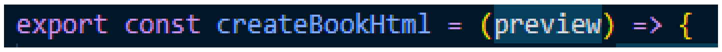
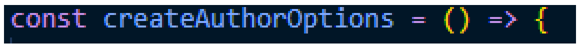
DWA_07.4 Knowledge Check_DWA7

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

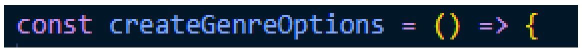


```
export const createBookHtml = (preview) => {
```

It is responsible for a single task: creating the HTML representation of a book preview (image, info, author, genre). Because it only has one purpose, it becomes easier to understand and potentially reuse. It obeys the Open/Closed Principle. The function can be extended (adding additional information to the preview) or written over without altering its base implementation.
It uses encapsulation by hiding the internal details of the HTML contributing to a clearer looking code, which is more maintainable in the long run.

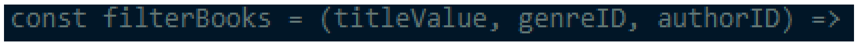


```
const createAuthorOptions = () => {
```

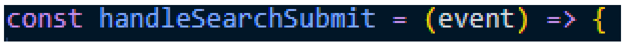


```
const createGenreOptions = () => {
```

They are responsible for singular tasks, such as populating the author options and genre options within a select element respectively. If there are edits to be made they can be done so without editing the core logic such as error handling or localisation (for authors with names not written in the latin alphabet). The same can be said for features like sorting genres by popular or user-generated tags (such as Ao3), or adjusting the visible top genres based on user preferences. These are still flawed.



```
const filterBooks = (titleValue, genreID, authorID) =>
```

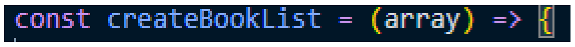
A good abstraction (had I completed it) would have been separating the book filtering feature from 

```
const handleSearchSubmit = (event) => {
```

as it currently manages both submitting the search, the return of applicable data, and display of said data in the same function. Through separation the filter can be made modular, and thus reusable elsewhere.

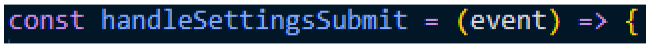
---

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



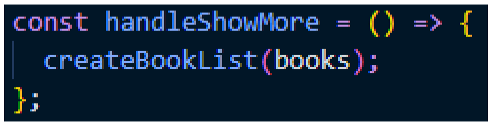
```
const createBookList = (array) => {
```

This function has multiple responsibilities: it extracts a set of books from an array, updates state, and creates HTML elements to be appended to the DOM.



```
const handleSettingsSubmit = (event) => {
```

It connects the toggle for changing between light and dark mode to form submission as well as an external toggle that opens and closes the settings menu after confirming a setting.

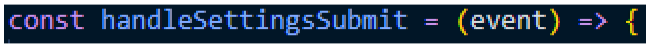


```
const handleShowMore = () => {
  createBookList(books);
};
```

Literally exists to call createBookList(books). It's responsible for handling the show more action and directly invokes the createBookList function.

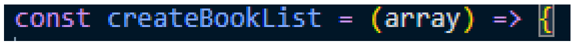
---

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



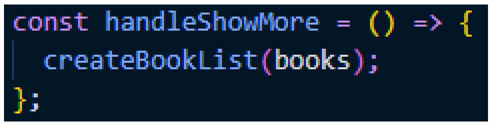
```
const handleSettingsSubmit = (event) => {
```

It currently is responsible for form submission and updating the theme. Extract the theme manipulation into a separate function independent of the setting being submitted.



```
const createBookList = (array) => {
```

It is currently responsible for creating a list of books, updating the state, and manipulating the DOM to display the book list. Refactor the function to focus solely on creating a list of books based on the provided array and separate out the rest. By separating it from interacting with DOM elements we can allow the code to be edited with substitute logic for book list creation. This allows extension and the addition of new behaviours promoting reusability.



```
const handleShowMore = () => {
  createBookList(books);
};
```

The responsibility of triggering the creation of the book list can be moved to a separate function leaving handleShowMore to only be focused on the 'Show More' action rather than the generation of the book list. Having them separate means you can adjust the behaviour of handleShowMore without impacting list generation, enabling edits like changing the number of books shown when 'Show More' is clicked.