Chapter 4
Control Flow

Mọi thứ trong Python là object/class

Classes chính là *blueprints* cho các **instance objects**.

Class cũng là object ở cấp độ cao hơn — Python dùng **metaclass** (`type` theo mặc định) để quản lý cách class được tạo và hoạt động.

Bốn nền tảng cốt lõi của OOP

Encapsulation (đóng gói)

Bundles dữ liệu (attributes) và hành vi (methods) vào trong class.

Python không ép truy cập private như Java; dùng convention `_protected` (prefix `_`) hoặc `__private` để ngăn truy cập ngoài lớp và tránh conflict thông qua name mangling - thuộc dạng **khai báo ý định**, không cấm hoàn toàn.

Abstraction (trừu tượng hóa)

Chỉ phơi giao diện cần thiết, ẩn đi chi tiết complex bên trong class.

Dùng `abc.ABC` và `@abstractmethod` để định nghĩa abstract class bắt buộc subclass triển khai (geekster.in).

Inheritance (kế thừa)

Cho phép subclass kế thừa method và attribute từ superclass, có thể override hoặc thêm mới.

Hỗ trợ Single, Multiple, Multilevel, Hierarchical Inheritance trong Python.

Polymorphism (đa hình)

Một method có thể hành xử khác nhau tùy loại object (method overriding).

Python dùng **duck typing**, bất cứ object có method phù hợp đều sử dụng được, không cần cùng class hierarchy.

Cách Python implement runtime

Python sử dụng **dynamic dispatch** gọi method tương ứng với loại instance tại runtime — ví dụ `pet.speak()` gọi vào `Dog.speak()` hay `Cat.speak()` tùy object cụ thể.

`super()` được dùng trong subclass để gọi hàm của superclass, thường là `__init__` để kế thừa state ban đầu.

Cơ sở ngữ nghĩa & custom behavior

Bạn có thể override các **special methods** như `__init__`, `__str__`, `__eq__`, `__lt__`, để điều chỉnh hành vi object mặc định.

Descriptors, `@property`, `@classmethod`, `@staticmethod`... là công cụ tinh chỉnh access control và interface.

Ví dụ tổng hợp — lớp `Shape`

```python
from abc import ABC, abstractmethod

class Shape(ABC):  # abstract
    @abstractmethod
    def area(self): ...
    @abstractmethod
    def perimeter(self): ...

class Rectangle(Shape):
    def __init__(self, width, height):
        self._w = width      # encapsulation: protected
        self._h = height

    def area(self): return self._w * self._h
    def perimeter(self): return 2 * (self._w + self._h)
```

Ví dụ tổng hợp — lớp `Circle` kế thừa từ lớp `Shape`

```
class Circle(Shape):
    def __init__(self, r):
        self.__r = r          # private via name-mangling

    def area(self): return 3.14 * self.__r ** 2
    def perimeter(self): return 2 * 3.14 * self.__r

def total_area(shapes):
    return sum(s.area() for s in shapes)  # duck typing + polymorphism
```

Hàm `total_area` làm việc với bất cứ object có method `area()` — không cần biết class là gì.

So sánh nhanh

Khái niệm	Bản chất lý thuyết	Triển khai trong Python
Encapsulation	Bundle data + behavior	Private/protected via naming convention
Abstraction	Hide implementation, expose interface	`ABC`, `@abstractmethod`, property
Inheritance	Reuse, extend superclass behavior	`class Sub(A):`, `super()`
Polymorphism	Same call, nhiều hành vi	Duck typing + overriding dynamic dispatch
Metaclass	Điều khiển cách class được tạo lập	Customize class behavior thông qua `type`

Kết luận

OOP trong Python là tập hợp các **object**, nơi **class** đóng vai trò cấu trúc thiết kế.

Python hỗ trợ **encapsulation**, **abstraction**, **inheritance**, và **polymorphism** qua cú pháp đơn giản, linh hoạt.

Tại runtime, Python phân phối method dựa trên loại thực tế của object — sử dụng dynamic dispatch và duck typing.

Bạn có thể customize behavior thông qua special methods, descriptors, properties, metaclasses...

cheers

cảm ơn

thank you!

muchas gracias

dziękuję

danke