Python Programming

Property Decorator

Mostafa S. Ibrahim
Teaching, Training and Coaching since more than a decade!

Artificial Intelligence & Computer Vision Researcher
PhD from Simon Fraser University - Canada
Bachelor / Msc from Cairo University - Egypt
Ex-(Software Engineer / ICPC World Finalist)

Issues in the property class

```
class Person:
    def __init__(self, full_name):
        # DRY Principle: DON'T repeat yourself!
        self.set_full_name(full_name)

    def get_full_name(self):
        return f'{self.first_name} {self.last_name}'

    def set_full_name(self, full_name):
        self.first_name, self.last_name = full_name.lower().split()

    # Create property object
    # On class level. No self.
    full_name = property(get_full_name, set_full_name)  # NOT set_full_name()
```

Issues in the property class

- You typically have to call the set method from __init__
  - To do some common verifications / changes
  - E.g. if salary, make sure it is > 0
- The outsiders now see 2 ways to change a variable
  - Bad design. There should be one way!
    - We may mangle to reduce the issue
- The elegant and recommended way is Property Decorator
  - We haven't study decorators yet. We will later

Property Decorator

```python
class Person:
    def __init__(self, full_name):
        self.full_name = full_name

    @property
    def full_name(self):
        return f'{self.first_name} {self.last_name}'

    @full_name.setter
    def full_name(self, value):
        self.first_name, self.last_name = value.lower().split()

def f1():
    person = Person('Mostafa Saad')
    # Now can see some attribute named full_name
    print(person.full_name)          # calls get
    person.full_name = 'Hello world'    # calls set
```

Set calls itself forever! … Be careful!

```
3   class Person:
4       def __init__(self, salary):
5           self.salary = salary    # calls set
6
7       @property
8       def salary(self):
9           return self.salary
10
11      @salary.setter
12      def salary(self, value):
13          if value < 0:
14              value = 0
15          self.salary = value # calls salary.setter again for ever!
16
17  def f1():
18      person = Person(100)
19  #   print(person.salary)
20      # person.salary = -200
21      # print(person.salary)
```

Get calls itself forever!

```python
class Person:
    def __init__(self, salary):
        self.__salary = salary    # calls set

    @property
    def salary(self):
        return self.salary    # calls salary.getter again for ever!

    @salary.setter
    def salary(self, value):
        if value < 0:
            value = 0
        self.__salary = value #

def f1():
    person = Person(100)
    print(person.salary)
    # person.salary = -200
    # print(person.salary)
```

Proper way

- You typically need a different variable name
  - In general, you need to make sure no cycles in calling
- Note: Recursive issue is same with Property class
- Observe: Getter Property with mangled name might also share intentions
  - Get, but don't set
  - Or provide controlled set/get

```
3   class Person:
4       def __init__(self, salary):
5           self.__salary = salary
6
7       @property
8       def salary(self):
9           return self.__salary
10
11      @salary.setter
12      def salary(self, value):
13          if value < 0:
14              value = 0
15          self.__salary = value
16
17  def f1():
18      person = Person(100)
19      print(person.salary)      # 100
20      person.salary = -200
21      print(person.salary)      # 0
```

Background

- The setter/getter methods are fundamental functions in C++/Java
- Why?
  - As these languages hides data as possible in their private section
  - Then users use the set/get to get info about the properties
  - This results in many written methods to just get/set
  - And a lot of debate about setters/getters being evil
- Python
  - By default we make things public
  - Typically no getters/setters
  - Have a good reason? Use Property Decorator, the most pythonic treatment

"Acquire knowledge and impart it to the people."

"Seek knowledge from the Cradle to the Grave."