DWA_07.4 Knowledge Check_DWA7

---

1. Which were the three best abstractions, and why?

- **Modules -** The import and export modules help in the sharing of code or even parts of the code between two or more files eliminating the need to rewrite code we want to access, by using these modules we can reduce code making it easier to read and maintain.

- **Function -** createSearchOption function is one of the best abstractions because of its modularity as it focuses on a single task: looping through objects then creating options from the objects' property values for a select element and also for its reusability by just calling it as many times as we want.

- **Inline if-statement -** is one of the best abstractions because it makes working with code that is driven by conditions a whole lot easier, for example, a single variable can be assigned a changeable value if the conditions have been met, Instead of nesting the whole code that needs that variable in its if statement.

---

2. Which were the three worst abstractions, and why?

- **Inconsistent Naming Conventions -** Poorly chosen or inconsistent naming conventions can make code harder to understand and maintain. Abstractions that fail to provide clear and intuitive names for variables, functions, or classes can hinder readability and introduce confusion.

- **Const Objects and Arrays -** arrays and objects can be assigned to constant variables but their properties are mutable and when we reuse the variable it would have the updated properties because we are not reassigning the const variable but changing the object or array itself, even this is not the worst it could be problematic in large projects.

- **Direct DOM Manipulation -** The code directly manipulates the DOM by creating elements and appending them to specific containers. This tightly couples the

JavaScript code with the HTML structure, making it harder to modify or refactor the HTML layout without impacting the JavaScript logic.

---

3. How can The three worst abstractions be improved via SOLID principles?

- **Single Responsibility Principle (SRP)**

  Inconsistent Naming Conventions - you can ensure that each class or module has a clear and focused purpose, which can lead to better naming conventions. Names should accurately reflect the specific responsibility or functionality of the entity.

- **Open-Closed Principle (OCP)**

  Const Objects and Arrays - Apply the open-closed principle by designing your code in a way that allows for the extension of array operations without modifying existing code.

- **Interface Segregation Principle (ISP)**

  Direct DOM Manipulation - Define clear and focused interfaces for specific DOM interactions or components. This ensures that clients only depend on the methods they require, reducing unnecessary coupling and promoting modularity.
