Lập trình hướng đối tượng

TS. Nguyễn Tuấn Anh

Trường Đại học Công nghệ Thông tin & Truyền thông

Khoa Công nghệ Thông tin

Tháng 2 năm 2023

Giới thiệu về OOP trong Python

Lập trình hướng đối tượng (OOP: Object-oriented programming) là một kỹ thuật hỗ trợ, cho phép lập trình viên trực tiếp làm việc với các đối tượng mà họ định nghĩa lên.

Lập trình hướng đối tượng là một khái niệm không thể thiếu trong hầu hết các ngôn ngữ thông dụng hiện nay.

Python cũng hỗ trợ lập trình hướng đối tượng với các khái niệm Class, Object, Override...

Ví dụ về Lớp và Đối tượng

- Hãng xe: Toyota
- Màu: Đỏ
- Nguyên liệu: Điện

- Hãng xe: Porsche
- Màu: Xanh
- Nguyên liệu: Gas

- Hãng xe: Lamborghini
- Màu: Vàng
- Nguyên liệu: Deisel

Class Car
- Loại xe
- Tên xe
- Màu sắc
- Nguyên liệu

Các nguyên lý OOP

Trong Python, khái niệm về OOP tuân theo một số nguyên lý cơ bản:

- Tính đóng gói
- Tính kế thừa
- Tính đa hình

Các nguyên lý OOP

Tính kế thừa: cho phép một lớp (class) có thể kế thừa các thuộc tính và phương thức từ các lớp khác đã được định nghĩa.

Tính đóng gói: là quy tắc yêu cầu trạng thái bên trong của một đối tượng được bảo vệ và tránh truy cập được từ bên ngoài.

Tính đa hình: là khái niệm mà hai hoặc nhiều lớp có những phương thức giống nhau nhưng có thể thực thi theo những cách thức khác nhau.

Khai báo một Class

Khai báo một class theo cú pháp sau:

```
class myclass([parentclass]):
    assignments
    def __init__(self):
        statements
    def method():
        statements
    def method2():
        statements
```

Ví dụ về Class và Object:

Ví dụ lớp Con người:

```
class ConNguoi:
    def __init__(self, ten, tuoi, gioiTinh):
        self.ten = ten
        self.tuoi = tuoi
        self.gioi_tinh = gioi_tinh

    def gioiThieu(self):
        print("Ten:", self.ten, ", tuoi: ", self.tuoi, ", giới tính:", self.gioiTi

# Tạo một đối tượng thuộc lớp ConNguoi
nguoi = ConNguoi("An", 30, "Nữ")

# Gọi phương thức gioiThieu
nguoi.gioiThieu()
```

Giải thích

Lớp **ConNguoi** có ba thuộc tính ten, tuoi, gioiTinh

Phương thức **__init__()** được gọi khi một đối tượng **ConNguoi** được khởi tạo để thiết lập các thuộc tính ban đầu.

Phương thức **gioiThieu** để in thông tin của một con người.

Kế thừa

Tính kế thừa cho phép một lớp (class) có thể kế thừa các thuộc tính và phương thức từ các lớp khác đã được định nghĩa. Lớp đã có gọi là lớp cha, lớp mới phát sinh gọi là lớp con.

Lớp con kế thừa tất cả thành phần của lớp cha, có thể mở rộng các thành phần kế thừa và bổ sung thêm các thành phần mới.

Lớp Sinh viên kế thừa từ lớp con người

```
class SinhVien(ConNguoi):
    def __init__(self, ten, tuoi, gioiTinh, lop, mssv):
        super().__init__(ten, tuoi, gioiTinh)
        self.lop = lop
        self.mssv = mssv
    def thongTin(self):
        print("Tên:", self.ten)
        print("Tuổi:", self.tuoi)
        print("Giới tính:", self.gioiTinh)
        print("Lớp:", self.lop)
        print("MSSV:", self.mssv)
# Tạo một đối tượng thuộc lớp SinhVien
sinhVien = SinhVien("Linh", 20, "Nữ", "K20", "123456")

# Gọi phương thức gioiThieu kế thừa từ lớp ConNguoi
sinh_vien.gioiThieu()

# Gọi phương thức thongTin của lớp SinhVien
sinh_vien.thongTin()
```

Đóng gói (Encapsulation)

Có thể hạn chế quyền truy cập vào các thuộc tính bên trong của đối tượng.

Điều này ngăn chặn dữ liệu bị sửa đổi trực tiếp, được gọi là đóng gói.

Trong Python, chúng ta biểu thị thuộc tính private này bằng cách sử dụng dấu gạch dưới làm tiền tố: "_" hoặc "__".

Lớp Sinh viên kế thừa từ lớp con người

```
class SinhVien:
    def __init__(self, ten, mssv):
        self.ten = ten
        self.__mssv = mssv  # dấu gạch chân đứng trước tên biến là biến private
    def getMssv(self):
        return self.__mssv
    def setMssv(self, mssv):
        self.__mssv = mssv
# Tạo một đối tượng thuộc lớp SinhVien
sinhVien = SinhVien("Dũng", "123456")

# Đặt giá trị mới cho thuộc tính mssv bằng phương thức setter
sinhVien.setMssv("654321")

# Truy cập lại thuộc tính mssv bằng phương thức getter
print("MSSV mới của sinh viên là:", sinhVien.getMssv())
```

Đa hình (Polymorphism)

Tính đa hình là khái niệm mà hai hoặc nhiều lớp có những phương thức giống nhau nhưng có thể thực thi theo những cách thức khác nhau.

Giả sử, chúng ta cần tô màu một hình khối, có rất nhiều lựa chọn cho hình của bạn như hình chữ nhật, hình vuông, hình tròn.

Tuy nhiên, bạn có thể sử dụng cùng một phương pháp để tô màu bất kỳ hình dạng nào.

Đa hình

```
class ConNguoi:
    def __init__(self, ten):
        self.ten = ten
    def thongTin(self):
        print("Tên của người này là", self.ten)

class SinhVien(ConNguoi):
    def __init__(self, ten, mssv):
        super().__init__(ten)
        self.mssv = mssv
    def thongTin(self):
        print("Tên của sinh viên này là", self.ten)
        print("MSSV của sinh viên này là", self.mssv)
```

Đa hình

```
# Tạo một đối tượng thuộc lớp ConNguoi
cn = ConNguoi("Nguyễn Văn A")


# Tạo một đối tượng thuộc lớp SinhVien
sinhVien = SinhVien("Bình", "123456")

# Gọi phương thức thông tin từ đối tượng con người
print("----Gọi phương thức thông tin của đối tượng Con người")
cn.thongTin()

print("----Gọi phương thức thông tin của đối tượng sinh viên")
# Gọi phương thức thông tin từ đối tượng sinh viên
sinhVien.thongTin()
```

Đa hình

```
----Gọi phương thức thông tin của đối tượng Con người
Tên của người này là Nguyễn Văn A
----Gọi phương thức thông tin của đối tượng sinh viên
Tên của sinh viên này là Bình
MSSV của sinh viên này là 123456
----Gọi phương thức thông tin của đối tượng Con người
Tên của sinh viên này là Bình
MSSV của sinh viên này là 123456
----Gọi phương thức thông tin của đối tượng sinh viên
Tên của sinh viên này là Bình
MSSV của sinh viên này là 123456
```