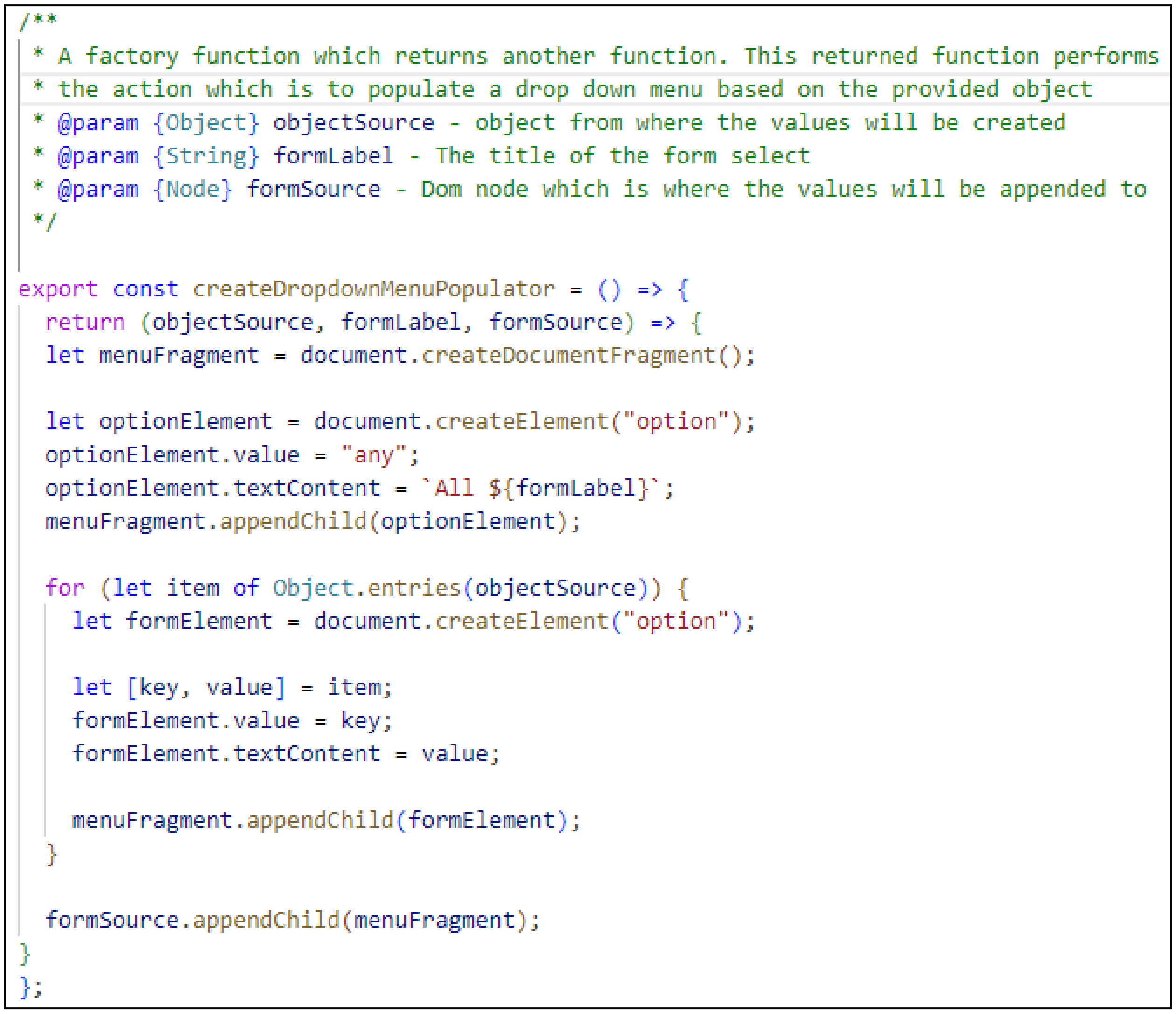
# DWA_08 Discussion Questions

In this module you will continue with your “Book Connect” codebase, and further iterate on your abstractions. You will be required to create an encapsulated abstraction of the book preview by means of a single factory function. If you are up for it you can also encapsulate other aspects of the app into their own abstractions.

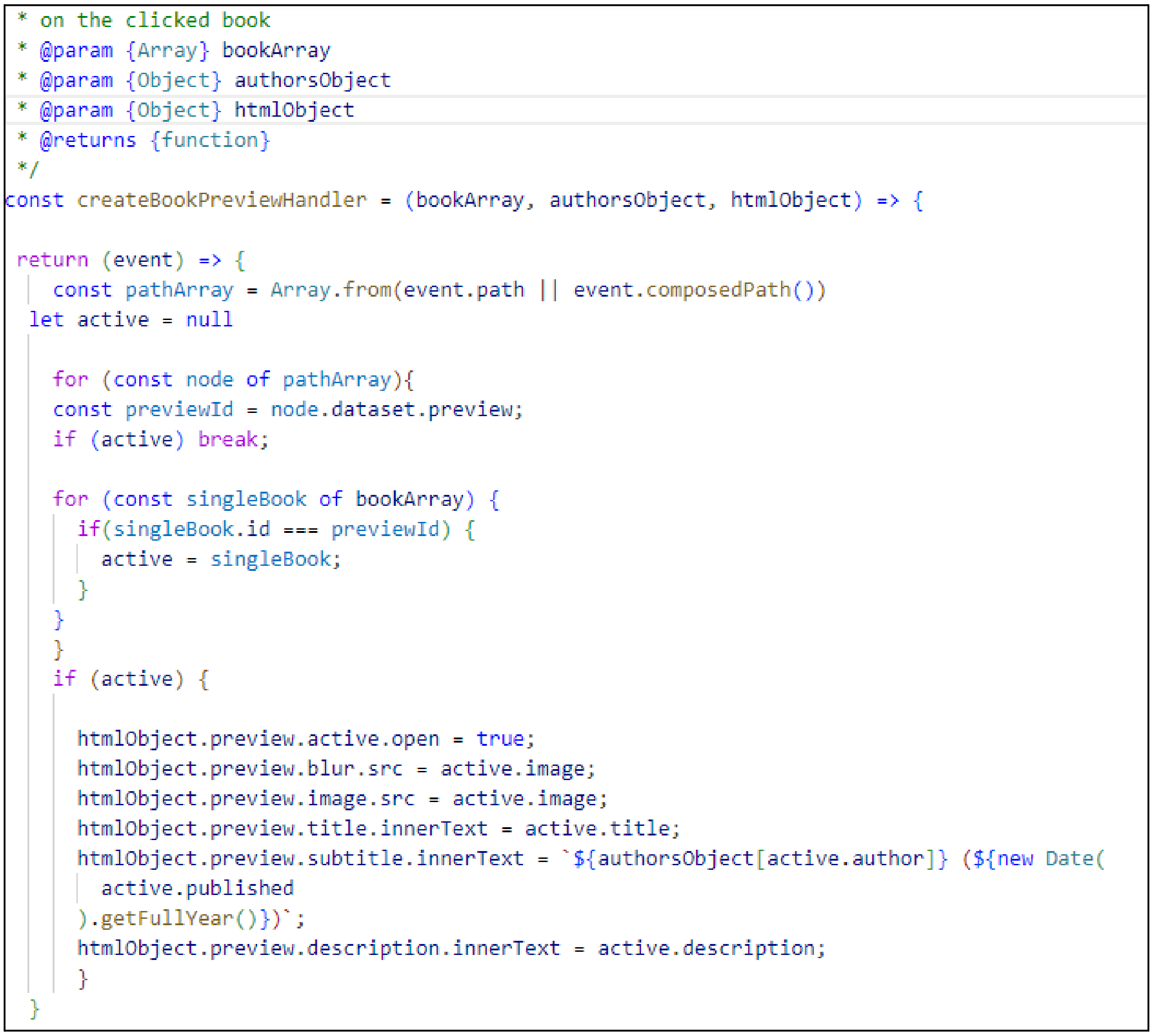
To prepare for your session with your coach, please answer the following questions. Then download this document as a PDF and include it in the repository with your code.

---

1. What parts of encapsulating your logic were easy?



```
/**
 * A factory function which returns another function. This returned function performs
 * the action which is to populate a drop down menu based on the provided object
 * @param {Object} objectSource - object from where the values will be created
 * @param {String} formLabel - The title of the form select
 * @param {Node} formSource - Dom node which is where the values will be appended to
 */

export const createDropdownMenuPopulator = () => {
  return (objectSource, formLabel, formSource) => {
  let menuFragment = document.createDocumentFragment();

  let optionElement = document.createElement("option");
  optionElement.value = "any";
  optionElement.textContent = `All ${formLabel}`;
  menuFragment.appendChild(optionElement);

  for (let item of Object.entries(objectSource)) {
    let formElement = document.createElement("option");

    let [key, value] = item;
    formElement.value = key;
    formElement.textContent = value;

    menuFragment.appendChild(formElement);
  }

  formSource.appendChild(menuFragment);
}
};
```

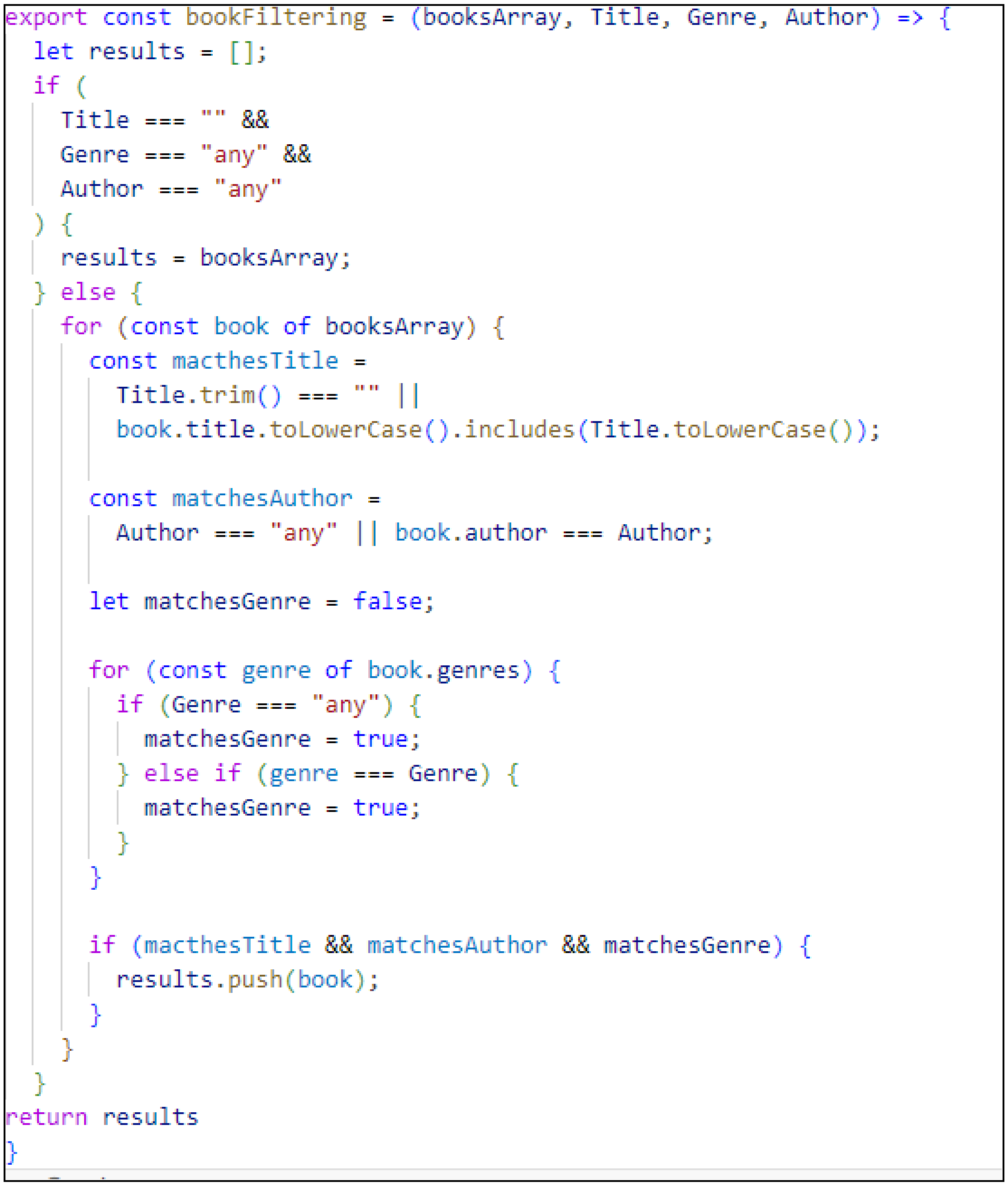


```
 * on the clicked book
 * @param {Array} bookArray
 * @param {Object} authorsObject
 * @param {Object} htmlObject
 * @returns {function}
 */
const createBookPreviewHandler = (bookArray, authorsObject, htmlObject) => {

 return (event) => {
    const pathArray = Array.from(event.path || event.composedPath())
  let active = null

    for (const node of pathArray){
    const previewId = node.dataset.preview;
    if (active) break;

    for (const singleBook of bookArray) {
      if(singleBook.id === previewId) {
        active = singleBook;
      }
    }
    }
    if (active) {

      htmlObject.preview.active.open = true;
      htmlObject.preview.blur.src = active.image;
      htmlObject.preview.image.src = active.image;
      htmlObject.preview.title.innerText = active.title;
      htmlObject.preview.subtitle.innerText = `${authorsObject[active.author]} (${new Date(
        active.published
      ).getFullYear()})`;
      htmlObject.preview.description.innerText = active.description;
      }
  }
```

---

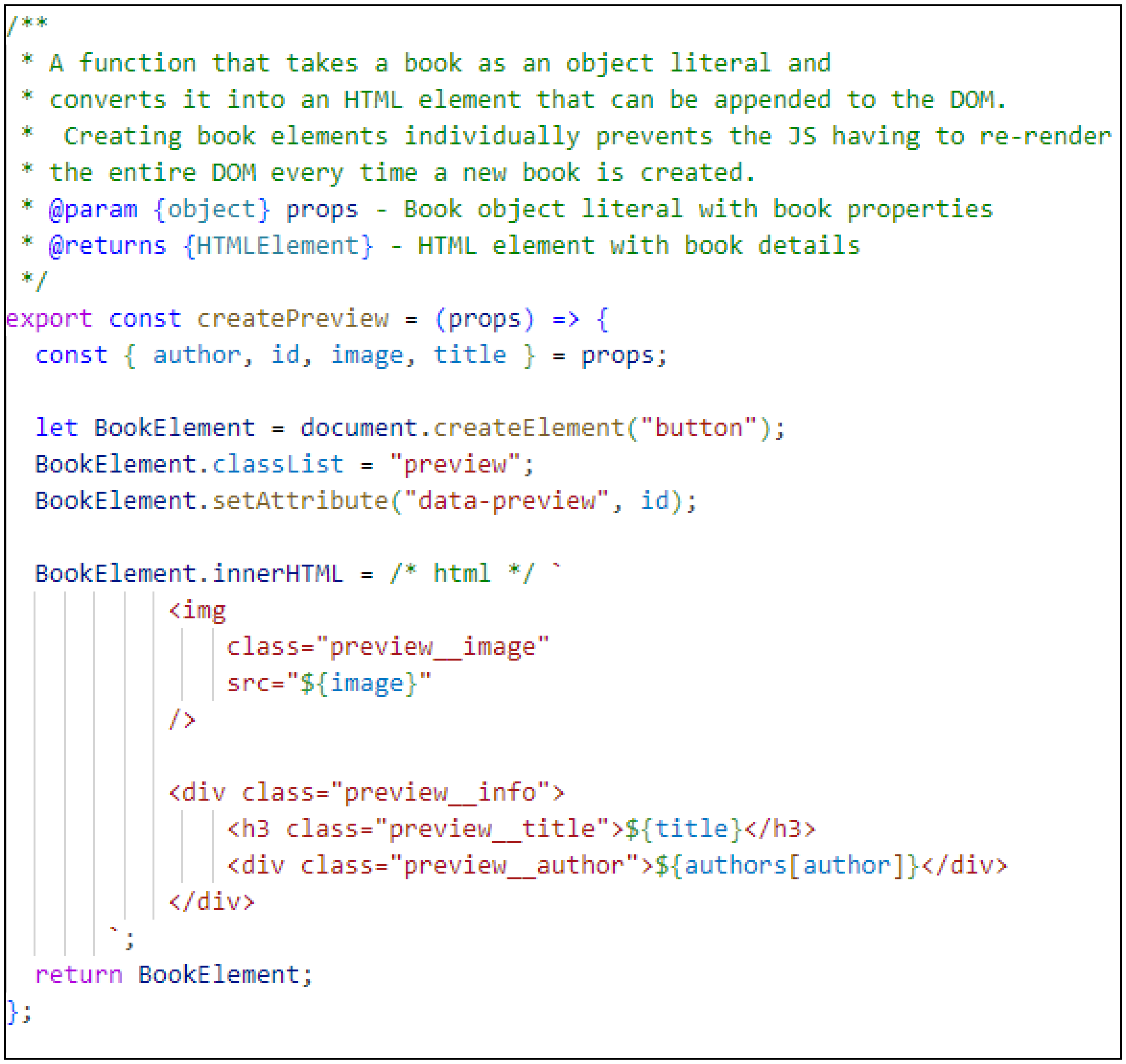
2. What parts of encapsulating your logic were hard?

For context, the function `bookFiltering` is a function which is used to filter books by genre, author and title. This function is used in the filter submission handler. Encapsulating this logic was hard because I do not see how I can change it to a factory function because this function returns an array of filtered books. Factory functions are used for creating objects or instances.



```
export const bookFiltering = (booksArray, Title, Genre, Author) => {
  let results = [];
  if (
    Title === "" &&
    Genre === "any" &&
    Author === "any"
  ) {
    results = booksArray;
  } else {
    for (const book of booksArray) {
      const macthesTitle =
        Title.trim() === "" ||
        book.title.toLowerCase().includes(Title.toLowerCase());

      const matchesAuthor =
        Author === "any" || book.author === Author;

      let matchesGenre = false;

      for (const genre of book.genres) {
        if (Genre === "any") {
          matchesGenre = true;
        } else if (genre === Genre) {
          matchesGenre = true;
        }
      }

      if (macthesTitle && matchesAuthor && matchesGenre) {
        results.push(book);
      }
    }
  }
return results
}
```

The createPreview function was hard to encapsulate because although it does return an object, it does not encapsulate the process of creating objects and customizing them in a way that factory functions do. It just simply changes the book object into an HTML element



```
/**
 * A function that takes a book as an object literal and
 * converts it into an HTML element that can be appended to the DOM.
 *  Creating book elements individually prevents the JS having to re-render
 * the entire DOM every time a new book is created.
 * @param {object} props - Book object literal with book properties
 * @returns {HTMLElement} - HTML element with book details
 */
export const createPreview = (props) => {
  const { author, id, image, title } = props;

  let BookElement = document.createElement("button");
  BookElement.classList = "preview";
  BookElement.setAttribute("data-preview", id);

  BookElement.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 BookElement;
};
```

---

3. Is abstracting the book preview a good or bad idea? Why?
Yes, reasons why it is a good is:

- You are able to reuse the function in certain parts of the codebase and it can be useful if you have multiple elements that also require the same preview functionality.
- It also allows you to separate the logic for creating a book preview from the event handler which makes the codebase more modular and easier to understand as each function has a specific responsibility.

- Also it allows you to test the function or make updates and changes independently without having to trigger a specific event.

---