Code presentation: Book Connect

Sinoxolo Nondala

Steps of debugging

1. Employ static code analysis tools: Utilize specialized software that scans your code for syntax errors and other issues before runtime. These tools can help you catch potential issues early on, saving you time and effort in the long run.

2. Implement automated testing: Use automated testing tools to ensure that any changes you make to your code don't introduce new errors or bugs. These tests can be run quickly and frequently, allowing you to catch issues before they become bigger problems.

3. Refactor your code: Rather than simply adding more code to fix a problem, consider refactoring your existing code to improve its overall structure and readability. This can help you identify the root cause of the problem and prevent similar issues from occurring in the future.

HTML

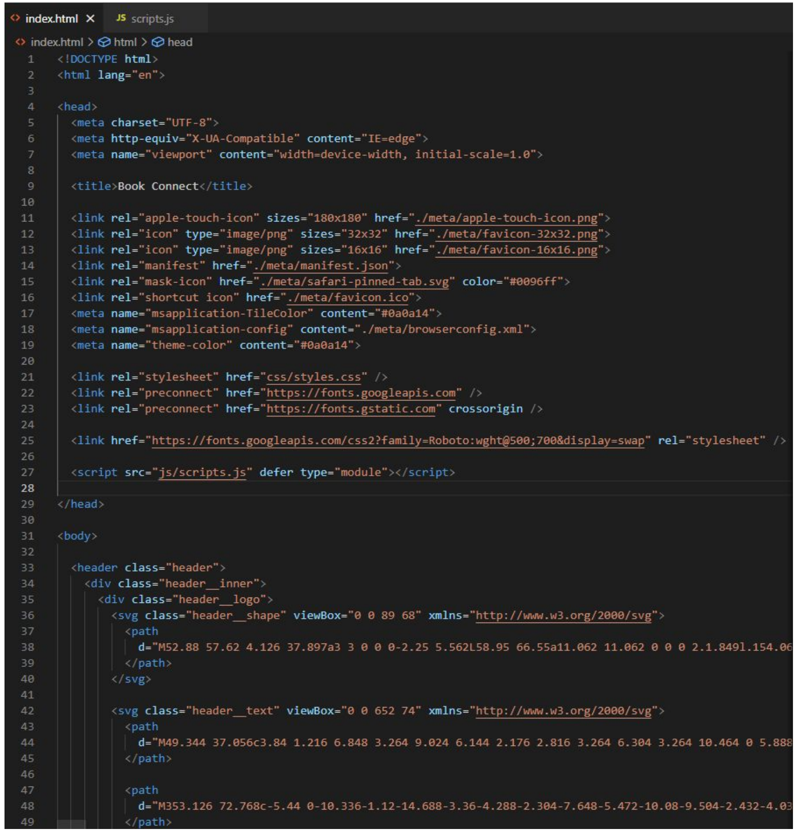
html

- The HTML document defines the structure and content of a web page.

- The <head> element contains metadata about the document, including the title and links to external resources.

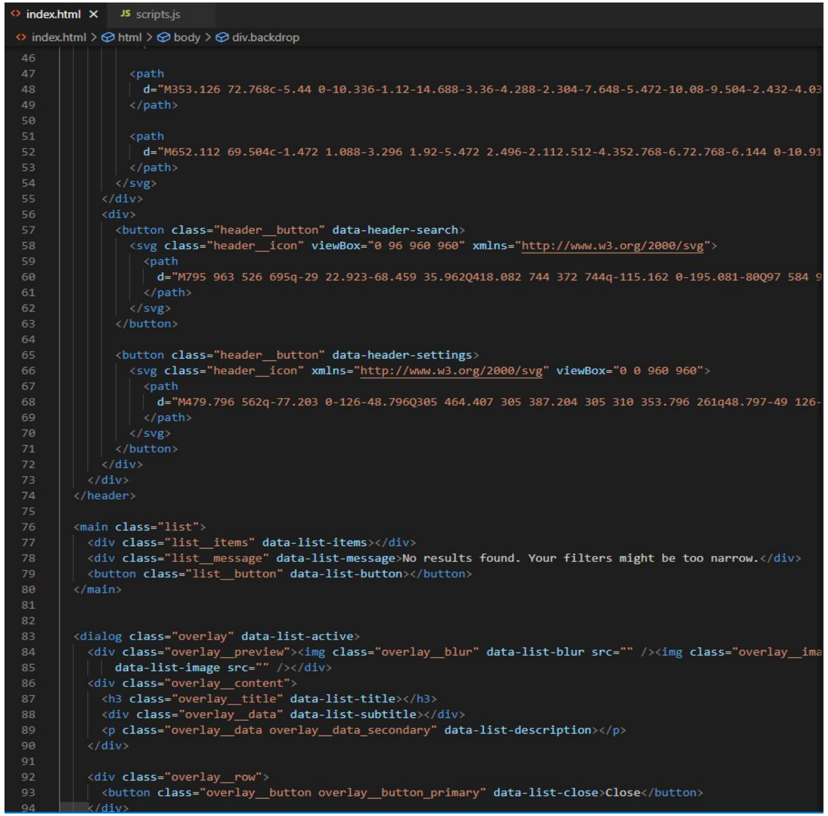
- The <body> element contains the visible content of the web page, including headings, paragraphs, images, and links.

- The <svg> element can be used to create scalable vector graphics, and the <path> element defines the path to be drawn on the SVG canvas.

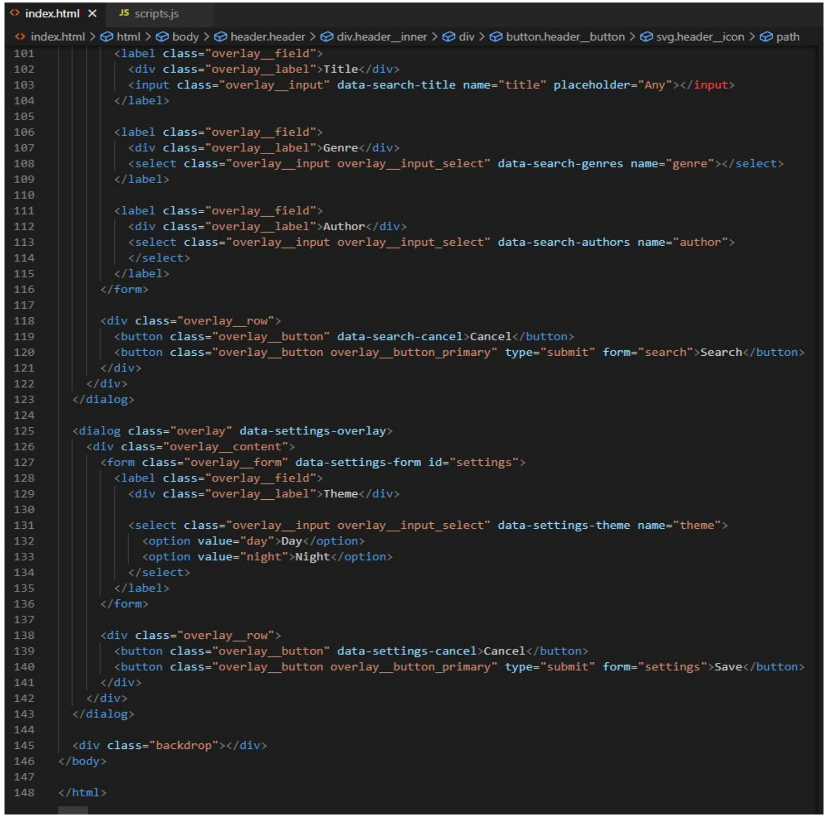


```html
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">

  <title>Book Connect</title>

  <link rel="apple-touch-icon" sizes="180x180" href="./meta/apple-touch-icon.png">
  <link rel="icon" type="image/png" sizes="32x32" href="./meta/favicon-32x32.png">
  <link rel="icon" type="image/png" sizes="16x16" href="./meta/favicon-16x16.png">
  <link rel="manifest" href="./meta/manifest.json">
  <link rel="mask-icon" href="./meta/safari-pinned-tab.svg" color="#0096ff">
  <link rel="shortcut icon" href="./meta/favicon.ico">
  <meta name="msapplication-TileColor" content="#0a0a14">
  <meta name="msapplication-config" content="./meta/browserconfig.xml">
  <meta name="theme-color" content="#0a0a14">

  <link rel="stylesheet" href="css/styles.css" />
  <link rel="preconnect" href="https://fonts.googleapis.com" />
  <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin />

  <link href="https://fonts.googleapis.com/css2?family=Roboto:wght@500;700&display=swap" rel="stylesheet" />

  <script src="js/scripts.js" defer type="module"></script>

</head>

<body>

  <header class="header">
    <div class="header__inner">
      <div class="header__logo">
        <svg class="header__shape" viewBox="0 0 89 68" xmlns="http://www.w3.org/2000/svg">
          <path
            d="M52.88 57.62 4.126 37.897a3 3 0 0 0-2.25 5.562L58.95 66.55a11.062 11.062 0 0 0 2.1.849l.154.06
          </path>
        </svg>

        <svg class="header__text" viewBox="0 0 652 74" xmlns="http://www.w3.org/2000/svg">
          <path
            d="M49.344 37.056c3.84 1.216 6.848 3.264 9.024 6.144 2.176 2.816 3.264 6.304 3.264 10.464 0 5.888
          </path>

          <path
            d="M353.126 72.768c-5.44 0-10.336-1.12-14.688-3.36-4.288-2.304-7.648-5.472-10.08-9.504-2.432-4.03
          </path>
```

html/2



```html
        <path
          d="M353.126 72.768c-5.44 0-10.336-1.12-14.688-3.36-4.288-2.304-7.648-5.472-10.08-9.504-2.432-4.03
        </path>

        <path
          d="M652.112 69.504c-1.472 1.088-3.296 1.92-5.472 2.496-2.112.512-4.352.768-6.72.768-6.144 0-10.91
        </path>
      </svg>
    </div>
    <div>
      <button class="header__button" data-header-search>
        <svg class="header__icon" viewBox="0 96 960 960" xmlns="http://www.w3.org/2000/svg">
          <path
            d="M795 963 526 695q-29 22.923-68.459 35.962Q418.082 744 372 744q-115.162 0-195.081-80Q97 584 9
          </path>
        </svg>
      </button>

      <button class="header__button" data-header-settings>
        <svg class="header__icon" xmlns="http://www.w3.org/2000/svg" viewBox="0 0 960 960">
          <path
            d="M479.796 562q-77.203 0-126-48.796Q305 464.407 305 387.204 305 310 353.796 261q48.797-49 126-
          </path>
        </svg>
      </button>
    </div>
  </div>
</header>

<main class="list">
  <div class="list__items" data-list-items></div>
  <div class="list__message" data-list-message>No results found. Your filters might be too narrow.</div>
  <button class="list__button" data-list-button></button>
</main>


<dialog class="overlay" data-list-active>
  <div class="overlay__preview"><img class="overlay__blur" data-list-blur src="" /><img class="overlay__ima
      data-list-image src="" /></div>
  <div class="overlay__content">
    <h3 class="overlay__title" data-list-title></h3>
    <div class="overlay__data" data-list-subtitle></div>
    <p class="overlay__data overlay__data_secondary" data-list-description></p>
  </div>

  <div class="overlay__row">
    <button class="overlay__button overlay__button_primary" data-list-close>Close</button>
  </div>
```


index.html
scripts.js
index.html > html > body > header.header > div.header__inner > div > button.header__button > svg.header__icon > path

```html
        <label class="overlay__field">
          <div class="overlay__label">Title</div>
          <input class="overlay__input" data-search-title name="title" placeholder="Any"></input>
        </label>

        <label class="overlay__field">
          <div class="overlay__label">Genre</div>
          <select class="overlay__input overlay__input_select" data-search-genres name="genre"></select>
        </label>

        <label class="overlay__field">
          <div class="overlay__label">Author</div>
          <select class="overlay__input overlay__input_select" data-search-authors name="author">
          </select>
        </label>
      </form>

      <div class="overlay__row">
        <button class="overlay__button" data-search-cancel>Cancel</button>
        <button class="overlay__button overlay__button_primary" type="submit" form="search">Search</button>
      </div>
    </div>
  </dialog>

  <dialog class="overlay" data-settings-overlay>
    <div class="overlay__content">
      <form class="overlay__form" data-settings-form id="settings">
        <label class="overlay__field">
          <div class="overlay__label">Theme</div>

          <select class="overlay__input overlay__input_select" data-settings-theme name="theme">
            <option value="day">Day</option>
            <option value="night">Night</option>
          </select>
        </label>
      </form>

      <div class="overlay__row">
        <button class="overlay__button" data-settings-cancel>Cancel</button>
        <button class="overlay__button overlay__button_primary" type="submit" form="settings">Save</button>
      </div>
    </div>
  </dialog>

  <div class="backdrop"></div>
</body>

</html>
```

Html/3

Header section:

- The header section of the code contains two buttons with different classes and data attributes for styling and functionality.
- The first button has an SVG image of a magnifying glass, which is a commonly used icon for search functionality. It also has a data attribute for targeting it with JavaScript.
- The second button has an SVG image of three horizontal lines, which is often used as a menu or settings icon. It also has a data attribute for targeting it with JavaScript.
-

2. Main section:

- The main section of the code contains a div element with the class list__items and a data attribute for storing a list of items. This is likely used for displaying a list of items on the web page.
- The main section also contains a div element with the class list__message and a data attribute for storing a message related to the list of items. This could be used for displaying a message to the user if there are no items in the list, for example.
- Finally, the main section contains a button with the class list__button and a data attribute for storing a button action related to the list of items. This button could be used for adding new items to the list or for triggering some other action.
-

3. Dialogs section:

- The dialogs section of the code contains two dialogs with the class overlay and data attributes for targeting them with JavaScript.
- The first dialog contains an image preview and some content related to the image. This could be used for displaying more information about an image when the user clicks on it, for example.
- The second dialog contains a search form with three input fields for title, genre, and author. Each input field has a data attribute for storing the search term related to that field. This could be used for searching for specific items in the list, for example.

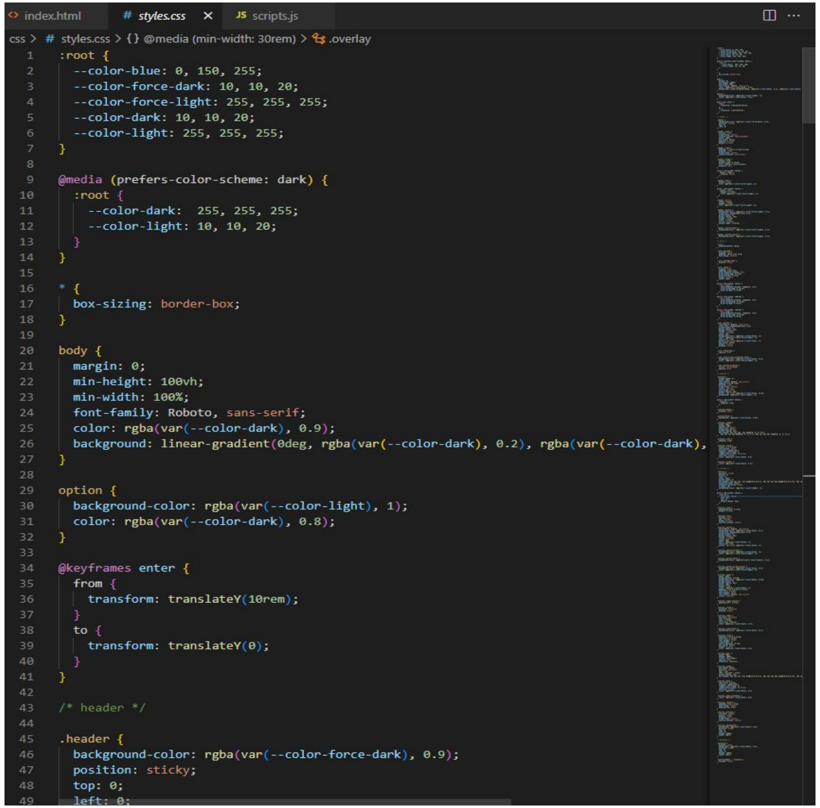
CSS

CSS

Header section

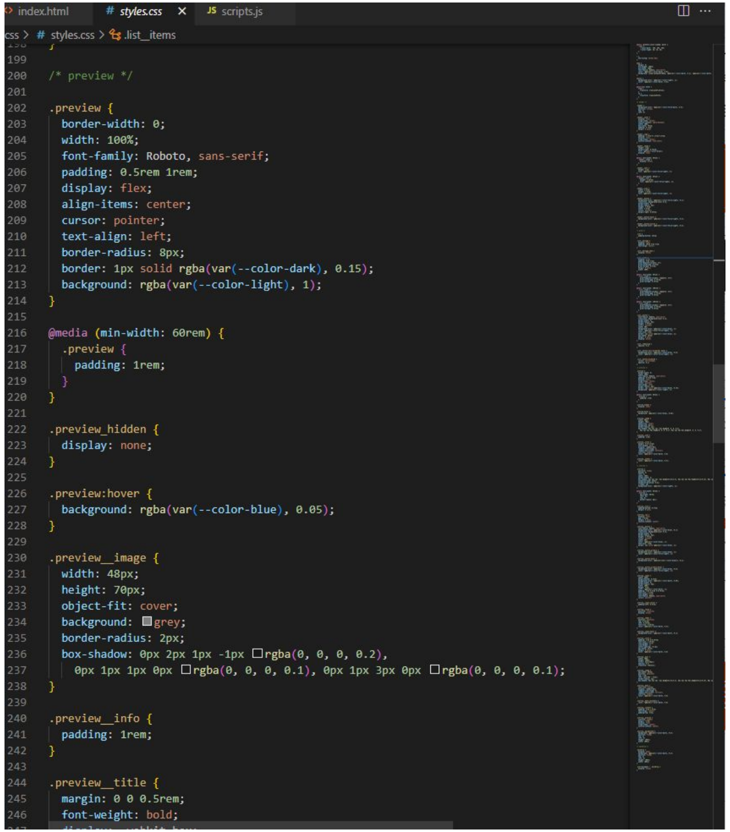
The code defines a set of CSS variables that define various colors used throughout the website.

The code includes media queries that adjust the color scheme based on the user's preference for dark mode or light mode.
The code also includes styling for the header of the website, including the logo, navigation buttons, and color scheme.

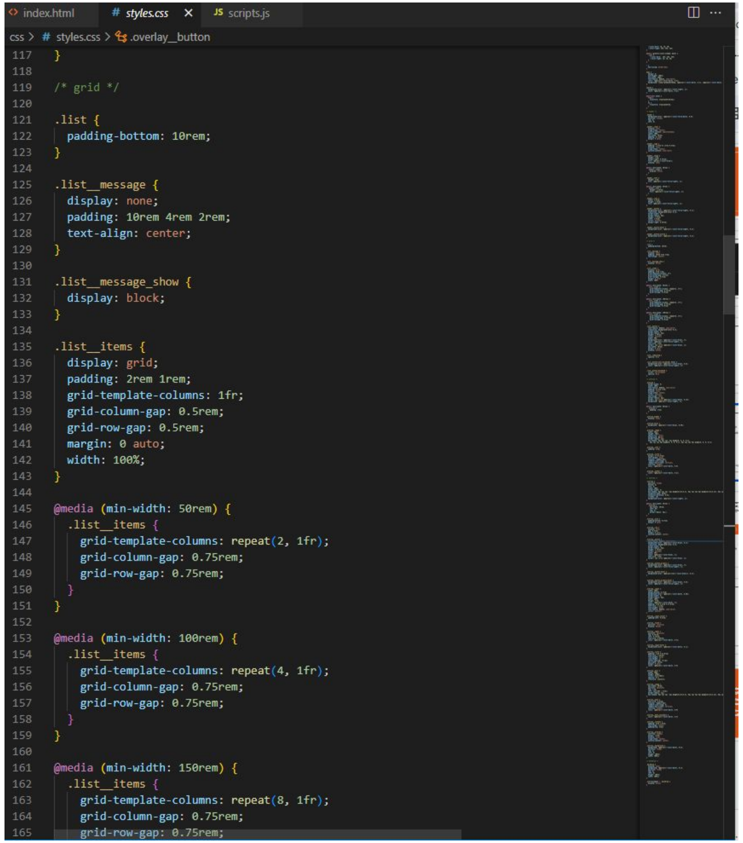

index.html
styles.css
scripts.js
css > # styles.css > {} @media (min-width: 30rem) > .overlay

```css
:root {
  --color-blue: 0, 150, 255;
  --color-force-dark: 10, 10, 20;
  --color-force-light: 255, 255, 255;
  --color-dark: 10, 10, 20;
  --color-light: 255, 255, 255;
}

@media (prefers-color-scheme: dark) {
  :root {
    --color-dark:  255, 255, 255;
    --color-light: 10, 10, 20;
  }
}

* {
  box-sizing: border-box;
}

body {
  margin: 0;
  min-height: 100vh;
  min-width: 100%;
  font-family: Roboto, sans-serif;
  color: rgba(var(--color-dark), 0.9);
  background: linear-gradient(0deg, rgba(var(--color-dark), 0.2), rgba(var(--color-dark),
}

option {
  background-color: rgba(var(--color-light), 1);
  color: rgba(var(--color-dark), 0.8);
}

@keyframes enter {
  from {
    transform: translateY(10rem);
  }
  to {
    transform: translateY(0);
  }
}

/* header */

.header {
  background-color: rgba(var(--color-force-dark), 0.9);
  position: sticky;
  top: 0;
  left: 0;
```

CSS/2



```css
}

/* preview */

.preview {
  border-width: 0;
  width: 100%;
  font-family: Roboto, sans-serif;
  padding: 0.5rem 1rem;
  display: flex;
  align-items: center;
  cursor: pointer;
  text-align: left;
  border-radius: 8px;
  border: 1px solid rgba(var(--color-dark), 0.15);
  background: rgba(var(--color-light), 1);
}

@media (min-width: 60rem) {
  .preview {
    padding: 1rem;
  }
}

.preview_hidden {
  display: none;
}

.preview:hover {
  background: rgba(var(--color-blue), 0.05);
}

.preview__image {
  width: 48px;
  height: 70px;
  object-fit: cover;
  background: grey;
  border-radius: 2px;
  box-shadow: 0px 2px 1px -1px rgba(0, 0, 0, 0.2),
    0px 1px 1px 0px rgba(0, 0, 0, 0.1), 0px 1px 3px 0px rgba(0, 0, 0, 0.1);
}

.preview__info {
  padding: 1rem;
}

.preview__title {
  margin: 0 0 0.5rem;
  font-weight: bold;
```



```css
}

/* grid */

.list {
  padding-bottom: 10rem;
}

.list__message {
  display: none;
  padding: 10rem 4rem 2rem;
  text-align: center;
}

.list__message_show {
  display: block;
}

.list__items {
  display: grid;
  padding: 2rem 1rem;
  grid-template-columns: 1fr;
  grid-column-gap: 0.5rem;
  grid-row-gap: 0.5rem;
  margin: 0 auto;
  width: 100%;
}

@media (min-width: 50rem) {
  .list__items {
    grid-template-columns: repeat(2, 1fr);
    grid-column-gap: 0.75rem;
    grid-row-gap: 0.75rem;
  }
}

@media (min-width: 100rem) {
  .list__items {
    grid-template-columns: repeat(4, 1fr);
    grid-column-gap: 0.75rem;
    grid-row-gap: 0.75rem;
  }
}

@media (min-width: 150rem) {
  .list__items {
    grid-template-columns: repeat(8, 1fr);
    grid-column-gap: 0.75rem;
    grid-row-gap: 0.75rem;
```

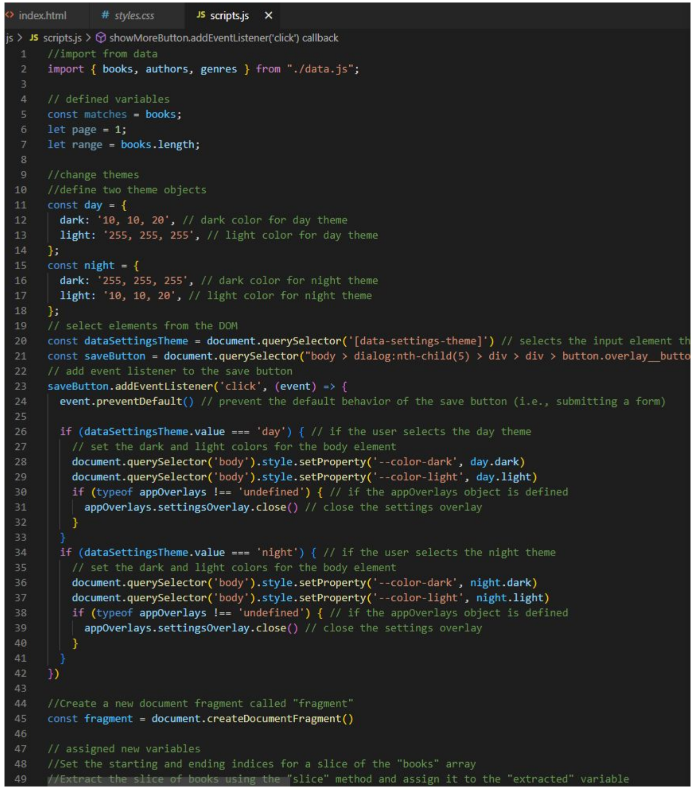
CSS/3

- This is CSS code used to style a web page.
- The first block of code sets custom color variables to be used throughout the page.
- The second block of code sets some styling specific to when the user prefers a dark color scheme.
- The third block of code sets the default box-sizing to border-box for all elements.
- The fourth block of code sets some general styles for the body element.
- The fifth block of code sets some specific styles for option elements.
- The sixth block of code defines a keyframe animation called "enter".
- The remaining code styles specific elements on the page, such as the header and a grid of items.

Javascript

Javascript

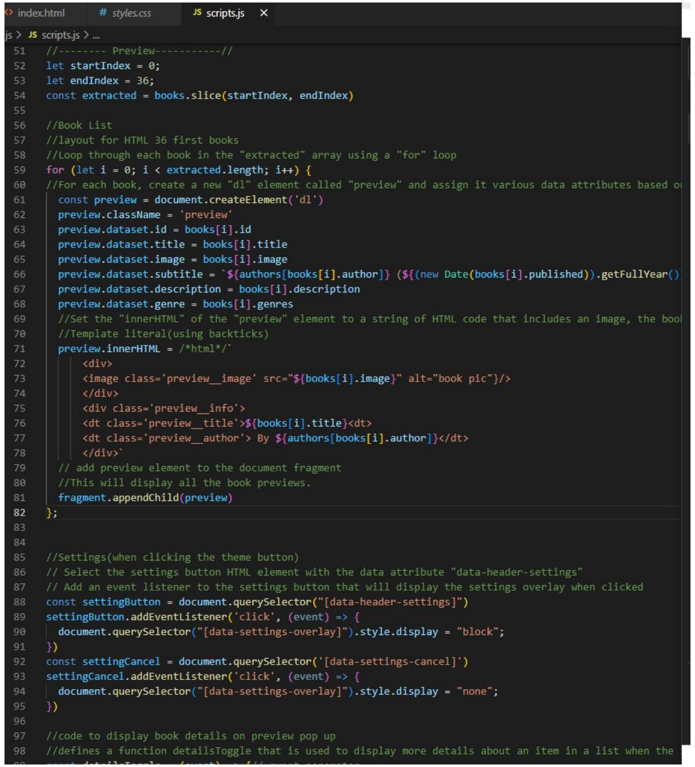
1. The code imports data, defines variables, and creates objects for day and night themes.
2. The code selects elements from the DOM, such as an input element and a save button, and adds an event listener to the save button.
3. When the save button is clicked, the code checks which theme the user selected and updates the colors of the body element accordingly.
4. Finally, the code creates a new document fragment.


index.html
styles.css
scripts.js
js > scripts.js > showMoreButton.addEventListener('click') callback

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

// defined variables
const matches = books;
let page = 1;
let range = books.length;

//change themes
//define two theme objects
const day = {
  dark: '10, 10, 20', // dark color for day theme
  light: '255, 255, 255', // light color for day theme
};
const night = {
  dark: '255, 255, 255', // dark color for night theme
  light: '10, 10, 20', // light color for night theme
};
// select elements from the DOM
const dataSettingsTheme = document.querySelector('[data-settings-theme]') // selects the input element th
const saveButton = document.querySelector("body > dialog:nth-child(5) > div > div > button.overlay__butto
// add event listener to the save button
saveButton.addEventListener('click', (event) => {
  event.preventDefault() // prevent the default behavior of the save button (i.e., submitting a form)

  if (dataSettingsTheme.value === 'day') { // if the user selects the day theme
    // set the dark and light colors for the body element
    document.querySelector('body').style.setProperty('--color-dark', day.dark)
    document.querySelector('body').style.setProperty('--color-light', day.light)
    if (typeof appOverlays !== 'undefined') { // if the appOverlays object is defined
      appOverlays.settingsOverlay.close() // close the settings overlay
    }
  }
  if (dataSettingsTheme.value === 'night') { // if the user selects the night theme
    // set the dark and light colors for the body element
    document.querySelector('body').style.setProperty('--color-dark', night.dark)
    document.querySelector('body').style.setProperty('--color-light', night.light)
    if (typeof appOverlays !== 'undefined') { // if the appOverlays object is defined
      appOverlays.settingsOverlay.close() // close the settings overlay
    }
  }
})

//Create a new document fragment called "fragment"
const fragment = document.createDocumentFragment()

// assigned new variables
//Set the starting and ending indices for a slice of the "books" array
//Extract the slice of books using the "slice" method and assign it to the "extracted" variable
```

Javascript

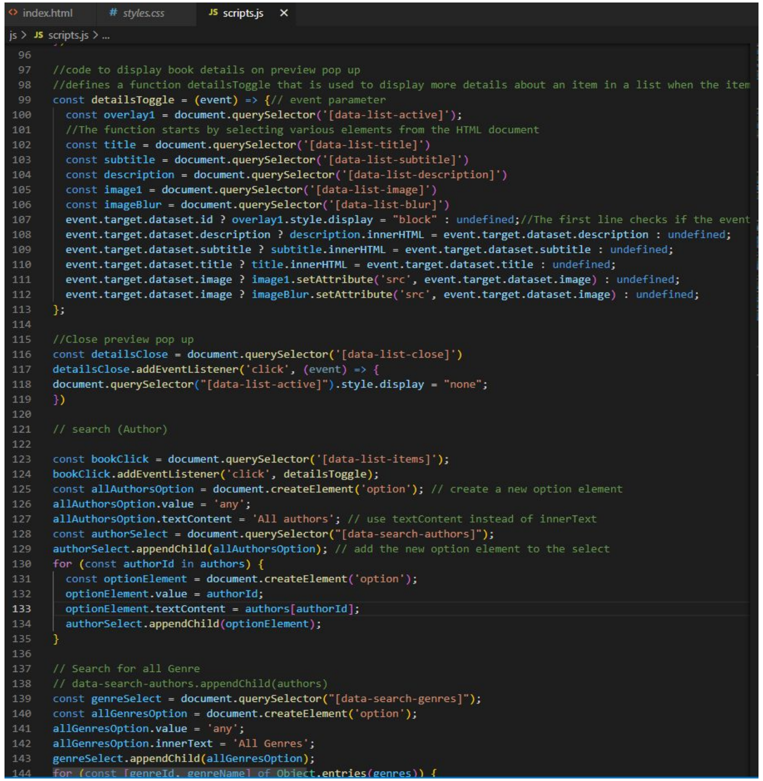
- This code creates a preview of the first 36 books in the "books" array.
- It uses a for loop to iterate through each book in the "extracted" array and creates a new "dl" element called "preview".
- The "preview" element is assigned various data attributes based on the book's properties, and its "innerHTML" is set to a string of HTML code that includes an image, the book title, and author name.
- The preview element is added to a document fragment.
- There are also settings buttons that display a settings overlay when clicked, and a cancel button that hides the overlay.



```javascript
//-------- Preview----------//
let startIndex = 0;
let endIndex = 36;
const extracted = books.slice(startIndex, endIndex)

//Book List
//layout for HTML 36 first books
//Loop through each book in the "extracted" array using a "for" loop
for (let i = 0; i < extracted.length; i++) {
//For each book, create a new "dl" element called "preview" and assign it various data attributes based o
  const preview = document.createElement('dl')
  preview.className = 'preview'
  preview.dataset.id = books[i].id
  preview.dataset.title = books[i].title
  preview.dataset.image = books[i].image
  preview.dataset.subtitle = `${authors[books[i].author]} (${(new Date(books[i].published)).getFullYear()
  preview.dataset.description = books[i].description
  preview.dataset.genre = books[i].genres
  //Set the "innerHTML" of the "preview" element to a string of HTML code that includes an image, the boo
  //Template literal(using backticks)
  preview.innerHTML = /*html*/`
      <div>
      <image class='preview__image' src="${books[i].image}" alt="book pic"}/>
      </div>
      <div class='preview__info'>
      <dt class='preview__title'>${books[i].title}<dt>
      <dt class='preview__author'> By ${authors[books[i].author]}</dt>
      </div>`
  // add preview element to the document fragment
  //This will display all the book previews.
  fragment.appendChild(preview)
};


//Settings(when clicking the theme button)
// Select the settings button HTML element with the data attribute "data-header-settings"
// Add an event listener to the settings button that will display the settings overlay when clicked
const settingButton = document.querySelector("[data-header-settings]")
settingButton.addEventListener('click', (event) => {
  document.querySelector("[data-settings-overlay]").style.display = "block";
})
const settingCancel = document.querySelector('[data-settings-cancel]')
settingCancel.addEventListener('click', (event) => {
  document.querySelector("[data-settings-overlay]").style.display = "none";
})

//code to display book details on preview pop up
//defines a function detailsToggle that is used to display more details about an item in a list when the
```

–Javascript

1. The `details Toggle` function is defined to display more details about an item in a list when the item is clicked. It selects various elements from the HTML document, checks if the event target has specific data attributes, and sets the innerHTML and src attributes of other elements based on those data attributes.
2. The `details Close` event listener is defined to close the pop-up window containing the book details when the close button is clicked. It selects the element with the `data-list-active` attribute and sets its display property to "none".
3. The `book Click` event listener is defined to call the `details Toggle` function when a book item is clicked. It selects the element with the `data-list-items` attribute.
4. Two `for` loops are defined to create and append option elements to two `select` elements in the HTML document. One loop creates and appends option elements for each author, and the other creates and appends option elements for each genre.



```javascript

//code to display book details on preview pop up
//defines a function detailsToggle that is used to display more details about an item in a list when the item
const detailsToggle = (event) => {// event parameter
  const overlay1 = document.querySelector('[data-list-active]');
  //The function starts by selecting various elements from the HTML document
  const title = document.querySelector('[data-list-title]')
  const subtitle = document.querySelector('[data-list-subtitle]')
  const description = document.querySelector('[data-list-description]')
  const image1 = document.querySelector('[data-list-image]')
  const imageBlur = document.querySelector('[data-list-blur]')
  event.target.dataset.id ? overlay1.style.display = "block" : undefined;//The first line checks if the event
  event.target.dataset.description ? description.innerHTML = event.target.dataset.description : undefined;
  event.target.dataset.subtitle ? subtitle.innerHTML = event.target.dataset.subtitle : undefined;
  event.target.dataset.title ? title.innerHTML = event.target.dataset.title : undefined;
  event.target.dataset.image ? image1.setAttribute('src', event.target.dataset.image) : undefined;
  event.target.dataset.image ? imageBlur.setAttribute('src', event.target.dataset.image) : undefined;
};

//Close preview pop up
const detailsClose = document.querySelector('[data-list-close]')
detailsClose.addEventListener('click', (event) => {
document.querySelector("[data-list-active]").style.display = "none";
})

// search (Author)

const bookClick = document.querySelector('[data-list-items]');
bookClick.addEventListener('click', detailsToggle);
const allAuthorsOption = document.createElement('option'); // create a new option element
allAuthorsOption.value = 'any';
allAuthorsOption.textContent = 'All authors'; // use textContent instead of innerText
const authorSelect = document.querySelector("[data-search-authors]");
authorSelect.appendChild(allAuthorsOption); // add the new option element to the select
for (const authorId in authors) {
  const optionElement = document.createElement('option');
  optionElement.value = authorId;
  optionElement.textContent = authors[authorId];
  authorSelect.appendChild(optionElement);
}

// Search for all Genre
// data-search-authors.appendChild(authors)
const genreSelect = document.querySelector("[data-search-genres]");
const allGenresOption = document.createElement('option');
allGenresOption.value = 'any';
allGenresOption.innerText = 'All Genres';
genreSelect.appendChild(allGenresOption);
for (const [genreId, genreName] of Object.entries(genres)) {
```

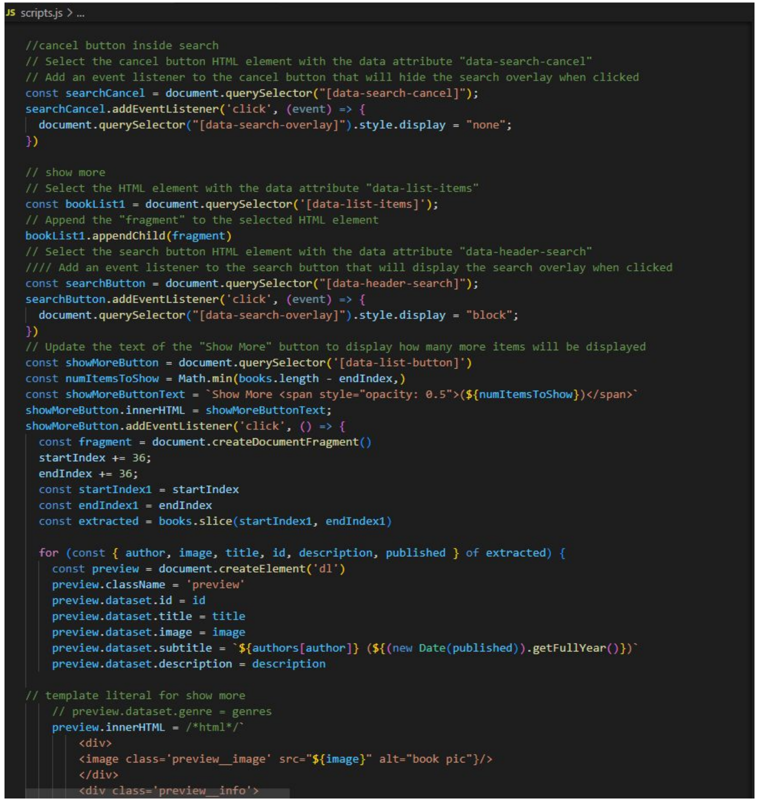
–Javascript

1. When the "cancel" button inside the search overlay is clicked, it will hide the search overlay.

2. When the "search" button is clicked, it will display the search overlay.

3. When the "show more" button is clicked, it will add more items to the list, and update the "Show More" button text to display how many more items will be displayed.

4. The code uses HTML data attributes to select specific elements in the HTML document.



```
//cancel button inside search
// Select the cancel button HTML element with the data attribute "data-search-cancel"
// Add an event listener to the cancel button that will hide the search overlay when clicked
const searchCancel = document.querySelector("[data-search-cancel]");
searchCancel.addEventListener('click', (event) => {
  document.querySelector("[data-search-overlay]").style.display = "none";
})

// show more
// Select the HTML element with the data attribute "data-list-items"
const bookList1 = document.querySelector('[data-list-items]');
// Append the "fragment" to the selected HTML element
bookList1.appendChild(fragment)
// Select the search button HTML element with the data attribute "data-header-search"
//// Add an event listener to the search button that will display the search overlay when clicked
const searchButton = document.querySelector("[data-header-search]");
searchButton.addEventListener('click', (event) => {
  document.querySelector("[data-search-overlay]").style.display = "block";
})
// Update the text of the "Show More" button to display how many more items will be displayed
const showMoreButton = document.querySelector('[data-list-button]')
const numItemsToShow = Math.min(books.length - endIndex,)
const showMoreButtonText = `Show More <span style="opacity: 0.5">(${numItemsToShow})</span>`
showMoreButton.innerHTML = showMoreButtonText;
showMoreButton.addEventListener('click', () => {
  const fragment = document.createDocumentFragment()
  startIndex += 36;
  endIndex += 36;
  const startIndex1 = startIndex
  const endIndex1 = endIndex
  const extracted = books.slice(startIndex1, endIndex1)

  for (const { author, image, title, id, description, published } of extracted) {
    const preview = document.createElement('dl')
    preview.className = 'preview'
    preview.dataset.id = id
    preview.dataset.title = title
    preview.dataset.image = image
    preview.dataset.subtitle = `${authors[author]} (${(new Date(published)).getFullYear()})`
    preview.dataset.description = description

// template literal for show more
    // preview.dataset.genre = genres
    preview.innerHTML = /*html*/`
        <div>
        <image class='preview__image' src="${image}" alt="book pic"}/>
        </div>
        <div class='preview__info'>
```

—Javascript

Explanation of the whole code

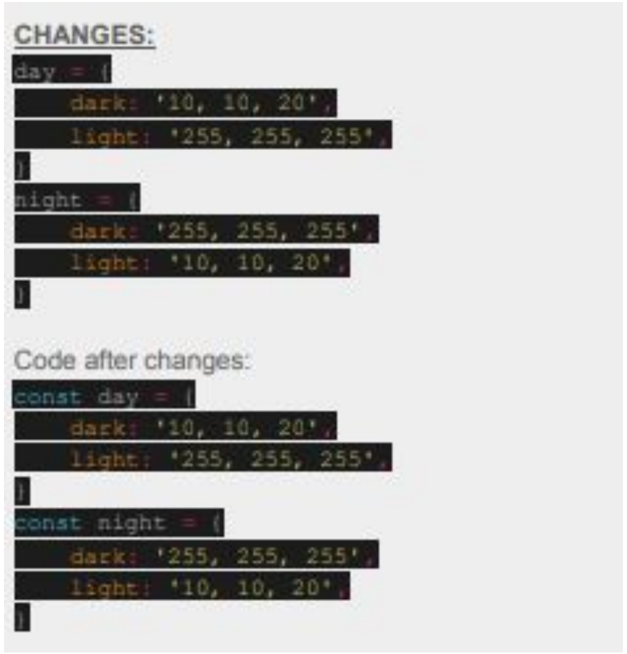
1. Import data from a separate file using ES6 modules
2. Define some variables, including two theme objects (day and night) and a matches variable set to the books array
3. Add an event listener to a save button that allows users to change themes
4. Create a new document fragment
5. Extract a slice of the books array to display in a book list
6. For each book in the extracted array, create a preview element with various data attributes based on the book's properties and add it to the document fragment
7. Add event listeners to the settings button and cancel button that display and hide the settings overlay when clicked
8. Add an event listener to each book preview element that displays more details about the book when clicked
9. Define a function that displays more details about a book when the preview is clicked
10. Check if the target of the event has a data-id attribute and display the overlay if it does. Otherwise, hide the overlay.

Changes

Changes

Theme ‘day’ and ‘night’ variables

- Two objects, day and night, are defined in the code.
- Each object has two properties, 'dark' and 'light', that determine the color values for a light and a dark theme.
- The variables 'day' and 'night' are declared with a constant 'const' declaration for the code to function.
- The RGB color code for the "dark" color is represented by the values '10, 10, 20', while '255, 255, 255' represent the RGB color code for the "light" color.



CHANGES:

```
day = {
    dark: '10, 10, 20',
    light: '255, 255, 255',
}
night = {
    dark: '255, 255, 255',
    light: '10, 10, 20',
}
```

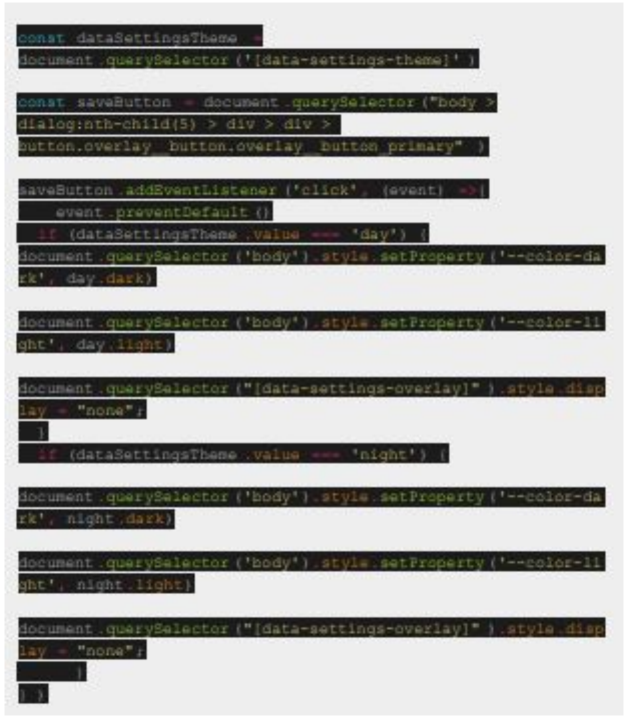
Code after changes:

```
const day = {
    dark: '10, 10, 20',
    light: '255, 255, 255',
}
const night = {
    dark: '255, 255, 255',
    light: '10, 10, 20',
}
```

Changes

DISPLAYS THEME ‘DAY’ AND ‘NIGHT’

- This code belongs to a web application called Book Connect that allows the user to customize the interface's color scheme.
- The first line of code selects the input element that the user interacts with to select the desired theme, and assigns it to the "dataSettingsTheme" constant variable.
- The second line selects the button element that saves the selected theme, which is the first child of the fifth dialog element in the HTML document's body.
- When the user clicks the "saveButton" element, an event listener function is called. It checks the value of the "dataSettingsTheme" input element to set the color scheme variables for the corresponding "day" or "night" theme, and hides the settings overlay accordingly.

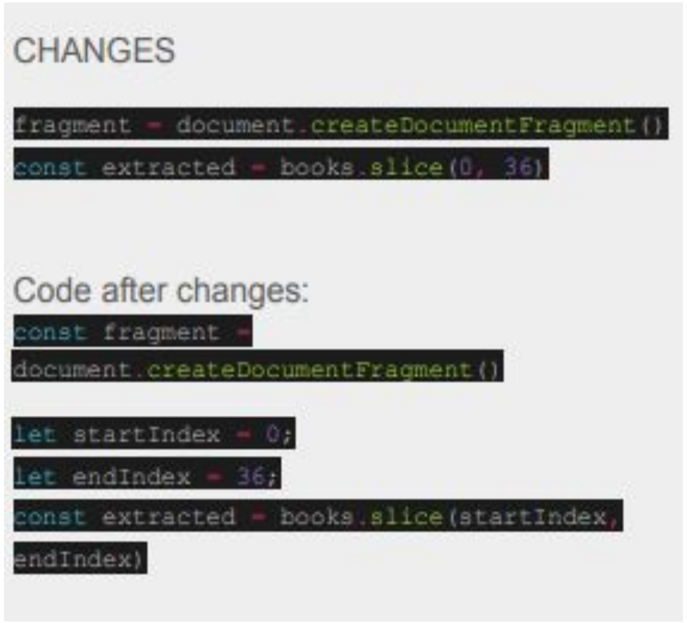


```
const dataSettingsTheme =
document .querySelector ('[data-settings-theme]' )

const saveButton = document .querySelector ("body >
dialog:nth-child(5) > div > div >
button.overlay__button.overlay__button_primary" )

saveButton .addEventListener ('click', (event) =>{
    event .preventDefault ()
  if (dataSettingsTheme .value === 'day') {
document .querySelector ('body') .style .setProperty ('--color-da
rk', day .dark)

document .querySelector ('body') .style .setProperty ('--color-li
ght', day .light)

document .querySelector ("[data-settings-overlay]" ) .style .disp
lay = "none";
  }
  if (dataSettingsTheme .value === 'night') {

document .querySelector ('body') .style .setProperty ('--color-da
rk', night .dark)

document .querySelector ('body') .style .setProperty ('--color-li
ght', night .light)

document .querySelector ("[data-settings-overlay]" ) .style .disp
lay = "none";
      }
} )
```

Changes

Creating a DocumentFragment

- To ensure proper initialization of the fragment variable, a 'const' declaration was added.
- A range of books to be displayed is specified using the variables startIndex and endIndex, and a DocumentFragment object called fragment is created.
- The slice() method is used to extract the specified range of books from the books array and store them in a new array called extracted.


CHANGES

```
fragment = document.createDocumentFragment()
const extracted = books.slice(0, 36)
```

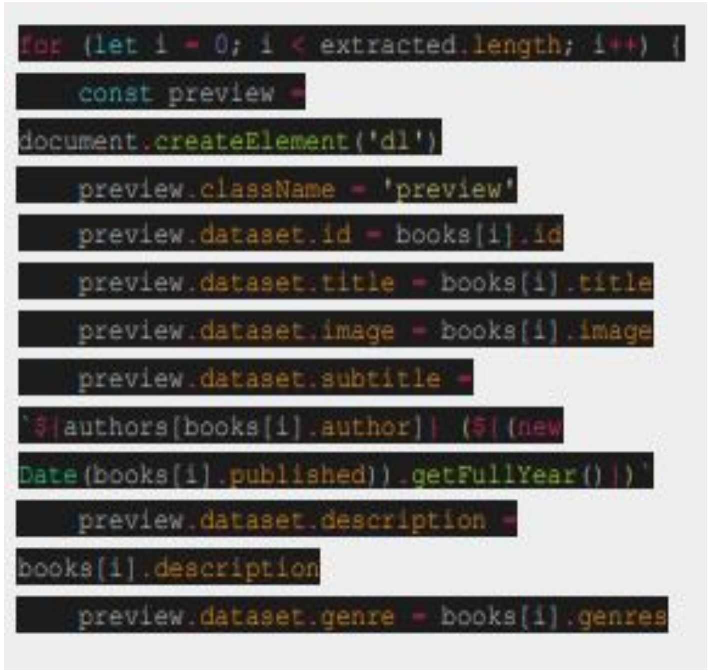
Code after changes:

```
const fragment =
document.createDocumentFragment()

let startIndex = 0;
let endIndex = 36;
const extracted = books.slice(startIndex,
endIndex)
```

Changes

After creating a DocumentFragment

- The for loop creates a preview element for each book item in the extracted array.
- The metadata for each book is stored as custom data attributes using the dataset property.
- The preview element has a class of "preview" and five data attributes: id, title, image, subtitle, and description.
- The subtitle data attribute links the author name and publication year of the book item.
- The genre data attribute is added to the preview element, but it is not used in this code snippet.

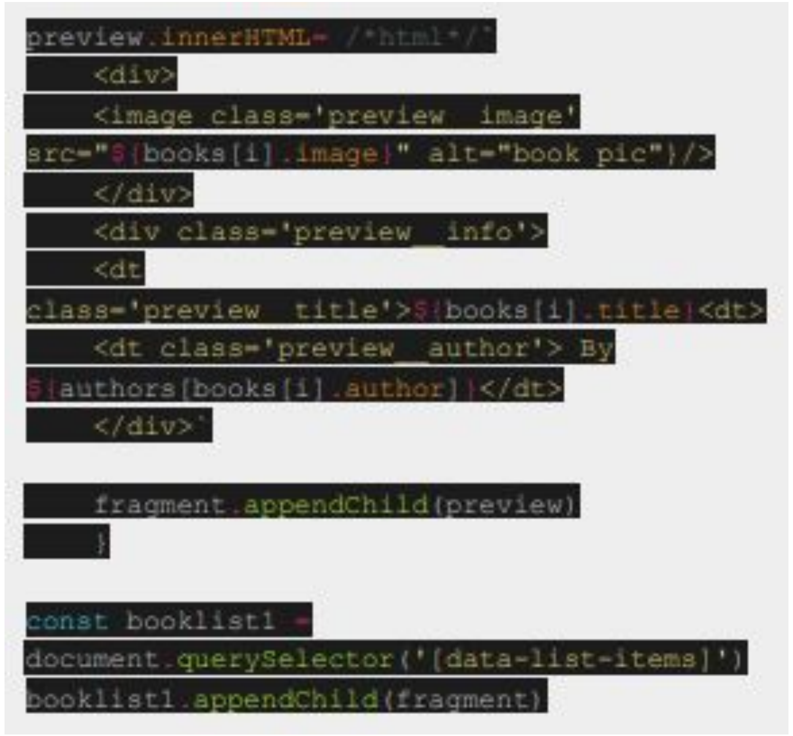


```
for (let i = 0; i < extracted.length; i++) {
    const preview =
document.createElement('dl')
    preview.className = 'preview'
    preview.dataset.id = books[i].id
    preview.dataset.title = books[i].title
    preview.dataset.image = books[i].image
    preview.dataset.subtitle =
`${authors[books[i].author]} (${(new
Date(books[i].published)).getFullYear()})`
    preview.dataset.description =
books[i].description
    preview.dataset.genre = books[i].genres
```

Changes

- Template literals are used to generate HTML markup for each book, which is then assigned to the preview element's innerHTML property. The preview element is then added to the fragment object.

- The fragment object is appended to an element with a data-list-items attribute, effectively rendering the books on the page.

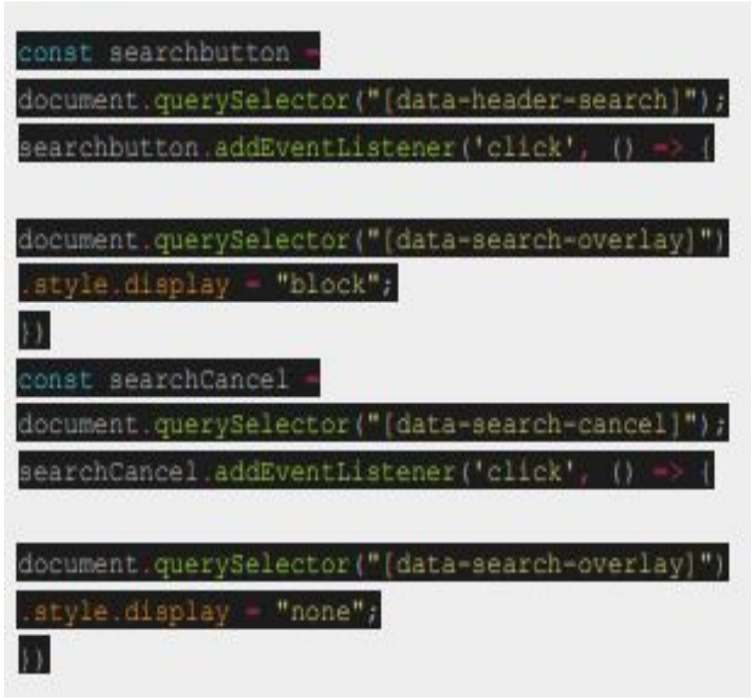


```
preview.innerHTML= /*html*/`
      <div>
      <image class='preview__image'
src="${books[i].image}" alt="book pic"}/>
      </div>
      <div class='preview__info'>
      <dt
class='preview__title'>${books[i].title}<dt>
      <dt class='preview__author'> By
${authors[books[i].author]}</dt>
      </div>`

      fragment.appendChild(preview)
      }

const booklist1 =
document.querySelector('[data-list-items]')
booklist1.appendChild(fragment)
```

Changes

SEARCH BUTTON

- The first event listener is triggered by a click on an HTML element with a data-header-search attribute, assigned to the searchbutton constant via querySelector(). When clicked, the function inside addEventListener() sets the display property of an HTML element with a data-search-overlay attribute to "block", making it visible.
- The second event listener is triggered by a click on an HTML element with a data-search-cancel attribute, assigned to the searchCancel constant via querySelector(). When clicked, the function inside addEventListener() sets the display property of an HTML element with a data-search-overlay attribute to "none", hiding it from view.



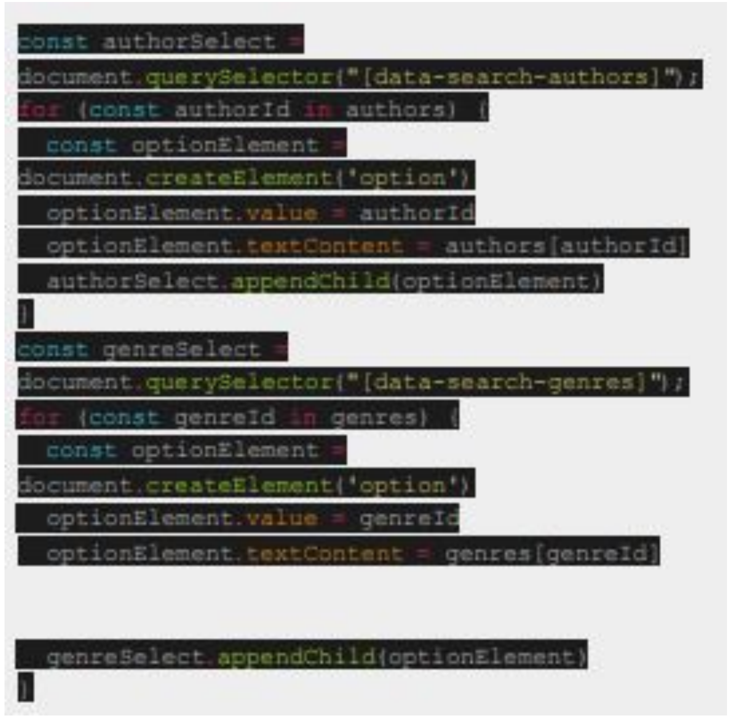
```
const searchbutton =
document.querySelector("[data-header-search]");
searchbutton.addEventListener('click', () => {

document.querySelector("[data-search-overlay]")
.style.display = "block";
})
const searchCancel =
document.querySelector("[data-search-cancel]");
searchCancel.addEventListener('click', () => {

document.querySelector("[data-search-overlay]")
.style.display = "none";
})
```

Changes

SEARCH OPTIONS FOR ‘ALL GENRES’ AND ‘ALL AUTHORS’

- This code uses the Fetch API to make an HTTP GET request to a JSON file, which contains an array of book objects. The book data is then parsed into a JavaScript array of objects, which is passed to a callback function for further processing. The JSON file is located at the specified URL.

- this code uses data from imported arrays to dynamically create two dropdown menus for authors and genres, by creating a new <option> element for each item in the arrays, setting its value and text content, and appending it to the corresponding dropdown menu using appendChild(). It then selects the relevant HTML elements using querySelector().



```
const authorSelect =
document.querySelector("[data-search-authors]");
for (const authorId in authors) {
  const optionElement =
document.createElement('option')
  optionElement.value = authorId
  optionElement.textContent = authors[authorId]
  authorSelect.appendChild(optionElement)
}
const genreSelect =
document.querySelector("[data-search-genres]");
for (const genreId in genres) {
  const optionElement =
document.createElement('option')
  optionElement.value = genreId
  optionElement.textContent = genres[genreId]


  genreSelect.appendChild(optionElement)
}
```

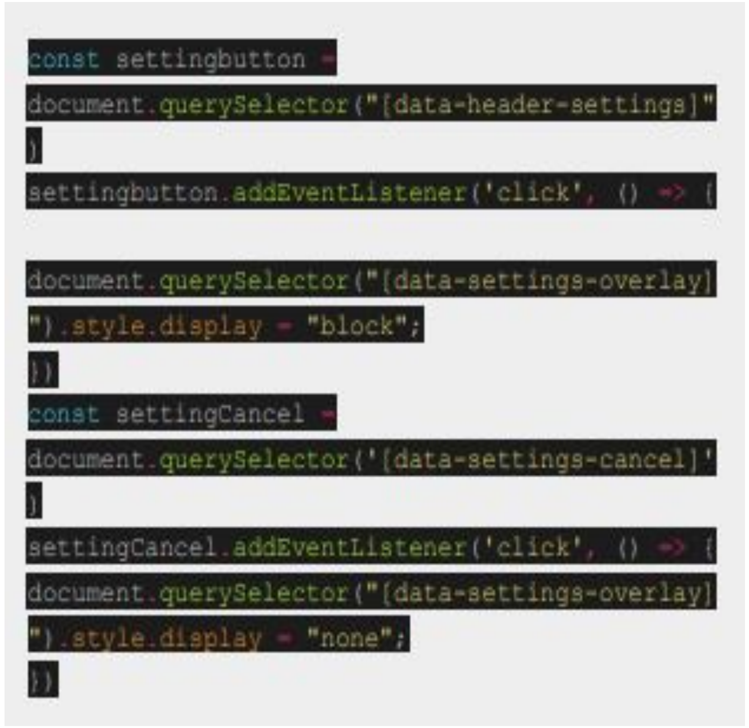
Changes

SETTINGS

- The code selects the DOM elements with certain data attributes and assigns them to variables using the querySelector() method.

- An event listener is attached to the settingbutton and settingCancel elements to listen for click events and execute arrow functions when they are clicked.

- When the settingbutton is clicked, the CSS display property of the element with the data-settings-overlay attribute is set to block, making it visible on the page. When the settingCancel is clicked, the arrow function is executed.

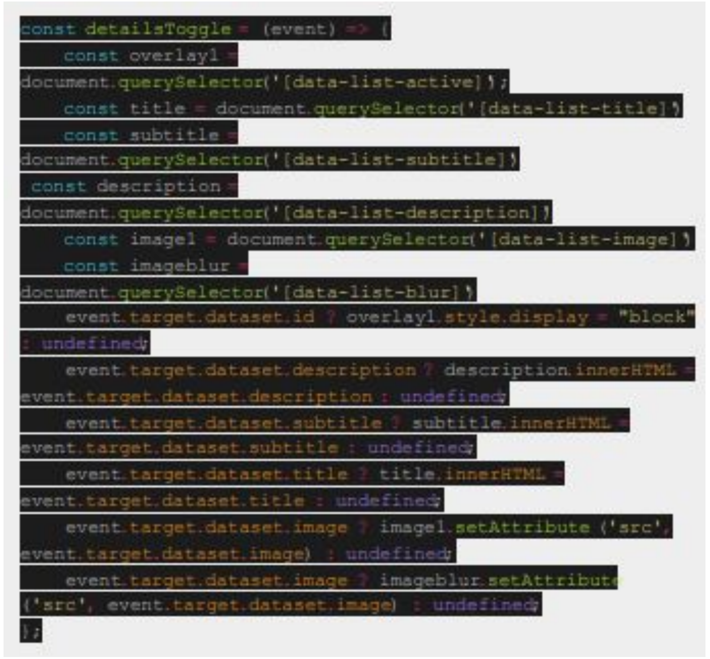


```
const settingbutton =
document.querySelector("[data-header-settings]"
)
settingbutton.addEventListener('click', () => {

document.querySelector("[data-settings-overlay]
").style.display = "block";
})
const settingCancel =
document.querySelector('[data-settings-cancel]'
)
settingCancel.addEventListener('click', () => {
document.querySelector("[data-settings-overlay]
").style.display = "none";
})
```

Changes

DISPLAYS BOOK DETAILS

1. The detailsToggle function toggles the visibility of a specific HTML element, which has the attribute data-list-active, based on the dataset attributes of the event target passed as an argument.
2. The function uses querySelector statements to select different HTML elements based on their attribute values, such as data-list-title, data-list-subtitle, data-list-description, data-list-image, and data-list-blur, which are used to display information about the selected item.
3. The function updates the content and visibility of certain HTML elements by setting their inner HTML or attribute values based on the dataset attributes of the event target, such as dataset.id, dataset.description, dataset.subtitle, dataset.title, and dataset.image.



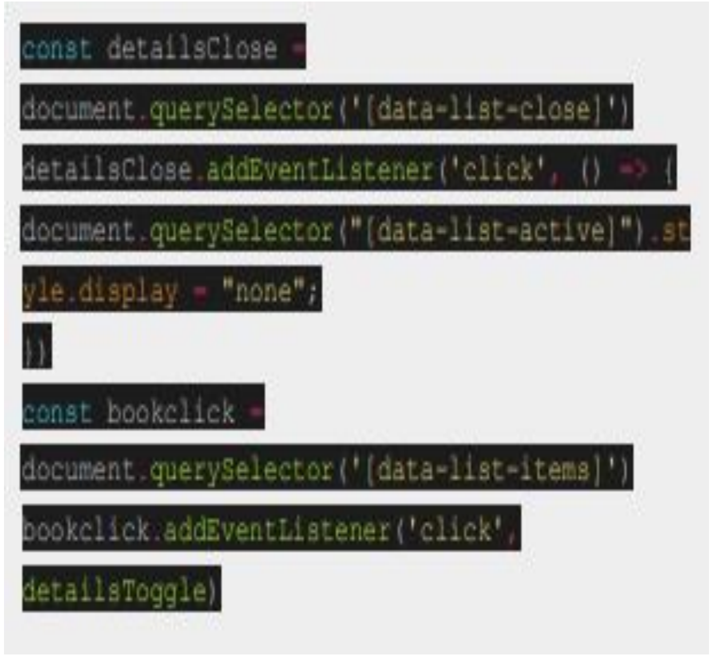
```
const detailsToggle = (event) => {
    const overlay1 =
document.querySelector('[data-list-active]');
    const title = document.querySelector('[data-list-title]')
    const subtitle =
document.querySelector('[data-list-subtitle]')
 const description =
document.querySelector('[data-list-description]')
    const image1 = document.querySelector('[data-list-image]')
    const imageblur =
document.querySelector('[data-list-blur]')
    event.target.dataset.id ? overlay1.style.display = "block"
: undefined;
    event.target.dataset.description ? description.innerHTML =
event.target.dataset.description : undefined;
    event.target.dataset.subtitle ? subtitle.innerHTML =
event.target.dataset.subtitle : undefined;
    event.target.dataset.title ? title.innerHTML =
event.target.dataset.title : undefined;
    event.target.dataset.image ? image1.setAttribute ('src',
event.target.dataset.image) : undefined;
    event.target.dataset.image ? imageblur.setAttribute
('src', event.target.dataset.image) : undefined;
};
```

Changes

1. This code sets up two event listeners for a list of books on a web page. One listener hides the book details overlay when the "detailsClose" button is clicked. The other listener calls the "detailsToggle" function when a book in the list is clicked.

2. The "detailsToggle" function retrieves elements such as the book title, subtitle, description, and images based on their data attributes. It then updates these elements based on the clicked book's data attributes and displays the book details overlay by setting the "display" style property of the "data-list-active" element to "block".

3.



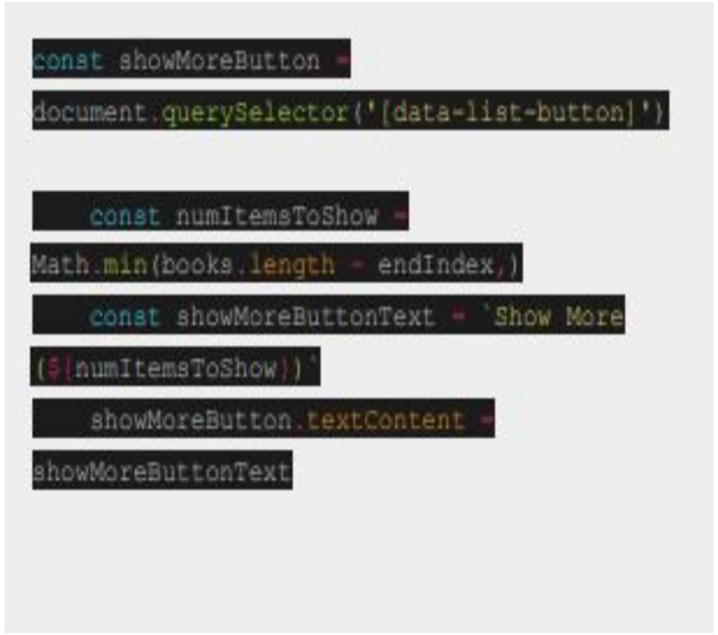
```
const detailsClose =
document.querySelector('[data-list-close]')
detailsClose.addEventListener('click', () => {
document.querySelector("[data-list-active]").st
yle.display = "none";
})
const bookclick =
document.querySelector('[data-list-items]')
bookclick.addEventListener('click',
detailsToggle)
```

Changes

SHOW MORE BUTTON

- The code is related to a "Show More" button that appears at the end of a list of items.

- The code selects the "Show More" button from the HTML page, calculates the number of items that will be shown when the button is clicked, creates a string that contains the text that will be displayed on the "Show More" button, and sets the text content of the button to the created string.



```
const showMoreButton =
document.querySelector('[data-list-button]')

    const numItemsToShow =
Math.min(books.length - endIndex,)
    const showMoreButtonText = `Show More
(${numItemsToShow})`
    showMoreButton.textContent =
showMoreButtonText
```

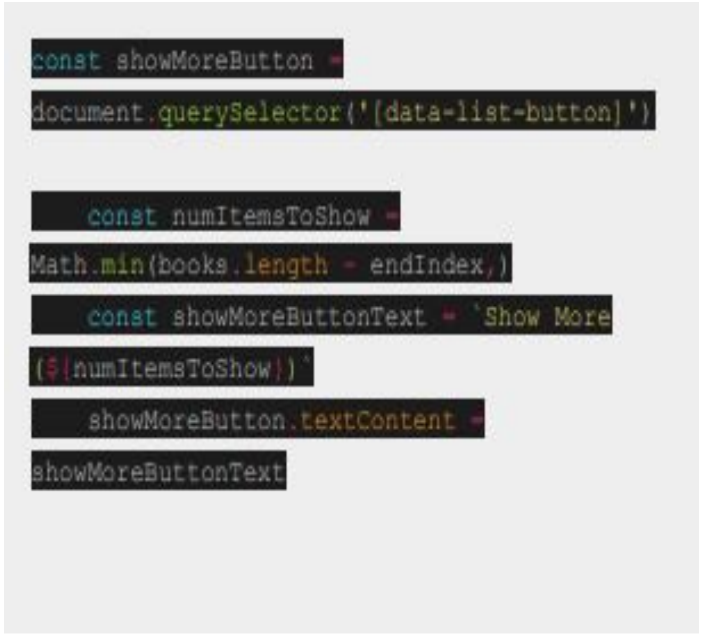
Changes

- An event listener is defined for the "showMoreButton" element that triggers when the user clicks on it.

- When the event listener is triggered, a new document fragment is created, and the "startIndex" and "endIndex" variables are increased by 36.

- The code then extracts a new set of book objects from the "books" array using the updated "startIndex" and "endIndex" values.

- For each extracted book object, a new "dl" element with the class name "preview" is created, and various dataset attributes are set to store information about the book, including its ID, title, image URL, author name, publication year, and description.



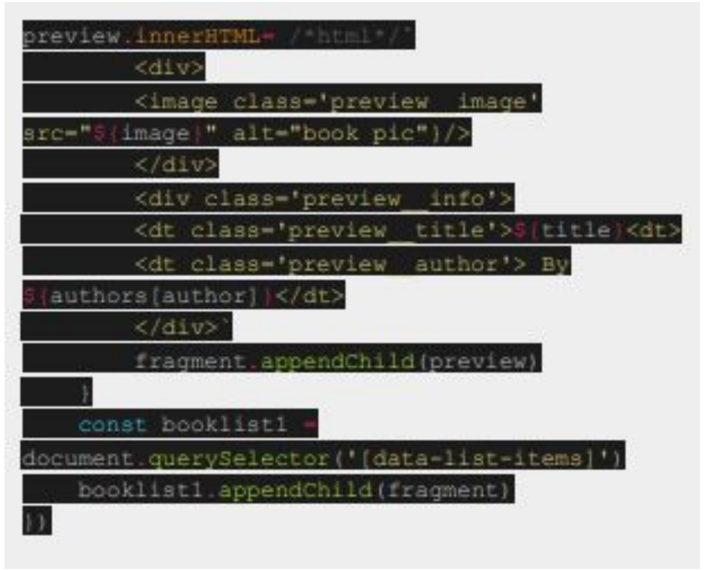
```
const showMoreButton =
document.querySelector('[data-list-button]')

    const numItemsToShow =
Math.min(books.length - endIndex,)
    const showMoreButtonText = `Show More
(${numItemsToShow})`
    showMoreButton.textContent =
showMoreButtonText
```

Changes

Conclusion

1. This code sets up an event listener for the showMoreButton element.

2. When the showMoreButton is clicked, the code creates a new document fragment, increases the startIndex and endIndex variables, extracts a new slice of book items, creates new preview elements for each book item, and appends them to the document fragment. Finally, the document fragment is appended to the booklist1 element to display the new book items on the page.



```
preview.innerHTML= /*html*/`
        <div>
        <image class='preview__image'
src="${image}" alt="book pic"}/>
        </div>
        <div class='preview__info'>
        <dt class='preview__title'>${title}<dt>
        <dt class='preview__author'> By
${authors[author]}</dt>
        </div>`
        fragment.appendChild(preview)
    }
    const booklist1 =
document.querySelector('[data-list-items]')
    booklist1.appendChild(fragment)
})
```