DWA_07.4 Knowledge Check_DWA7

---

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

Functions mainly because I am familiar with them .Functions are easy to write and also easy to read and trace them.

Classes are similar to functions. After doing some research on classes and found out that they work like functions but can be used to handle bigger projects. I don't really understand how to write, maybe even read classes but I do believe that they are worth learning and I will do that.

Modules could be my favorite of all out of them, it is easier to search for what you are looking for without going through a long script.

---

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

Generators personally I find them very useful at times but looking at a scenario where I have to work with someone and they have a user for loop it would be very difficult to debug if we had a bug.

Libraries and framework they are very useful, its downfall is that they can be difficult to modify

---

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

Generators
Single Responsibility Principle (SRP) Ensure that the generator function has a clear responsibility. Avoid mixing unrelated functionality within the generator function.

Open-Closed Principle (OCP) Design the generator function in a way that allows for extension without modification.

Liskov Substitution Principle (LSP), Ensure that the generator function behaves consistently and predictably. Avoid introducing unexpected side effects or behavior changes that might affect how it is used or debugged.

Interface Segregation Principle (ISP) Keep the generator function focused on a specific purpose. Provide a clean interface that clearly does its purpose.

Dependency Inversion Principle (DIP) Apply dependency injection principles to the generator function, allowing the flexibility to inject dependencies and facilitate easier testing and debugging

Libraries and Frameworks

Single Responsibility Principle (SRP) Design libraries and frameworks with clear responsibilities.

Open-Closed Principle (OCP) Provide extension points and well-defined APIs that allow modification.

Liskov Substitution Principle (LSP), Ensure that libraries and frameworks adhere to established conventions and interfaces, making it easier to swap out components without affecting the overall functionality.

Interface Segregation Principle (ISP) 2Provide smaller, focused interfaces that can be used independently, allowing developers to choose only the functionalities they need.

Dependency Inversion Principle (DIP) Design libraries and frameworks to depend on abstractions rather than concrete implementations. Allow users to easily replace or modify underlying dependencies to suit their specific requirements.