Python2018

compscicenter.ru

aleksey.kladov@gmail.com

Лекция 6

Классы I

Простой класс

```
class Counter:
    """I am a Counter, I count stuff."""

    def __init__(self, initial_count=0):
        self.count = initial_count

    def get(self):
        return self.count

    def increment(self):
        self.count += 1

c = Counter(initial_count=91)
c.increment()
print(c.get())
```

Простой класс

```
class Counter:
    """I am a Counter, I count stuff."""

    def __init__(self, initial_count=0):
        self.count = initial_count

    def get(self):
        return self.count

    def increment(self):
        self.count += 1

c = Counter(initial_count=91)
c.increment()
print(c.get())
```

Атрибуты

```
class Counter:
    all_counters = []  # class attribute

    def __init__(self, initial_count=0):
        Counter.all_counters.append(self)
        # no explicit field declaration
        self.count = initial_count


c1 = Counter(92)
c2 = Counter(62)
assert len(Counter.all_counters) == 2
assert c1.all_counters is c2.all_counters
```

__dict__

```
>>> c = Counter(92)
>>> c.__class__
<class '__main__.Counter'>
>>> c.__dict__
{'count': 92}
>>> c.count == c.__dict__["count"]
True
>>> c.__dict__["foo"] = 62
>>> c.foo
62
>>> del c.foo
>>> del c.__dict__["count"] # ~= .pop("count")
>>> vars(c)  # ~= c.__dict__
{}
```

Класс это объект

```
>>> (Counter.__name__, Counter.__doc__, Counter.__module__)
('Counter', 'I am a Counter.', '__main__')
>>> Counter.__bases__
(<class 'object'>,)
>>> Counter.__dict__
mappingproxy({
    'all_counters': [],
    '__init__': <function Counter.__init__ ...>,
    'get': <function Counter.get ...>,
    'increment': <function Counter.increment ...>,
})
```

Класс это statement

```
>>> class Weird:
...     f1, f2 = 0, 1
...     for _ in range(10):
...         f1, f2 = f2, f1 + f2
...
>>> Weird.f1
55
```

__dict__ класса -- результат выполнения тела класса

Поиск атрибутов

```
>>> class A:
...     x = 92
...
>>> a = A()
>>> vars(a)
{}
>>> a.x
92
>>> a.x = 62
>>> vars(a)
{'x': 62}
>>> a.x
62
>>> A.x
92
```

Поиск атрибутов

- ищем в __dict__ экземпляра
- ищем в __dict__ класса (базовых классов)
- присваиваем в __dict__ экземпляра
- утверждение на слайде -- ложь

Bound Methods

```
>>> class A:
...     def foo(self):
...         pass
...
>>> a = A()
>>> a.foo
<bound method A.foo of <__main__.A object ... >>
>>> A.foo
<function A.foo ...>
>>> a.foo is A.foo
False
```

Bound Methods

```
>>> a.foo()
92
>>> A.foo(a)
92
>>> f = a.foo
>>> g = partial(A.foo, a)
>>> f()
92
>>> g()
92
```

Bound Methods

```
obj.foo(bar)
obj.__class__.foo(obj, bar)
type(obj).foo(bar)
```

Properties

```
class Counter:
    def __init__(self, initial_count=0):
        self.count = initial_count

    def increment(self):
        self.count += 1

    @property
    def is_zero(self):
        return self.count == 0


c = Counter()
assert c.is_zero  # нет `()`
c.increment()
assert not c.is_zero
```

```
class Temperature:
    def __init__(self, *, celsius=0):
        self.celsius = celsius

    @property
    def fahrenheit(self):
        return self.celsius * 9 / 5 + 32

    @fahrenheit.setter
    def fahrenheit(self, value):
        self.celsius = (value - 32) * 5 / 9

    @fahrenheit.deleter
    def fahrenheit(self):
        del self.celsius


c = Temperature()
c.fahrenheit = 451
assert c.celsius == 232.77777777777777
```

Поле? Свойство? Функция?

- геттер -- нет
- поле -- если нет инварианта
- NamedTuple
- свойство -- инвариант, read-only, инструментария
- функция -- side-effects, долгие вычисления

__slots__

```
>>> class A:
...     __slots__ = ["x", "y"]  # экономим память
...
>>> a = A()
>>> a.__dict__
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'A' object has no attribute '__dict__'
>>> a.x = 92
>>> a.z = 92
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'A' object has no attribute 'z'
```

Управление доступом

Соглашения и mangling

```
class A:
    def __init__(self):
        self.pub = 92
        self._priv = 62
        self.__mangled = 42


a = A()
assert a.pub == 92
assert a._priv == 62
assert a._A__mangled == 42
```

Наследование

```
class Counter:
    def __init__(self, initial_count=0):
        self.count = initial_count

    def get(self):
        return self.count


class SquaredCounter(Counter):
    def get(self):
        return super().get() ** 2


c = SquaredCounter(91)
assert c.get() == 8281
```

Наследование

```
assert isinstance(c, Counter)
assert issubclass(SquaredCounter, Counter)
assert issubclass(Counter, (str, object))
```

Наследование

```
class A:
    def f(self):
        print("A")

class B:
    def f(self):
        print("B")

class C(A, B):
    pass

C().f()
# A
```

```
class Base:
    def f(self):
        print("Base")

class A(Base):
    def f(self):
        print("A")
        super().f()  # super is dynamic!

class B(Base):
    def f(self):
        print("B")
        super().f()

class C(A, B):
    pass

C().f()
# A
# B
# Base

assert C.mro() == [C, A, B, Base, object]
```

super caxap

```
class A:
    def foo(self):
        super().foo()
```

super caxap

```
class A:
    def foo(self):
        super(A, self).foo()
```

Mixin

```
class DoublingMixing:  # !!!
    def increment(self):
        super().increment()
        super().increment()


class DoublingCounter(DoublingMixing, Counter):
    pass


c = DoublingCounter()
assert c.count == 0
c.increment()
assert c.count == 2
```

Декораторы

```
def doubling(cls):
    orig_increment = cls.increment
    @functools.wraps(orig_increment)
    def increment(self):
        orig_increment(self)
        orig_increment(self)

    cls.increment = increment
    return cls

@doubling
class DoublingCounter(Counter):
    pass

c = DoublingCounter()
assert c.count == 0
c.increment()
assert c.count == 2
```

“Магические” ✨ методы

```
class Counter:
    def __init__(self, initial_count):
        self.count = initial_count

    def __lt__(self, other):
        return self.count < other.count

    def __eq__(self, other):
        return self.count == other.count


c1 = Counter(62)
c2 = Counter(92)
assert c1 < c2
assert (62).__lt__(92)
assert c2 >= c1  # упадёт, нет __ge__
```

“Магические” ✨ методы

```
class Counter:
    def __init__(self, initial_count):
        self.count = initial_count

    def __repr__(self):
        return "Counter({})".format(self.count)

    def __str__(self):
        return "Counted to {}".format(self.count)


c = Counter(92)
assert str(c) == f"{c}" == "Counted to 92"
assert repr(c) == f"{c!r}" == "Counter(92)"
```

“Магические” ✨ методы

```
class Counter:
    def __init__(self, initial_count):
        self.count = initial_count

    def __format__(self, format_spec):
        if format_spec == "bold":
            return f"**{self.count}**"
        return str(self.count)


c = Counter(92)
assert f"{c:bold}" == "**92**"
```

“Магические” ✨ методы

```
class Counter:
    def __init__(self, initial_count):
        self.count = initial_count

    def __hash__(self):
        # NB: a == b => hash(a) == hash(b)
        return hash(self.count)

    def __eq__(self, other):
        return self.count == other.count


assert len({Counter(92), Counter(92)}) == 1
```

“Магические” ✨ методы

```
class Counter:
    def __init__(self, initial_count):
        self.count = initial_count

    def __bool__(self):
        return self.count > 0


c = Counter(0)

if not c:
    print("empty")
```

“Магические” ✨ методы

```
class Counter:
    def __init__(self, initial_count):
        self.count = initial_count

    def __add__(self, other):
        if not isinstance(other, int):
            return NotImplemented
        return Counter(self.count + other)

    def __radd__(self, other):
        return self + other


c = Counter(0)

assert (c + 1).count == 1
assert (1 + c).count == 1
```

“Магические” ✨ методы

```
class Identity:
    def __call__(self, x):
        return x


assert Identity()(92) == 92
```

“Магические” ✨ методы

```
class n_times:
    def __init__(self, n):
        self.n = n

    def __call__(self, f):
        @functools.wraps(f)
        def inner(*args, **kwargs):
            for _ in range(self.n)
                f(*args, **kwargs)

        return inner
```

“Магические” ✨ методы

```
>>> c = Counter(10)
>>> c = Counter.__call__(10)
>>> c = type(Counter).__call__(Counter, 10)
>>> c.count
10
```

“Магические” ✨✨ методы

```
class Silly:
    def __init__(self):
        self.__dict__['data'] = \
            collections.defaultdict(lambda: 42)

    def __getattr__(self, item):
        return self.data[item]  # this works!

    def __setattr__(self, key, value):
        value = 42
        self.__dict__[key] = value

    def __delattr__(self, item):
        self.data.pop(item, None)

s = Silly()
assert s.foo == 42
s.foo = 92
assert s.foo == 42
del s.foo
```

“Магические” ✨✨ методы

```
class EvenMoreSilly:
    def __init__(self):
        __dict__ = super().__getattribute__("__dict__")
        __dict__["data"] = collections.defaultdict(lambda: 42)

    def __getattribute__(self, item):
        data = super().__getattribute__("data")
        return data[item]

    def __setattr__(self, key, value):
        value = 42
        data = super().__getattribute__("data")
        data[key] = value


s = EvenMoreSilly()
assert s.foo == 42
s.foo = 92
assert s.foo == 42
```

“Магические” ✨✨ методы

```
class EvenMoreSilly:
    def __init__(self):
        __dict__ = super().__getattribute__("__dict__")
        __dict__["data"] = collections.defaultdict(lambda: 42)

    def __getattribute__(self, item):
        data = super().__getattribute__("data")
        return data[item]

    def __setattr__(self, key, value):
        value = 42
        data = super().__getattribute__("data")
        data[key] = value


s = EvenMoreSilly()
assert s.foo == 42
s.foo = 92
assert s.foo == 42
```

Почитать в транспорте

https://rszalski.github.io/magicmethods/