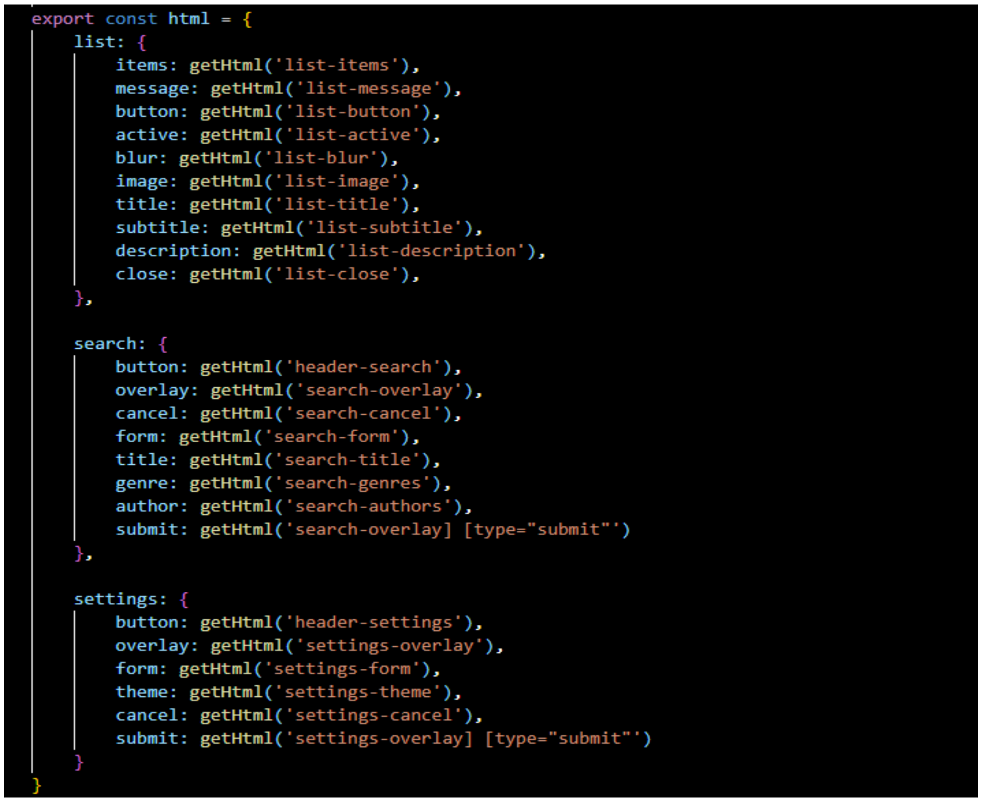
DWA_07.4 Knowledge Check_DWA7

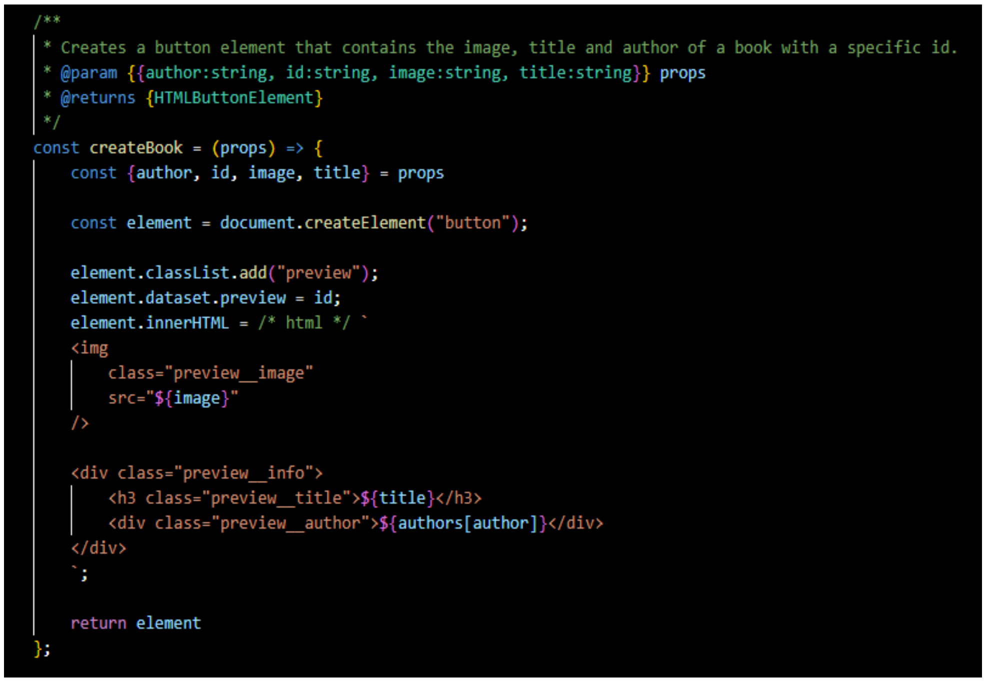
1. Which were the three best abstractions, and why?

1. Placing all of the DOM references in one single object. Grouping all of these references makes for easier access and usability of the code.



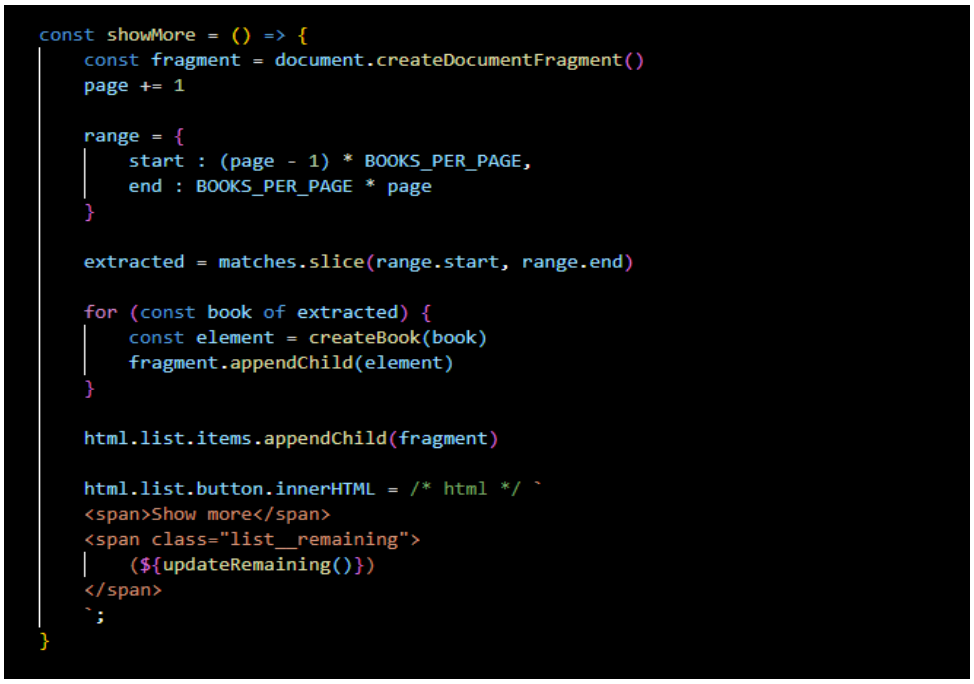
```
export const html = {
    list: {
        items: getHtml('list-items'),
        message: getHtml('list-message'),
        button: getHtml('list-button'),
        active: getHtml('list-active'),
        blur: getHtml('list-blur'),
        image: getHtml('list-image'),
        title: getHtml('list-title'),
        subtitle: getHtml('list-subtitle'),
        description: getHtml('list-description'),
        close: getHtml('list-close'),
    },

    search: {
        button: getHtml('header-search'),
        overlay: getHtml('search-overlay'),
        cancel: getHtml('search-cancel'),
        form: getHtml('search-form'),
        title: getHtml('search-title'),
        genre: getHtml('search-genres'),
        author: getHtml('search-authors'),
        submit: getHtml('search-overlay] [type="submit"')
    },

    settings: {
        button: getHtml('header-settings'),
        overlay: getHtml('settings-overlay'),
        form: getHtml('settings-form'),
        theme: getHtml('settings-theme'),
        cancel: getHtml('settings-cancel'),
        submit: getHtml('settings-overlay] [type="submit"')
    }
}
```

2. A function that has one purpose. It makes code easier to maintain and for comes making errors.



```
/**
 * Creates a button element that contains the image, title and author of a book with a specific id.
 * @param {{author:string, id:string, image:string, title:string}} props
 * @returns {HTMLButtonElement}
 */
const createBook = (props) => {
    const {author, id, image, title} = props

    const element = document.createElement("button");

    element.classList.add("preview");
    element.dataset.preview = id;
    element.innerHTML = /* html */ `
    <img
        class="preview__image"
        src="${image}"
    />

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

    return element
};
```

3. Functions that are made up of smaller functions that handle different functionality. It breaks down the complexity into more maintainable pieces.

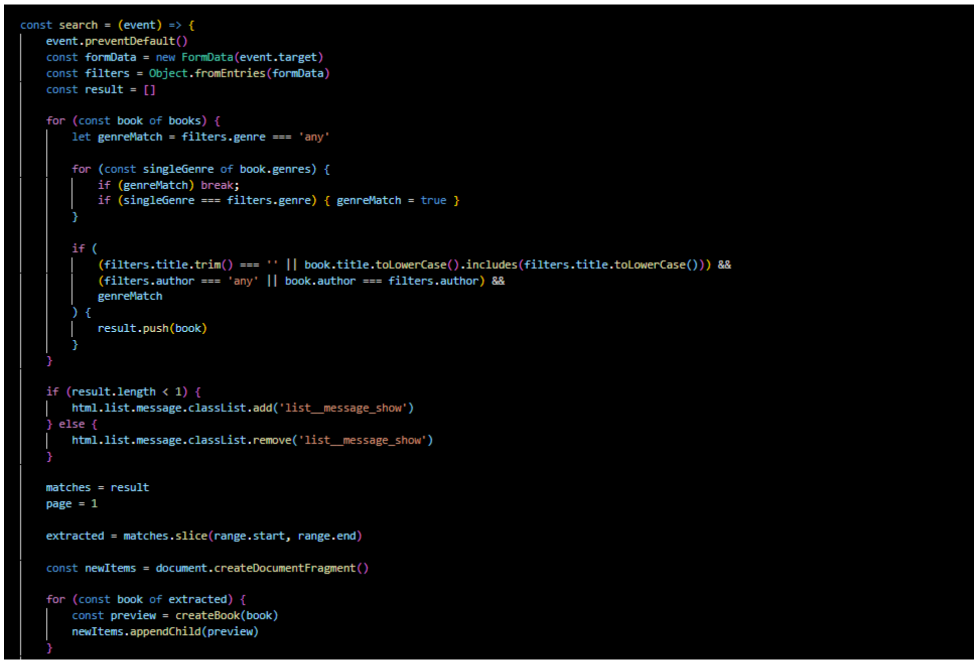


```
const showMore = () => {
    const fragment = document.createDocumentFragment()
    page += 1

    range = {
        start : (page - 1) * BOOKS_PER_PAGE,
        end : BOOKS_PER_PAGE * page
    }

    extracted = matches.slice(range.start, range.end)

    for (const book of extracted) {
        const element = createBook(book)
        fragment.appendChild(element)
    }

    html.list.items.appendChild(fragment)

    html.list.button.innerHTML = /* html */ `
    <span>Show more</span>
    <span class="list__remaining">
        (${updateRemaining()})
    </span>
    `;
}
```

---

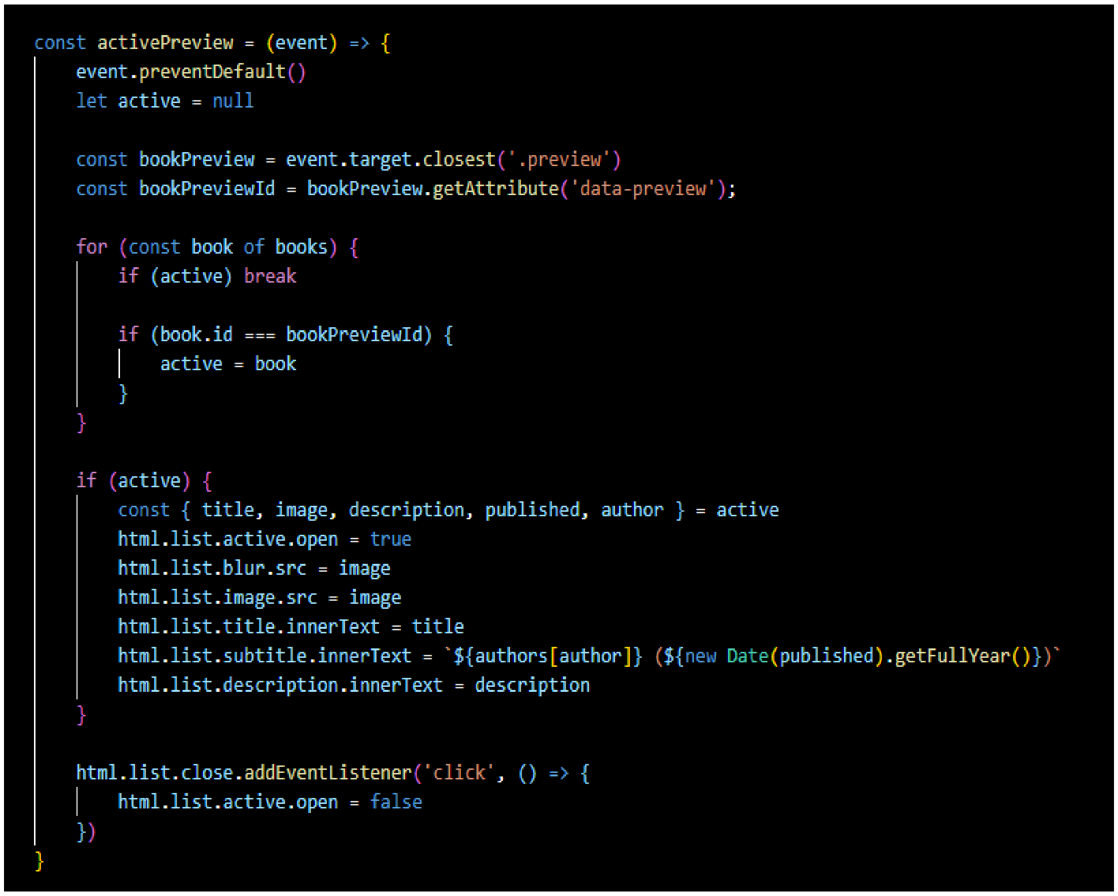
2. Which were the three worst abstractions, and why?

1. Functions made up of multiple if statements and for loops. It places too much complexity inside of one function.



```
const search = (event) => {
    event.preventDefault()
    const formData = new FormData(event.target)
    const filters = Object.fromEntries(formData)
    const result = []

    for (const book of books) {
        let genreMatch = filters.genre === 'any'

        for (const singleGenre of book.genres) {
            if (genreMatch) break;
            if (singleGenre === filters.genre) { genreMatch = true }
        }

        if (
            (filters.title.trim() === '' || book.title.toLowerCase().includes(filters.title.toLowerCase())) &&
            (filters.author === 'any' || book.author === filters.author) &&
            genreMatch
        ) {
            result.push(book)
        }
    }

    if (result.length < 1) {
        html.list.message.classList.add('list__message_show')
    } else {
        html.list.message.classList.remove('list__message_show')
    }

    matches = result
    page = 1

    extracted = matches.slice(range.start, range.end)

    const newItems = document.createDocumentFragment()

    for (const book of extracted) {
        const preview = createBook(book)
        newItems.appendChild(preview)
    }
```

2. A Function performing multiple actions. It increases the complexity inside one function which is unnecessary.



```
const activePreview = (event) => {
    event.preventDefault()
    let active = null

    const bookPreview = event.target.closest('.preview')
    const bookPreviewId = bookPreview.getAttribute('data-preview');

    for (const book of books) {
        if (active) break

        if (book.id === bookPreviewId) {
            active = book
        }
    }

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

    html.list.close.addEventListener('click', () => {
        html.list.active.open = false
    })
}
```

---

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

1. The **SRP** principle can be used to improve both abstractions mentioned in the previous question. The **SRP** principle states that a class or module should have only one reason to change. Each function should be responsible for a single functionality or feature, and should not be coupled with other functionalities. These functions needs to be broken up into smaller functions and should only handle one specific task or responsibility.
