IWA19 PRESENTATION TSHEGOFATSO TSITSI

What Issues did I have to solve in this challenge

*__HTML__ - Script.js and data.js ware not “defer”

*__Data.js__ - The data was not exported, so I had to “export const”

*__Script.js__ - Import from data.js, defined the few variables,created preview elements functions, created button to display element and sets it classes and data attributes, Appends the fragment to the DOM elements with the ‘data-list-items’ attribute.

This is just a little summary of some of the changes that I had to make, but there are more than what I have just mentioned. So let’s get into it.

```
import { books, authors, BOOKS_PER_PAGE, genres } from "./data.js";

let page = 1;

const matches = books;
```

```
function createPreview({ author, id, image, title }) {
  let element = document.createElement('button');
  element.classList = 'preview';
  element.setAttribute('data-preview', id);
  element.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 element;
}
```

* I started by importing data from a file called data.js, which contains four variables and this allows JavaScript to import functionality from other files.
* Then defined the two variables page and matches

*And the function creates a priveaw elements for a book, it creates a button elements and sets its class and data attributes.
*And in this function we define `creatPreview` that takes an object as an argument with properties `author`, `id`, `images`, and `title`.
*New HTML `button` element and assigns it to variable

*Then we set the `innerHTML` property of the `button` element to a string containing HTML code. The code includes an `img` element with the `src` attribute set to the `image` property from the augment object, and a `div` element containing an `h3` element with the `title` property from the argument object as its text, and a `div` element with the author name obtained from the`authors` object using the `author` property from the argument object.

*Within this code we create a document fragment and populate it with the first 36 books from the books from the `books` array.
It does this by calling the `creatPriveaw` function for each book and appending the resulting preview button to the fragment. Then, it appends the fragment to the DOM element with the `data-list-items` attribute.

Then, the code sets up a click event listener for the ‘show more’ button, which will display more books when clicked. When the button is clicked, it gets the DOM element with the `data-list-items` atribbuteslice the`matches` array to get the remaining books to be displayed.

```
let fragment = document.createDocumentFragment();
const extracted = books.slice(0, 36);
for (const { author, title, image, id } of extracted) {
  const preview = createPreview({ author, id, image, title });
  fragment.appendChild(preview);
}
const dataListItems = document.querySelector("[data-list-items]");
dataListItems.appendChild(fragment);
```

```
const moreBooks = document.querySelector("[data-list-button]");
let showMore = page * BOOKS_PER_PAGE;
moreBooks.addEventListener("click", () => {
  const dataListItems = document.querySelector("[data-list-items]");
  const remaining = matches.slice(showMore, matches.length);
  const fragment = document.createDocumentFragment();
  for (const { author, title, image, id } of remaining) {
    const preview = createPreview({ author, id, image, title });
    fragment.appendChild(preview);
  }
  dataListItems.appendChild(fragment);
  showMore += remaining.length;
  moreBooks.disabled = !(matches.length - showMore > 0);
});
```

```
// Display "Show more" button
const moreBooks = document.querySelector("[data-list-button]");
moreBooks.innerHTML = /* html */ `
  <span>Show more</span> (
  <span class="list__remaining">${
    matches.length - showMore > 0 ? matches.length - showMore : 0
  }</span> )
`;

// Handle preview click
document.querySelector("[data-list-items]").addEventListener("click", (even
  // Code to handle preview click goes here
});

 //Search modal show
 document.querySelector('[data-header-search]').addEventListener('click', ()
   document.querySelector('[data-search-overlay]').open = true ;
   data-search-title.focus();
 })
```

*On this code creates ‘show more’ button and display the number of remaining items to show based on the length of the `matches` array and the value of the`showMore` variable.

*Then code attaches a click event listener to the list of books. When the user click on book, it retrieves the book data and displays it in a preview moddal

*And then code displays the search modal when the user clicks on the search button in the header

```
// Code to filter books goes here


hide book list
ument.querySelector('[data-list-items]').style.display = 'none'

clear message area
ument.querySelector('[data-list-message]').innerHTML = ''

get form data
st formData = new FormData(event.target)
st title1 = formData.get('title');
st genre1 = formData.get('genre');
st author1 = formData.get('author');

array to store filtered books
st filteredBooks = [];

loop through all books
(let i = 0; i < books.length; i++) {
onst book = books[i];
/ Code to filter books based on user's search criteria goes here


display filtered books
ument.querySelector('[class="list__message"]').style.display = 'block'
```

user's search criteria from the form data, and initializes an empty array to store the filtered books. It then loops through all books and filters them based on the user's search criteria. And it displays the filtered books in the mmessage area.

```
// create fragment to hold filtered books
const fragment2 = document.createDocumentFragment()

for (const {author ,image, title, id , description, published} of filteredBo
  const preview = document.createElement('button')
  // Code to create preview button with book information goes here
  fragment2.appendChild(preview)
}

// add filtered books to message area
const booklist2 = document.querySelector('[class="list__message"]')
booklist2.append(fragment2)
// Drop down for genres
const dataSearchGenres = document.querySelector("[data-search-genres]");
const allGenresOption = document.createElement("option");
allGenres
```

⋆This function creates a preview element for a book. It takes four parameters - author, id, image, and title. It creates a button element and sets its class and data attributes. It then sets the inner HTML of the button to display the book image, title, and author. Finally, it returns the button element.

*In this code we selecting an HTML element with the attribute `data-search-genres` using the `documents.querySelector` method and storing it in the `dataSearchGenres` variable, this will allow us to manipulate the selected element using JavaScript.

```
const css = {
  day: {
    dark: '10, 10, 20',
    light: '255, 255, 255',
  },
  night: {
    dark: '255, 255, 255',
    light: '10, 10, 20',
  }
}
const form = document.getElementById('settings');
//const themeSelect = document.querySelector('[data-settings-theme]');
form.addEventListener('submit', (event) => {
  event.preventDefault();
  const theme = themeSelect.value;
  document.documentElement.style.setProperty('--color-dark', css[theme].dark);
  document.documentElement.style.setProperty('--color-light', css[theme].light);
});
// Initialize theme based on user's OS theme preference
const prefersDarkMode = window.matchMedia('(prefers-color-scheme: dark)').matches;
const initialTheme = prefersDarkMode ? 'night' : 'day';
document.documentElement.style.setProperty('--color-dark', css[initialTheme].dark);
document.documentElement.style.setProperty('--color-light', css[initialTheme].light)
```

*This code defines a JavaScript object css that contains two themes (day and night) with their corresponding dark and light colors. It also retrieves an HTML form element with the id of "settings" and adds an event listener to it. This event listener listens for a form submission and prevents the default form submission behavior.
When the form is submitted, the current value of the themeSelect element is retrieved and used to update the --color-dark and --color-light CSS custom properties of the root element of the HTML document.

Finally, the code initializes the theme based on the user's operating system's preferred color scheme using the window.matchMedia method. If the user prefers a dark mode, the initial theme is set to "night" and if they prefer a light mode, the initial theme is set to "day". Then the --color-dark and --color-light CSS custom properties of the root element of the HTML document are set to the corresponding colors of the initial theme.