Describing Descriptors

EuroPython 2023

by Rodrigo Girão Serrão

Rodrigo 🐍🚀

🐦 @mathsppblog

Rodrigo 🐍🚀

@mathsppblog

Textualize

mathspp.com

Slides:

github.com/mathspp/talks

Rules

1. Ask questions

1. Ask questions
2. Answer *my* questions

1. Ask questions
2. Answer *my* questions
3. OK to interrupt

1. Ask questions
2. Answer *my* questions
3. OK to interrupt
4. Write the code

1. Ask questions
2. Answer *my* questions
3. OK to interrupt
4. Write the code
5. Laugh at my jokes

Attributes

```python
class Person:
    def __init__(self, first, last):
        self.first = first
        self.last = last
        self.name = f"{first} {last}"
```

```python
class Person:
    def __init__(self, first, last):
        self.first = first
        self.last = last
        self.name = f"{first} {last}"
```

Attribute `name` depends on other attributes.

Dynamic attribute computation

Fix this with `@property`

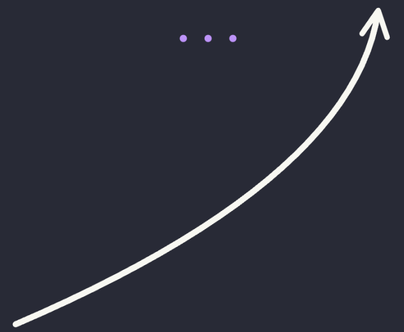
`@property` “turns methods into attributes.”

Your first descriptor

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

    @property
    def name(self):
        return f"{self.first} {self.last}"
```

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

    @property
    def name(self):
        return f"{self.first} {self.last}"
```

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

class NameDescriptor:
    ...
```



We create a class for the descriptor.

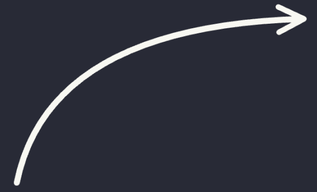
```python
class Person:
    def __init__(self, first, last):
        self.first = first
        self.last = last

    @property
    def name(self):
        return f"{self.first} {self.last}"
```

```python
class Person:
    name = NameDescriptor()

    def __init__(self, first, last):
        self.first = first
        self.last = last

class NameDescriptor:
    ...
```

We assign the descriptor to the class attribute `name`.

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

    @property
    def name(self):
        return f"{self.first} {self.last}"
```

```python
class Person:
    name = NameDescriptor()

    def __init__(self, first, last):
        self.first = first
        self.last = last

class NameDescriptor:
    def __get__(self, person, cls):
        ...
```



The dunder method `__get__` is called

when we try to “get” the attribute...

```
class Person:
    def __init__(self, first, last):
        self.first = first
        self.last = last

    @property
    def name(self):
        return f"{self.first} {self.last}"
```

```
class Person:
    name = NameDescriptor()

    def __init__(self, first, last):
        self.first = first
        self.last = last

class NameDescriptor:
    def __get__(self, person, cls):
        return f"{person.first} {person.last}"
```

... so it must return the computed value.

```
class Person:
    def __init__(self, first, last):
        self.first = first
        self.last = last

    @property
    def name(self):
        return f"{self.first} {self.last}"
```

```
class Person:
    name = NameDescriptor()

    def __init__(self, first, last):
        self.first = first
        self.last = last

class NameDescriptor:
    def __get__(self, person, cls):
        return f"{person.first} {person.last}"
```

`__get__` in action

... so it must return the computed value.

Add prints & see the descriptor being used.

Add attributes `birthyear` and `age` to `Person`.

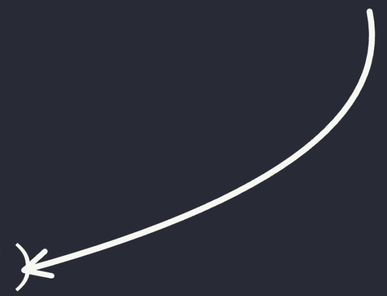
Use a descriptor to dynamically compute the age from the year of birth.

```
import datetime as dt

print(dt.date.today().year)  # 2023
```

The descriptor's `__get__` runs
when we access the attribute.

A more general descriptor

Implement this with properties `r`, `g`, and `b`, depending on the attribute `hex`.



```
>>> red = Colour("#ff0000")
>>> red.r
255
>>> red.g, red.b
(0, 0)
```

Descriptors can be customised with parameters when they're instantiated.

Create a new `FullNamePerson` class. Instances accept the full name.

Create a descriptor that uses an index to access the correct name.

```
class FullNamePerson:
    first = NamePart(0)
    middle = NamePart(1)
    last = NamePart(2)

    ...


person = FullNamePerson(
    "Mary Anne Smith"
)
print(person.first)  # Mary
print(person.middle)  # Anne
print(person.last)  # Smith
```

```
class Person:
    def __init__(self, first, last):
        self.first = first
        self.last = last

    @property
    def full_name(self):
        return f"{self.first} {self.last}"
```

Dynamic attribute assignment

How to use `@property.setter` to the the name in the first `Person` version?

Dynamic attribute assignment with descriptors

How to use a descriptor to set the name in the first version of `Person`?

Use a descriptor to set the `first`, `middle`, and `last` names of `FullNamePerson`.

Read-only attributes with descriptors

Challenges

Challenges:

- add `__set__` to `RGBComponent`
- add `__set__` to `ReadOnlyAttr` (make it error)
- create `LogAttrAccess` descriptor

mathspp.com/pydonts

rodrigo@mathspp.com