QL-5) – Untestable Code: Key Thoughts and Snippets

Mocking vs Spying

- **Mocks** create scaffolding by simulating methods or entire objects from scratch. You define the behaviour explicitly - what to return, how to respond - without using real implementations.
- **Spies** wrap around existing objects, allowing you to monitor or override specific behaviours while preserving the original implementation for the rest.

Verification

Verification checks whether certain methods were called, how many times, and with what arguments. It focuses on interactions, not state. In Python, both mocks and spies (via unittest.mock.Mock or MagicMock) can be used to verify behaviour using assertions like assert_called_once_with().

Return Values with Mocks

Python's Mock and MagicMock objects can return fixed values or use a lambda/function to compute return values dynamically using the side_effect or return_value parameters.

Mocking Classes vs Interfaces

While Python doesn't have formal interfaces like Java, any class (including concrete ones) can be mocked. This is often referred to as "spying" when you mock parts of the behaviour while allowing the rest to behave normally using wraps.

Advanced Mocking in Python

While Python's standard library doesn't have a When...Do equivalent like NSubstitute, similar behaviour can be achieved with side_effect or by combining patch() with custom functions or lambdas to simulate dynamic responses or side effects.