Tóm tắt Phương pháp Lập trình

Vũ Thái Sơn

Tháng 6, 2025

Hành trình "lột xác" tư duy lập trình Python: Từ code rối rắm đến chuẩn mực và chuyên nghiệp

1 Mục đích bài viết

Bạn từng cảm thấy bối rối khi nhìn vào những dòng code cũ của mình, hoặc tự hỏi vì sao code của đồng nghiệp lại "sạch sẽ" và dễ đọc đến thế? Nếu bạn đang trên hành trình học Python — đặc biệt là với mục tiêu làm Data Science, AI — thì việc nắm vững các phương pháp coding chuẩn mực sẽ là "vũ khí bí mật" giúp bạn phát triển nhanh hơn, làm việc nhóm hiệu quả hơn và tiết kiệm vô số thời gian bảo trì về sau.

Bài blog này tổng hợp những kiến thức trọng tâm từ buổi học về Coding Methodology, giúp bạn hiểu và áp dụng các nguyên tắc code sạch, Pythonic, cũng như những nền tảng quan trọng như interface và abstraction — đặc biệt phù hợp cho người mới bắt đầu hoặc chưa có nền tảng lập trình vững chắc.

Các tiêu chí của bài viết:

- Tính đúng đắn kiến thức chuyên môn
- Cấu trúc trình bày rõ ràng
- Lý thuyết, công thức, mã nguồn minh họa
- Hình minh họa, ví dụ thực tế
- Kiến thức mở rộng

2 Clean Code & PEP-8 — Viết code sạch và chuẩn mực

Clean Code là mã nguồn rõ ràng, dễ đọc, dễ bảo trì, có khả năng mở rộng và dễ kiểm thử. Python có chuẩn **PEP-8** — bộ quy tắc vàng về định dạng code: từ cách đặt tên biến, hàm (`snake_case`), class (`PascalCase`), hằng số (`UPPER_CASE`), cho đến căn lề, khoảng trắng, cách tổ chức import và cấu trúc hàm/class.

```
# Non-standard
def tinhTong(a, b):
   return a + b
```

```python
# PEP-8 compliant
def tinh_tong(a: int, b: int) -> int:
    """Calculate the sum of two numbers."""
    return a + b
```

Listing 1: Ví dụ đặt tên và định dạng theo PEP-8

Ngoài ra, hãy tài liệu hóa code bằng docstring, sử dụng type annotation, và tích hợp các công cụ kiểm tra tự động như Flake8, Black, Pylint, Mypy để nâng cao chất lượng dự án.

3 Viết code Pythonic — Tận dụng tối đa "chất" Python

Pythonic nghĩa là viết code ngắn gọn, rõ ràng, tận dụng đặc trưng của Python như list/dict/set comprehensions, slicing, context manager (`with`), properties, decorators, và các quy tắc truthiness.

```python
# List comprehension
squares = [n**2 for n in range(1, 6)] # Output: [1, 4, 9, 16, 25]

# Context manager
with open("file.txt") as f:
    content = f.read()

# Comparison with None
if x is None:
    print("x is None")
```

Listing 2: Ví dụ code Pythonic

Các kỹ thuật này giúp code đẹp hơn, tăng hiệu suất và giảm lỗi khó chịu về sau.

4 Nguyên lý vàng để viết code tốt: DRY, KISS, YAGNI...

- **DRY** (Don't Repeat Yourself): Tránh lặp lại code, hãy tách thành hàm hoặc module riêng.
- **KISS** (Keep It Simple, Stupid): Luôn ưu tiên giải pháp đơn giản, dễ hiểu.
- **YAGNI** (You Aren't Gonna Need It): Đừng thêm tính năng khi chưa thực sự cần.
- **Defensive Programming**: Luôn kiểm tra đầu vào, xử lý ngoại lệ rõ ràng, không "nuốt lỗi".
- **Separation of Concerns**: Phân chia rõ trách nhiệm giữa các module/hàm.
- **Logging**: Sử dụng logging thay vì print để kiểm soát và lưu trữ log hiệu quả.

```python
def divide(a, b):
    if b == 0:
        raise ValueError("Cannot divide by zero.")
    return a / b
```

Listing 3: Ví dụ kiểm tra đầu vào

5 SOLID & Design Patterns — Khi bạn muốn code "level up"

- **SOLID** gồm 5 nguyên tắc: Single Responsibility, Open/Closed, Liskov Substitution, Interface Segregation, Dependency Inversion.
- **Design Patterns**: Factory, Singleton, Adapter, Decorator, Command, Template Method...

Các nguyên tắc này giúp code dễ mở rộng, bảo trì và kiểm thử, đặc biệt khi làm việc nhóm hoặc xây dựng hệ thống lớn. Python hỗ trợ rất tốt nhờ duck typing, ABC module, và composition.

6 Kiến thức mở rộng: Interface & Abstraction — "Chìa khóa" cho code chuyên nghiệp

Nếu bạn là người mới học lập trình, chắc hẳn khi đọc đến các ví dụ về interface hay abstract trong tài liệu, bạn sẽ cảm thấy khá "lạc lõng" vì những khái niệm này thường chỉ được nhắc sơ qua. Tuy nhiên, đây lại là nền tảng cực kỳ quan trọng để xây dựng code có cấu trúc rõ ràng, dễ mở rộng và dễ bảo trì – đặc biệt khi làm việc với các dự án lớn hoặc phát triển theo hướng chuyên nghiệp.

Hãy cùng mình "giải mã" hai khái niệm này theo cách dễ hiểu nhất nhé!

6.1 Abstraction (Trừu tượng hóa): "Làm cà phê"

Hãy tưởng tượng bạn pha cà phê bằng máy tự động: bạn chỉ cần nhấn nút, không cần biết bên trong máy vận hành ra sao. Đó chính là **abstraction**: chỉ "phơi bày" những gì cần thiết, còn chi tiết bên trong thì ẩn đi.

```python
from abc import ABC, abstractmethod

class CoffeeMachine(ABC):
    @abstractmethod
    def make_coffee(self):
        pass

    def clean_machine(self):
        print("Cleaning the machine...") # Concrete method

class MyCoffeeMachine(CoffeeMachine):
    def make_coffee(self):
        print("Brewing coffee...")

machine = MyCoffeeMachine()
machine.make_coffee() # Brewing coffee...
machine.clean_machine() # Cleaning the machine...
```

Listing 4: Ví dụ về Abstraction

Giải thích:

- `make_coffee` là **abstract method** (bắt buộc class con phải định nghĩa).
- `clean_machine` là **concrete method** (class con có thể dùng lại hoặc override).

- Bạn **không thể** tạo trực tiếp đối tượng từ abstract class (`CoffeeMachine()` sẽ báo lỗi).

6.2 Interface: "Luật đội mũ bảo hiểm"

Giả sử mọi người tham gia giao thông đều phải đội mũ bảo hiểm. Đó là một "luật chung" — ai cũng phải tuân thủ, nhưng mỗi người có thể chọn loại mũ, màu sắc, kiểu dáng khác nhau. Interface trong lập trình cũng như vậy: định nghĩa *mọi đối tượng* (class) phải có những hành động nào, nhưng không quan tâm cách thực hiện chi tiết.

```python
from abc import ABC, abstractmethod

class HelmetRule(ABC):
    @abstractmethod
    def wear_helmet(self):
        pass

class MotorbikeDriver(HelmetRule):
    def wear_helmet(self):
        print("Wear type A helmet")

class Cyclist(HelmetRule):
    def wear_helmet(self):
        print("Wear type B helmet")
```

Listing 5: Ví dụ về Interface

Nếu thiếu phương thức `wear_helmet`, Python sẽ báo lỗi khi khởi tạo.

6.3 So sánh rõ ràng: Interface vs Abstract Class

Đặc điểm	Interface (Python)	Abstract Class (Python)
Chứa concrete methods	Không	Có thể có
Chứa abstract methods	Luôn luôn	Có thể có
Bắt buộc định nghĩa	Tất cả `@abstractmethod`	Tất cả `@abstractmethod`
Kế thừa concrete methods	Không có để kế thừa	Có thể dùng lại hoặc override

Lưu ý:

- Nếu class con **không định nghĩa** tất cả các hàm có `@abstractmethod` thì sẽ **không khởi tạo được đối tượng** — Python báo lỗi ngay.
- Abstract class có thể có cả phương thức đã định nghĩa sẵn (concrete methods), còn interface thì thường chỉ gồm các abstract method.

6.4 Ví dụ đời sống giúp dễ hình dung

- **Abstraction** giống như chiếc máy pha cà phê: bạn chỉ cần nhấn nút, mọi thứ phức tạp đã được "giấu" bên trong.
- **Interface** giống như luật đội mũ bảo hiểm: ai cũng phải làm, nhưng cách làm thì tùy mỗi người.

6.5 Tại sao người mới học cần hiểu rõ interface & abstraction?

- **Giúp code rõ ràng, dễ kiểm thử:** Khi bạn định nghĩa interface/abstract class, các thành viên khác trong nhóm chỉ cần nhìn vào đó là biết class nào cần làm gì.
- **Dễ mở rộng:** Khi cần thêm chức năng, chỉ cần tạo class mới kế thừa mà không phải sửa code cũ.
- **Tránh lỗi:** Python sẽ báo lỗi nếu bạn quên định nghĩa phương thức bắt buộc — giúp phát hiện bug sớm.

7 Tổng kết

Buổi học Coding Methodology đã giúp nắm được:

- Các nguyên tắc Clean Code, PEP-8, Pythonic để code rõ ràng, dễ đọc, dễ bảo trì.
- Áp dụng các nguyên lý DRY, KISS, YAGNI, Defensive Programming để code bền vững, dễ mở rộng.
- Hiểu và thực hành tổ chức code theo chuẩn, sử dụng công cụ kiểm tra tự động.
- Làm quen với SOLID, Design Patterns, và đặc biệt là phân biệt rõ interface — abstraction trong Python, nền tảng cho lập trình hướng đối tượng hiện đại.

Nếu bạn vẫn nghĩ code đẹp chỉ là chuyện hình thức, hãy thử áp dụng những nguyên tắc trên vào dự án thực tế — bạn sẽ thấy hiệu quả tăng lên rõ rệt, teamwork mượt mà hơn rất nhiều. Hãy bắt đầu từ những thay đổi nhỏ nhất — chỉnh lại tên biến, thêm docstring, thử list comprehension — và bạn sẽ bất ngờ về sự "lột xác" của chính mình!

Chúc bạn sớm trở thành coder chuẩn chỉnh trong mắt đồng nghiệp và chính bản thân mình!