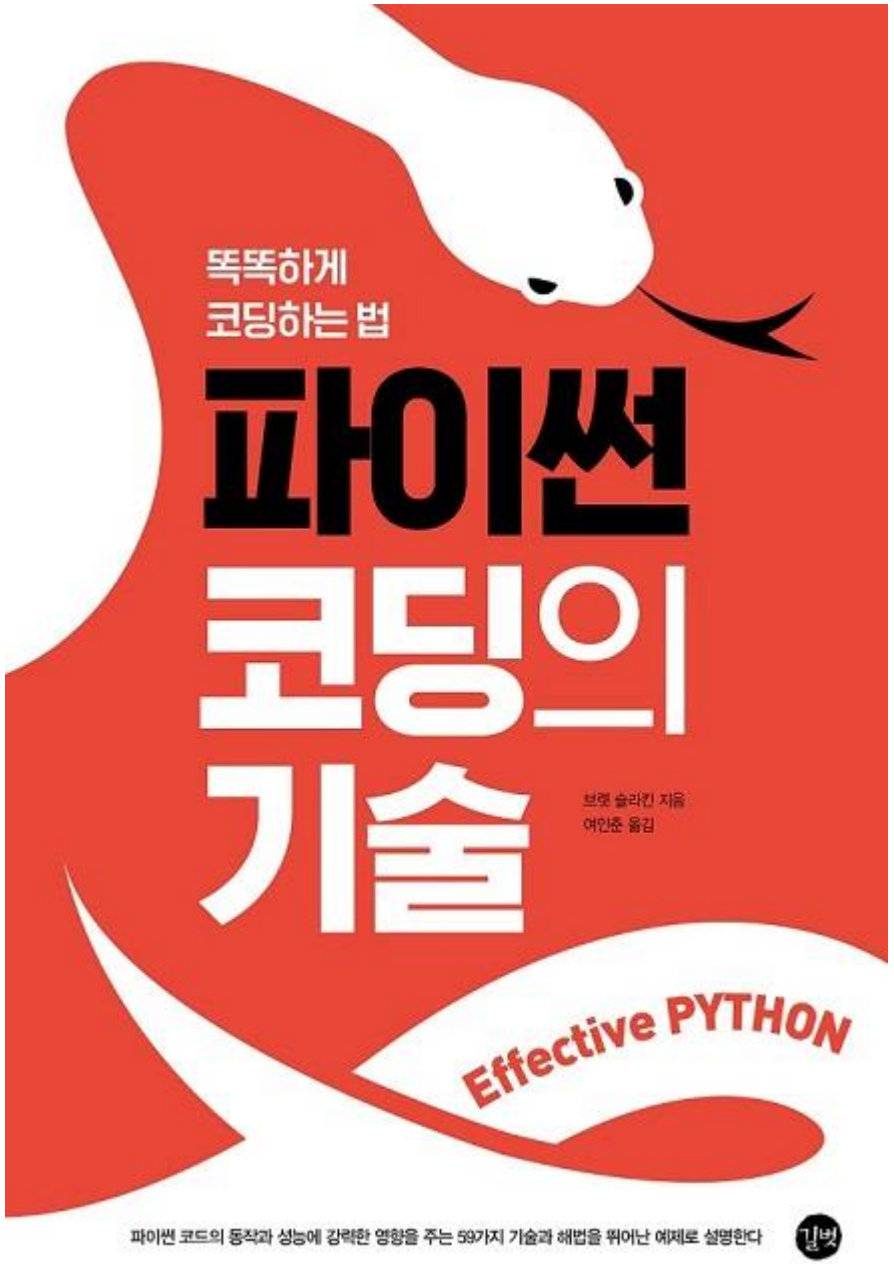
Effective python

강서연

2020. 03. 12



Chapter4. 메타클래스와 속성

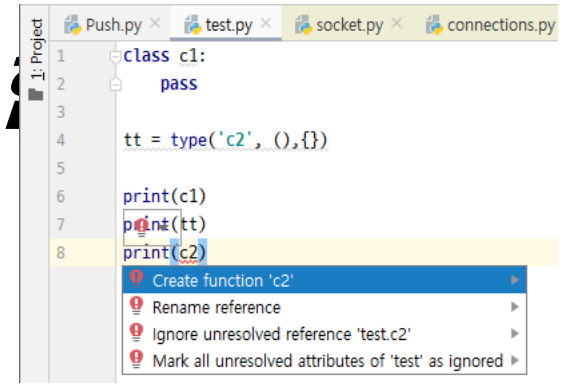
- ✓ 29 게터와 세터 메서드 대신에 일반 속성을 사용하자
- ✓ 30 속성을 리팩토링하는 대신 @property를 고려하자
- ✓ 31 재사용 가능한 @property 메서드에서는 디스크립터를 사용하자
- ✓ 32 지연속성에는 다양한 함수를 사용하자
- ✓ 33 메타클래스로 서브클래스를 검증하자
- ✓ 34 메타클래스로 클래스의 존재를 등록하자
- ✓ 35 메타클래스로 클래스 속성에 주석을 달자

메타클래스? - 클래스를 객체로

- Python은 smalltalk언어에서 따온 매우 특별한 클래스 구성이 존재
- 파이썬에서 클래스는 객체
  - Class 키워드를 사용하면 python이 실행하면서 객체를 만들어냄.
  - 클래스 객체는 새로운 객체(인스턴스)를 만들 수 있음

  - 변수에 할당할 수도 있고
  - 복사할 수도 있고
  - 새로운 속성을 추가할 수도 있고
  - 함수의 인자로 넘길 수도 있습니다.

메타클래스? – 동적으로 클래스 생성 I type



- Type 함수
  1. 객체의 타입 반환
  2. 클래스의 정의를 인자로 받아서 클래스를 반환

```
type(name of the class,
     tuple of the parent class (for inheritance, can be empty),
     dictionary containing attributes names and values)
```

```
class c1:
    pass

tt = type('c2', (), {})


print(c1)
print(tt)
print(type('c3', (), {}))
```

- 클래스 객체 반환
- 할당한 변수가 클래스의 레퍼런스를 갖는다

```
<class '__main__.c1'>
<class '__main__.c2'>
<class '__main__.c3'>
```

```
print(c1())
print(tt())
```

```
<__main__.c1 object at 0x00000243D6D23E80>
<__main__.c2 object at 0x00000243D6D23E80>
```

https://tech.ssut.me/understanding-python-metaclasses/

잠깐 딴소리… - 인스턴스 메모리 할당

Example #1

```
>>> for i in range(3):
...     print(DemoClass())
...
<test.DemoClass instance at 0x288b248>   A
<test.DemoClass instance at 0x288b248>   A
<test.DemoClass instance at 0x288b248>   A
```

For loop의 iteration마다 print(DemoClass())가 호출되는데, 호출이 끝날 때마다 사용된 메모리주소는 해제되어 heap의 top을 반환함.

다음 iteration이 실행되는 동안, 처음에 할당한 주소 다시 사용.

Example #2

```
>>> for j in [DemoClass() for i in range(3)]:
...     print(j)
...                                          A
<test.DemoClass instance at 0x288bcf8>       B
<test.DemoClass instance at 0x288b290>       C
<test.DemoClass instance at 0x288b638>
```

```
>>> print DemoClass()
<test.DemoClass instance at 0x288bcf8>   A
```

리스트에 먼저 Democlass 객체들을 할당하고서 리스트를 순회하므로 주소 3개를 사용하게 됨.

루프가 끝나고, 리스트가 담긴 메모리가 해제됨.

다시 호출된다면 이전에 사용된 메모리가 다시 사용됨.

https://stackoverflow.com/questions/40514829/different-class-instances-use-same-memory-location

잠깐 딴소리… - 인스턴스 메모리 할당

Example #3

```
>>> for i in range(4):
...     Demo = DemoClass()
...     print(Demo)
...
<test.DemoClass instance at 0x288bcf8> A
<test.DemoClass instance at 0x288b290> B
<test.DemoClass instance at 0x288bcf8> A
<test.DemoClass instance at 0x288b290> B
```

이 예제에서는 매번 Demo라는 변수에 Democlass 인스턴스가 할당?되고, 메모리의 세그먼트 하나에 Demo가 할당됨.

하지만 print(Demo)가 호출되고 메모리가 해제되지 않음. 다음 루프의 시작에서 새로운 메모리 세그먼트에 Demo가 할당되고, Demo는 **덮어 쓰여지면서** 기존의 메모리주소는 heap의 top으로 돌아감. 두개의 메모리 주소가 번갈아 가면서 사용됨.

https://stackoverflow.com/questions/40514829/different-class-instances-use-same-memory-location

메타클래스? – 동적으로 클래스 생성 I type함수

`type`은 클래스의 속성을 정의하기 위해 `dict`를 인자로 받습니다, 예로:

```
>>> class Foo(object):
...       bar = True
```

는 다음과 같이 쓸 수 있으며:

```
>>> Foo = type('Foo', (), {'bar':True})
```

일반적인 클래스로 사용할 수 있습니다:

```
>>> print(Foo)
<class '__main__.Foo'>
>>> print(Foo.bar)
True
>>> f = Foo()
>>> print(f)
<__main__.Foo object at 0x8a9b84c>
>>> print(f.bar)
True
```

```
15
16    import types
17    c3 = type('c3', (), {'m':lambda a,b :a+b})
18 1  print(type(c3.m))
19
20 2  print(isinstance(c3.m, types.LambdaType))
21 3  print(c3.m.__name__)
22
23 4  print(c3.m(1,2))
```

```
1  <class 'function'>
2  True
3  <lambda>
4  3
```

```
x = lambda x: None
def y(): pass
print y.__name__
# y
print x.__name__
# <lambda>
```

메타클래스? - 동적으로 클래스 생성 I type함수

그리고 물론 상속도 받을 수 있습니다, 예로:

```
>>>   class FooChild(Foo):
...         pass
```

는 다음 코드가 됩니다:

```
>>> FooChild = type('FooChild', (Foo,), {})
>>> print(FooChild)
<class '__main__.FooChild'>
>>> print(FooChild.bar) # bar is inherited from Foo
True
```

https://tech.ssut.me/understanding-python-metaclasses/

메타클래스? – 동적으로 클래스 생성 I type함수

클래스에 메소드를 추가하고 싶을 때에는 적절한 모양으로 함수를 만들고 인자로 넘기기만 하면 됩니다.

```
>>> def echo_bar(self):
...         print(self.bar)
...
>>> FooChild = type('FooChild', (Foo,), {'echo_bar': echo_bar})
>>> hasattr(Foo, 'echo_bar')
False
>>> hasattr(FooChild, 'echo_bar')
True
>>> my_foo = FooChild()
>>> my_foo.echo_bar()
True
```

Self 안 넣으면 오류발생

```
6   def meth():
7       print('meth')
8
9   import types
10  c3 = type('c3', (), {'m1':lambda a,b :a+b, 'm2': meth})
11  print(type(c3.m1), " / ", type(c3.m2))
12
13  print(isinstance(c3.m1, types.LambdaType), " / ", isinstance(c3.m2, types.LambdaType),
14        " / ", isinstance(c3.m2, types.FunctionType))
15
16  print(c3.m1.__name__, " / ", c3.m2.__name__)
17
```

➔

```
<class 'function'>  /  <class 'function'>
True  /  True  /  True
<lambda>  /  meth
```

https://tech.ssut.me/understanding-python-metaclasses/

메타클래스? - 동적으로 클래스 생성 I type함수

물론 이렇게 동적으로 클래스를 만들고 나서 일반 클래스와 동일하게 새로운 메소드나 속성을 추가할 수도 있습니다.

```
>>> def echo_bar_more(self):
...       print('yet another method')
...
>>> FooChild.echo_bar_more = echo_bar_more
>>> hasattr(FooChild, 'echo_bar_more')
True
```

hasattr(*object*, *name*)

Object내에 name에 해당하는 attribute가 있으면 True, 없으면 False를 리턴.

메타클래스?

- 메타클래스는 클래스객체를 만드는 '무언가'

Type은 python이 사용하는 내장된 메타클래스.

모든 것, Python에서의 모든 것은 객체입니다. 여기에는 정수, 문자열, 함수, 클래스를 포함합니다. 이 모든 것들은 객체입니다. 그리고 이 모든 것들은 클래스로부터 생성됩니다:

```
>>> age = 35
>>> age.__class__
<type 'int'>
>>> name = 'bob'
>>> name.__class__
<type 'str'>
>>> def foo(): pass
>>> foo.__class__
<type 'function'>
>>> class Bar(object): pass
>>> b = Bar()
>>> b.__class__
<class '__main__.Bar'>
```

자, 아무 `__class__`의 `__class__`는 무엇일까요?

```
>>> age.__class__.__class__
<type 'type'>
>>> name.__class__.__class__
<type 'type'>
>>> foo.__class__.__class__
<type 'type'>
>>> b.__class__.__class__
<type 'type'>
```

메타클래스? - __metaclass__ 속성

우리는 클래스 코드를 작성할 때 `__metaclass__` 속성을 직접 추가할 수 있습니다.

```
class Foo(object):
  __metaclass__ = something...
  [...]
```

이렇게 하면 Python은 `Foo` 클래스를 생성하기 위해 직접 설정된 메타클래스를 사용하게 됩니다. 다만 약간의 주의가 필요합니다.

우리는 `class Foo(object)` 코드를 먼저 작성했습니다, 하지만 `Foo` 클래스 객체는 메모리상에 아직 생성되지 않은 상태입니다.

Python은 클래스 정의에 `__metaclass__`가 있는지 먼저 확인하게 될거고, 발견된 경우에 `Foo` 클래스를 만들기 위해 해당 메타클래스를 사용합니다. 발견하지 못한 경우에는 클래스를 만들기 위해 `type`을 사용하게 됩니다.

메타클래스? - __metaclass__ 속성

```
class Foo(Bar):
    pass
```

Python은 다음과 같이 작동합니다:

- `Foo` 에 `__metaclass__` 속성이 있나요?
  - 있으면, `__metaclass__` 에 있는 걸로 `Foo` 클래스 객체(클래스 객체라고 말했습니다.)를 만듭니다.
  - Python이 `__metaclass__` 를 찾지 못했으면, Python은 `__metaclass__` 를 **모듈** 레벨에서 찾고 위와 같은 방법으로 작동합니다. (단 상속받지 않은 클래스에 한해서만 그렇습니다, old-style classes 말이죠 - py2.)
  - 그래도 `__metaclass__` 를 찾지 못했으면, Python은 `Bar` (가장 첫번째 부모 클래스)가 가진 메타클래스(여기서는 기본 메타클래스인 `type`)를 사용해서 클래스 객체를 만듭니다.

여기서 조심해야 할 부분은 `__metaclass__` 속성은 상속되지 않는다는 점입니다, 부모 (`Bar.__class__`)의 메타클래스도 말이죠. 만약 `Bar` 가 (`type.__new__()` 가 아닌) `type()` 을 이용해 `Bar` 를 만든 `__metaclass__` 속성을 사용했다면, 서브클래스는 해당 행동을 상속받지 않습니다.

__metaclass__엔 클래스를 만드는 Type 또는 서브클래스, type을 사용하는 모든 것이 올 수 있음.

메타클래스? - 커스텀 메타클래스

자 이제 큰 질문을 하나 해봅시다, `__metaclass__`에 무엇을 넣을 수 있을까요?

답은: 클래스를 만드는 '무언가'입니다.

그러면 무엇이 클래스를 만들 수 있을까요? `type` 또는 서브클래스나 `type`을 사용하는 모든 것이 해당됩니다.

메타클래스의 주 목적은 클래스가 만들어질 때 클래스를 자동으로 바꾸기 위한 것입니다. 우리는 보통 현재 컨텍스트와 알맞는 클래스를 만들기 위해 API와 같은 곳에 커스텀 메타클래스를 사용합니다.

메타클래스?

- 파이썬에서는 클래스도 객체이다. 클래스를 만드는 또 다른 클래스가 "메타클래스"
  - 클래스로 객체를 만들 듯이 메타클래스로 클래스를 만들 수 있음

- type()

```
>>> type(3)
<class 'int'>
```

튜플 | 딕셔너리

```
>>> temp = type('temp', (), {})
>>> temp
<class '__main__.temp'>
```

```
>>> type(int)
<class 'type'>
```

1. 동적으로 클래스를 만들 수 있음

```
class temp:
    a = 3

    def m(self, m, n):
        return m + n

ins = temp()
print(ins.m(3,4))
#결과
7
```

이 클래스를 type()이용해 만들면 다음과 같다.

```
>>> ins = type('temp', (object,), {'a':3, 'm':lambda a, b: a+b})
>>> print(ins.m(3,4))
7
```

메타클래스?

2. 커스텀 메타 클래스 생성

→클래스를 컨트롤해 원하는 방향으로 클래스가 생성되게 할 수 있음

→type클래스를 상속받고, type클래스가 가지고 있는 new 메서드를 오버라이드하여 생성

```
class myMetaclass(type):

    def __new__(cls, clsname, bases, dct):
        assert type(dct['a']) is int, 'a속성이 정수가 아니에요.'
        return type.__new__(cls, clsname, bases, dct)

class temp(metaclass=myMetaclass):
    a = 3.14

ins = temp()
#결과
AssertionError: a속성이 정수가 아니에요..
```

Dct : 메타클래스로 클래스를 만들 때 속성과 메서드를 이곳에 명시하면 됨. Type()으로 클래스를 만들 때 맨 끝 인자와 같음.

New메서드에서 반환해야 할 값 : type.new() 메서드로 받는 클래스