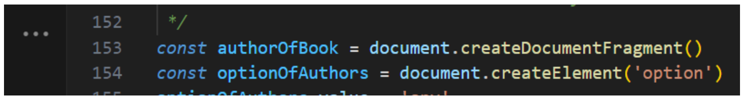


```
/*------------------Show more button------------------------------*/
html.list.button.innerHTML = /* html */[
    `<span>Show more</span>
    <span class="list__remaining"> (${matches.length -
// @ts-ignore
    [page * BOOKS_PER_PAGE] > 0 ? matches.length - [page * BOOKS_PER_PAGE] : 0})</span>`,
]

html.list.button == `Show more (${books.length} - ${BOOKS_PER_PAGE})`
html.list.button.disabled = !(matches.length - (page * BOOKS_PER_PAGE) > 0)
```

This code combines multiple responsibilities into a single block of code which can cause issues with modification in the future. It would be best to separate the responsibilities into separate functions making the code more modular and easier to understand.

2. Multiple fragments being created



```
*/
const authorOfBook = document.createDocumentFragment()
const optionOfAuthors = document.createElement('option')
```

There are multiple fragments being created and serves the same purpose for ‘author’, ‘genre’ and ‘preview’.

How can the three worst abstractions be improved via SOLID principles?

1. The Single Responsibility Principle can be used since the HTML list code serves the same purpose, which means it should only have one purpose or responsibility and should not be affected when making changes to the code.

2. Multiple fragments. You can apply Liskov Substitution Principle in which you can make a fragment with a default argument and the callbacks can added where needed. The parent can be passed to the child and its contents can be changed without the main functionality of the parent be affected.