1.Which were the three best abstractions, and why?
Separation of Concerns: The code separates different functionalities into separate functions. For example, renderBooks() is responsible for rendering the books on the page, applyTheme() handles the theme changes, and handleSearchFormSubmit() handles the search form submission. This separation allows for better organization, modularity, and code reuse.

Modularization: The code modularizes related tasks into functions. For example, the creation of a book element is encapsulated in the createBookElement() function. This abstraction promotes code readability, maintainability, and reusability.

Encapsulation of State: The code encapsulates the state variables page and matches within the module. This encapsulation restricts direct access to the state variables from outside the module and provides controlled access through functions. It helps maintain data integrity and prevents unintended modifications to the state.

2.Which were the three worst abstractions, and why?
Lack of Dependency Injection: The code directly references the books, authors, and genres arrays from the imported module within multiple functions. This creates a tight coupling between the functions and the specific data source, limiting flexibility and testability. It would be better to pass the required data as parameters to the functions or use dependency injection to decouple the functions from specific data sources.

Mixing UI Manipulation and Data Manipulation: Some functions in the code, such as handleSearchFormSubmit() and handleListButtonClick(), perform both UI manipulation (e.g., updating the DOM) and data manipulation (e.g., filtering books). This violates the Single Responsibility Principle (SRP) by combining unrelated concerns. It would be better to separate UI manipulation and data manipulation into distinct functions, following the SRP.

Implicit Dependencies: Some functions in the code rely on global variables and DOM traversal to access elements and perform actions. For example, renderBooks() accesses the DOM directly using document.querySelector(). This creates implicit dependencies on the DOM structure and makes the functions less modular and harder to test. Instead, the functions should explicitly declare their dependencies or receive necessary elements as function arguments.

3.How can the three worst abstractions be improved via SOLID principles?
Dependency Injection: Refactor the functions to accept the required data as parameters instead of relying on global variables or imported modules directly. This promotes loose coupling and facilitates easier testing and reuse.

Single Responsibility Principle (SRP): Split the functions with mixed concerns into separate functions, each responsible for a single task. For example, handleSearchFormSubmit() can be split into separate functions for handling form submission and filtering books. This improves code organization, readability, and maintainability.

Interface Segregation Principle (ISP): Identify functions that depend on a large interface (e.g., DOM traversal and manipulation) and break them down into smaller, more focused functions. This prevents functions from depending on unnecessary or unrelated methods and promotes better separation of concerns.