Python OOP - Classes & Objects Basics

> **Chủ đề 1: Nền tảng của lập trình hướng đối tượng trong Python**

📋 Mục Lục

Thời lượng học: 6 giờ

1. Classes và Objects - Khái Niệm Cơ Bản

1.1. Đặt Vấn Đề

😨 Vấn đề với Procedural Programming

Giả sử bạn cần quản lý thông tin học sinh:

```
# ❌ Cách cũ - dùng biến riêng lẻ
student1_name = "Nguyễn Văn An"
student1_age = 20
student1_score = 85

student2_name = "Trần Thị Bình"
student2_age = 21
student2_score = 92

# Vấn đề:
# - Quá nhiều biến rời rạc
# - Khó quản lý khi có nhiều học sinh
# - Không có cách tổ chức rõ ràng
# - Khó mở rộng (thêm thuộc tính mới)
```

Hoặc dùng dictionary:

```
# ❌ Cách tốt hơn một chút - dùng dict
student1 = {
    "name": "Nguyễn Văn An",
    "age": 20,
    "score": 85
}

student2 = {
    "name": "Trần Thị Bình",
    "age": 21,
    "score": 92
}

# Vấn đề:
# - Không có validation (có thể gán sai kiểu dữ liệu)
# - Không có methods (hành vi)
# - Mỗi dict độc lập, không có "template" chung
```

💡 Giải pháp: Classes

Class là một **blueprint** (bản thiết kế) để tạo objects.

```
# ☑ Cách OOP - dùng Class
class Student:
    """Blueprint cho học sinh"""

    def __init__(self, name, age, score):
        self.name = name
        self.age = age
        self.score = score

    def display_info(self):
        print(f"{self.name}, {self.age} tuổi, điểm: {self.score}")

# Tạo objects từ class
student1 = Student("Nguyễn Văn An", 20, 85)
student2 = Student("Trần Thị Bình", 21, 92)

student1.display_info()  # Nguyễn Văn An, 20 tuổi, điểm: 85
student2.display_info()  # Trần Thị Bình, 21 tuổi, điểm: 92
```

Ưu điểm:

- ☑ Tổ chức rõ ràng
- ☑ Dễ mở rộng
- ☑ Code reuse
- ☑ Encapsulation (đóng gói data + behavior)

1.2. Class vs Object

Analogy (So sánh):

```
Class = Blueprint (bản thiết kế nhà)
Object = Actual House (ngôi nhà thực tế)

1 blueprint → có thể xây nhiều nhà
1 class → có thể tạo nhiều objects
```

Ví dụ:

```python
# Class - Blueprint
class Car:
    def __init__(self, brand, model):
        self.brand = brand
        self.model = model

# Objects - Actual instances
car1 = Car("Toyota", "Camry")
car2 = Car("Honda", "Civic")
car3 = Car("Toyota", "Corolla")

print(car1.brand)  # Toyota
print(car2.brand)  # Honda
print(car3.brand)  # Toyota
```

Thuật ngữ:

- **Class** = blueprint, template
- **Object** = instance (thực thể)
- **Instantiation** = quá trình tạo object từ class

1.3. Tạo Class Đầu Tiên

Syntax Cơ Bản

```python
class ClassName:
    """Docstring - mô tả class"""

    # Class body
    pass
```

Quy tắc đặt tên:

- ☑ **PascalCase** (viết hoa chữ cái đầu mỗi từ)

- ☑ Descriptive (mô tả rõ ràng)
- ❌ Không dùng snake_case, camelCase

```
# ☑ GOOD
class Student:
    pass

class BankAccount:
    pass

class ShoppingCart:
    pass

# ❌ BAD
class student:  # Không viết hoa
    pass

class bank_account:  # snake_case
    pass

class shoppingCart:  # camelCase
    pass
```

Empty Class

```
class EmptyClass:
    """Một class trống - chưa có gì"""
    pass

# Tạo object
obj = EmptyClass()
print(obj)  # <__main__.EmptyClass object at 0x...>
print(type(obj))  # <class '__main__.EmptyClass'>
```

---

2. Constructor - `__init__` Method

2.1. Tại Sao Cần Constructor?

```
# ❌ Không có constructor
class Student:
    pass

# Tạo object và gán attributes thủ công
student = Student()
student.name = "An"
student.age = 20
```

```
student.score = 85

# Vấn đề:
# - Phải gán từng attribute riêng
# - Dễ quên gán một số attributes
# - Không có validation
```

```
# ☑ Có constructor
class Student:
    def __init__(self, name, age, score):
        """Constructor - khởi tạo object"""
        self.name = name
        self.age = age
        self.score = score

# Tạo object - gọn gàng, đầy đủ ngay!
student = Student("An", 20, 85)
```

2.2. `__init__` Method Chi Tiết

Syntax:

```
class ClassName:
    def __init__(self, param1, param2, ...):
        self.attribute1 = param1
        self.attribute2 = param2
        # Initialize object
```

Đặc điểm:

- `__init__` là **special method** (magic method)
- Tự động được gọi khi tạo object
- `self` là tham số đầu tiên (luôn luôn!)
- Có thể có nhiều parameters khác

Ví dụ đầy đủ:

```
class Rectangle:
    def __init__(self, width, height):
        """
        Initialize a rectangle

        Args:
            width: Chiều rộng (int hoặc float)
            height: Chiều cao (int hoặc float)
        """
```

```
        print("Đang tạo hình chữ nhật...")

        self.width = width
        self.height = height

        print(f"Đã tạo hình {width}x{height}")

# Khi tạo object
rect = Rectangle(10, 5)

# Output:
# Đang tạo hình chữ nhật...
# Đã tạo hình 10x5
```

2.3. Default Parameters

```
class Student:
    def __init__(self, name, age=18, score=0):
        """
        Constructor với default values

        Args:
            name: Tên (bắt buộc)
            age: Tuổi (mặc định 18)
            score: Điểm (mặc định 0)
        """
        self.name = name
        self.age = age
        self.score = score

# Các cách tạo object
s1 = Student("An")                    # age=18, score=0
s2 = Student("Bình", 20)              # score=0
s3 = Student("Chi", 19, 85)           # full params
s4 = Student("Dũng", score=90)        # keyword argument

print(f"{s1.name}: {s1.age} tuổi, {s1.score} điểm")
# An: 18 tuổi, 0 điểm
```

2.4. Validation trong Constructor

```
class BankAccount:
    def __init__(self, account_number, balance):
        """Tạo tài khoản ngân hàng với validation"""

        # Validate account number
        if not account_number:
```

```
            raise ValueError("Account number không được rỗng!")

        # Validate balance
        if balance < 0:
            raise ValueError("Balance không thể âm!")

        self.account_number = account_number
        self.balance = balance
        print(f"Tài khoản {account_number} được tạo với số dư: ${balance}")

# ☑ Valid
account1 = BankAccount("ACC001", 1000)

# ❌ Invalid - sẽ raise exception
try:
    account2 = BankAccount("", 500)
except ValueError as e:
    print(f"Lỗi: {e}")  # Lỗi: Account number không được rỗng!

try:
    account3 = BankAccount("ACC002", -100)
except ValueError as e:
    print(f"Lỗi: {e}")  # Lỗi: Balance không thể âm!
```

---

3. The `self` Parameter

3.1. `self` là Gì?

`self` đại diện cho **instance hiện tại** của class.

```
class Dog:
    def __init__(self, name):
        self.name = name  # self = instance đang được tạo
        print(f"self là: {self}")
        print(f"self.name là: {self.name}")

dog1 = Dog("Buddy")
# self là: <__main__.Dog object at 0x...>
# self.name là: Buddy

dog2 = Dog("Max")
# self là: <__main__.Dog object at 0x...> (địa chỉ khác!)
# self.name là: Max
```

Analogy:

```
self = "chính tôi", "bản thân"
```

```
Khi nói: "Tôi tên là An" (self.name = "An")
Khi nói: "Tôi 20 tuổi" (self.age = 20)

"self" luôn refer đến object đang được xử lý
```

3.2. Tại Sao Phải Có `self`?

```python
class Counter:
    def __init__(self):
        self.count = 0  # Attribute của instance

    def increment(self):
        self.count += 1  # Truy cập attribute qua self
        return self.count

# Tạo 2 counters độc lập
counter1 = Counter()
counter2 = Counter()

counter1.increment()  # self = counter1
counter1.increment()
print(counter1.count)  # 2

counter2.increment()  # self = counter2
print(counter2.count)  # 1

# Mỗi object có data riêng!
```

Không có `self` → không phân biệt được instances:

```python
# ❌ Nếu không có self (giả định)
class Counter:
    count = 0  # Chỉ có 1 biến chung!

    def increment():
        count += 1  # count của ai???

# Không thể có nhiều counters độc lập
```

3.3. `self` Convention

`self` là quy ước, không phải keyword:

```python
# ☑ Chuẩn - dùng "self"
class Dog:
```

```
    def __init__(self, name):
        self.name = name

# ☑ Hoạt động - nhưng KHÔNG NÊN
class Cat:
    def __init__(this, name):  # "this" thay vì "self"
        this.name = name

# ☑ Hoạt động - nhưng KHÔNG NÊN
class Bird:
    def __init__(myself, name):  # "myself"
        myself.name = name

# Dù hoạt động, LUÔN LUÔN dùng "self" theo convention!
```

Tại sao dùng `self`?

- ☑ Python convention (PEP 8)
- ☑ Mọi developer đều hiểu
- ☑ Code đồng nhất, dễ đọc
- ❌ Dùng tên khác → confusing!

---

4. Instance Variables vs Class Variables

4.1. Instance Variables

Định nghĩa: Variables thuộc về **từng instance riêng biệt**.

```
class Student:
    def __init__(self, name, score):
        self.name = name    # Instance variable
        self.score = score  # Instance variable

# Mỗi instance có data riêng
student1 = Student("An", 85)
student2 = Student("Bình", 92)

print(student1.name)   # An
print(student2.name)   # Bình

# Thay đổi instance1 không ảnh hưởng instance2
student1.score = 90
print(student1.score)  # 90
print(student2.score)  # 92 (không đổi!)
```

Đặc điểm:

- Defined trong `__init__` hoặc instance methods
- Unique cho mỗi object

- Access qua `self.variable_name`

4.2. Class Variables

Định nghĩa: Variables **shared by all instances** của class.

```
class Student:
    school_name = "THPT ABC"  # Class variable (shared)

    def __init__(self, name, score):
        self.name = name    # Instance variable (unique)
        self.score = score  # Instance variable (unique)

# Tất cả students đều cùng school
student1 = Student("An", 85)
student2 = Student("Bình", 92)

print(student1.school_name)  # THPT ABC
print(student2.school_name)  # THPT ABC

# Thay đổi class variable ảnh hưởng TẤT CẢ instances
Student.school_name = "THPT XYZ"

print(student1.school_name)  # THPT XYZ (đổi!)
print(student2.school_name)  # THPT XYZ (đổi!)
```

Đặc điểm:

- Defined ngay trong class body (ngoài methods)
- Shared by all instances
- Access qua `ClassName.variable` hoặc `self.variable`

4.3. So Sánh Instance vs Class Variables

```
class Dog:
    species = "Canis familiaris"  # Class variable - chung
    count = 0                     # Class variable - đếm số dogs

    def __init__(self, name, age):
        self.name = name  # Instance variable - riêng
        self.age = age    # Instance variable - riêng
        Dog.count += 1    # Tăng counter

dog1 = Dog("Buddy", 3)
dog2 = Dog("Max", 5)

print("=== Class Variables (Shared) ===")
print(f"Species: {Dog.species}")    # Canis familiaris
```

```
print(f"Total dogs: {Dog.count}")   # 2

print("\n=== Instance Variables (Unique) ===")
print(f"Dog 1: {dog1.name}, {dog1.age} years")  # Buddy, 3 years
print(f"Dog 2: {dog2.name}, {dog2.age} years")  # Max, 5 years
```

Bảng so sánh:

Aspect	Instance Variable	Class Variable
Định nghĩa	Trong `__init__` hoặc methods	Trong class body
Prefix	`self.`	`ClassName.`
Scope	Unique cho mỗi instance	Shared by all instances
Use case	Data riêng (name, age)	Data chung (species, constants)
Memory	Mỗi instance một copy	Chỉ 1 copy chung

4.4. Khi Nào Dùng Class Variables?

☑ Use Cases cho Class Variables:

1. Constants (Hằng số chung):

```
class Circle:
    PI = 3.14159  # Class variable - constant

    def __init__(self, radius):
        self.radius = radius

    def area(self):
        return Circle.PI * self.radius ** 2

circle = Circle(5)
print(circle.area())  # 78.53975
```

2. Counters (Đếm số instances):

```
class User:
    user_count = 0  # Class variable - counter

    def __init__(self, username):
        self.username = username
        User.user_count += 1

user1 = User("alice")
user2 = User("bob")
user3 = User("charlie")
```

```
print(f"Total users: {User.user_count}")  # 3
```

3. Default Values:

```
class BankAccount:
    interest_rate = 0.05  # Class variable - default 5%

    def __init__(self, balance):
        self.balance = balance

    def calculate_interest(self):
        return self.balance * BankAccount.interest_rate

account = BankAccount(1000)
print(account.calculate_interest())  # 50.0

# Change interest rate for ALL accounts
BankAccount.interest_rate = 0.07
print(account.calculate_interest())  # 70.0
```

4.5. ⚠ Pitfall: Shadowing Class Variables

```
class MyClass:
    shared = 10  # Class variable

obj1 = MyClass()
obj2 = MyClass()

print(obj1.shared)  # 10 (từ class)
print(obj2.shared)  # 10 (từ class)

# ⚠ Tạo instance variable cùng tên!
obj1.shared = 20  # Tạo obj1.shared (instance variable)

print(obj1.shared)       # 20 (instance variable)
print(obj2.shared)       # 10 (vẫn là class variable)
print(MyClass.shared)    # 10 (class variable không đổi)

# obj1 giờ có CẢ HAI:
# - obj1.shared (instance) = 20
# - MyClass.shared (class) = 10
```

Best Practice: Dùng `ClassName.variable` để modify class variables:

```python
class MyClass:
    shared = 10

obj1 = MyClass()
obj2 = MyClass()

# ☑ Modify class variable
MyClass.shared = 20

print(obj1.shared)  # 20 (cả 2 đều đổi)
print(obj2.shared)  # 20
```

---

5. Instance Methods

5.1. Instance Methods Basics

Định nghĩa: Methods hoạt động trên instance data.

```python
class Rectangle:
    def __init__(self, width, height):
        self.width = width
        self.height = height

    def area(self):  # Instance method
        """Tính diện tích"""
        return self.width * self.height

    def perimeter(self):  # Instance method
        """Tính chu vi"""
        return 2 * (self.width + self.height)

    def scale(self, factor):  # Instance method với parameter
        """Phóng to/thu nhỏ"""
        self.width *= factor
        self.height *= factor

# Sử dụng
rect = Rectangle(10, 5)

print(f"Area: {rect.area()}")          # 50
print(f"Perimeter: {rect.perimeter()}") # 30

rect.scale(2)  # Phóng to 2 lần
print(f"New area: {rect.area()}")       # 200
```

Đặc điểm:

- Luôn có `self` là parameter đầu tiên

- Access instance data qua `self`
- Call qua object: `object.method()`

5.2. Methods với Parameters

```
class BankAccount:
    def __init__(self, balance):
        self.balance = balance

    def deposit(self, amount):
        """Nạp tiền"""
        if amount > 0:
            self.balance += amount
            print(f"Nạp ${amount}. Số dư: ${self.balance}")
        else:
            print("Số tiền phải > 0!")

    def withdraw(self, amount):
        """Rút tiền"""
        if amount > self.balance:
            print("Không đủ tiền!")
        elif amount <= 0:
            print("Số tiền phải > 0!")
        else:
            self.balance -= amount
            print(f"Rút ${amount}. Số dư: ${self.balance}")

# Demo
account = BankAccount(1000)

account.deposit(500)    # Nạp $500. Số dư: $1500
account.withdraw(300)   # Rút $300. Số dư: $1200
account.withdraw(2000)  # Không đủ tiền!
```

5.3. Methods Calling Other Methods

```
class Student:
    def __init__(self, name, scores):
        self.name = name
        self.scores = scores  # List of scores

    def average(self):
        """Tính điểm trung bình"""
        return sum(self.scores) / len(self.scores)

    def grade(self):
        """Xếp loại dựa vào average"""
        avg = self.average()  # Gọi method khác!
```

```
        if avg >= 90:
            return "A"
        elif avg >= 80:
            return "B"
        elif avg >= 70:
            return "C"
        else:
            return "F"

    def display_report(self):
        """In báo cáo - gọi nhiều methods"""
        print(f"Student: {self.name}")
        print(f"Average: {self.average():.2f}")
        print(f"Grade: {self.grade()}")

# Demo
student = Student("An", [85, 90, 88, 92])
student.display_report()

# Output:
# Student: An
# Average: 88.75
# Grade: B
```

5.4. Modifying Instance State

```
class Counter:
    def __init__(self):
        self.count = 0

    def increment(self):
        """Tăng counter"""
        self.count += 1

    def decrement(self):
        """Giảm counter"""
        self.count -= 1

    def reset(self):
        """Reset về 0"""
        self.count = 0

    def get_value(self):
        """Lấy giá trị hiện tại"""
        return self.count

# Demo
counter = Counter()
```

```
counter.increment()
counter.increment()
counter.increment()
print(counter.get_value())  # 3

counter.decrement()
print(counter.get_value())  # 2

counter.reset()
print(counter.get_value())  # 0
```

---

6. Class Methods - @classmethod

6.1. Tại Sao Cần Class Methods?

Instance methods work with instance data.
Class methods work with **class data** hoặc create instances.

```
class Student:
    school_name = "THPT ABC"  # Class variable

    def __init__(self, name):
        self.name = name

    # Instance method - works with instance data
    def introduce(self):
        print(f"Tôi là {self.name}")

    # Class method - works with class data
    @classmethod
    def get_school_name(cls):
        return cls.school_name

    @classmethod
    def change_school(cls, new_school):
        cls.school_name = new_school

# Gọi class method KHÔNG CẦN object
print(Student.get_school_name())  # THPT ABC

Student.change_school("THPT XYZ")
print(Student.get_school_name())  # THPT XYZ
```

6.2. @classmethod Decorator

Syntax:

```python
class MyClass:
    class_variable = "value"

    @classmethod
    def class_method(cls, param):
        # cls = class itself
        # có thể access class variables
        print(cls.class_variable)
```

Đặc điểm:

- Dùng `@classmethod` decorator
- Parameter đầu tiên là `cls` (not `self`)
- `cls` = class itself
- Có thể gọi qua class hoặc instance

6.3. Factory Methods

Use case phổ biến: Alternative constructors

```python
class Date:
    def __init__(self, year, month, day):
        self.year = year
        self.month = month
        self.day = day

    @classmethod
    def from_string(cls, date_string):
        """Factory method - tạo Date từ string"""
        year, month, day = map(int, date_string.split('-'))
        return cls(year, month, day)  # cls() = Date()

    @classmethod
    def today(cls):
        """Factory method - tạo Date hôm nay"""
        import datetime
        today = datetime.date.today()
        return cls(today.year, today.month, today.day)

    def __str__(self):
        return f"{self.year}-{self.month:02d}-{self.day:02d}"

# Regular constructor
date1 = Date(2024, 1, 15)
print(date1)  # 2024-01-15

# Factory method từ string
date2 = Date.from_string("2024-02-20")
print(date2)  # 2024-02-20
```

```
# Factory method hôm nay
date3 = Date.today()
print(date3)  # 2024-01-23 (hôm nay)
```

6.4. Class Methods vs Instance Methods

```
class Employee:
    company_name = "Tech Corp"  # Class variable
    employee_count = 0

    def __init__(self, name, salary):
        self.name = name
        self.salary = salary
        Employee.employee_count += 1

    # Instance method
    def get_info(self):
        """Lấy thông tin employee cụ thể"""
        return f"{self.name} - ${self.salary}"

    # Class method
    @classmethod
    def get_company_name(cls):
        """Lấy tên công ty (class data)"""
        return cls.company_name

    @classmethod
    def get_employee_count(cls):
        """Đếm số nhân viên (class data)"""
        return cls.employee_count

    @classmethod
    def change_company_name(cls, new_name):
        """Đổi tên công ty (modify class data)"""
        cls.company_name = new_name

# Instance methods cần object
emp1 = Employee("Alice", 5000)
emp2 = Employee("Bob", 6000)

print(emp1.get_info())  # Alice - $5000

# Class methods không cần object
print(Employee.get_company_name())    # Tech Corp
print(Employee.get_employee_count())  # 2

Employee.change_company_name("New Tech")
print(Employee.get_company_name())    # New Tech
```

7. Static Methods - `@staticmethod`

7.1. Tại Sao Cần Static Methods?

Static methods không cần instance data hay class data - chỉ là utility functions.

```
class MathUtils:
    @staticmethod
    def add(a, b):
        """Cộng 2 số - không cần self hay cls"""
        return a + b

    @staticmethod
    def multiply(a, b):
        """Nhân 2 số"""
        return a * b

    @staticmethod
    def is_even(number):
        """Check số chẵn"""
        return number % 2 == 0

# Gọi static methods không cần object
print(MathUtils.add(5, 3))        # 8
print(MathUtils.multiply(4, 7))   # 28
print(MathUtils.is_even(10))      # True
```

Khi nào dùng:

- Utility functions liên quan logic đến class
- Không cần access instance hay class data
- Organize code tốt hơn (group related functions)

7.2. So Sánh 3 Loại Methods

```
class MyClass:
    class_variable = "Class level"

    def __init__(self, value):
        self.instance_variable = value

    # 1. Instance method - cần self
    def instance_method(self):
        print(f"Instance: {self.instance_variable}")
        print(f"Can access class: {self.class_variable}")

    # 2. Class method - cần cls
    @classmethod
```

```
    def class_method(cls):
        print(f"Class: {cls.class_variable}")
        # print(self.instance_variable)  # ❌ Không có self!

    # 3. Static method - không cần gì
    @staticmethod
    def static_method(x, y):
        print(f"Static: {x} + {y} = {x + y}")
        # print(self.instance_variable)  # ❌ Không có self!
        # print(cls.class_variable)      # ❌ Không có cls!

# Demo
obj = MyClass("Instance value")

# Instance method - qua object
obj.instance_method()

# Class method - qua class hoặc object
MyClass.class_method()
obj.class_method()

# Static method - qua class hoặc object
MyClass.static_method(5, 3)
obj.static_method(5, 3)
```

Bảng so sánh:

Type	Decorator	First Param	Access Instance Data	Access Class Data	Call Via
Instance	None	`self`	☑ Yes	☑ Yes	Object
Class	`@classmethod`	`cls`	❌ No	☑ Yes	Class or Object
Static	`@staticmethod`	None	❌ No	❌ No	Class or Object

7.3. Real-World Example

```
class User:
    min_password_length = 8  # Class variable

    def __init__(self, username, password):
        self.username = username

        # Validate password using static method
        if not User.is_valid_password(password):
            raise ValueError("Password không hợp lệ!")

        self.password = password
```

```
    # Instance method
    def change_password(self, new_password):
        """Đổi password"""
        if not User.is_valid_password(new_password):
            raise ValueError("Password mới không hợp lệ!")

        self.password = new_password
        print("Password đã được đổi!")

    # Static method - utility validation
    @staticmethod
    def is_valid_password(password):
        """
        Validate password:
        - Ít nhất 8 ký tự
        - Có chữ và số
        """
        if len(password) < User.min_password_length:
            return False

        has_letter = any(c.isalpha() for c in password)
        has_digit = any(c.isdigit() for c in password)

        return has_letter and has_digit

    # Class method - change rule for all users
    @classmethod
    def update_min_length(cls, new_length):
        """Đổi độ dài tối thiểu cho tất cả users"""
        cls.min_password_length = new_length
        print(f"Min length updated to {new_length}")

# Demo
user = User("alice", "pass1234")  # ✓ Valid

try:
    user2 = User("bob", "short")  # ❌ Invalid
except ValueError as e:
    print(e)  # Password không hợp lệ!

# Static method có thể gọi độc lập
print(User.is_valid_password("abc123def"))  # True
print(User.is_valid_password("tooshort"))   # False

# Class method thay đổi rule cho tất cả
User.update_min_length(10)
```

---

8. String Representation - `__str__` và `__repr__`

8.1. Vấn Đề Khi Print Object

```python
class Student:
    def __init__(self, name, age):
        self.name = name
        self.age = age

student = Student("An", 20)

# ❌ Không có __str__
print(student)
# <__main__.Student object at 0x7f8b9c...> - không hữu ích!
```

8.2. __str__ Method

Mục đích: Human-readable string representation

```python
class Student:
    def __init__(self, name, age, score):
        self.name = name
        self.age = age
        self.score = score

    def __str__(self):
        """String representation cho end-users"""
        return f"{self.name}, {self.age} tuổi, điểm: {self.score}"

student = Student("An", 20, 85)

# ☑ Có __str__
print(student)  # An, 20 tuổi, điểm: 85
print(str(student))  # An, 20 tuổi, điểm: 85
```

8.3. __repr__ Method

Mục đích: Unambiguous representation cho developers

```python
class Student:
    def __init__(self, name, age, score):
        self.name = name
        self.age = age
        self.score = score

    def __str__(self):
        """For print() - human readable"""
        return f"{self.name}, {self.age} tuổi, điểm: {self.score}"

    def __repr__(self):
```

```
        """For debugging - unambiguous"""
        return f"Student('{self.name}', {self.age}, {self.score})"

student = Student("An", 20, 85)

print(student)        # An, 20 tuổi, điểm: 85 (dùng __str__)
print(repr(student))  # Student('An', 20, 85) (dùng __repr__)

# Trong interactive shell
student  # Student('An', 20, 85) (dùng __repr__)

# Trong list
students = [student]
print(students)  # [Student('An', 20, 85)] (dùng __repr__)
```

8.4. Best Practices

Quy tắc vàng:

```
class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def __str__(self):
        """Friendly, human-readable"""
        return f"Point at ({self.x}, {self.y})"

    def __repr__(self):
        """
        Unambiguous, ideally recreate object
        Nên có thể eval(repr(obj)) == obj
        """
        return f"Point({self.x}, {self.y})"

# Demo
point = Point(3, 4)

print(point)        # Point at (3, 4)
print(repr(point))  # Point(3, 4)

# Recreate object from repr
point_copy = eval(repr(point))
print(point_copy)   # Point at (3, 4)
```

Khi nào implement:

- ☑ **Luôn implement** `__repr__` - for debugging
- ☑ **Implement** `__str__` nếu cần user-friendly output

- ⚠ Nếu chỉ có `__repr__`, `str()` sẽ fallback dùng `__repr__`

---

9. Best Practices & Common Mistakes

9.1. ☑ Best Practices

1. Naming Conventions

```
# ☑ GOOD
class Student:          # PascalCase for class names
    def __init__(self, name):
        self.name = name     # snake_case for attributes

    def get_info(self):      # snake_case for methods
        return self.name

# ❌ BAD
class student:  # Lowercase
    pass

class Student_Class:  # Snake_case
    pass
```

2. Docstrings

```
class BankAccount:
    """
    Represents a bank account with deposit/withdraw capabilities.

    Attributes:
        account_number (str): Unique account identifier
        balance (float): Current account balance
    """

    def __init__(self, account_number, initial_balance=0):
        """
        Initialize a bank account.

        Args:
            account_number (str): Account ID
            initial_balance (float): Starting balance (default 0)
        """
        self.account_number = account_number
        self.balance = initial_balance

    def deposit(self, amount):
        """
        Deposit money into account.
```

```
        Args:
            amount (float): Amount to deposit

        Returns:
            float: New balance
        """
        self.balance += amount
        return self.balance
```

3. Initialize All Attributes trong `__init__`

```
# ☑ GOOD - tất cả attributes defined trong __init__
class User:
    def __init__(self, username):
        self.username = username
        self.email = None       # Explicitly set to None
        self.is_active = False  # Default value

# ❌ BAD - attributes defined outside __init__
class User:
    def __init__(self, username):
        self.username = username

    def activate(self):
        self.is_active = True  # Attribute xuất hiện đột ngột!
```

9.2. ❌ Common Mistakes

Mistake 1: Quên `self`

```
# ❌ BAD
class Counter:
    def __init__(self):
        count = 0  # ❌ Local variable, không phải attribute!

    def increment(self):
        count += 1  # ❌ NameError!

# ☑ GOOD
class Counter:
    def __init__(self):
        self.count = 0  # ✓ Instance attribute

    def increment(self):
        self.count += 1  # ✓ Access via self
```

Mistake 2: Mutable Default Arguments

```
# ❌ BAD - DANGEROUS!
class Student:
    def __init__(self, name, scores=[]):  # ❌ Shared list!
        self.name = name
        self.scores = scores

s1 = Student("An")
s2 = Student("Bình")

s1.scores.append(85)
print(s2.scores)  # [85] - WTF? Cùng list!

# ☑ GOOD
class Student:
    def __init__(self, name, scores=None):
        self.name = name
        self.scores = scores if scores is not None else []

s1 = Student("An")
s2 = Student("Bình")

s1.scores.append(85)
print(s2.scores)  # [] - Correct!
```

Mistake 3: Truy cập class variables sai cách

```
class MyClass:
    class_var = 10

obj1 = MyClass()
obj2 = MyClass()

# ❌ BAD - creates instance variable
obj1.class_var = 20  # Tạo instance var, không modify class var!

print(obj1.class_var)      # 20 (instance)
print(obj2.class_var)      # 10 (class)
print(MyClass.class_var)   # 10 (class)

# ☑ GOOD - modify class variable
MyClass.class_var = 20

print(obj1.class_var)      # 20
print(obj2.class_var)      # 20
print(MyClass.class_var)   # 20
```

Mistake 4: Không implement `__str__` hoặc `__repr__`

```
# ❌ BAD
class Student:
    def __init__(self, name):
        self.name = name

student = Student("An")
print(student)  # <__main__.Student object at 0x...> - useless!

# ☑ GOOD
class Student:
    def __init__(self, name):
        self.name = name

    def __str__(self):
        return f"Student: {self.name}"

    def __repr__(self):
        return f"Student('{self.name}')"

student = Student("An")
print(student)  # Student: An - useful!
```

---

10. Bài Tập Thực Hành

Bài 1: Rectangle Class

```
"""
Tạo class Rectangle với:

Attributes:
- width (chiều rộng)
- height (chiều cao)

Methods:
- area(): tính diện tích
- perimeter(): tính chu vi
- is_square(): check có phải hình vuông không
- scale(factor): phóng to/thu nhỏ
- __str__(): string representation

Test:
rect = Rectangle(10, 5)
print(rect.area())        # 50
print(rect.perimeter())   # 30
print(rect.is_square())   # False

rect.scale(2)
```

```
print(rect)               # Rectangle: 20x10
"""

# Solution
class Rectangle:
    def __init__(self, width, height):
        """Initialize rectangle"""
        self.width = width
        self.height = height

    def area(self):
        """Calculate area"""
        return self.width * self.height

    def perimeter(self):
        """Calculate perimeter"""
        return 2 * (self.width + self.height)

    def is_square(self):
        """Check if it's a square"""
        return self.width == self.height

    def scale(self, factor):
        """Scale rectangle"""
        self.width *= factor
        self.height *= factor

    def __str__(self):
        return f"Rectangle: {self.width}x{self.height}"

    def __repr__(self):
        return f"Rectangle({self.width}, {self.height})"

# Test
rect = Rectangle(10, 5)
print(f"Area: {rect.area()}")            # 50
print(f"Perimeter: {rect.perimeter()}")  # 30
print(f"Is square: {rect.is_square()}")  # False

rect.scale(2)
print(rect)  # Rectangle: 20x10

square = Rectangle(5, 5)
print(f"Is square: {square.is_square()}")  # True
```

Bài 2: Student Class với Class Variables

```
"""
Tạo class Student với:
```

```
Class Variables:
- school_name = "THPT ABC"
- student_count = 0 (tăng khi tạo student mới)

Instance Attributes:
- name
- age
- scores (list)

Instance Methods:
- average(): tính điểm TB
- add_score(score): thêm điểm
- __str__(): hiển thị info

Class Methods:
- get_student_count(): trả về số lượng students
- change_school(new_name): đổi tên trường

Test:
s1 = Student("An", 20, [85, 90])
s2 = Student("Bình", 21, [92, 88])

print(Student.get_student_count())  # 2
print(s1.average())                 # 87.5

Student.change_school("THPT XYZ")
print(Student.school_name)          # THPT XYZ
"""

# Solution
class Student:
    school_name = "THPT ABC"  # Class variable
    student_count = 0         # Class variable

    def __init__(self, name, age, scores=None):
        """Initialize student"""
        self.name = name
        self.age = age
        self.scores = scores if scores is not None else []

        # Increment counter
        Student.student_count += 1

    def average(self):
        """Calculate average score"""
        if not self.scores:
            return 0
        return sum(self.scores) / len(self.scores)

    def add_score(self, score):
        """Add a score"""
        self.scores.append(score)

    def __str__(self):
```

```
        return f"{self.name} ({self.age}): avg {self.average():.2f}"

    def __repr__(self):
        return f"Student('{self.name}', {self.age}, {self.scores})"

    @classmethod
    def get_student_count(cls):
        """Get total number of students"""
        return cls.student_count

    @classmethod
    def change_school(cls, new_name):
        """Change school name for all students"""
        cls.school_name = new_name

# Test
s1 = Student("An", 20, [85, 90])
s2 = Student("Bình", 21, [92, 88])

print(f"Total students: {Student.get_student_count()}")  # 2
print(f"School: {Student.school_name}")                  # THPT ABC

print(f"{s1.name}'s average: {s1.average()}")            # 87.5

s1.add_score(95)
print(f"{s1.name}'s new average: {s1.average()}")        # 90.0

Student.change_school("THPT XYZ")
print(f"New school: {Student.school_name}")              # THPT XYZ
print(s1)  # An (20): avg 90.00
```

Bài 3: Date Class với Factory Methods

```
"""
Tạo class Date với:

Attributes:
- day, month, year

Instance Methods:
- is_valid(): check date có hợp lệ không
- __str__(): format DD/MM/YYYY

Class Methods (Factory):
- from_string(date_str): tạo từ "DD-MM-YYYY"
- today(): tạo date hôm nay

Static Methods:
- is_leap_year(year): check năm nhuận
```

```
Test:
date1 = Date(15, 1, 2024)
print(date1)                          # 15/01/2024

date2 = Date.from_string("20-02-2024")
print(date2)                          # 20/02/2024

print(Date.is_leap_year(2024))        # True
"""

# Solution
class Date:
    def __init__(self, day, month, year):
        """Initialize date"""
        self.day = day
        self.month = month
        self.year = year

        if not self.is_valid():
            raise ValueError("Invalid date!")

    def is_valid(self):
        """Check if date is valid"""
        if self.month < 1 or self.month > 12:
            return False

        if self.day < 1:
            return False

        # Days in each month
        days_in_month = [31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31]

        # Leap year adjustment
        if Date.is_leap_year(self.year):
            days_in_month[1] = 29

        if self.day > days_in_month[self.month - 1]:
            return False

        return True

    @classmethod
    def from_string(cls, date_str):
        """
        Create Date from string DD-MM-YYYY

        Example: Date.from_string("15-01-2024")
        """
        day, month, year = map(int, date_str.split('-'))
        return cls(day, month, year)

    @classmethod
    def today(cls):
        """Create Date for today"""
```

```
        import datetime
        today = datetime.date.today()
        return cls(today.day, today.month, today.year)

    @staticmethod
    def is_leap_year(year):
        """Check if year is leap year"""
        return (year % 4 == 0 and year % 100 != 0) or (year % 400 == 0)

    def __str__(self):
        return f"{self.day:02d}/{self.month:02d}/{self.year}"

    def __repr__(self):
        return f"Date({self.day}, {self.month}, {self.year})"

# Test
date1 = Date(15, 1, 2024)
print(date1)  # 15/01/2024

date2 = Date.from_string("20-02-2024")
print(date2)  # 20/02/2024

date3 = Date.today()
print(f"Today: {date3}")

print(f"2024 is leap year: {Date.is_leap_year(2024)}")  # True
print(f"2023 is leap year: {Date.is_leap_year(2023)}")  # False

# Invalid date test
try:
    invalid = Date(31, 2, 2024)  # Feb 31 - invalid!
except ValueError as e:
    print(e)  # Invalid date!
```

---

11. Tổng Kết

☑ Checklist - Classes & Objects Basics

Foundation:

- ☐ Hiểu class vs object
- ☐ Tạo class với đúng naming convention (PascalCase)
- ☐ Viết constructor `__init__`
- ☐ Hiểu và sử dụng `self` đúng cách

Variables:

- ☐ Phân biệt instance variables vs class variables
- ☐ Biết khi nào dùng loại nào
- ☐ Tránh pitfall: shadowing class variables

Methods:

- [ ] Viết instance methods
- [ ] Dùng `@classmethod` cho class-level operations
- [ ] Dùng `@staticmethod` cho utility functions
- [ ] Gọi methods từ methods khác

String Representation:

- [ ] Implement `__str__` cho user-friendly output
- [ ] Implement `__repr__` cho debugging
- [ ] Hiểu sự khác nhau giữa 2 methods

Best Practices:

- [ ] Viết docstrings cho classes và methods
- [ ] Initialize tất cả attributes trong `__init__`
- [ ] Tránh mutable default arguments
- [ ] Validation trong constructor

Key Takeaways

1. **Class = Blueprint**, Object = Instance
2. `__init__` tự động gọi khi tạo object
3. `self` = instance hiện tại
4. **Instance variables** = unique, **Class variables** = shared
5. **Instance methods** có `self`, **Class methods** có `cls`, **Static methods** không có gì
6. `__str__` cho users, `__repr__` cho developers

Next Steps

Sau khi master Chủ đề 1, bạn đã sẵn sàng cho:

- **Chủ đề 2: Encapsulation** - `@property`, private attributes
- **Chủ đề 3: Inheritance** - Code reuse, `super()`
- **Chủ đề 4: Polymorphism** - Duck typing, ABC
