CHEAT SHEET · ALIJABBARY.COM

Python OOP & Design Patterns

Modern object-oriented Python for the 2026 stack — dataclasses, Protocols, and patterns that earn their keep.

01 · FOUNDATIONS

Dataclasses replace 30 lines of boilerplate

Stop writing __init__ / __repr__ / __eq__ by hand. Modern Python 3.10+ uses @dataclass with slots=True for memory-efficient, immutable-friendly records.

- @dataclass(slots=True, frozen=True) is the default for value objects
- Use kw_only=True when a class has 4+ fields — positional args become a footgun
- field(default_factory=list) for mutable defaults — never `= []` directly
- __post_init__ for validation that has to happen after the fields are set
- Switch to Pydantic only when data crosses a trust boundary (HTTP / files / users)

02 · TYPE HINTS

PEP 604 union syntax + Protocols

Python 3.10 brought the pipe-union syntax. Python 3.12 brought generic class syntax. Modern code is shorter and clearer than the typing-module-heavy code of 2020.

str | None	Instead of Optional[str]
list[Foo]	Instead of List[Foo]
dict[str, int]	Instead of Dict[str, int]
Iterable[T]	Import from collections.abc, not typing
TypedDict	Use for dict-shaped data with known keys
Protocol	Structural typing — duck typing with static checking

03 · COMPOSITION

Prefer Protocols to abstract base classes

ABCs are still useful for shared implementation. But for 'anything that quacks like a duck', Protocols are the modern tool — no inheritance required.

- class Repository(Protocol): def get(self, id: str) → Foo: …
- Any class with a matching get() method satisfies the Protocol — no `extends` needed
- @runtime_checkable when you actually need isinstance() to work at runtime

- ABCs (abc.ABC + @abstractmethod) still win for shared default implementations
- Composition > inheritance: hold a `_logger: Logger` field, don't extend `LoggerMixin`

04 · STRATEGY & FACTORY

Two patterns that actually compound

Most design patterns are 1990s Java in disguise. These two have aged well in dynamic, function-first Python.

- Strategy: pass behaviour as a callable. `discount: Callable[[Order], Decimal]` beats a DiscountStrategy class hierarchy
- Factory: a module-level `def make_client(env: str) → Client` is usually all you need — no AbstractFactory-Factory
- Singleton in Python = module-level instance. There is no other right answer
- Observer = a `list[Callable[[Event], None]]` and a for-loop. That's it
- When tempted to reach for a Visitor, reach for a match statement instead

05 · DEPENDENCY INJECTION

Constructor injection, no framework

Inject collaborators as constructor parameters with Protocol-typed contracts. Tests pass fakes; production passes real implementations. No DI container required.

- class OrderService: def __init__(self, repo: Repository, mailer: Mailer): …
- Top-level `composition root` (often main.py) wires the real implementations
- Tests instantiate the service with fakes — no monkey-patching, no global state
- If you find yourself wanting a DI container, the answer is usually `more functions, fewer classes`
- Reserve full DI frameworks (dependency-injector, kink) for codebases past 50k LOC

06 · ANTI-PATTERNS

What modern Python OOP avoids

- Inheritance chains 3+ deep — flatten with composition or Protocols
- Getters/setters by reflex — Python attributes are public; use @property only when you actually compute or validate
- Mixins as a way to share state — they're a way to share behaviour, not data
- `*args, **kwargs` everywhere — you lose type safety and IDE autocomplete
- Inheriting from dict / list — wrap them in a dataclass field instead

Beyond the basics

COMMON PITFALLS

- ! Mutable default arguments — `def foo(items=[]):` is the classic Python footgun; use `items: list | None = None` and resolve in the body
- ! Forgetting `frozen=True` on value objects — accidental mutation in one corner of the codebase breaks invariants elsewhere
- ! Reaching for inheritance to share code — composition + Protocols are almost always the better answer
- ! Treating Protocols as a substitute for runtime validation — they're STATIC contracts; data from the wire still needs Pydantic / zod-style validation
- ! Over-applying patterns — most Python code is happier as small functions than as a class hierarchy

FURTHER READING

- → PEP 695 — Type Parameter Syntax (Python 3.12) (https://peps.python.org/pep-0695/)
- → PEP 544 — Protocols (structural subtyping) (https://peps.python.org/pep-0544/)
- → Python typing module reference (https://docs.python.org/3/library/typing.html)
- → Hynek Schlawack — Subclassing in Python redux (https://hynek.me/articles/python-subclassing-redux/)