Lập trình hướng đối tượng (OOP) - Phần 1

W28 - Thứ 5 sáng (09:00-12:00)

Mục tiêu bài học:

- Hiểu khái niệm lập trình hướng đối tượng (OOP)
- Nắm vững cách tạo và sử dụng class
- Hiểu về objects và cách tạo instances
- Làm quen với methods trong class

Kiến thức cần có:

- Functions (đã học w28 đầu tuần)
- Variables và data types
- Lists và dictionaries
- Loops và conditions

1. Lập trình hướng đối tượng là gì?

Tại sao cần OOP?

Trước khi học OOP, chúng ta đã viết code theo kiểu **procedural programming** (lập trình thủ tục):

- Dữ liệu và functions riêng biệt
- Khó quản lý khi chương trình lớn
- Khó tái sử dụng code

OOP giải quyết vấn đề như thế nào?

- **Đóng gói** (Encapsulation): Gom dữ liệu và functions liên quan lại với nhau
- **Tái sử dụng**: Tạo nhiều objects từ một class
- **Dễ bảo trì**: Code có tổ chức, dễ hiểu

Ví dụ thực tế:

Thay vì viết:

```
# Cách cũ - procedural
student_name = "Nguyễn Văn A"
student_age = 20
```

```
student_grade = 8.5

def print_student_info(name, age, grade):
    print(f"Tên: {name}, Tuổi: {age}, Điểm: {grade}")
```

Chúng ta có thể viết:

```
# Cách mới - OOP
class Student:
    def __init__(self, name, age, grade):
        self.name = name
        self.age = age
        self.grade = grade

    def print_info(self):
        print(f"Tên: {self.name}, Tuổi: {self.age}, Điểm: {self.grade}")
```

2. Class là gì?

Khái niệm:

- **Class** là một "bản thiết kế" (blueprint) để tạo ra objects
- **Object** là một "thực thể" (instance) được tạo từ class

Ví dụ dễ hiểu:

- **Class Car** = Bản thiết kế xe hơi
- **Object my_car** = Chiếc xe Toyota Camry cụ thể
- **Object your_car** = Chiếc xe Honda Civic cụ thể

Cú pháp cơ bản:

In [ ]:
```
# Cú pháp tạo class đơn giản
class ClassName:
    # Nội dung class
    pass

# Ví dụ class đơn giản nhất
class Dog:
    pass

# Tạo object từ class
my_dog = Dog()
print(type(my_dog))  # <class '__main__.Dog'>
print(my_dog)        # <__main__.Dog object at 0x...>
```

💡 TODO 1: Tạo class đầu tiên

Hãy tạo một class tên `Person` và tạo một object từ class này:

In [ ]:
```python
# TODO 1: Tạo class Person và tạo object
# Viết code của bạn ở đây
class Person:
    pass

# Test code
person1 = Person()
print(f"Đã tạo object: {person1}")
print(f"Kiểu dữ liệu: {type(person1)}")
```

💡 **Đáp án TODO 1:**

```python
class Person:
    pass
```

Giải thích:

- `class Person:` - Tạo class tên Person
- `pass` - Từ khóa để tạo class rỗng (chưa có nội dung)
- `person1 = Person()` - Tạo object từ class Person

3. Attributes (Thuộc tính)

Khái niệm:

- **Attributes** là các biến bên trong class
- Lưu trữ dữ liệu của object
- Mỗi object có thể có giá trị attributes khác nhau

Cách thêm attributes:

In [ ]:
```python
# Cách 1: Thêm attributes sau khi tạo object
class Dog:
    pass

# Tạo object
my_dog = Dog()

# Thêm attributes
my_dog.name = "Buddy"
my_dog.age = 3
my_dog.breed = "Golden Retriever"

print(f"Tên chó: {my_dog.name}")
print(f"Tuổi: {my_dog.age}")
print(f"Giống: {my_dog.breed}")
```

In [ ]:
```python
# Cách 2: Định nghĩa attributes trong class
class Dog:
    # Class attributes (chung cho tất cả objects)
    species = "Canis lupus"
```

```
    # Instance attributes sẽ học trong phần __init__

dog1 = Dog()
dog2 = Dog()

print(f"Dog1 species: {dog1.species}")
print(f"Dog2 species: {dog2.species}")

# Thay đổi class attribute
Dog.species = "Domestic Dog"
print(f"Sau khi thay đổi - Dog1: {dog1.species}")
print(f"Sau khi thay đổi - Dog2: {dog2.species}")
```

💡 TODO 2: Thực hành với attributes

Tạo class `Car` với các attributes và test:

In [ ]:
```
# TODO 2: Tạo class Car với attributes
class Car:
    # Thêm class attribute wheels = 4
    wheels = 4

# Tạo object và thêm attributes
my_car = Car()
# Thêm attributes: brand, model, year, color
my_car.brand = "Toyota"
my_car.model = "Camry"
my_car.year = 2023
my_car.color = "Đỏ"

# In thông tin xe
print(f"Xe của tôi:")
print(f"Hãng: {my_car.brand}")
print(f"Mẫu: {my_car.model}")
print(f"Năm: {my_car.year}")
print(f"Màu: {my_car.color}")
print(f"Số bánh: {my_car.wheels}")
```

💡 **Đáp án TODO 2:**

```
class Car:
    wheels = 4  # Class attribute - chung cho tất cả xe

my_car = Car()
my_car.brand = "Toyota"    # Instance attribute
my_car.model = "Camry"     # Instance attribute
my_car.year = 2023         # Instance attribute
my_car.color = "Đỏ"        # Instance attribute
```

Giải thích:

- `wheels = 4` - Class attribute (chung cho tất cả objects)
- `my_car.brand = "Toyota"` - Instance attribute (riêng cho object này)

- Có thể truy cập cả hai loại attributes bằng `object.attribute_name`

4. Methods (Phương thức)

Khái niệm:

- **Methods** là các functions bên trong class
- Định nghĩa các hành động mà object có thể thực hiện
- Luôn có tham số đầu tiên là `self`

Tại sao cần `self`?

- `self` đại diện cho object hiện tại
- Cho phép method truy cập attributes của object
- Python tự động truyền `self` khi gọi method

In [ ]:
```python
# Ví dụ methods cơ bản
class Dog:
    species = "Canis lupus"

    def bark(self):
        return "Woof! Woof!"

    def sleep(self):
        return "Zzz..."

    def eat(self, food):
        return f"Đang ăn {food}... Ngon quá!"

# Sử dụng methods
my_dog = Dog()
print(my_dog.bark())      # Woof! Woof!
print(my_dog.sleep())     # Zzz...
print(my_dog.eat("xương")) # Đang ăn xương... Ngon quá!
```

In [ ]:
```python
# Methods truy cập attributes
class Dog:
    def set_info(self, name, age):
        self.name = name
        self.age = age

    def get_info(self):
        return f"Tên: {self.name}, Tuổi: {self.age}"

    def have_birthday(self):
        self.age += 1
        return f"{self.name} đã {self.age} tuổi! Chúc mừng sinh nhật!"

# Test methods
dog = Dog()
```

```
dog.set_info("Buddy", 3)
print(dog.get_info())       # Tên: Buddy, Tuổi: 3
print(dog.have_birthday())  # Buddy đã 4 tuổi! Chúc mừng sinh nhật!
print(dog.get_info())       # Tên: Buddy, Tuổi: 4
```

💡 TODO 3: Thực hành với methods

Hoàn thiện class `Calculator` với các methods cơ bản:

In [ ]:

```
# TODO 3: Hoàn thiện class Calculator
class Calculator:
    def __init__(self):
        self.result = 0

    def add(self, number):
        # Thêm number vào result
        self.result += number
        return self.result

    def subtract(self, number):
        # Trừ number từ result
        self.result -= number
        return self.result

    def multiply(self, number):
        # Nhân result với number
        self.result *= number
        return self.result

    def divide(self, number):
        # Chia result cho number (kiểm tra chia cho 0)
        if number == 0:
            print("Lỗi: Không thể chia cho 0!")
            return self.result
        self.result /= number
        return self.result

    def clear(self):
        # Reset result về 0
        self.result = 0
        return self.result

    def get_result(self):
        # Trả về result hiện tại
        return self.result

# Test Calculator
calc = Calculator()
print(f"Kết quả ban đầu: {calc.get_result()}")

calc.add(10)
print(f"Sau khi +10: {calc.get_result()}")

calc.multiply(5)
print(f"Sau khi *5: {calc.get_result()}")
```

```
calc.divide(2)
print(f"Sau khi /2: {calc.get_result()}")

calc.clear()
print(f"Sau khi clear: {calc.get_result()}")
```

💡 **Đáp án TODO 3:**

```
def add(self, number):
    self.result += number
    return self.result

def subtract(self, number):
    self.result -= number
    return self.result

def divide(self, number):
    if number == 0:
        print("Lỗi: Không thể chia cho 0!")
        return self.result
    self.result /= number
    return self.result
```

Giải thích:

- `self.result += number` - Truy cập attribute qua self
- `return self.result` - Trả về kết quả để có thể in ra
- Kiểm tra chia cho 0 để tránh lỗi
- Mỗi method đều làm việc với cùng một attribute `self.result`

5. Tạo nhiều objects từ một class

Lợi ích của OOP:

Từ một class, chúng ta có thể tạo nhiều objects khác nhau:

In [ ]:
```
# Ví dụ: Tạo nhiều students
class Student:
    school = "Trường THPT ABC"  # Class attribute

    def set_info(self, name, age, grade):
        self.name = name
        self.age = age
        self.grade = grade

    def study(self, subject):
        return f"{self.name} đang học {subject}"

    def get_info(self):
        return f"Học sinh: {self.name}, Tuổi: {self.age}, Điểm: {self.grade}"
```

```
    def is_passed(self):
        return self.grade >= 5.0

# Tạo nhiều students
student1 = Student()
student1.set_info("Nguyễn Văn A", 18, 8.5)

student2 = Student()
student2.set_info("Trần Thị B", 17, 4.2)

student3 = Student()
student3.set_info("Lê Văn C", 18, 9.0)

# Test các students
students = [student1, student2, student3]

for student in students:
    print(student.get_info())
    print(f"Kết quả: {'Đậu' if student.is_passed() else 'Rớt'}")
    print(student.study("Python"))
    print(f"Trường: {student.school}")
    print("-" * 40)
```

💡 TODO 4: Quản lý danh sách objects

Tạo class `BankAccount` và quản lý nhiều tài khoản:

In [ ]:

```
# TODO 4: Tạo class BankAccount
class BankAccount:
    bank_name = "Ngân hàng ABC"  # Class attribute

    def create_account(self, account_number, owner_name, initial_balance=0):
        # Tạo tài khoản với số tài khoản, tên chủ, số dư ban đầu
        self.account_number = account_number
        self.owner_name = owner_name
        self.balance = initial_balance
        print(f"Đã tạo tài khoản {account_number} cho {owner_name}")

    def deposit(self, amount):
        # Nạp tiền vào tài khoản
        if amount > 0:
            self.balance += amount
            print(f"Đã nạp {amount:,} VND. Số dư: {self.balance:,} VND")
        else:
            print("Số tiền nạp phải lớn hơn 0!")

    def withdraw(self, amount):
        # Rút tiền (kiểm tra số dư)
        if amount <= 0:
            print("Số tiền rút phải lớn hơn 0!")
        elif amount > self.balance:
            print(f"Không đủ tiền! Số dư hiện tại: {self.balance:,} VND")
        else:
            self.balance -= amount
            print(f"Đã rút {amount:,} VND. Số dư: {self.balance:,} VND")
```

```
    def get_balance(self):
        # Trả về số dư hiện tại
        return self.balance

    def get_info(self):
        # In thông tin tài khoản
        print(f"TK: {self.account_number} | Chủ: {self.owner_name} | Số dư: {self.b

# Tạo và test nhiều tài khoản
account1 = BankAccount()
account1.create_account("001", "Nguyễn Văn A", 1000000)

account2 = BankAccount()
account2.create_account("002", "Trần Thị B", 500000)

# Test các thao tác
print("=== Thông tin ban đầu ===")
account1.get_info()
account2.get_info()

print("\n=== Giao dịch ===")
account1.deposit(200000)
account1.withdraw(150000)
account2.deposit(300000)

print("\n=== Thông tin sau giao dịch ===")
account1.get_info()
account2.get_info()
```

💡 **Đáp án TODO 4:**

```
def create_account(self, account_number, owner_name, initial_balance=0):
    self.account_number = account_number
    self.owner_name = owner_name
    self.balance = initial_balance

def withdraw(self, amount):
    if amount <= 0:
        print("Số tiền rút phải lớn hơn 0!")
    elif amount > self.balance:
        print(f"Không đủ tiền! Số dư: {self.balance:,} VND")
    else:
        self.balance -= amount
        print(f"Đã rút {amount:,} VND")
```

Giải thích:

- `self.account_number = account_number` - Lưu thông tin tài khoản
- `if amount > self.balance:` - Kiểm tra số dư trước khi rút
- `{amount:,}` - Format số với dấu phẩy (1,000,000)
- Validation input để tránh lỗi logic

6. So sánh Procedural vs OOP

Ví dụ minh họa:

Cùng một bài toán quản lý thông tin sinh viên:

In [ ]:
```python
# Cách 1: Procedural Programming
print("=== PROCEDURAL PROGRAMMING ===")

# Dữ liệu
students_data = [
    {"name": "Nguyễn Văn A", "age": 18, "grades": [8, 7, 9]},
    {"name": "Trần Thị B", "age": 17, "grades": [6, 8, 7]}
]

# Functions
def calculate_average(grades):
    return sum(grades) / len(grades)

def print_student_info(student):
    avg = calculate_average(student["grades"])
    print(f"{student['name']} - Tuổi: {student['age']} - Điểm TB: {avg:.1f}")

def add_grade(student, grade):
    student["grades"].append(grade)

# Sử dụng
for student in students_data:
    print_student_info(student)

add_grade(students_data[0], 10)
print("\nSau khi thêm điểm:")
print_student_info(students_data[0])
```

In [ ]:
```python
# Cách 2: Object-Oriented Programming
print("=== OBJECT-ORIENTED PROGRAMMING ===")

class Student:
    def __init__(self, name, age, grades=None):
        self.name = name
        self.age = age
        self.grades = grades if grades else []

    def add_grade(self, grade):
        self.grades.append(grade)

    def calculate_average(self):
        if not self.grades:
            return 0
        return sum(self.grades) / len(self.grades)

    def print_info(self):
        avg = self.calculate_average()
```

```
        print(f"{self.name} - Tuổi: {self.age} - Điểm TB: {avg:.1f}")

# Sử dụng
student1 = Student("Nguyễn Văn A", 18, [8, 7, 9])
student2 = Student("Trần Thị B", 17, [6, 8, 7])

students = [student1, student2]

for student in students:
    student.print_info()

student1.add_grade(10)
print("\nSau khi thêm điểm:")
student1.print_info()
```

Ưu điểm của OOP:

1. **Tổ chức tốt hơn**: Dữ liệu và functions liên quan được nhóm lại
2. **Dễ bảo trì**: Thay đổi logic chỉ cần sửa trong class
3. **Tái sử dụng**: Tạo nhiều objects từ một class
4. **Mở rộng**: Dễ dàng thêm tính năng mới
5. **Trực quan**: Gần với cách suy nghĩ thực tế

7. Bài tập thực hành tổng hợp

💡 TODO 5: Tạo class Library Management System

Tạo hệ thống quản lý thư viện đơn giản:

In [ ]:
```
# TODO 5: Hoàn thiện Library Management System
class Book:
    def __init__(self, title, author, isbn):
        # Khởi tạo sách với tiêu đề, tác giả, ISBN
        # Trạng thái mặc định: available = True
        self.title = title
        self.author = author
        self.isbn = isbn
        self.available = True

    def borrow(self):
        # Mượn sách (đổi available thành False)
        if self.available:
            self.available = False
            return True
        return False

    def return_book(self):
        # Trả sách (đổi available thành True)
        self.available = True

    def get_info(self):
```

```
        # In thông tin sách và trạng thái
        status = "Có sẵn" if self.available else "Đã mượn"
        return f"'{self.title}' - {self.author} ({self.isbn}) - {status}"

class Library:
    def __init__(self, name):
        self.name = name
        self.books = []  # Danh sách các sách

    def add_book(self, book):
        # Thêm sách vào thư viện
        self.books.append(book)
        print(f"Đã thêm sách: {book.get_info()}")

    def find_book(self, title):
        # Tìm sách theo tiêu đề
        for book in self.books:
            if book.title.lower() == title.lower():
                return book
        return None

    def list_available_books(self):
        # Liệt kê các sách có thể mượn
        print("\n=== Sách có sẵn ===")
        available_books = [book for book in self.books if book.available]
        if available_books:
            for book in available_books:
                print(f"- {book.get_info()}")
        else:
            print("Không có sách nào có sẵn")

    def borrow_book(self, title):
        # Mượn sách theo tiêu đề
        book = self.find_book(title)
        if book:
            if book.borrow():
                print(f"Đã mượn: {book.title}")
            else:
                print(f"Sách '{title}' đã được mượn rồi!")
        else:
            print(f"Không tìm thấy sách '{title}'")

    def return_book(self, title):
        # Trả sách theo tiêu đề
        book = self.find_book(title)
        if book:
            book.return_book()
            print(f"Đã trả: {book.title}")
        else:
            print(f"Không tìm thấy sách '{title}'")

# Test hệ thống
library = Library("Thư viện Trung tâm")

# Thêm sách
book1 = Book("Python Programming", "John Doe", "123-456-789")
```

```
book2 = Book("Data Science", "Jane Smith", "987-654-321")

library.add_book(book1)
library.add_book(book2)

# Test các chức năng
print(f"=== {library.name} ===")
library.list_available_books()

print("\n=== Mượn sách ===")
library.borrow_book("Python Programming")
library.list_available_books()

print("\n=== Trả sách ===")
library.return_book("Python Programming")
library.list_available_books()
```

💡 **Đáp án TODO 5:**

```
class Book:
    def __init__(self, title, author, isbn):
        self.title = title
        self.author = author
        self.isbn = isbn
        self.available = True

    def borrow(self):
        if self.available:
            self.available = False
            return True
        return False

class Library:
    def find_book(self, title):
        for book in self.books:
            if book.title.lower() == title.lower():
                return book
        return None
```

Giải thích:

- `__init__` - Constructor để khởi tạo object với dữ liệu
- `self.available = True` - Trạng thái mặc định của sách
- `book.title.lower()` - So sánh không phân biệt hoa thường
- `return None` - Trả về None nếu không tìm thấy
- List comprehension để lọc sách có sẵn

8. Tóm tắt bài học

Kiến thức đã học:

1. **Lập trình hướng đối tượng (OOP)**:

- Cách tổ chức code tốt hơn
- Gom dữ liệu và functions liên quan
- Dễ tái sử dụng và bảo trì

2. **Class và Objects**:
   - Class = bản thiết kế
   - Object = thực thể từ class
   - Cú pháp: `class ClassName:`
3. **Attributes (Thuộc tính)**:
   - Biến bên trong class
   - Class attributes: chung cho tất cả objects
   - Instance attributes: riêng cho từng object
4. **Methods (Phương thức)**:
   - Functions bên trong class
   - Luôn có tham số `self` đầu tiên
   - Truy cập attributes qua `self`

Lợi ích của OOP:

- **Tổ chức**: Code có cấu trúc rõ ràng
- **Tái sử dụng**: Tạo nhiều objects từ một class
- **Bảo trì**: Dễ sửa đổi và mở rộng
- **Trực quan**: Gần với thực tế

9. Bài tập về nhà

💡 Bài tập 1: Class Person cơ bản

Tạo class `Person` với:

- Attributes: name, age, city
- Methods: introduce(), have_birthday(), move_to(new_city)

💡 Bài tập 2: Class Rectangle

Tạo class `Rectangle` với:

- Attributes: width, height
- Methods: calculate_area(), calculate_perimeter(), is_square()

💡 Bài tập 3: Class ShoppingCart

Tạo class `ShoppingCart` với:

- Attributes: items (list), total_price
- Methods: add_item(name, price), remove_item(name), calculate_total(), display_cart()

💡 Bài tập 4: Mở rộng Library System

Thêm vào Library Management System:

- Class `Member` với thông tin thành viên
- Theo dõi ai mượn sách nào
- Tính phí trễ hạn

💡 Bài tập 5: Tự do sáng tạo

Tạo một hệ thống OOP cho một chủ đề bạn quan tâm:

- Game đơn giản (Player, Enemy, Weapon)
- Quản lý cửa hàng (Product, Customer, Order)
- Mạng xã hội mini (User, Post, Comment)

Lưu ý: Hãy bắt đầu đơn giản, tập trung vào việc hiểu rõ class, object, attributes và methods trước khi làm những bài phức tạp.

10. Chuẩn bị cho bài học tiếp theo

Bài học tiếp theo (W28 - Thứ 5 chiều):

"Student class, Simple OOP demo, object creation"

Kiến thức sẽ học:

1. **Constructor (`__init__`)**:
   - Khởi tạo object với dữ liệu ban đầu
   - Tham số và default values
   - Validation trong constructor
2. **Instance vs Class attributes**:
   - Phân biệt rõ ràng hơn
   - Khi nào dùng loại nào
   - Memory và performance
3. **Method types**:
   - Instance methods
   - Class methods (preview)

- Static methods (preview)

4. **Thực hành với Student class**:

- Quản lý thông tin sinh viên
- Tính điểm trung bình
- Xếp loại học tập

Ôn tập trước khi đến lớp:

- [ ] Xem lại cách tạo class và object
- [ ] Thực hành viết methods với `self`
- [ ] Làm ít nhất 2 bài tập về nhà
- [ ] Nghĩ về các ví dụ thực tế áp dụng OOP

Câu hỏi tự kiểm tra:

1. Class và Object khác nhau như thế nào?
2. Tại sao methods luôn có tham số `self`?
3. Khi nào nên dùng OOP thay vì Procedural Programming?
4. Làm thế nào để truy cập attributes từ bên trong methods?
5. Có thể tạo bao nhiêu objects từ một class?

Hẹn gặp lại trong buổi học tiếp theo! 🚀

📚 Tài liệu tham khảo thêm:

Online Resources:

- Python.org OOP Tutorial
- Real Python - OOP in Python 3
- W3Schools Python Classes

Sách tiếng Việt:

- "Lập trình Python cho người mới bắt đầu" - Chương OOP
- "Python cơ bản đến nâng cao" - Phần hướng đối tượng

Video hướng dẫn:

- YouTube: "Python OOP Tutorial for Beginners"
- Coursera: "Python for Everybody - Object Oriented Programming"

Bài giảng được soạn cho khóa đào tạo Python W28 - Thứ 5 sáng
Giảng viên: Nguyễn Mạnh Trung, Nguyễn Văn Anh

Thời gian: 09:00-12:00