Object-Oriented Programming

Lập trình hướng đối tượng trong Python

Mục tiêu

- Hiểu các khái niệm cơ bản về lập trình hướng đối tượng trong Python
- Biết cách triển khai các kỹ thuật lập trình hướng đối tượng
- Xây dựng chương trình đơn giản theo mô hình hướng đối tượng

Nội dung

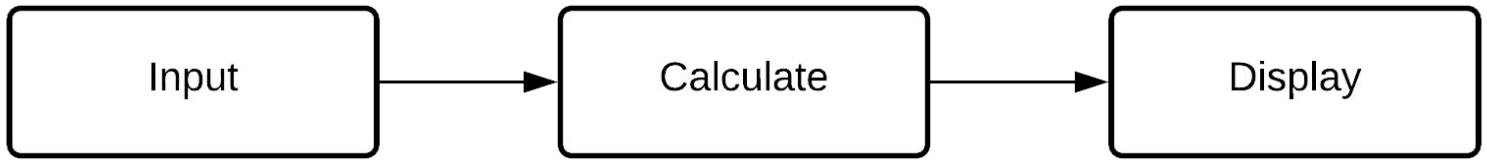
- Giới thiệu về lập trình hướng đối tượng
- Các tính chất của lập trình hướng đối tượng
- Classes vs Objects
- Magic Methods (các hàm đặc biệt)
- Inheritance
- Composition
- Polymorphism

Giới thiệu

Chương trình là gì?

Chương trình - program - là một tập hợp các câu lệnh hay chỉ dẫn để thao tác với dữ liệu theo một cách nào đó

Ví dụ chương trình tính tuổi



Mô hình lập trình - Programming paradigm

Mô hình lập trình là một cách tiếp cận để tổ chức và cấu trúc mã trong chương trình hay nói cách khác, nó là một phong cách lập trình.

Mô hình lập trình phổ biến:

- Lập trình thủ tục (hàm) - Procedural Programming
- Lập trình hướng đối tượng - Object-Oriented Programming (OOP)

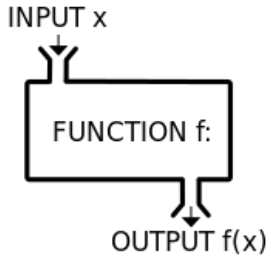
** Các ngôn ngữ lập trình cũng được thiết kế để cung cấp các tính năng cho một (hoặc nhiều) mô hình lập trình cụ thể*

Lập trình thủ tục

Lập trình thủ tục (hàm): là quy trình thực hiện các công việc theo thứ tự để giải quyết một vấn đề cụ thể

Mỗi công việc được thực hiện bởi các khối mã (được gọi là hàm - chức năng), dữ liệu được chuyển qua các chức năng và biến đổi cho ra kết quả cuối cùng

Ngôn ngữ phổ biến: C, Perl, ...



Lập trình hướng đối tượng

Trong **lập trình hướng đối tượng**, các khái niệm hay thực thể trong thế giới thực được mô hình hóa trong ngôn ngữ lập trình bằng các đối tượng (object)

Mỗi đối tượng bao gồm:

- Trạng thái (state) hay thuộc tính: là thông tin, đặc điểm mô tả về đối tượng
- Hành vi (behaviour): là hành động, hay chức năng mà đối tượng có thể thực hiện

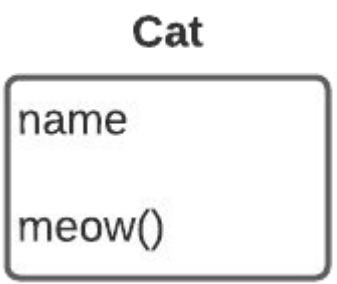
Lập trình hướng đối tượng

```
# Properties
breed
name
age
color
```

```
# Behaviors
eat()
meow()
run()
sleep()
```

Lập trình hướng đối tượng

Class



Objects

Lập trình hướng đối tượng

```
# Properties
breed
name
age
color

# Behaviors
eat()
meow()
run()
sleep()
```

```
breed = "Persian"
name = "Lucy"
age = 2
color = "White"
```

```
breed = "British"
name = "Kitty"
age = 1
color = "Grey"
```

Lập trình hướng đối tượng

Lớp - Class - đại diện cho một tập hợp các đối tượng có thuộc tính và hành vi chung. Lớp cung cấp bản thiết kế hay mô tả các đối tượng được tạo ra từ nó.

Khai báo một lớp cũng là khai báo một kiểu dữ liệu mới (*) (**Class Object**) và các giá trị từ kiểu dữ liệu mới này được gọi là **Object hoặc Instance**

* Tương tự như các kiểu dữ liệu có sẵn (built-in) trong Python

Lập trình hướng đối tượng

- Instantiation (khởi tạo)
- State → Properties hay Data members
- Behaviour → Methods (phương thức)
- Encapsulation (tính đóng gói)
- Inheritance (tính kế thừa)
- Polymorphims (tính đa hình)
- Abstraction (tính trừu tượng)

Lập trình hướng đối tượng

- **Encapsulation**: Tất cả trạng thái và phương thức được lưu trong đối tượng, thông tin có thể được che giấu để hạn chế truy cập từ mã bên ngoài
- **Inheritance**: Xây dựng các lớp mới từ lớp hiện có, mở rộng các thuộc tính và phương thức có sẵn
- **Polymorphism**: Các đối tượng khác nhau thực thi cùng một phương thức nhưng theo cách khác nhau
- **Abstraction**: Tổng quát hóa, loại bỏ các chi tiết không cần thiết mà chỉ quan tâm tới điểm quan trọng nhất

Lập trình hướng đối tượng

Ưu điểm của lập trình hướng đối tượng:

- Mô hình hóa những hệ thống phức tạp dưới dạng cấu trúc đơn giản
- Code có thể tái sử dụng dễ dàng
- Tính bảo mật cao, thông tin có thể được bảo vệ
- Dễ dàng mở rộng dự án

Class vs Object

Class vs Object

```
class ClassName:                # Khai báo class
    def method(self):           # Khai báo phương thức
        pass

obj = ClassName()               # Khởi tạo instance object
obj.x = 123                     # Khai báo instance variable
obj.method()                    # Gọi phương thức
print(obj.x)                    # Truy cập instance variable
```

Class vs Object

- Khi khai báo class, Python tạo ra một class object và gán cho tên class
- Khởi tạo một class tạo ra một instance object mới
- Cả class object và instance object đều có thể sử dụng giống như những giá trị thông thường (truyền vào hàm, lưu trong list, return từ hàm, ...)
- Phương thức được khai báo trong class với cú pháp khai báo hàm thông thường với tham số đầu tiên là **self**
- Có thể truy cập đến thuộc tính (hay **instance variables**) và phương thức khác thông qua **self**

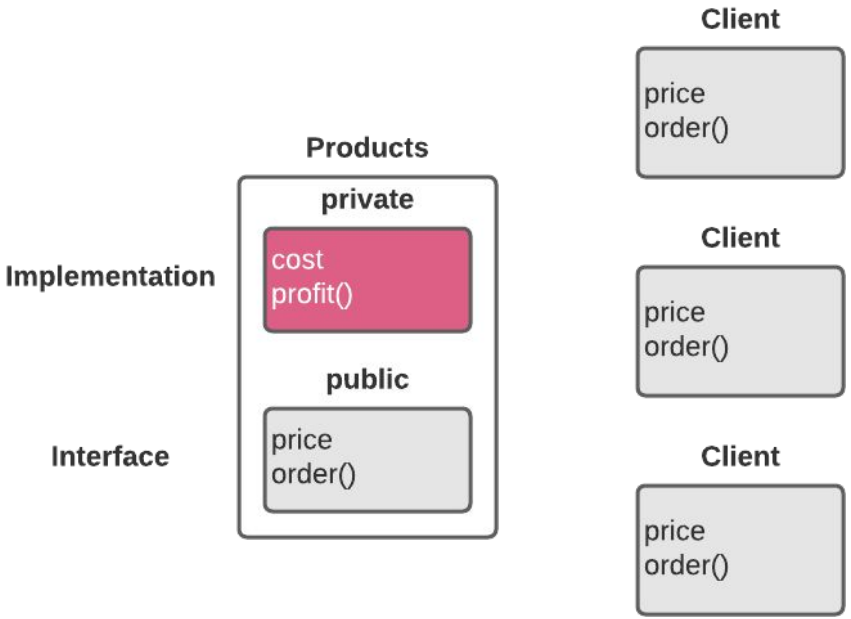
Hàm khởi tạo

```
class ClassName:
    def __init__(self, x):          # Hàm khởi tạo
        self.x = x                  # Khai báo instance variables

    def method(self):
        print(self.x)               # Truy cập instance variable

obj = ClassName(123)                # Truyền giá trị cho __init__
obj.method()
```

Hàm khởi tạo

- **__init__()** được sử dụng để khai báo instance variables, hoặc thực thi một số câu lệnh cần thiết khác **ngay sau khi đối tượng được tạo**
- Ngoài **__init__()**, còn nhiều phương thức đặc biệt khác nữa (magic methods)
- Mỗi class chỉ có một phương thức **__init__()** (Python không hỗ trợ method overloading như Java)

Data hiding


Client
price
order()
Products
private
cost
profit()
Implementation
Client
price
order()
public
price
order()
Interface
Client
price
order()

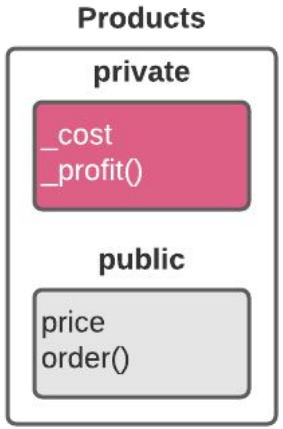
Data hiding

Interface - lớp giao tiếp cho biết cách mà một đối tượng tương tác với mã bên ngoài, và thường được xác định từ đầu, với đảm bảo là nó sẽ không thay đổi.

Implementation - lớp triển khai thì ngược lại, nó có thể bị thay đổi, hay xóa bỏ mà không làm ảnh hưởng những hệ thống đang sử dụng.

Data hiding

Python không hỗ trợ phương pháp hay toán tử hạn chế quyền truy cập nào. Chỉ có quy ước đặt tên cho biết một giá trị hay phương thức là riêng tư:



@property

```python
@property                           # Getter
def prop_name(self):                # Khai báo thuộc tính `pro_name`
    pass

@prop_name.setter                   # Setter
def prop_name(self, new_value):
    pass

@prop_name.deleter                   # Deleter
def prop_name(self):
    pass
```

@property

- Property là các phương thức cho phép truy cập đến instance variable
- Property được sử dụng giống như instance variable thông thường
- Sử dụng Property để kiểm tra kiểu và xác thực dữ liệu
- Có thể biến một instance variable thành read-only hoặc write-only bằng cách chỉ triển khai các phương thức tương ứng
- Tạo instance variable được tính toán tự động
- Cho phép áp dụng các thay đổi với instance variable mà không cần thay đổi mã đang sử dụng nó

Class variables

```
class ClassName:
    class_variable = 123          # Class variable - class attribute

    def method(self):
        pass

ClassName.class_variable          # Truy cập trực tiếp từ class

obj = ClassName()
obj.class_variable                # Hoặc thông qua object
```

Class variables

- Class variables là các thông tin thuộc về class, được chia sẻ bởi tất cả instance object
- Class variables được khai báo ở bên ngoài tất cả phương thức
- Class variables có thể truy cập qua class object hoặc instance object
- Khi muốn thay đổi giá trị của class variable, phải truy cập thông qua class
- Class variables có thể truy cập mà không cần bất kỳ instance object nào được tạo

@classmethod

```
class ClassName:

    @classmethod            # Đánh dấu một method thuộc về class
    def method(cls):        # Tham số đầu tiên theo quy ước là `cls`
        print(cls)          # `cls` tham chiếu tới class thay vì instance

ClassName.method()          # Gọi class method
obj = ClassName()
obj.method()                # Cũng có thể gọi qua instance object
```

Class methods

- Class methods là các phương thức thuộc về lớp
- Class method được đánh dấu với @**classmethod**
- Tham số đầu tiên của class method là **cls**, tham chiếu đến class
- Class methods không thể tham chiếu đến instance variables
- Class methods hữu ích để tạo **Factory methods**