[Hindi] Property Decorators, Setters & Deleters? | Object Oriented Programming Using Python #9

What are Decorators?

A decorator is the design pattern that adds additional responsibilities to an object dynamically. It's used to add additional features to the function without modifying it.

To continue learning about what decorators are, consider the following snippet of code.

```
class Employee:

    def __init__(self,fname,lname,salary):
        self.fname=fname
        self.lname=lname
        self.salary=salary
        self.increment=1.4

    @classmethod
    def change_increment(cls,amount):
        cls.increment = amount
```

```
if __name__ == '__main__':
    harry = Employee("harry", "jackson", 9400)
    rohan = Employee("rohan", "aagarwal", 9)
```

In this code snippet, **harry** and **rohan** are the objects of class **Employee.**
There are different types of decorators such as Property, setter, deletor, and many more...

Let's elaborate on some of these decorators using our code snippet.

1. **Property Decorator**

Property Decorator just means that by initializing **@property** before our method, we force the method to *behave like an* ***attribute***.
Let's work with the property Decorator:
In the above code block, suppose we want an email from the object.
This can be achieved by creating some other instance variable or method within the class. As follows:

```
self.email = self.fname +“.”+ self.lname + "@gmail.com"
```

Note: Email is in the format: fname.lname@gmail.com
But, as the email is dynamically created using fname and lname.
It will be mind-tempting to customize the email field according to our desired need with all its dependencies.
To perform this very well **property decorator** can be taken into consideration.
You only need to code **@property** prior to the class function.

```
@property()
        def email(self):
        return self.fname + "." + self.lname + "@gmail.com"
```

As in this case, our method has become an attribute.
Let's try to change the **lname** of the object dynamically.

```
print(rohan.email)
    rohan.lname ="khanna" # dynamically changing the lname
```

```
print( rohan.email)
```

Output:

```
rohan.aagarwal@gmail.com
rohan.aagarwal@gmail.com
```

It seems that here our email declaration did not function properly. We thought it was going to be "**rohan.khanna@gmail.com**", but the email hasn't changed.

2. **To achieve this, we will use a setter.**

Setter means setting the value.
In the following method, setter is performed for the Employee class.

```
# setting the email
    @email.setter
    def email(self,given_email):
        self.email = given_email
```

Note: given_email is our provided argument.
Here, as setter directly defines or changes the email, this will prompt us with an error, as the email is derived from fname and lname.
To pursue the solution. Since the email provided is in the specified format, we can retrieve fname and lname separately in our setter method and can be used for dynamical allocation. The **split** functionality of Python can be considered.

Creating setter:

```
@email.setter
    def email(self,given_email):
        # self.email = given_email
        change_email = given_email.split("@")[0].split(".")[0]
        self.fname= change_email[0]  # as split return a list
        self.lname =change_email[1]
```

This function simply takes email as its argument. After the split function is implemented, it declares fname and lname as shown.
Getting the output:

```
rohan.email= "khanna.rohan@gmail.com"
print(rohan.email)
```

Output:

```
khanna.rohan@gmail.com
```

Here we get our desired output.

3. **Deletor Decorator**

We also have a delete decorator that is used to **remove** the value assigned by the **setter**.
Here, the value assigned by the setter is **email**.
Let's try and remove the email field.
To pursue this, we will create a **deletor decorator**

```
@email.deleter
    def email(self):
        self.fname=None
        self.lname =None
```

Note: None is used here because in OOP we generally initialize the object with none rather than deleting it.

```
del rohan.email
            print(rohan.email)
```

No email field will be printed.

That's it for this article.
Keep coding :)

Previous

Next

CodeWithHarry