# DWA\_07.4 Knowledge Check\_DWA7

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

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

Functions: encapsulate code into reusable blocks. They promote the DRY (Don't Repeat Yourself) principle, making code modular, maintainable, and testable. Functions can take parameters and return values, allowing for flexible and dynamic code execution.

Objects and Classes: support encapsulation and inheritance, core principles of object-oriented programming. They allow you to model real-world entities and their behaviors, leading to organized and manageable code structures. Classes provide a blueprint for creating objects, enabling code reuse and extensibility.

Modules: help in structuring code by splitting it into separate files and logical units. They enable code reuse, scope isolation, and dependency management. By using import and export statements, developers can control the visibility and access of various parts of the code, leading to cleaner and more maintainable projects.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

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

Global Variables: Lead to namespace pollution, debugging difficulties, and maintenance issues.

Callbacks for Asynchronous Code: This can result in "callback hell," making code difficult to read, manage, and debug.

'with' Statement: Causes ambiguity, performance issues, and is incompatible with strict mode.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

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

Global Variables: Lead to namespace pollution, debugging difficulties, and maintenance issues.

* Ensure that each module or component has only one reason to change. Instead of relying on global variables, encapsulate related data and behavior within classes or modules.

Callbacks for Asynchronous Code: This can result in "callback hell," making code difficult to read, manage, and debug.

* The code should be open for extension but closed for modification. Using promises and async/await makes it easier to extend and maintain asynchronous code without modifying existing code.

'with' Statement: Causes ambiguity, performance issues, and is incompatible with strict mode.

* Each object or function should have a single responsibility. Avoiding the 'with' statement and explicitly referencing object properties ensures clarity and single responsibility

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_