Bài: Setters, Getters và Deleters trong lập trình hướng đối tượng Python

Xem bài học trên website để ủng hộ Kteam: Setters, Getters và Deleters trong lập trình hướng đối tượng 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, bạn đọc đã tìm hiểu về CÁC PHƯƠNG THỨC ĐẶC BIỆT TRONG LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG PYTHON.

Còn ở bài này, bạn đọc sẽ biết tới một số những phương thức nữa, nhưng những phương thức này không hẳn gọi là phương thức. Đó là **Setters, Getters, Deleters**

Để cho bài này được giải thích một cách đơn giản. Bạn đọc tạo sẵn một **class** mới thay vì **class** Siêu Nhân mà chúng ta đã làm việc trong suốt series.

Python:

```
class Kter:
    def __init__(self, ho, ten):
        self.ho = ho
        self.ten = ten
        self.email = ho + '-' + ten + '@kteam.com'
```

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
- CÁC PHƯƠNG THỨC LỚP TRONG LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG PYTHON
- TẠO LỚP KẾ THỪA TRONG LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG VỚI PYTHON
- PHƯƠNG THỨC ĐẶC BIỆT TRONG LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG PYTHON

Trong bài này, Bạn và Kteam sẽ cùng tìm hiểu những nội dụng sau đây

- Getter
- Setter
- Deleter

Getter

Giờ ta tạo một đối tượng như sau

Python:

```
kter_A = Kter("Tran", "Long")
```

Ở lớp Kter, ta viết thêm một **regular method**

Python:

```
def ho_va_ten(self):
    return '{} {}'.format(self.ho, self.ten)
```

Sau đó ta thử in một số thứ cơ bản

Python:

```
print(kter_A.ho)
print(kter_A.ten)
print(kter_A.email)
print(kter_A.ho_va_ten())
```

Kết quả:

Python:

```
Tran
Long
Tran-Long@kteam.com
Tran Long
```

Giờ là lúc mà ta nghịch ngợm một tí, bằng cách trước khi in ra những thông tin, ta thử tay đổi họ và tên của đối tượng **kter** ta vừa tạo.

Python:

```
kter_A.ho = "Nguyen"
kter_A.ten = "Giau"

print(kter_A.ho)
print(kter_A.ten)
print(kter_A.email)
print(kter_A.ho_va_ten())
```

Kết quả:

Python:

```
Nguyen
Giau
Tran-Long@kteam.com
Nguyen Giau
```

Bạn đọc để ý, sau khi thay đổi lại họ và tên thì phương thức **ho_va_ten** trả lại họ và tên đúng như sau khi đã được thay đổi. Tuy nhiên về **email** thì lại không phải thế. Và dĩ nhiên rồi, bạn muốn nó thay đổi thì nó cũng phải được đổi lại theo bằng "cách thủ công".

Bạn muốn **email** cũng được thay đổi như **ho_va_ten**? Vậy thì ta viết cho nó một phương thức như **ho_va_ten:**

Python:

```
def email(self):
    return self.ho + '-' + self.ten + '@kteam.com'
```

Dĩ nhiên khi có phương thức này rồi thì bạn nhớ xóa thuộc tính **email** được gán trong hàm **constructor**

Khi ta chạy đoạn code:

Python:

```
kter_A.ho = "Nguyen"
kter_A.ten = "Giau"

print(kter_A.ho)
print(kter_A.ten)
print(kter_A.email())
print(kter_A.ho_va_ten())
```

Kết quả:

Python:

```
Nguyen
Giau
Nguyen-Giau@kteam.com
Nguyen Giau
```

Nhưng bạn thấy một điều là, bản chất nhìn vào nó là một phương thức, thật không đúng với cái ban đầu ta muốn rằng nó là một thuộc tính. Để nó trở thành một thuộc tính, thì ta làm một thao tác như tạo **classmethod**. Bạn đọc xem ví dụ:

Python:

```
@property
def email(self):
    return self.ho + '-' + self.ten + '@kteam.com'
@property
def ho_va_ten(self):
    return '{} {}'.format(self.ho, self.ten)
```

Giờ đây, ta đã có hai thuộc tính là **email** và **ho_va_ten**.

Python:

```
kter_A.ho = "Nguyen"
kter_A.ten = "Giau"

print(kter_A.ho)
print(kter_A.ten)
print(kter_A.email) # thuoc tinh nen khong can () nhu luc nay
print(kter_A.ho_va_ten
```

Kết quả:

Python:

```
Nguyen
Giau
Nguyen-Giau@kteam.com
Nguyen Giau
```

Setter

Nãy ta có một thuộc tính **ho_va_ten** sau khi sử dụng **getter**. Vậy giả sử ta muốn gán lại **ho_va_ten** thì chuyện gì xảy ra?

Python:

```
class Kter:
    def __init__(self, ho, ten):
        self.ho = ho
        self.ten = ten
    @property
    def email(self):
        return self.ho + '-' + self.ten + '@kteam.com'
    @property
    def ho_va_ten(self):
        return '{} {}'.format(self.ho, self.ten)

kter_A = Kter("Tran", "Long")

kter_A.ho_va_ten = "Nguyen Giau"
```

Và kết quả là có lỗi vì nó mâu thuẫn với thuộc tính **ho_va_ten** của chúng ta. Vì sao mâu thuẫn? Bản chất **ho_va_ten** là một phương thức, không thể gán cho một giá trị ngang như vậy được. Vì lí do đó, **setter** đã được sinh ra, chúng ta hãy thêm phương thức này vào lớp:

Python:

```
class Kter:
    def __init__(self, ho, ten):
        self.ho = ho
        self.ten = ten
    @property
    def email(self):
        return self.ho + '-' + self.ten + '@kteam.com'
    @property
    def ho_va_ten(self):
        return '{} {}'.format(self.ho, self.ten)
    @ho_va_ten.setter
    def ho_va_ten(self, ten_moi):
        ho_moi, ten_moi = ten_moi.split(' ')
        self.ho = ho_moi
        self.ten = ten_moi

kter_A = Kter("Tran", "Long")

kter_A.ho_va_ten = "Nguyen Giau" # day la argument cho parameter ten_moi

print(kter_A.ho_va_ten)
```

Kết quả:

Python:

```
Nguyen Giau
```

Deleter

Nếu bạn muốn dùng xong xóa, thì **deleter** là công cụ giúp bạn xóa những thuộc tính bạn vừa gán. **Xóa ở đây là việc gán một giá trị rác** (giá trị không có ý nghĩa) **cho cái chúng ta muốn xóa**. Và thường trong Python giá trị đó được sử dụng là **None** (đôi lúc là 0, hoặc là một list rỗng, tùy từng thuộc tính mà ta muốn xóa)

Cú pháp gần như tương tự với **setter** (**deleter** không sử dụng **parameter** nào khác ngoài **parameter self**). Ta thêm một **deleter** vào một lớp và sử dụng **deleter** như sau.

Mời bạn đọc xem ví dụ:

Python:

```
class Kter:
    def __init__(self, ho, ten):
        self.ho = ho
        self.ten = ten
    @property
    def email(self):
        return self.ho + '-' + self.ten + '@kteam.com'
    @property
    def ho_va_ten(self):
        return '{} {}'.format(self.ho, self.ten)
    @ho_va_ten.setter
    def ho_va_ten(self, ten_moi):
        ho_moi, ten_moi = ten_moi.split(' ')
        self.ho = ho_moi
        self.ten = ten_moi
    @ho_va_ten.deleter
    def ho_va_ten(self):
        self.ho = None
        self.ten = None
        print('Da xoa')

kter_A = Kter("Tran", "Long")

kter_A.ho_va_ten = "Nguyen Giau"

print(kter_A.ho_va_ten)

del kter_A.ho_va_ten

print(kter_A.ho)
print(kter_A.ten)
```

Kết quả:

Python:

```
Nguyen Giau
Da xoa
None
None
```

Kết luận

Bài này đã giúp bạn đọc tìm hiểu về getter, setter và deleter của một lớp

Như vậy chúng ta đã kết thúc series LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG PYTHON. Đây là những kiến thức nền tảng cơ bản (tuy không quá chuyên sâu) mà nhờ vào đó các bạn có thể tiếp tục học các model của Python một cách dễ dàng. Nếu bạn đọc có hứng thú về **class** trong **Python** nhiều thì có thể tìm hiểu siêu lớp (**meta class**) ở một số tài liệu trên mạng.

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**"