LAB 1 — Discover Duck Typing

Objective

Understand that **shared behavior**, not shared type, enables polymorphism.

Instructions

1. Write a function that:
    - accepts a single argument
    - calls **one method** on that argument
2. Call the function with:
    - a built-in type
    - a custom class you create yourself
3. The function must work **without modification** for both.

Constraints

- Do NOT use isinstance
- Do NOT check types explicitly

LAB 2 — Break Duck Typing on Purpose

Objective

Understand the **risk** of duck typing.

Instructions

1. Take your function from Lab 1.
2. Pass an object that does **not** implement the expected behavior.
3. Observe the error.
4. Explain:
    - when the error occurs

LAB 3 — EAFP vs LBYL

Objective

Practice **EAFP-style error handling**.

Instructions

1. Write a function that:
    - performs an operation on two inputs
    - may fail depending on the inputs

2. Implement it using:
    - try
    - except
3. Test the function with:
    - valid inputs
    - invalid inputs
    - edge cases

Constraints

- Do NOT use isinstance
- Do NOT pre-check types

LAB 4 — Build Polymorphism with Inheritance

Objective

Understand **method overriding** and runtime dispatch.

Instructions

1. Design a base class representing a **general concept**.
2. The base class must define a method but **not implement it**.
3. Create at least **three subclasses** that:
    - inherit from the base class
    - override the method with different behavior
4. Write a function that:
    - accepts the base class
    - calls the method
    - does NOT use conditionals
5. Call the function with each subclass.

Constraints

- Do NOT use if, elif, or isinstance

LAB 5 — Feel the Pain of isinstance

Objective

Recognize why type-check chains do not scale.

Instructions

1. Write a function that:
    - accepts an object
    - behaves differently depending on the object's type
    - uses isinstance
2. Add a new class that should be supported.
3. Identify **all places** that must be modified.
4. Redesign the solution using polymorphism so:
    - new classes require no changes to existing functions

LAB 6 — Abstract Base Class Enforcement

Objective

Understand **why ABCs exist**.

Instructions

1. Design an abstract base class that:
    - represents a **role or capability**
    - defines at least one abstract method
2. Attempt to:
    - create a subclass that does **not** implement the method
    - instantiate it
3. Observe and explain the error.
4. Create:
    - one valid subclass
    - a second valid subclass with different behavior
5. Write a function that:
    - accepts the abstract base class
    - calls the abstract method

LAB 7 — Duck Typing vs ABCs (Comparison Lab)

Objective

Compare **flexibility vs safety**.

Instructions

1. Solve the same problem twice:
    - once using duck typing
    - once using an abstract base class
2. Intentionally violate the expected interface in both versions.
3. Compare:
    - error timing
    - error messages
    - developer experience