It's not magic:

Descriptors exposed

(the descriptors, not us, don't scare)

Joaquín Sorianello
@_joac

Facundo Batista
@facundobatista

Let's play

Meta-play

```
class Strength:

    def break_wall(self, width):
        return self > width * 50

    def jump_hole(self, length):
        return self > length * 10


class Magic:

    def spell(self, resistance):
        return self > resistance
```

```
class Character:

    strength = Strength()
    magic = Magic()

    def __init__(self, strength=0,
                 magic=0):
        self.strength = strength
        self.magic = magic
```

We want to do this

```
>>> gimli = Character(strength=800)
>>> gimli.strength.break_wall(width=20)  # can Gimli break the wall?
False
>>> gimli.strength = 1500
>>> gimli.strength
1500
>>> gimli.strength += 100
>>> gimli.strength
1600
>>> gimli.strength.break_wall(width=20)  # can Gimli on steroids break the wall?
True
>>> gimli.magic.spell(120)  # can Gimli charm a tree?
False
```

And this

```
>>> gandalf = Character(strength=25, magic=100)
>>> gandalf.magic.spell(12)  # can Gandalf the Grey charm a tree?
True
>>> gandalf.magic.spell(300)  # can Gandalf the Grey make Saruman bite the dust?
False
>>> gandalf.magic = 500
>>> gandalf.magic.spell(300)  # can Gandalf the White make Saruman bite the dust?
True
```

In short, we want to be able to do:

```
>>> character.power = 123

>>> character.power

123

>>> character.power.action()

<... something happens ...>
```

It's weird, but...

It's not magic

©Peter Chong

We use Descriptors

RUN, YOU FOOLS!!!

> *“In general, a descriptor is an object attribute with binding behavior, one whose attribute access has been overridden by methods in the descriptor protocol.”*
>
> \- Raymond Hettinger

Wait...

what?!

In simpler words:

We can take control of…

```
>>> someobject.attribute = 42  # set
>>> someobject.attribute       # get
42
>>> del someobject.attribute   # del
```

…and make it to **execute our code**

But how?

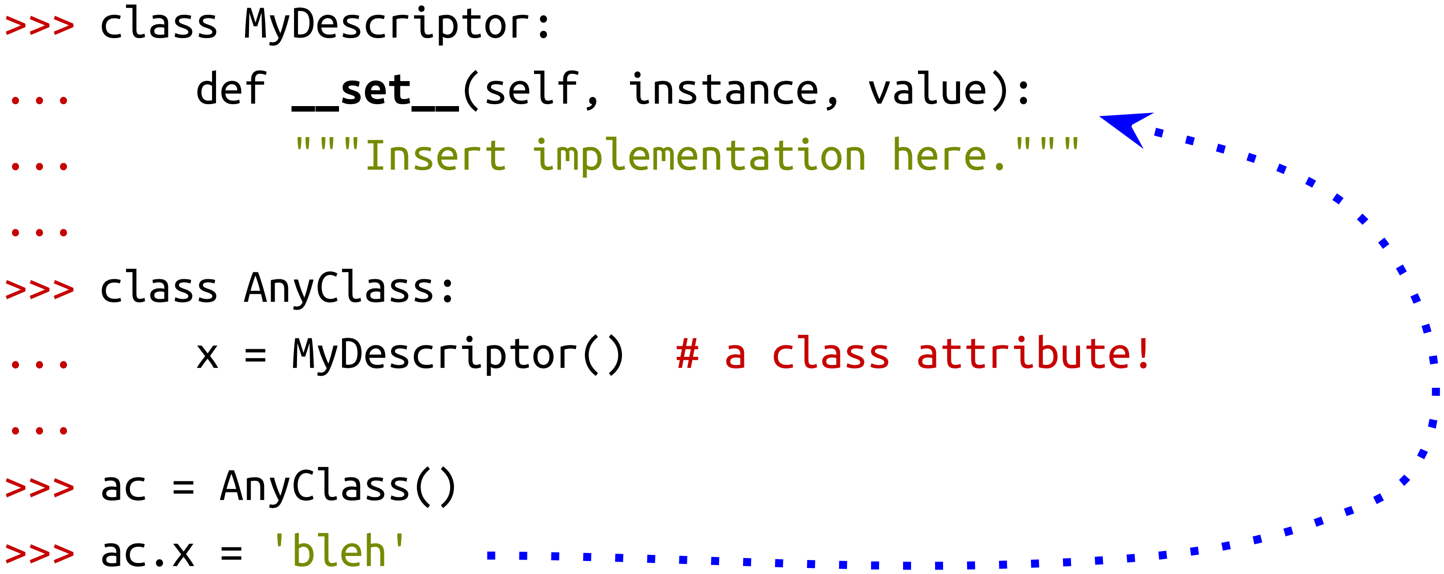
This is a descriptor in its simplest form:

```python
class HelloWorldDescriptor:
    def __get__(self, instance, cls):
        return "Hello World"
```

Using the descriptor

```
>>> class HelloWorldDescriptor:
...     def __get__(self, instance, cls):
...         return "Hello World"
>>>
>>> class AnyClass:
...     x = HelloWorldDescriptor()  # a class attribute!
>>>
>>> ac = AnyClass()
>>> ac.x
"Hello World"
```

Flourishing the idea



```
>>> class MyDescriptor:
...     def __set__(self, instance, value):
...         """Insert implementation here."""
...
>>> class AnyClass:
...     x = MyDescriptor()  # a class attribute!
...
>>> ac = AnyClass()
>>> ac.x = 'bleh'
```

Going for more

```
class Hailer:
    def __get__(self, instance, cls):
        who = instance.__dict__.get(
            'who', 'Unknown')
        return "Hello {}".format(who)

    def __set__(self, instance, value):
        instance.__dict__['who'] = value
```

```
>>> class HelloWorld2:
...     greet = Hailer()
...
>>> hailer = HelloWorld2()
>>> hailer.greet
"Hello Unknown"
>>> hailer.greet = "EuroPython"
>>> hailer.greet
"Hello EuroPython"
```

"There are 10 types of Descriptors: those that understand binary, and those that don't"

- B. B. King

Two types of Descriptors

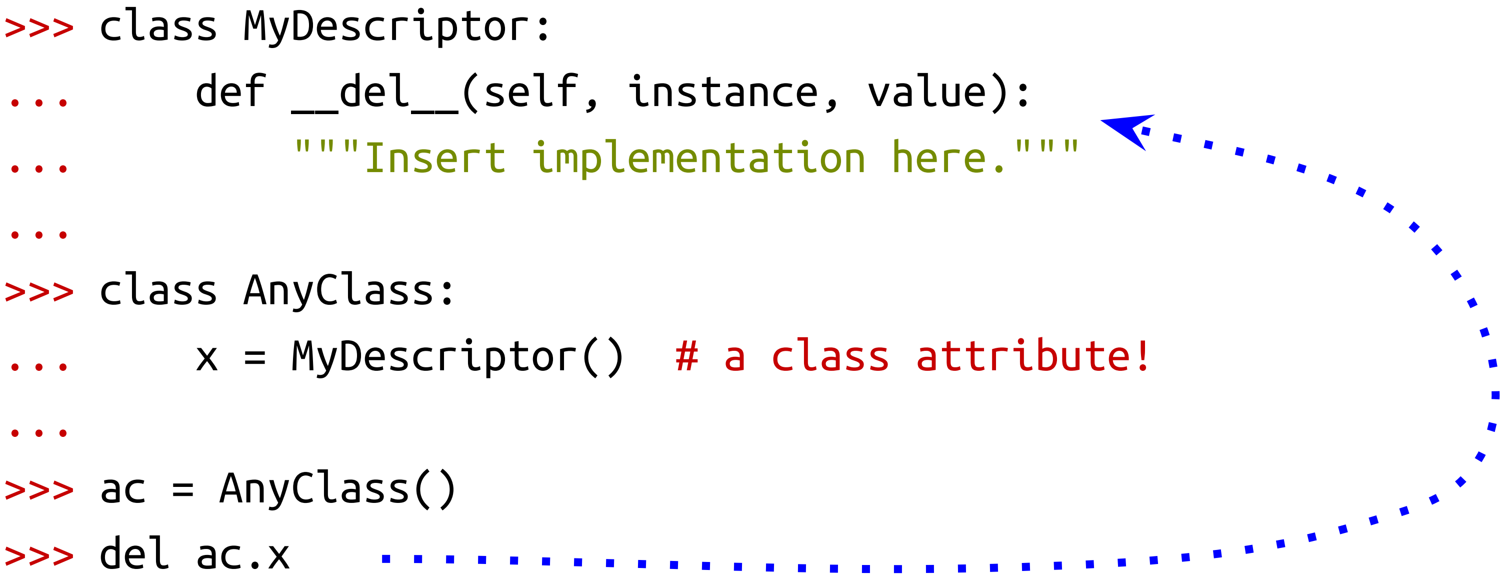
"Overriding" (or "data")

```
>>> class D:
...   def __get__(self, inst, cls):
...     ...
...   def __set__(self, inst, value):
...     ...
...
>>> class C:
...   d = D()
...
>>> c = C()
>>> c.d  # executes the __get__
>>> c.d = 123  # executes the __set__
```

"Non-overriding" (or "non-data")

```
>>> class D:
...   def __get__(self, inst, cls):
...     ...
...
>>> class C:
...   d = D()
...
>>> c = C()
>>> c.d  # executes the __get__
>>> c.d = 123  # overwrote it!!!!
```

For Descriptor API completeness



```
>>> class MyDescriptor:
...     def __del__(self, instance, value):
...         """Insert implementation here."""
...
>>> class AnyClass:
...     x = MyDescriptor()  # a class attribute!
...
>>> ac = AnyClass()
>>> del ac.x
```

"I can do that very same thing with **@property** *and feel sexier"*

- **Brad Pitt**

Let's go back to wizards and dwarves

Remember this?

```python
class Strength:
    def break_wall(self, width):
        ...

class Magic:
    def spell(self, resistance):
        ...

class Character:
    strength = Strength()
    magic = Magic()
    ...
```

```python
>>> gimli = Character(strength=800)
>>> gimli.strength.break_wall(width=20)
False
>>> gimli.strength = 1500
>>> gimli.strength
1500
>>> gandalf = Character(strength=25, magic=100)
>>> gandalf.magic.spell(12)
True
>>> gandalf.magic.spell(300)
False
```

How can we make that work?

"The key of a good offense and a solid defense: descriptors and class decorators."

- **Michael Jordan**

Our descriptor

```
class PowerDescriptor:

    def __init__(self, name, power_class):
        self._name = name
        self._power = power_class

    def __set__(self, instance, value):
        instance.__dict__[self._name] = self._power(value)

    def __get__(self, instance, klass):
        return instance.__dict__[self._name]
```

Convert functionalities

@power takes the class, registers it as "power", and makes it also a "number"

```
@power
class Strength:
    def break_wall(self, width):
        return self > width * 50
    def jump_hole(self, length):
        return self > length * 10

@power
class Magic:
    def spell(self, resistance):
        return self > resistance
```

@character makes class attributes to automagically be descriptors

```
@character
class Character:
    strength = Strength()
    magic = Magic()

    def __init__(self, strength=0, magic=0):
        self.strength = strength
        self.magic = magic
```

INTO THE WILD

Python methods

```python
class Foo:
    def method(self, a, b):
        pass
```

- Python methods are **non-overriding descriptors**
- When you do `foo.method(1, 2)` a ***descriptor*** is executed, that calls our function adding `self`
- Elegant, right?

Django's models and forms fields

```
class Users(models.Model):
    name = models.CharField(...)
```

When you use __slots__

```python
class Point:
    __slots__ = ('x', 'y')

    def __init__(self, x, y):
        self.x = x
        self.y = y
```

Detail: it's not implemented in Python, but uses the descriptors API from C

And in a lot

more places!

Bonus track

Class decorator

KISS: a class decorator is **a function** that **receives a class** and **returns a class**, doing in the middle whatever it wants

It's the same than a function decorator... but for classes :p

Say what?

Normal definition:

```
class Foo:
    pass
```

`Foo` is is the class we defined

With a decorator:

```
@decorator
class Foo:
    pass
```

`Foo` is the class returned by `decorator` (that received the class we defined and did whatever it wanted with it)

Is the same than: `Foo = decorator(Foo)`

How do we use it?

We make powers to also be a `float` and register them

```python
_powers = {}

def power(klass):
    t = type(klass.__name__, (klass, float), {})
    _powers[klass.__name__.lower()] = t
    return t

@power
class Magic:
    def spell(self, resistance):
        return self > resistance
```

How do we use it?

We transform the Character attributes into descriptors

```
def character(klass):
    for name, power_class in _powers.items():
        power_instance = getattr(klass, name, None)
        if power_instance is not None:
             setattr(klass, name,
                     PowerDescriptor(name, power_instance.__class__))
    return cls

@character
class Character:
    strength = Strength()
    magic = Magic()
```

That's all!

It wasn't that hard, right?

role.py

```
_powers = {}


def power(klass):
    t = type(klass.__name__, (klass, float), {})
    _powers[klass.__name__.lower()] = t
    return t


class PowerDescriptor:

    def __init__(self, name, power_class):
        self._name = name
        self._power = power_class

    def __get__(self, instance, klass):
        if instance is None:
            return self
        else:
            return instance.__dict__[self._name]

    def __set__(self, instance, value):
        instance.__dict__[self._name] = self._power(value)


def character(klass):
    for name, power_class in _powers.items():
        power_instance = getattr(klass, name, None)
        if power_instance is not None:
            setattr(klass, name, PowerDescriptor(name, power_instance.__class__))
    return klass
```

example.py

```python
import role


@role.power
class Strength:

    def break_wall(self, width):
        return self > width * 50

    def jump_hole(self, length):
        return self > length * 10


@role.power
class Magic:

    def spell(self, resistance):
        return self > resistance


@role.character
class Character:
    strength = Strength()
    magic = Magic()

    def __init__(self, strength=0, magic=0):
        self.strength = strength
        self.magic = magic


gimli = Character(strength=800)
print("Can Gimli break the wall?", gimli.strength.break_wall(width=20))
gimli.strength = 1500
print("New Gimli strength", gimli.strength)
gimli.strength += 100
print("Newest Gimli strength", gimli.strength)
print("Can Gimli on steroids break the wall?", gimli.strength.break_wall(width=20))
print("Can Gimli charm a tree?", gimli.magic.spell(120))

gandalf = Character(strength=25, magic=100)
print("Can Gandalf the Grey charm a tree?", gandalf.magic.spell(12))
print("Can Gandalf the Grey make Saruman bite the dust?", gandalf.magic.spell(300))
gandalf.magic = 500
print("Can Gandalf the White make Saruman bite the dust?", gandalf.magic.spell(300))
```

Legal stuff

B.B. King, Brad Pitt and Michael Jordan may not have said what we said they said.

Images taken from:

https://en.wikipedia.org/wiki/D20_System#/media/File:Dice_%28typical_role_playing_game_dice%29.jpg

https://commons.wikimedia.org/wiki/File:Adjustable_spanner_20101109.jpg

http://images1.fanpop.com/images/photos/2300000/Map-of-Middle-Earth-lord-of-the-rings-2329809-1600-1200.jpg

https://eulogies4theliving.files.wordpress.com/2012/09/here-i-come-to-save-the-day2.jpg

https://thebrotherhoodofevilgeeks.files.wordpress.com/2013/11/gimli_helms_deep.jpg

http://iliketowastemytime.com/sites/default/files/gandalf-the-grey-hd-wallpaper.jpg

https://thequeentonie.files.wordpress.com/2014/04/magic_mist.jpg

https://s-media-cache-ak0.pinimg.com/originals/3b/ce/ef/3bceef8d4d5b3c6c1d7b5d78f03b08c2.jpg

https://scalegalaxy.files.wordpress.com/2012/08/gimli-11.jpg

http://static.comicvine.com/uploads/original/3/39280/787576-gandalf.jpg

https://upload.wikimedia.org/wikipedia/commons/6/69/NASA-HS201427a-HubbleUltraDeepField2014-20140603.jpg

https://s-media-cache-ak0.pinimg.com/736x/40/95/fc/4095fc068ce7012ee07baf11a8ef3a0f.jpg

http://www.hdwallpapers.in/walls/purple_magenta_flower-wide.jpg

Questions, Answers, etc

(you know how it works)

Joaquín Sorianello
@_joac

Facundo Batista
@facundobatista

slides

http://is.gd/KrFosR