Cloud Contraptions
TRAINING & CONSULTING

To discuss this course and customizations:
Call: 434-509-5680 or Email: sales@cloudcontraptions.com

Design Patterns and SOLID with Python

Class Duration

21 hours of live training delivered over 3 days.

Student Prerequisites

- Professional experience writing Python
- Comfort with classes, functions, and modules in Python
- Familiarity with type hints is helpful but not required
- No prior design patterns knowledge required

Target Audience

This course is designed for working Python developers who want to make better design decisions. It suits engineers maintaining growing codebases, developers moving from scripting into application and library design, and team leads who review code — including code produced by AI assistants — and need a shared vocabulary for evaluating structure. Participants should write Python professionally; no prior patterns background is assumed.

Description

Design patterns were catalogued for a 1990s object-oriented world, and applying them verbatim in Python produces code that fights the language. This course teaches the SOLID principles and the Gang of Four catalog the way they should be taught in modern Python: selectively, honestly, and with the language's own features doing as much work as possible. Participants learn the roughly fifteen patterns that still earn their keep in contemporary code — split across creational, structural, and behavioral families — and, just as importantly, which classic patterns dissolve into a first-class function, a decorator, a context manager, a module, or a dataclass.

The course is built around refactoring rather than recitation. Each pattern is motivated by a real code smell, introduced as the refactoring that resolves it, and weighed against simpler alternatives so participants learn when a pattern is overengineering. Pythonic idioms get equal billing with the classics: Strategy via plain functions, Singleton via modules and the Borg idiom, Factory via classmethods, interfaces via `typing.Protocol` and duck typing, all expressed with modern type hints in the Python 3.13/3.14 era. The

To discuss this course and customizations:
Call: 434-509-5680 or Email: sales@cloudcontraptions.com

final day addresses design in the age of AI-generated code: recognizing pattern use (and misuse) in agent output, reviewing generated designs against SOLID, and steering coding assistants toward structures your team can maintain.

Learning Outcomes

- Apply each SOLID principle to real Python code and recognize when a principle is being over-applied
- Identify the design problem each major GoF pattern solves and select among creational, structural, and behavioral options
- Replace classic class-based patterns with Pythonic equivalents: functions for Strategy and Command, modules for Singleton, classmethods for Factory
- Define interfaces with `typing.Protocol` and abstract base classes, and choose between nominal and structural approaches
- Use decorators, context managers, and dataclasses as first-class design tools rather than syntax tricks
- Favor composition over inheritance and refactor inheritance-heavy hierarchies toward delegation
- Apply dependency injection in Python without a framework, and evaluate when a DI library adds value
- Refactor toward patterns incrementally from concrete code smells, keeping tests green throughout
- Recognize anti-patterns and overengineering, and argue for the simplest design that works
- Express pattern implementations with accurate modern type hints, including generics and `Protocol` types
- Review AI-generated Python for structural quality, naming the patterns it uses and the SOLID violations it hides
- Write prompts and review checklists that steer coding agents toward maintainable designs

Training Materials

Comprehensive courseware is distributed online at the start of class. All students receive a downloadable MP4 recording of the training.

Software Requirements

- Python 3.13 or later

To discuss this course and customizations:
Call: 434-509-5680 or Email: sales@cloudcontraptions.com

- Text editor or IDE (VS Code or PyCharm recommended)
- Git for version control
- Access to an AI coding assistant for the code review exercises (any major tool)

Training Topics

Design Principles Foundations

- What design patterns are and what they are not
- Coupling, cohesion, and the cost of change
- Reading the GoF catalog critically in 2026
- Why Python changes the catalog
- The vocabulary value of patterns on a team

SOLID Principles, Applied Honestly

- Single Responsibility: reasons to change, not line counts
- Open/Closed via polymorphism, functions, and plugins
- Liskov Substitution and Python's duck typing
- Interface Segregation with small protocols
- Dependency Inversion without ceremony
- When SOLID is over-applied: speculative abstraction

Interfaces and Typing in Modern Python

- `typing.Protocol` for structural interfaces
- Abstract base classes vs. protocols
- Duck typing and when to formalize it
- Generics and type variables in pattern code
- Pattern idioms with Python 3.13/3.14 type hints

Creational Patterns

- Factory Method and Abstract Factory: when they pay off
- Factory via classmethods and plain functions
- Builder vs. dataclasses and keyword arguments
- Singleton: modules, the Borg idiom, and why you rarely need either
- Object pools and caching with `functools`

Structural Patterns

- Adapter for integrating third-party code
- Facade for taming complex subsystems

To discuss this course and customizations:
Call: 434-509-5680 or Email: sales@cloudcontraptions.com

- Decorator pattern vs. Python decorators
- Proxy, lazy loading, and `__getattr__`
- Composite for tree structures
- Flyweight, interning, and `__slots__`

Behavioral Patterns

- Strategy via first-class functions and protocols
- Command, undo stacks, and closures
- Observer, callbacks, and event systems
- Template Method vs. composition and hooks
- State machines: enums, classes, and libraries
- Iterator and generator functions
- Context managers as a resource pattern

Composition and Dependency Injection

- Composition over inheritance in practice
- Refactoring deep hierarchies toward delegation
- Constructor injection with plain Python
- Wiring object graphs at the application edge
- DI containers in Python: costs and benefits
- Testability as a design feedback signal

Refactoring Toward Patterns

- Code smells that motivate each pattern family
- Incremental refactoring with tests as a safety net
- Case study: untangling a God class
- Case study: replacing conditional sprawl with Strategy
- Knowing when to stop: rule of three

Anti-Patterns and Overengineering

- Pattern abuse: Java idioms transplanted into Python
- Premature abstraction and speculative generality
- The simplest thing that could possibly work
- YAGNI, KISS, and deletion as a design tool

Patterns in the Age of AI-Generated Code

- Structural habits of AI coding assistants
- Recognizing pattern use and misuse in agent output

To discuss this course and customizations:
Call: 434-509-5680 or Email: sales@cloudcontraptions.com

- Reviewing generated code against SOLID
- Steering agents with design-aware prompts and conventions
- Keeping architectural ownership when agents write the code