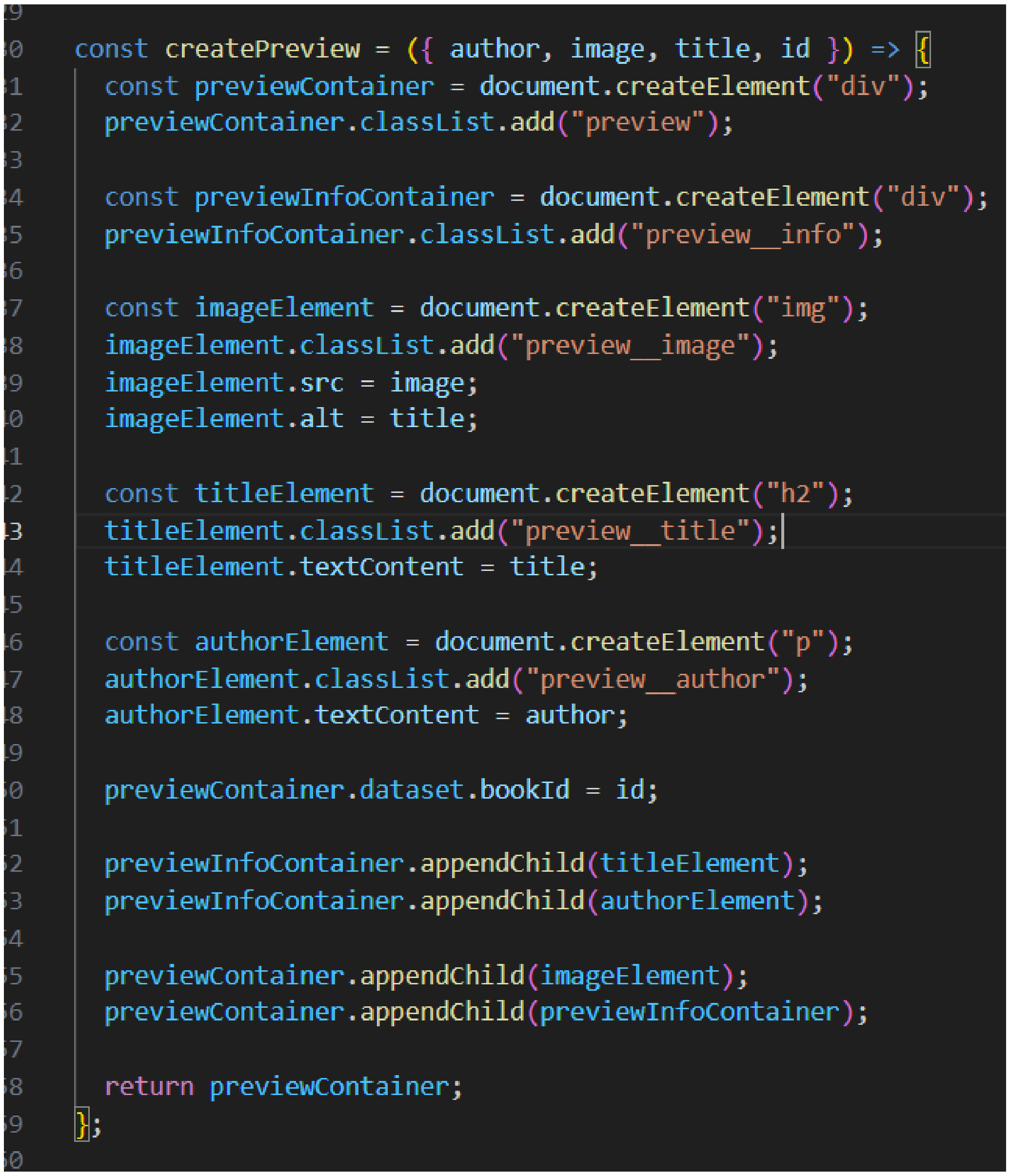
DWA_08 Discussion Questions



```
const createPreview = ({ author, image, title, id }) => {
  const previewContainer = document.createElement("div");
  previewContainer.classList.add("preview");

  const previewInfoContainer = document.createElement("div");
  previewInfoContainer.classList.add("preview__info");

  const imageElement = document.createElement("img");
  imageElement.classList.add("preview__image");
  imageElement.src = image;
  imageElement.alt = title;

  const titleElement = document.createElement("h2");
  titleElement.classList.add("preview__title");
  titleElement.textContent = title;

  const authorElement = document.createElement("p");
  authorElement.classList.add("preview__author");
  authorElement.textContent = author;

  previewContainer.dataset.bookId = id;

  previewInfoContainer.appendChild(titleElement);
  previewInfoContainer.appendChild(authorElement);

  previewContainer.appendChild(imageElement);
  previewContainer.appendChild(previewInfoContainer);

  return previewContainer;
};
```

---

1. What parts of encapsulating your logic were easy?

Making the code readable and maintainable by encapsulating the logic into a single, coherent block. Future modifications, bug fixes, or enhancements can be localized within this encapsulated block, reducing the risk of unintended side effects elsewhere in the application.

Abstracting away the details of DOM manipulation from the rest of the application. This separation allows developers to work on the logic or presentation aspects of the application independently, improving the development workflow.

Integration with Data and binding data attributes (like data-bookId in this case) directly within the encapsulated logic streamlines interactions with the created elements, such as event handling or data retrieval, making it straightforward to associate data with DOM elements.

---

2. What parts of encapsulating your logic were hard?

Handling Complexity: As the requirements for the preview component become more complex (e.g., adding interactivity, handling errors, or dynamically updating content), managing this complexity within a single function can become cumbersome. Ensuring the function remains readable and maintainable while accommodating additional features may require careful planning and refactoring.

Performance Considerations: Creating DOM elements programmatically and manipulating them can be performance-intensive, especially if done frequently or for a large number of elements. Optimizing performance while keeping the encapsulated logic clean and efficient can be challenging, particularly in applications that require high responsiveness or operate on lower-powered devices.

Scalability and Extensibility: Designing the encapsulated function in a way that it remains flexible and easy to extend or modify can be challenging. As new requirements emerge, the function might need to handle additional parameters, alternative layouts, or different kinds of content, requiring a careful balance between specificity and flexibility.

---

3. Is abstracting the book preview a good or bad idea? Why?

Abstracting the book preview is a good idea because it allows you to use the code multiple times if need be, it also reduces seeing the same function over and over again and allows the code to be more maintainable and easy to change.
