olsen software ltd

+44 7989 401397
info@olsensoft.com

SOLID Python Development (3 days)

Course overview

Python is one of the most popular programming languages on the planet at the moment. At the heart of its appeal is its simple syntax and rich set of libraries.

It's quite easy to write Python code, but it's much harder to write *high-quality* Python code. How should you structure your code? How can you design for change, extensibility, consistency, and pluggability? How can you manage dependencies in a large code-base, so that your code is modular, stable, and manageable? How can you verify behaviour and type-safety?

This course shows how you can apply the "SOLID" principles to achieve all these goals.

What you'll learn

- Achieving type safety in Python via Python typing
- Using object orientation effectively
- Understanding and applying the Single Responsibility Principle (SRP)
- Understanding and applying the Open/Closed Principle (OCP)
- Understanding and applying the Liskov Substitution Principle (LSP)
- Understanding and applying the Interface Segregation Principle (ISP)
- Understanding and applying the Dependency Inversion Principle (DIP)
- Test automation
- Using Pydantic to implement model validation

Prerequisites

- Comfortable with Python programming

Course details

- Python Typing: The need for Python typing; Defining and verifying Python types; What types are available; Typing techniques
- Object-Oriented Programming: Essential Concepts; Defining and Using a Class; Class-Wide Members
- The Single Responsibility Principle (SRP): Understanding the SRP; Functional example of SRP; OO example of SRP
- The Open/Closed Principle (OCP): Understanding the OCP; Achieving the OCP via inheritance; Defining abstract classes/methods; Multiple inheritance; Understanding method resolution order
- The Liskov Substitution Principle (LSP): Understanding the LSP; Using the LSP; Polymorphism

- The Interface Segregation Principle (ISP): Setting the scene; Interfaces in Python; Interfaces and the ISP
- The Dependency Inversion Principle (DIP): Setting the scene; Dependency injection; Managing dependencies with the DIP
- Test Automation: Getting started with testing; Using PyHamcrest matchers; Mocking; Additional testing techniques
- Using Pydantic: Setting the scene; Specifying simple validation rules; Specifying complex validation rules