DWA_07.4 Knowledge Check_DWA7

---

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

Encapsulation is the process of bundling data and methods that operate on that data within a single unit, such as an object.

Naming conventions are a set of rules or guidelines used to name variables, functions, and other elements in a program. They help to make the code more readable and maintainable by providing a consistent way to identify the purpose and meaning of different elements.

Functional abstraction is the process of defining functions to perform specific tasks, hiding the implementation details and exposing only the necessary information to the user. These functions hide the details of creating and appending elements to the DOM, and expose only the necessary information to the outside world.

---

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

Over-abstraction: This occurs when code is abstracted to the point where it becomes difficult to understand and maintain. It can result in code that is overly complex and difficult to work with.
Leaky abstractions: This refers to abstractions that do not completely hide the underlying implementation details. As a result, users of the abstraction may need to have knowledge of the underlying implementation in order to use it effectively.
The wrong abstraction: This occurs when an abstraction is created that does not accurately represent the underlying concept or object. It can result in code that is difficult to understand and maintain, and may require significant refactoring to fix.

---

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

Single Responsibility Principle: This principle states that a class should have one and only one reason to change, meaning that a class should have only one job. By adhering to this principle, over-abstraction can be avoided as each class will have a specific and well-defined responsibility.
Open-Closed Principle: This principle states that software entities (classes, modules, functions, etc.) should be open for extension but closed for modification1. By adhering to this principle, leaky abstractions can be avoided as the underlying implementation details will be hidden and the abstraction will be properly encapsulated.
Liskov Substitution Principle: This principle states that objects of a superclass should be able to be replaced with objects of a subclass without altering the correctness of the program1. By adhering to this principle, the wrong abstraction can be avoided as the abstraction will accurately represent the underlying concept or object.
