DWA_07.4 Knowledge Check_DWA7

---

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

Comments: provide an overview of what a particular piece of code does, its inputs and outputs, and any specific behavior or requirements, so it makes it for someone using your code to see what the code does.

Objects: Objects serve as a powerful abstraction in JavaScript, enabling you to protect related data and functionality into a single entity.

Functions: allow you to define a block of code that can be reused multiple times throughout your program.

---

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

Inconsistent naming conventions: poorly chosen naming conventions can hinder code comprehension and lead to confusion.

Code Reusability and Modularity(Global variables): By tightly coupling code to global variables, it becomes harder to extract and reuse specific functionalities or components in different contexts.

Readability and maintainability in Callbacks:Callback-based code can be more difficult to read and maintain, especially as the number of callbacks increases.

---

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

Single Responsibility Principle (SRP): According to SRP, a class or module should have a single responsibility. To address the issue of global variables, you can create modules or classes that encapsulate related data and functionality.

Dependency Inversion Principle (DIP): DIP promotes loose coupling and flexibility. Instead of tightly coupling your code to specific callback implementations, consider abstracting the callback functionality into separate modules or classes.

By using dependency injection or creating abstractions for callbacks, you decouple your code from specific implementations. This reduces dependencies and enhances readability, as the code becomes more focused on its main logic rather than specific callback handling.

Open-Closed Principle (OCP): Follow the principle of OCP by using descriptive and intuitive names that are easy to understand and reason about. Avoid cryptic or abbreviated names that may confuse other developers or make it harder to comprehend the code's behavior.
