Bài: Các phương thức lớp trong lập trình hướng đối tượng với Python

Xem bài học trên website để ủng hộ Kteam: Các phương thức lớp trong lập trình hướng đối tượng với Python

Mọi vấn đề về lỗi website làm ảnh hưởng đến bạn hoặc thắc mắc, mong muốn khóa học mới, nhằm hỗ trợ cải thiện Website. Các bạn vui lòng phản hồi đến Fanpage How Kteam nhé!

Dẫn nhập

Ở các bài trước, ta đã biết được cách TẠO & SỬ DỤNG PHƯƠNG THỨC CỦA LỚP. Đặc điểm chung của các phương thức đó là luôn sẽ có tối thiểu một **argument** được gửi vào đó chính là đối đượng gọi phương thức.

Thông thường **parameter** được nhận nhiệm vụ nhân **argument** đó ta sẽ đặt là **self**. Những phương thức mà có mặc định **parameter self** người ta gọi đó là những **regular method** (phương thức thường).

Còn ở bài hôm nay, các bạn biết thêm về hai loại phương thức nữa đó chính là **class method và static method**.

Nội dung

Để theo dõi bài này một cách tốt nhất, bạn nên có có những kiến thức cơ bản về Python trong khóa LẬP TRÌNH PYTHON CƠ BẢN

Nếu bạn chưa có thời gian để học hết khóa trên thì hãy đảm bảo đã tìm hiểu những kiến thức sau đây

- BIẾN và CÁC KIỂU DỮ LIỆU CƠ BẢN của Python (Số, chuỗi, List, Tuple, Dict, Set, Range)
- Một số toán tử cơ bản (+, -, *, /, %)
- Khối lệnh điều kiện Khối vòng lặp như VÒNG LẶP FOR, VÒNG LẶP IF)
- HÀM

Và đương nhiên để học tiếp bài sau, bạn phải nắm vững kiến thức ở các bài trước:

- LỚP & ĐỐI TƯỢNG TRONG LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG PYTHON
- KHAI BÁO THUỘC TÍNH LỚP TRONG LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG PYTHON

Bạn và Kteam sẽ cùng tìm hiểu những nội dụng sau đây

- Class method
- Static method

Class method

Nếu **regular method** có **argument** đầu tiên tự động đưa vào là đối tượng đó và được nhận bởi **parameter self** thì ở **class method**, **argument** đầu tiên tự động đưa vào chính lớp gọi phương thức đó hoặc là lớp của đối tượng gọi phương thức đó.

Theo quy ước thì **parameter** nhận **argument** này sẽ là **cls**.

Bạn còn nhớ ví dụ này ở bài trước chứ?

:

```
class SieuNhan:
    suc_manh = 50
    def __init__(self, para_ten, para_vu_khi, para_mau_sac):
        self.ten = para_ten
        self.vu_khi = para_vu_khi
        self.mau_sac = para_mau_sac

sieu_nhan_A = SieuNhan("Sieu nhan do", "Kiem", "Do")

SieuNhan.suc_manh = 40
```

Như bạn biết thì khi làm như vậy, thuộc tính ở lớp cũng như tất cả các đối tượng thuộc lớp đó sẽ được cập nhật lại với giá trị mới.

Tuy nhiên, cách này thường không đường sử dụng, mà thay vào đó họ sẽ sử dụng **class method**. Sử dụng làm sao thì mời bạn đọc đến với ví dụ sau:

:

```
class SieuNhan:
    suc_manh = 50
    def __init__(self, para_ten, para_vu_khi, para_mau_sac):
        self.ten = para_ten
        self.vu_khi = para_vu_khi
        self.mau_sac = para_mau_sac
    @classmethod
    def cap_nhat_suc_manh(cls, smanh):
        cls.suc_manh = smanh
sieu_nhan_A = SieuNhan("Sieu nhan do", "Kiem", "Do")

print(SieuNhan.suc_manh)
print(sieu_nhan_A.suc_manh)

SieuNhan.cap_nhat_suc_manh(40)

print(SieuNhan.suc_manh)
print(sieu_nhan_A.suc_manh)
```

Kết quả:

:

```
50
50
40
40
```

Để Python biết được phương thức nào là **class method** thì bạn thên **@classmethod** ngay trên dòng khai báo hàm. Và như đã nói, mặc định sẽ luôn có một argument được gửi vào đó chính là lớp gọi phương thức đó.

Và dĩ nhiên cũng có thể là lớp của đối tượng gọi phương thức đó

:

```
class SieuNhan:
    suc_manh = 50
    def __init__(self, para_ten, para_vu_khi, para_mau_sac):
        self.ten = para_ten
        self.vu_khi = para_vu_khi
        self.mau_sac = para_mau_sac
    @classmethod
    def cap_nhat_suc_manh(cls, smanh):
        cls.suc_manh = smanh
sieu_nhan_A = SieuNhan("Sieu nhan do", "Kiem", "Do")

print(SieuNhan.suc_manh)
print(sieu_nhan_A.suc_manh)

sieu_nhan_A.cap_nhat_suc_manh(40) # sử dụng đối tượng thay vì lớp

print(SieuNhan.suc_manh)
print(sieu_nhan_A.suc_manh)
```

Kết quả:

:

```
50
50
40
40
```

Tuy nhiên, đây không phải là ứng dụng chính của **class method**. **Class method** thường được dùng để tạo đối tượng.

Đặt vấn đề, ta muốn khởi tạo một siêu nhân, tuy nhiên một số siêu nhân lại có các thông tin không được tường minh rõ ràng mà lại được lưu dưới dạng một list, hay một chuỗi ta có thể xử lí để lấy các thông tin. Và như vậy, bạn thấy rằng ta cần phải có một bước tiền xử lí trước khi ra được các thông tin của một siêu nhân sau đó mới tạo được đối tượng

Giả sử ta phải xử lí các thông tin của siêu nhân là một chuỗi. Các thông tin được nối với nhau bằng một kí tư "**-**".

:

```
>>> s = "Sieu nhan do - Kiem - Do"
```

Ở đây ta cần phải có một chút kiến thức về XỬ LÝ CHUỖI. Đầu tiên ta cần tách bằng kí tự "-", nghĩa ngay tới phương thức **split** của chuỗi

:

```
>>> lst = s.split('-')
>>> lst
['Sieu nhan do ', ' Kiem ', ' Do']
```

Ta được các chuỗi đã tách xong, tuy nhiên một số chuỗi bị dư khoảng trắng, ta dùng phương thức **strip** kết hợp **list comprehension**

:

```
>>> new_lst = [st.strip() for st in lst]
>>> new_lst
['Sieu nhan do', 'Kiem', 'Do']
```

Và việc cuối cùng là gán các thông tin sau khi xử lí trong cho các biến

:

```
>>> ten, vu_khi, mau_sac
('Sieu nhan do', 'Kiem', 'Do')
```

Giờ ta chỉ cần áp dụng **class method** xử lí của chúng ta trong lớp nào

:

```
class SieuNhan:
    suc_manh = 50
    def __init__(self, para_ten, para_vu_khi, para_mau_sac):
        self.ten = para_ten
        self.vu_khi = para_vu_khi
        self.mau_sac = para_mau_sac
    @classmethod
    def from_string(cls, s): # thường những phương thức xử lí thế này hay có tên là from…
        lst = s.split('-')
        new_lst = [st.strip() for st in lst]
        ten, vu_khi, mau_sac = new_lst
        return cls(ten, vu_khi, mau_sac)

infor_str = "Sieu nhan do - Kiem - Do"
sieu_nhan_A = SieuNhan.from_string(infor_str)
print(sieu_nhan_A.__dict__)
```

Kết quả:

:

```
{'ten': 'Sieu nhan do', 'vu_khi': 'Kiem', 'mau_sac': 'Do'}
```

Như bạn thấy, đối tượng sieu_nhan_A giờ đây không còn được khởi tạo theo cách thông thường mà giờ đây lớp **SieuNhan** qua một bức xử lí các thông tin, sau đó khởi tạo một đối tượng ngay trong phương thức, rồi mới trả về lại cho sieu_nhan_A.

Static method

Regular method được ngầm gửi vào **argument** là chính đối tượng gọi phương thức và ta sử dụng **parameter self** để xử lí những vấn đề khác, **class method** được ngầm gửi vào **argument** là chính **class** gọi phương thức và ta sử dụng **parameter cls** để xử lí những vấn đề khác thì **static method** chẳng ngầm gửi cái gì vào cả, nó như một hàm bình thường.

Câu hỏi ở đây là, ta chẳng cần tạo **static method** mà tạo luôn hàm ở ngoài lớp được không? Chẳng vấn đề gì. Tuy nhiên **static method** vẫn tồn tại vì đôi lúc ta cần sự khoa học, logic, một số phương thức chẳng có sử dụng tí gì tới những thông tin của đối tượng thuộc lớp đó tuy nhiên nó vẫn có gì đó liên quan nên vẫn được đặt ở trong lớp đó.

Bạn đọc xem ví dụ sau đây:

:

```
class SieuNhan:
    suc_manh = 50
    def __init__(self, para_ten, para_vu_khi, para_mau_sac):
        self.ten = para_ten
        self.vu_khi = para_vu_khi
        self.mau_sac = para_mau_sac
    @staticmethod
    def bien_hinh():
        print("1, 2, 3. Sieu nhan bien hinh")

sieu_nhan_A = SieuNhan("Sieu nhan do", "Kiem", "Do")
sieu_nhan_A.bien_hinh()
```

Kết quả:

:

```
1, 2, 3. Sieu nhan bien hinh
```

Nếu bạn vẫn phân vân khi nào dùng **regular method**, **class method**, **static method** thì bạn chỉ cần nhớ thế này:

- Nếu bạn dựng một phương thức cần sử dụng đối tượng đó thì dùng **regular method**
- Nếu bạn cần dùng class thì dùng **class method**
- Trường hợp còn lại (tức là không dùng gì) thì dùng **static method**

> **Lưu ý:** Mình thấy một số bạn luôn luôn dùng **regular method** cho lớp, lúc nào cũng có một **parameter self** cho dù trong phương thức đó chẳng bao giờ sử dụng tới. Điều này không sai, tuy nhiên khi đọc code sẽ thấy sự thiếu chuyên nghiệp.

Kết luận

Bài này đã giúp bạn nắm được rõ hơn về các loại phương thức trong lớp

Ở bài tiếp theo, ta sẽ tìm hiểu về TÍNH KẾ THỪA TRONG LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG PYTHON.

Cảm ơn các bạn đã theo dõi bài viết. Hãy để lại bình luận hoặc góp ý của mình để phát triển bài viết tốt hơn. Đừng quên "**Luyện tập – Thử thách – Không ngại khó**"