LẬP TRÌNH PYTHON CƠ BẢN

BÀI 5

LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG TRONG PYTHON

NỘI DUNG

1. Giới thiệu
2. Cài đặt lớp
3. Quan hệ kế thừa
4. Quan hệ hợp thành
5. Lớp cơ sở trừu tượng
6. Tính đa hình

1. GIỚI THIỆU

1. GIỚI THIỆU

Lập trình hướng đối tượng trong Python

- Python là ngôn ngữ lập trình đa năng hỗ trợ nhiều phong cách lập trình khác nhau, bao gồm lập trình hướng đối tượng (OOP) thông qua việc sử dụng các đối tượng và lớp.
- Đối tượng là bất kỳ thực thể nào nào có thuộc tính và hành vi. Ví dụ, một con vẹt là một đối tượng. Nó có
  - Thuộc tính: tên, tuổi, màu sắc,...
  - Hành vi: nhảy, hót,...
- Có thể nói lớp là một thiết kế cấu trúc của đối tượng.

1. GIỚI THIỆU

Ưu điểm của OOP trong Python

- Tạo ra các mô hình và giải quyết các vấn đề phức tạp trong thực tế.
- Sử dụng lại mã và tránh lặp lại.
- Đóng gói dữ liệu và các thao tác liên quan thành một gói duy nhất.
- Trừu tượng hóa các chi tiết trong thực tế của các khái niệm, đối tượng.

=> Tóm lại, sử dụng lớp trong Python giúp viết code có tổ chức, có cấu trúc, dễ bảo trì, có thể tái sử dụng, linh hoạt và thân thiện với người dùng hơn.

1. GIỚI THIỆU

Một số trường hợp không nên sử dụng lớp trong Python

- Một chương trình, file mã nguồn nhỏ và đơn giản không yêu cầu cấu trúc dữ liệu hoặc logic phức tạp.
- Một chương trình quan trọng về hiệu suất.
- Sử dụng lại mã code không lập trình theo hướng đối tượng.
- Một team nhưng có nhiều phong cách code khác nhau, cần có sự thống nhất về phong cách viết để đảm bảo tính nhất quán trong dự án.

1. GIỚI THIỆU

Định nghĩa lớp trong Python

- Sử dụng từ khóa ***class*** để định nghĩa lớp:

```
class ClassName:
    # Class body
    pass
```

- Trong thân lớp, có thể định nghĩa các thuộc tính và phương thức cần thiết.
- Chỉ có thể truy cập các thuộc tính và phương thức thông qua lớp hoặc các đối tượng của nó.

Định nghĩa lớp

- Ví dụ: Định nghĩa một lớp Circle

```
import math

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

    def calculate_area(self):
        return round(math.pi * self.radius ** 2, 2)
```

- **__init__** là phương thức khởi tạo của lớp trong Python
- Đối số đầu tiên trong hầu hết các phương thức là ***self*** tham chiếu đến đối tượng hiện tại của lớp (tương tự con trỏ this trong C/C++)

Khởi tạo đối tượng

- Khởi tạo đối tượng dựa trên lớp đã được cài đặt

```
circle_1 = Circle(42)
circle_2 = Circle(7)

print(circle_1)
print(circle_2)
```

```
<circle.Circle object at 0x0000020B5A175850>
<circle.Circle object at 0x0000020B5A1750D0>
```

- Khi khai báo một đối tượng, hàm khởi tạo được gọi và cần truyền vào các đối số tương ứng.
- Gọi hàm tạo của lớp với các giá trị đối số khác nhau sẽ cho phép tạo các đối tượng hoặc thể hiện khác nhau của lớp đích.

Truy cập thuộc tính và phương thức

- Trong Python, có thể truy cập các thuộc tính và phương thức của một đối tượng bằng cách sử dụng ký hiệu dấu chấm.

```
obj.attribute_name
obj.method_name()
```

- Lưu ý: để gọi một hàm hoặc phương thức, cần sử dụng một cặp dấu ngoặc đơn và các đối số nếu cần.

Truy cập thuộc tính và phương thức

- Ví dụ:

```
circle_1.radius
circle_1.calculate_area()

circle_2.radius
circle_2.calculate_area()
```

```
42
5541.77
7
153.94
```

- Có thể sử dụng ký hiệu dấu chấm và câu lệnh gán để thay đổi giá trị hiện tại của thuộc tính

```
circle_1.radius = 100
print(circle_1.radius)
print(circle_1.calculate_area())
```

```
100
31415.93
```

Đặt tên khi định nghĩa lớp

- Thông thường lập trình viên thường đặt tên theo dạng snake_case, sử dụng _ để ngăn cách các từ trong tên.
- Nên đặt tên lớp trong Python theo dạng PascalCase, viết hoa chữ cái đầu của các từ.
- Python không phân biệt các thuộc tính thành 3 phạm vi truy cập **private, protected** và **public** như một số ngôn ngữ khác.
- Trong Python, mọi thuộc tính đều có thể truy cập được theo các cách khác nhau.

Đặt tên khi định nghĩa lớp

- Quy ước đặt tên để thể hiện phạm vi truy cập của thành viên trong hay ngoài lớp.
- Quy ước đặt tên khi định nghĩa lớp:

Thành viên	Đặt tên	Ví dụ
Công khai	Sử dụng cách đặt tên thông thường	radius, calculate_area()
Không công khai	Sử dụng dấu _ ở đầu tên thành viên	_radius, _calculate_area()

- *Lưu ý: Sử dụng _ trước tên thành viên, thành viên vẫn có thể được truy cập từ bên ngoài lớp.*

Đặt tên khi định nghĩa lớp

Name mangling:

- Khi một thuộc tính hoặc phương thức trong một lớp Python bắt đầu bằng hai dấu gạch dưới (__), Python sẽ thay đổi tên của thành viên đó bằng cách thêm tên của lớp đó trước tên của thành viên.
- Điều này thường được sử dụng để tạo ra các thành viên có tên duy nhất với mục đích tránh xung đột tên giữa các lớp con.

```
class MyClass:
    def __init__(self):
        self.__private_attribute = 10

class MySubClass(MyClass):
    def get_attribute(self):
        # Tên thuộc tính là _MyClass__private_attribute
        return self._MyClass__private_attribute

obj = MySubClass()
print(obj.get_attribute())  # Kết quả: 10
print(obj.__private_attribute) # Lỗi
```

Đặt tên khi định nghĩa lớp

- Khi viết các lớp, chưa thể quyết định một thành viên là công khai hay không công khai
- Nên viết thành viên ở chế độ không công khai để đảm bảo an toàn dữ liệu.
- Sau đó, công khai những thành viên nếu đặt trong trường hợp cần thiết phải sử dụng thành viên đó bên ngoài lớp.

2. CÀI ĐẶT LỚP

Thuộc tính (Attributes)

- Có 2 kiểu thuộc tính trong Python:
  - **Thuộc tính lớp (Class attributes)**: là biến thuộc về lớp chứ không phải của một đối tượng cụ thể, được chia sẻ giữa tất cả các đối tượng của lớp và được định nghĩa bên ngoài hàm khởi tạo.
  - **Thuộc tính đối tượng (Instance attributes)**: Là biến thuộc về một và chỉ một đối tượng; chỉ có thể truy cập trong phạm vi của đối tượng và được xác định bên trong hàm khởi tạo.

Thuộc tính (Attributes)

- ***Thuộc tính lớp (Class attributes):***
  - Các biến được xác định nghĩa bên trong lớp nhưng bên ngoài tất cả các phương thức.
  - Được chia sẻ với tất cả các đối tượng của lớp => thay đổi sẽ ảnh hưởng đến tất cả các đối tượng.

```
class ObjectCounter:
    num = 0
    def __init__(self):
        ObjectCounter.num += 1
```

```
obj1 = ObjectCounter()
print(obj1.num)

obj2 = ObjectCounter()
print(obj2.num)

print("Current counter:", ObjectCounter.num)
```

```
1
2
Current counter: 2
```

Thuộc tính (Attributes)

- ***Thuộc tính lớp (Class attributes):***
  - Có thể sử dụng hàm **type()** để lấy ra lớp hiện tại thông qua **self.** Hàm **type()** trong Python trả về lớp hoặc kiểu của đối số truyền vào.

```
class ObjectCounter:
    num = 0
    def __init__(self):
        type(self).num += 1
```

  - Nếu muốn thay đổi thuộc tính lớp, cần sử dụng chính lớp đó chứ không phải là đối tượng thuộc lớp.

```
class ObjectCounter:
    num = 0
    def __init__(self):
        self.num += 1
```

Thuộc tính (Attributes)

- ***Thuộc tính đối tượng (Instance attributes):***
  - Là các biến được gắn với một đối tượng cụ thể của một lớp nhất định. Giá trị của một thuộc tính được gắn vào chính đối tượng đó.
  - Giá trị của thuộc tính dành riêng cho đối tượng.

```
class Car:
    def __init__(self, make, model, year, color):
        self.make = make
        self.model = model
        self.year = year
        self.color = color
        self.started = False
        self.speed = 0
        self.max_speed = 200
```

```
toyota_camry = Car("Toyota", "Camry", 2022, "Red")
print(toyota_camry.make)
print(toyota_camry.model)
print(toyota_camry.color)
print(toyota_camry.speed)
```

```
Toyota
Camry
Red
0
```

Thuộc tính (Attributes)

- ***Thuộc tính đối tượng (Instance attributes):***
  - Không giống như thuộc tính lớp, bạn không thể truy cập thuộc tính đối tượng thông qua lớp. Bạn cần truy cập chúng thông qua đối tượng chứa chúng:

```
print(Car.make)
```

```
Traceback (most recent call last):
    ...
AttributeError: type object 'Car' has no attribute 'make'
```

Thuộc tính (Attributes)

- ***Thuộc tính .__dict__***
  - Là thuộc tính đặc biệt có trong cả lớp và đối tượng.
  - Xét ví dụ:

```
class SampleClass:
    class_attr = 100

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

    def method(self):
        print(f"Class attribute: {self.class_attr}")
        print(f"Instance attribute: {self.instance_attr}")
```

Thuộc tính (Attributes)

- ***Thuộc tính .__dict__***
  - Trong một lớp, .__dict__ chứa các thuộc tính và phương thức. Khi truy cập thành viên lớp thông qua đối tượng, Python sẽ tìm kiếm tên thành viên trong lớp .__dict__.

```
pprint(SampleClass.__dict__)
```

```
mappingproxy({
    '__module__': '__main__',
    'class_attr': 100,
    '__init__': <function SampleClass.__init__ at 0x1036c62a0>,
    'method': <function SampleClass.method at 0x1036c56c0>,
    '__dict__': <attribute '__dict__' of 'SampleClass' objects>,
    '__weakref__': <attribute '__weakref__' of 'SampleClass' objects>,
    '__doc__': None
})
```

Thuộc tính (Attributes)

- ***Thuộc tính .__dict__***
  - Trong một đối tượng, .__dict__ chứa các thuộc tính đối tượng. Tương tự, khi truy cập một thành viên thông qua một đối tượng cụ thể của một lớp, Python sẽ tìm tên thành viên đó trong .__dict__.
  - Có thể thay đổi giá trị của các thuộc tính thông qua .__dict__.

```
instance = SampleClass("Hello!")
print(instance.__dict__)
print(instance.__dict__["instance_attr"])
instance.__dict__["instance_attr"] = "Hello, Pythonista!"
print(instance.instance_attr)
```

```
{'instance_attr': 'Hello!'}
Hello!
Hello, Pythonista!
```

Thuộc tính (Attributes)

- ***Thuộc tính động***
  - Trong Python, có thể thêm các thuộc tính mới vào các lớp và đối tượng một cách linh hoạt.
  - Xét ví dụ sau, nhằm mục đích lưu trữ dữ liệu từ CSDL hoặc file CSV:
  - Trong lớp này, chưa xác định bất kỳ thuộc tính hoặc phương thức nào vì không biết lớp sẽ lưu trữ dữ liệu gì.

```
class Record:
    """Hold a record of data."""
```

Thuộc tính (Attributes)

- ***Thuộc tính động***
  - Sử dụng hàm **setattr()** tích hợp để thêm tuần tự từng trường làm thuộc tính cho đối tượng **john_record**.

```
class Record:
    """Hold a record of data."""

john = {
    "name": "John Doe",
    "position": "Python Developer",
    "department": "Engineering",
    "salary": 80000,
    "hire_date": "2020-01-01",
    "is_manager": False,
}
```

```
john_record = Record()

for field, value in john.items():
    setattr(john_record, field, value)

print(john_record.__dict__)
```

```
{
    'name': 'John Doe',
    'position': 'Python Developer',
    'department': 'Engineering',
    'salary': 80000,
    'hire_date': '2020-01-01',
    'is_manager': False
}
```

Phương thức (Methods)

- Trong lớp Python, bạn có thể định nghĩa ba loại phương thức khác nhau:
  - ***Phương thức đối tượng***: lấy đối tượng hiện tại, **self**, làm đối số đầu tiên
  - ***Phương thức lớp***: lấy lớp hiện tại, **cls**, làm đối số đầu tiên
  - ***Phương thức tĩnh***: không dùng đối số là lớp hay đối tượng

Phương thức (Methods)

- ***Phương thức đối tượng***:
  - Đối số self giữ một tham chiếu đến phiên bản hiện tại, cho phép truy cập đối tượng đó từ bên trong các phương thức. Quan trọng hơn, thông qua self, có thể truy cập và sửa đổi các thuộc tính của đối tượng cũng như gọi các phương thức khác trong lớp.
  - Xét lớp Car

```
class Car:
    def __init__(self, make, model, year, color):
        self.make = make
        self.model = model
        self.year = year
        self.color = color
        self.started = False
        self.speed = 0
        self.max_speed = 200
```

```
class Car:
    def __init__(self, make, model, year, color):
        self.make = make
        self.model = model
        self.year = year
        self.color = color
        self.started = False
        self.speed = 0
        self.max_speed = 200
```

Phương thức (Methods)

- ***Phương thức đối tượng***:
  - Bổ sung 2 phương thức **start()** và **stop()** sử dụng **self** để truy cập các thuộc tính đối tượng.

```
def start(self):
    print("Starting the car...")
    self.started = True

def stop(self):
    print("Stopping the car...")
    self.started = False
```

```
class Car:
    def __init__(self, make, model, year, color):
        self.make = make
        self.model = model
        self.year = year
        self.color = color
        self.started = False
        self.speed = 0
        self.max_speed = 200
```

Phương thức (Methods)

- ***Phương thức đối tượng***:
  - Bổ sung 2 phương thức **start()** và **stop()**

    sử dụng **self** để truy cập các thuộc tính đối tượng.
  - Bổ sung 2 phương thức **accelerate()** và **brake()**

```
def accelerate(self, value):
    if not self.started:
        print("Car is not started!")
        return
    if self.speed + value <= self.max_speed:
        self.speed += value
    else:
        self.speed = self.max_speed
    print(f"Accelerating to {self.speed} km/h...")
```

```
def brake(self, value):
    if self.speed - value >= 0:
        self.speed -= value
    else:
        self.speed = 0
    print(f"Braking to {self.speed} km/h...")
```

Phương thức (Methods)

- ***Phương thức đối tượng***:
  - Bổ sung 2 phương thức **start()** và **stop()** sử dụng **self** để truy cập các thuộc tính đối tượng.
  - Bổ sung 2 phương thức **accelerate()** và **brake().**
  - Sử dụng các phương thức.

```
class Car:
    def __init__(self, make, model, year, color):
        self.make = make
        self.model = model
        self.year = year
        self.color = color
        self.started = False
        self.speed = 0
        self.max_speed = 200
```

```
toyota_camry = Car("Toyota", "Camry", 2022, "Red")

toyota_camry.started()
toyota_camry.accelerate(100)
toyota_camry.brake(50)
toyota_camry.brake(80)
toyota_camry.stop()

toyota_camry.accelerate(100)
```

```
Car.start(toyota_camry)
Car.accelerate(toyota_camry, 100)
Car.brake(toyota_camry, 50)
Car.brake(toyota_camry, 80)
Car.stop(toyota_camry)
```

```
Starting the car...
Accelerating to 100 km/h...
Braking to 50 km/h...
Braking to 0 km/h...
Stopping the car...

Car is not started!
```

Phương thức (Methods)

- ***Phương thức đối tượng***:
  - ***Phương thức đặc biệt (dunder/magic methods):*** các phương thức được tự động gọi để đáp ứng các thao tác cụ thể
    - __init__(): hàm tạo/phương thức khởi tạo
    - __str__(): biểu diễn chuỗi khi in ra của một đối tượng

```
def __str__(self):
    return f"{self.make}, {self.model}, {self.color}: ({self.year})"
```

Phương thức (Methods)

- ***Phương thức đối tượng***:
  - ***Giao thức (protocol):*** bao gồm một hoặc nhiều phương thức đặc biệt hỗ trợ một tính năng hoặc chức năng nhất định

Giao thức	Tính năng	Phương thức đặc biệt
Vòng lặp	Tạo ra các đối tượng vòng lặp	.__iter__() and .__next__()
Cho phép lặp	Làm cho đối tượng có khả năng lặp	.__iter__()
Mô tả	Thao tác lên các thuộc tính	.__get__() , .__set__(), .__delete__(), .__set_name__()
Quản lý ngữ cảnh	Làm cho đối tượng thao tác được với câu lệnh with	.__enter__() and .__exit__()

Phương thức (Methods)

- ***Phương thức lớp***:
  - Có thể tạo các phương thức lớp bằng cách sử dụng ***@classmethod***.
  - Ví dụ: giả sử bạn muốn thêm một hàm tạo thay thế vào lớp ThreeDPoint để có thể nhanh chóng tạo điểm từ bộ dữ liệu hoặc danh sách tọa độ:

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

    @classmethod
    def from_sequence(cls, sequence):
        return cls(*sequence)

    def __str__(self):
        return f"{type(self).__name__}({self.x}, {self.y}, {self.z})"
```

```
print(ThreeDPoint.from_sequence((4, 8, 16)))
point = ThreeDPoint(7, 14, 21)
print(point.from_sequence((3, 6, 9)))
print(point)
```

```
ThreeDPoint(4, 8, 16)
ThreeDPoint(3, 6, 9)
ThreeDPoint(7, 14, 21)
```

Phương thức (Methods)

- ***Phương thức tĩnh***:
  - Các lớp cũng có thể có các phương thức tĩnh sử dụng ***@staticmethod***. Các phương thức này không lấy đối tượng hoặc lớp làm đối số.
  - Ví dụ: Bổ sung phương thức tĩnh vào lớp ThreeDPoint

```
@staticmethod
def show_intro_message(name):
    print(f"Hey {name}! This is your 3D Point!")
```

  - Phương thức tĩnh hoạt động như các hàm độc lập được bao bọc trong một lớp.

BÀI TẬP 5.1

Viết chương trình hướng đối tượng bằng Python thực hiện các yêu cầu sau

- Tạo một lớp Nguoi gồm:
  - Thuộc tính: tên, tuổi, cân nặng, chiều cao
  - Phương thức:
    - __init__
    - __str__
    - Chúc mừng sinh nhật (tuổi của đối tượng tăng lên 1)
- Khởi tạo một đối tượng thuộc lớp Nguoi. In ra màn hình thông tin của người đó. Chúc mừng sinh nhật đối tượng và in lại thông tin ra màn hình.

BÀI TẬP 5.1

```
class Person:
    def __init__(self, name, age, height, weight):
        self.name = name
        self.age = age
        self.height = height
        self.weight = weight

    def __str__(self):
        return (f"Xin chào, tôi là {self.name} và tôi {self.age} tuổi. "
                f"Tôi cao {self.height}cm, nặng {self.weight}kg.")

    def celebrate_birthday(self):
        self.age += 1
        print("Chúc mừng sinh nhật! Tôi đã", self.age, "tuổi.")
```

```
# Tạo một đối tượng Person mới
person1 = Person("John", 30)

# Gọi phương thức introduce()
print(person1)

# Gọi phương thức celebrate_birthday() để tăng tuổi
person1.celebrate_birthday()

# Kiểm tra xem tuổi đã được tăng chưa
print(person1)
```

KẾT THÚC