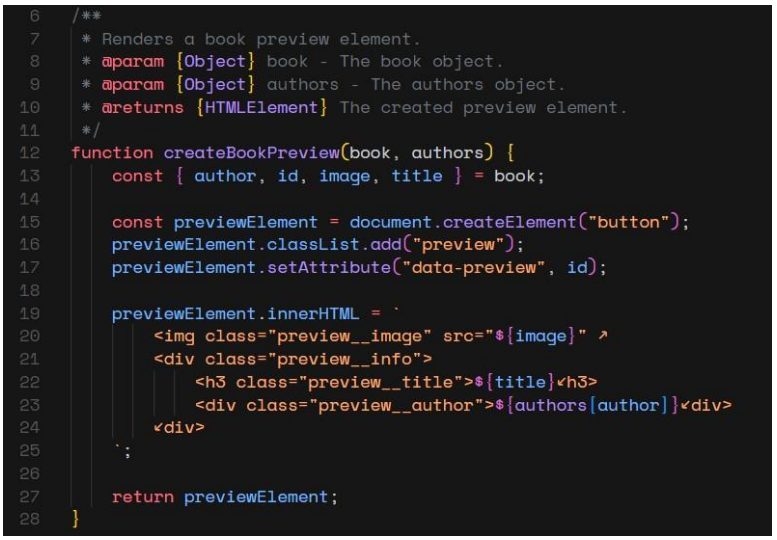
DWA07 Challenge

Which were the three best abstractions, and why?

createBookPreview:

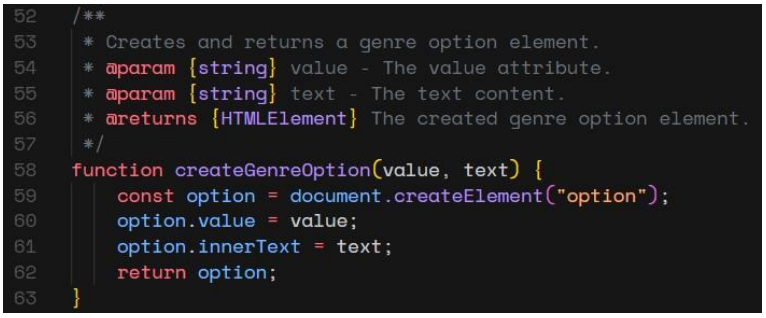
This function makes it easier to create book preview elements. It hides the complex details of creating the preview's look and structure.



```
/**
 * Renders a book preview element.
 * @param {Object} book - The book object.
 * @param {Object} authors - The authors object.
 * @returns {HTMLElement} The created preview element.
 */
function createBookPreview(book, authors) {
    const { author, id, image, title } = book;

    const previewElement = document.createElement("button");
    previewElement.classList.add("preview");
    previewElement.setAttribute("data-preview", id);

    previewElement.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>
    `;

    return previewElement;
}
```

createGenreOption:

This function simplifies making genre options. It takes care of the nitty-gritty parts of creating an option element for different genres.



```
/**
 * Creates and returns a genre option element.
 * @param {string} value - The value attribute.
 * @param {string} text - The text content.
 * @returns {HTMLElement} The created genre option element.
 */
function createGenreOption(value, text) {
    const option = document.createElement("option");
    option.value = value;
    option.innerText = text;
    return option;
}
```

createAuthorOption:

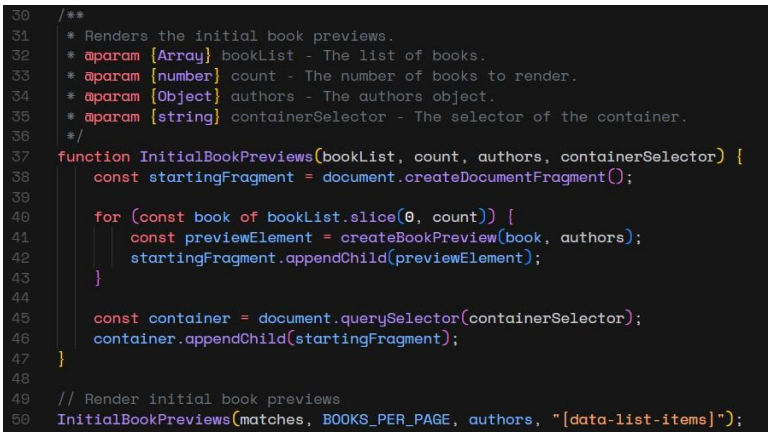
Similar to createGenreOption, this function simplifies creating author options, handling the details so you don't have to.

```
/**
 * Creates and returns an author option element.
 * @param {string} value - The value attribute.
 * @param {string} text - The text content.
 * @returns {HTMLElement} The created author option element.
 */
function createAuthorOption(value, text) {
    const option = document.createElement("option");
    option.value = value;
    option.innerText = text;
    return option;
}
```

Which were the three worst abstractions, and why?

InitialBookPreviews:

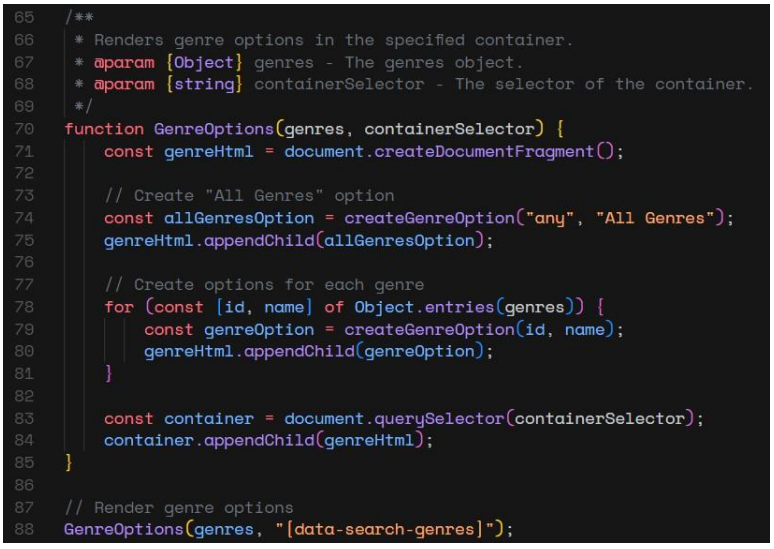
This function tries to do too much, both rendering previews and managing the initial setup. It would be better to split these tasks into separate functions.



```
/**
 * Renders the initial book previews.
 * @param {Array} bookList - The list of books.
 * @param {number} count - The number of books to render.
 * @param {Object} authors - The authors object.
 * @param {string} containerSelector - The selector of the container.
 */
function InitialBookPreviews(bookList, count, authors, containerSelector) {
    const startingFragment = document.createDocumentFragment();

    for (const book of bookList.slice(0, count)) {
        const previewElement = createBookPreview(book, authors);
        startingFragment.appendChild(previewElement);
    }

    const container = document.querySelector(containerSelector);
    container.appendChild(startingFragment);
}

// Render initial book previews
InitialBookPreviews(matches, BOOKS_PER_PAGE, authors, "[data-list-items]");
```

GenreOptions:

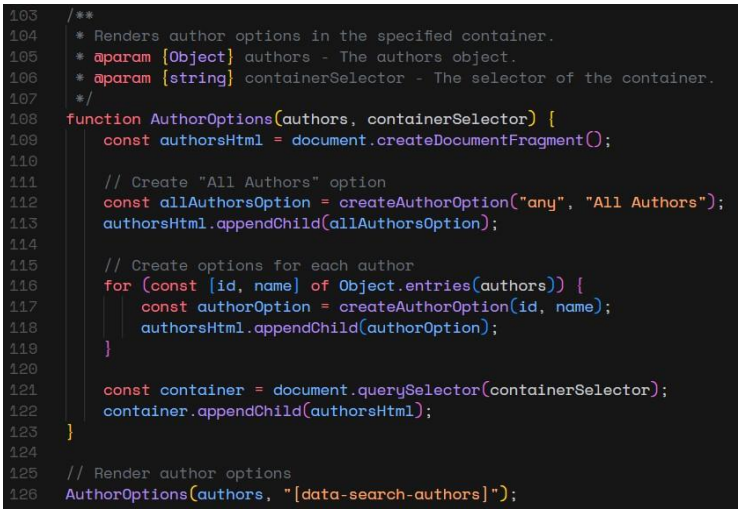
Similar to the first one, this function also tries to do two things at once, creating options and putting them on the screen. It's better to separate these actions.



```
/**
 * Renders genre options in the specified container.
 * @param {Object} genres - The genres object.
 * @param {string} containerSelector - The selector of the container.
 */
function GenreOptions(genres, containerSelector) {
    const genreHtml = document.createDocumentFragment();

    // Create "All Genres" option
    const allGenresOption = createGenreOption("any", "All Genres");
    genreHtml.appendChild(allGenresOption);

    // Create options for each genre
    for (const [id, name] of Object.entries(genres)) {
        const genreOption = createGenreOption(id, name);
        genreHtml.appendChild(genreOption);
    }

    const container = document.querySelector(containerSelector);
    container.appendChild(genreHtml);
}

// Render genre options
GenreOptions(genres, "[data-search-genres]");
```

AuthorOptions:

Just like the other two, this function mixes two tasks, making author options and showing them. It's clearer if these actions are separate.



```
/**
 * Renders author options in the specified container.
 * @param {Object} authors - The authors object.
 * @param {string} containerSelector - The selector of the container.
 */
function AuthorOptions(authors, containerSelector) {
    const authorsHtml = document.createDocumentFragment();

    // Create "All Authors" option
    const allAuthorsOption = createAuthorOption("any", "All Authors");
    authorsHtml.appendChild(allAuthorsOption);

    // Create options for each author
    for (const [id, name] of Object.entries(authors)) {
        const authorOption = createAuthorOption(id, name);
        authorsHtml.appendChild(authorOption);
    }

    const container = document.querySelector(containerSelector);
    container.appendChild(authorsHtml);
}

// Render author options
AuthorOptions(authors, "[data-search-authors]");
```

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

InitialBookPreviews:

Single Responsibility Principle:

Single Responsibility Principle:
Split this into two functions: one for creating book previews (createBookPreviews) and another for handling initial setup (setupInitialPreviews).

Open/Closed Principle:
Let the preview creation be more flexible. Allow users to customize each preview's look without changing the main code.

GenreOptions and AuthorOptions:

Single Responsibility Principle:
Divide this into two parts: one for making options (makeGenreOptions or makeAuthorOptions) and another for showing options (displayOptions).

Open/Closed Principle:
Make it easy to add new options without touching the code that displays them. You can pass a list of options to the display function instead of hardcoding it.