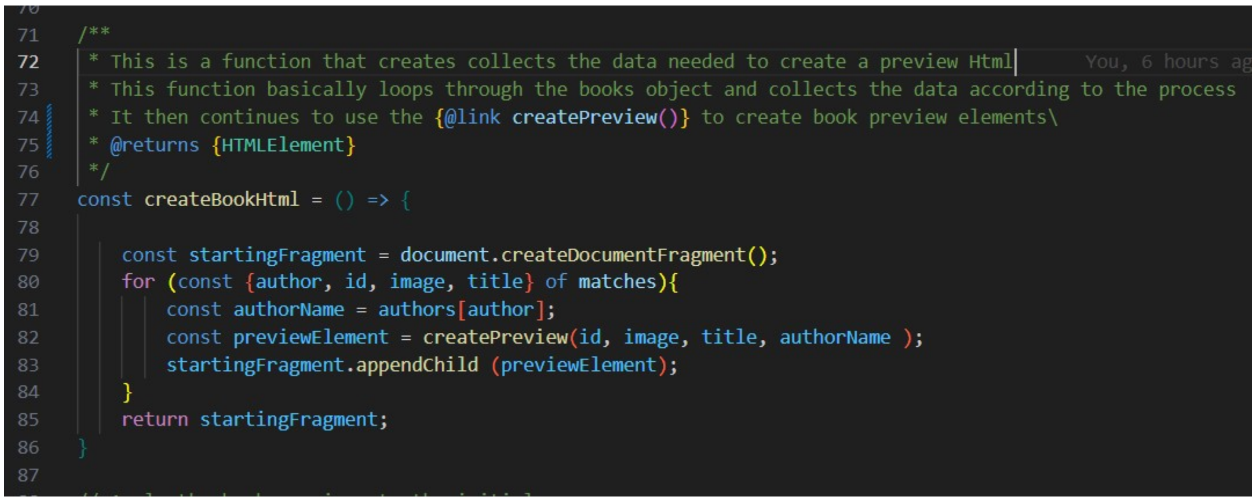
DWA_07.4 Knowledge Check_DWA7

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

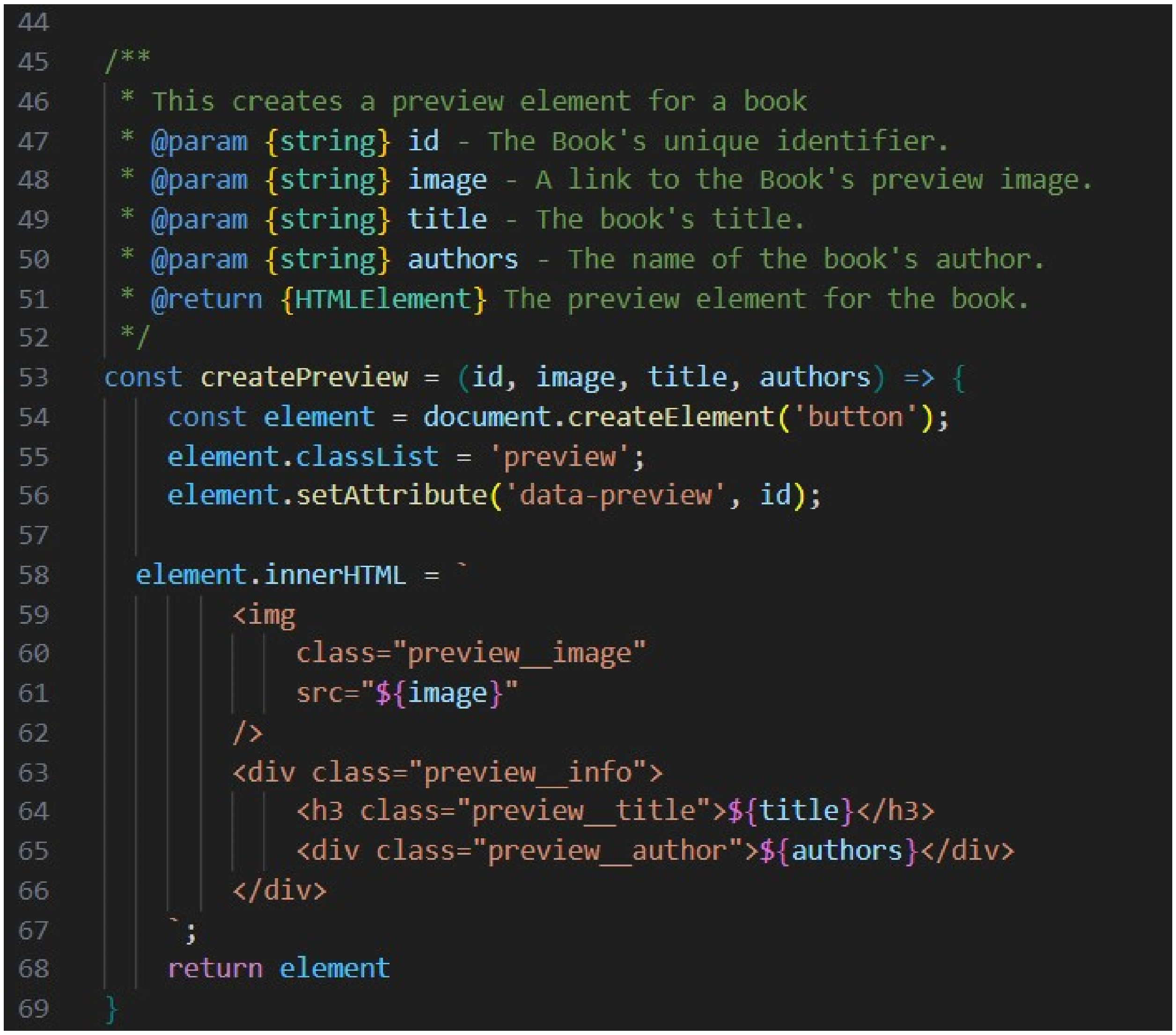


```
/**
 * This is a function that creates collects the data needed to create a preview Html        You, 6 hours ag
 * This function basically loops through the books object and collects the data according to the process
 * It then continues to use the {@link createPreview()} to create book preview elements\
 * @returns {HTMLElement}
 */
const createBookHtml = () => {

    const startingFragment = document.createDocumentFragment();
    for (const {author, id, image, title} of matches){
        const authorName = authors[author];
        const previewElement = createPreview(id, image, title, authorName );
        startingFragment.appendChild (previewElement);
    }
    return startingFragment;
}

```

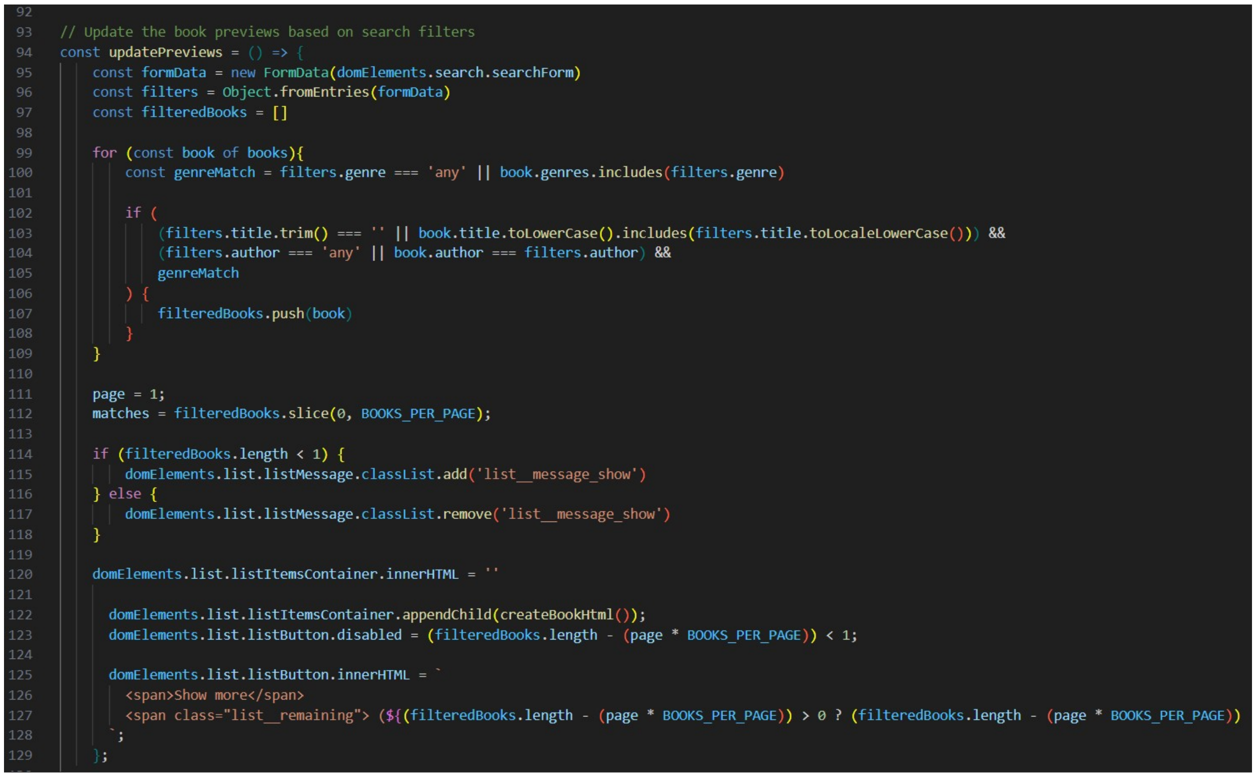
The function above abstracts the process of creating a preview Html element for a book. This code enables reusability and improves code readability by encapsulating the details of preview element creation.



```

/**
 * This creates a preview element for a book
 * @param {string} id - The Book's unique identifier.
 * @param {string} image - A link to the Book's preview image.
 * @param {string} title - The book's title.
 * @param {string} authors - The name of the book's author.
 * @return {HTMLElement} The preview element for the book.
 */
const createPreview = (id, image, title, authors) => {
    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}</div>
          </div>
    `;
    return element
}
```



```javascript

// Update the book previews based on search filters
const updatePreviews = () => {
    const formData = new FormData(domElements.search.searchForm)
    const filters = Object.fromEntries(formData)
    const filteredBooks = []

    for (const book of books){
        const genreMatch = filters.genre === 'any' || book.genres.includes(filters.genre)

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

    page = 1;
    matches = filteredBooks.slice(0, BOOKS_PER_PAGE);

    if (filteredBooks.length < 1) {
        domElements.list.listMessage.classList.add('list__message_show')
    } else {
        domElements.list.listMessage.classList.remove('list__message_show')
    }

    domElements.list.listItemsContainer.innerHTML = ''

      domElements.list.listItemsContainer.appendChild(createBookHtml());
      domElements.list.listButton.disabled = (filteredBooks.length - (page * BOOKS_PER_PAGE)) < 1;

      domElements.list.listButton.innerHTML = `
        <span>Show more</span>
        <span class="list__remaining"> (${(filteredBooks.length - (page * BOOKS_PER_PAGE)) > 0 ? (filteredBooks.length - (page * BOOKS_PER_PAGE))
      `;
    };
```

The above function is responsible for updating the books previews based on search filteres. This abstraction allows the code to focus on the result of the filtering process rather than the specific filtering logic.

---

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

a.The code directly accesses and modifies global variables. This is a bother as it can cause an issues over time if the application grows and adapts new features.

b.The code has some inconsistencies in the naminng convention for instance some variables use camelCase while others use snake_case. This can lead to confusion and make the codebase more difficult to read.

c.The code logic is mixed together meaning, logic that has a specific functions is mixed together with the others makes the code less modular and less maintainable.

---

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

a.It would be better to encapsulate the data within modules or objects and provide controlled access through interfaces or methods.

b.By applying the SRP and IRP you will be able to define clear and consistent naming conventions within each module or class. Each module or class focuses on a single responsibility, making it easier to choose appropriate and consistent names.

C. Using the SRP to separate UI-related code from core logic by creating separate modules or classes for UI interactions and business logic.