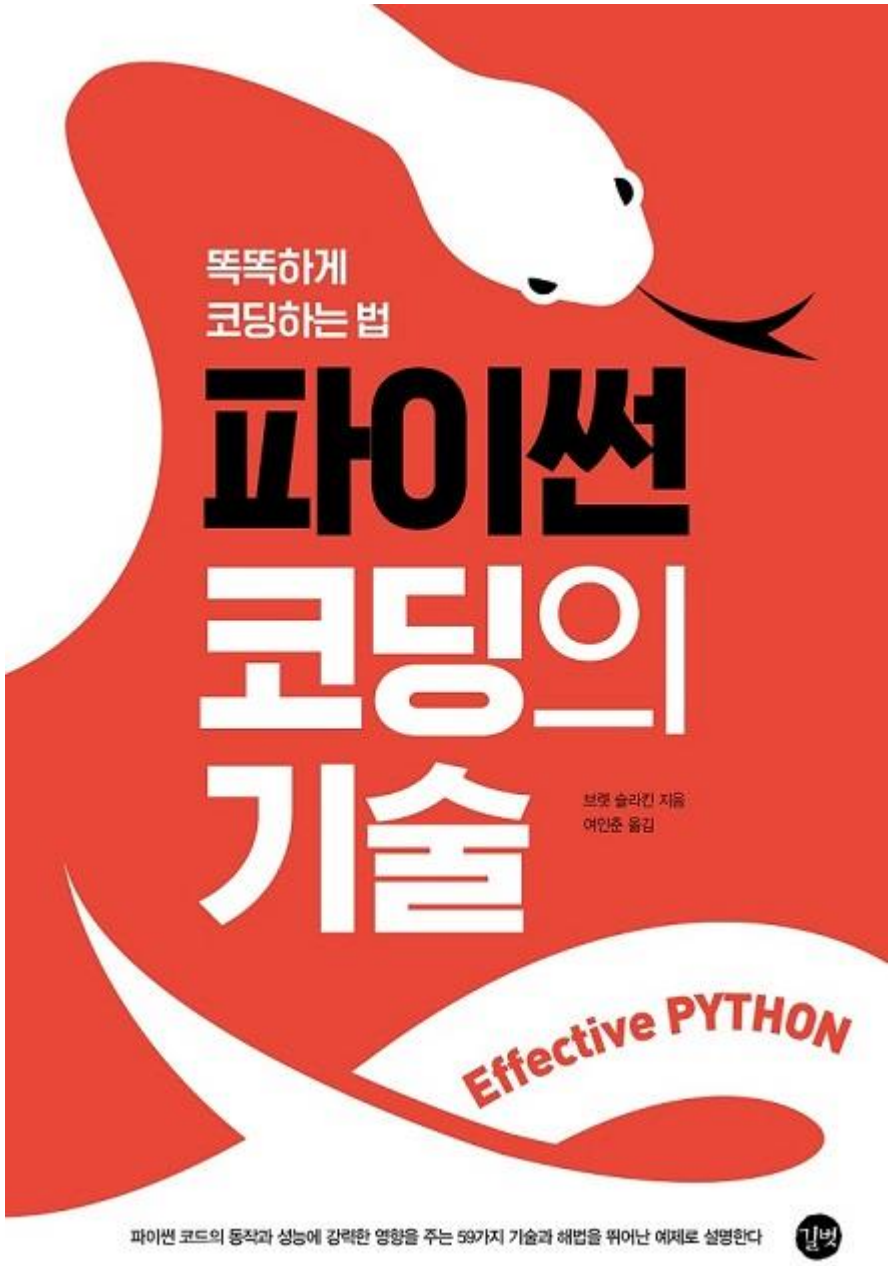
Effective python

강서연

2020. 02. 13



BetterWay27. 공개 속성보다는 비공개 속성을 사용하자.

- 파이썬 클래스의 속성의 가시성 – public or private

  - Under bar 2개 이용해서 private 표현
  - Classmethod도 같은 클래스 내의 비공개 속성에 접근 가능

```
class MyOtherObject(object):
    def __init__(self):
        self.__private_field = 71

    @classmethod
    def get_private_field_of_instance(cls, instance):
        return instance.__private_field

bar = MyOtherObject()
assert MyOtherObject.get_private_field_of_instance(bar) == 71
```

BetterWay27. 공개 속성보다는 비공개 속성을 사용하자.

- 서브클래스에서 부모클래스의 private 필드 접근 시

  - 파이썬 컴파일러는 클래스의 메서드에서 비공개 속성에 접근하는 코드 발견하면 _classname__private_field 에 접근하는 코드로 변환

```
class MyParentObject(object):
    def __init__(self):
        self.__private_field = 71

class MyChildObject(MyParentObject):
    def get_private_field(self):
        return self.__private_field

baz = MyChildObject()
baz.get_private_field()
```

```
AttributeError: 'MyChildObject' object has no attribute '_MyChildObject__private_field'
```

```
class MyChildObject(MyParentObject):
    def get_private_field(self):
        return self.private_field
```

```
AttributeError: 'MyChildObject' object has no attribute 'private_field'
```

BetterWay27. 공개 속성보다는 비공개 속성을 사용하자.

- 이와 같이 작성하면 비공개 속성에 접근 가능 → 객체 내부 조작 가능하도록 설계됨

```
assert baz._MyParentObject__private_field == 71
```

- 무조건 비공개 필드로 작성하기 보다는 _protected_field처럼 보호필드로 취급하여 외부사용자들에게 신중하게 사용하도록 작성하는게 좋음, 그리고 문서화해서 설명 작성

- 비공개필드로만 작성시 확장성이 떨어짐

```
class MyClass(object):
    def __init__(self, value):
        self.__value = value

    def get_value(self):
        return str(self.__value)


# Example 10
class MyIntegerSubclass(MyClass):
    def get_value(self):
        return int(self._MyClass__value)
```

```
class MyBaseClass(object):
    def __init__(self, value):
        self.__value = value

    def get_value(self):
        return self.__value

class MyClass(MyBaseClass):
    def get_value(self):
        return str(super().get_value())

class MyIntegerSubclass(MyClass):
    def get_value(self):
        return int(self._MyClass__value)
```

BetterWay27. 공개 속성보다는 비공개 속성을 사용하자.

- 비공개필드 사용할만한 경우
  - 서브클래스와 이름이 충돌할 가능성
    →자식클래스가 부모클래스에서 정의된 필드를 다시 정의할 때

이름을 겹치지 않게 작성하여 해결

```
class ApiClass(object):
    def __init__(self):
        self._value = 5

    def get(self):
        return self._value

class Child(ApiClass):
    def __init__(self):
        super().__init__()
        self._value = 'hello'  # Conflicts

a = Child()
print(a.get(), 'and', a._value, 'should be different')
```

```
hello and hello should be different
```

```
class ApiClass(object):
    def __init__(self):
        self.__value = 5

    def get(self):
        return self.__value

class Child(ApiClass):
    def __init__(self):
        super().__init__()
        self._value = 'hello'  # OK!

a = Child()
print(a.get(), 'and', a._value, 'are different')
```

```
5 and hello are different
```

BetterWay27. 공개 속성보다는 비공개 속성을 사용하자.

- 파이썬 컴파일러는 비공개 속성을 엄격하게 강요하지 않는다.
- 서브클래스가 내부 API와 속성에 접근하지 못하게 막기보다는 처음부터 내부 API와 속성으로 더 많은 일을 할 수 있게 설계하자.
- 비공개 속성에 대한 접근을 강제로 제어하지 말고 보호 필드를 문서화해서 서브클래스에 필요한 지침을 제공하자.
- 직접 제어할 수 없는 서브클래스와 이름이 충돌하지 않게 할 때만 비공개 속성을 사용하는 방안을 고려하자.

BetterWay28. 커스텀 컨테이너 타입은 collections.abc의 클래스를 상속받게 만들자

- 커스텀 컨테이너 타입

  : 내장 컨테이너 타입(리스트, 튜플, 세트, 딕셔너리 등)에 프로그래머가 커스텀 동작을 위해 메서드를 추가한 타입

- 커스텀 리스트타입 예제 - 멤버의 빈도를 세는 메서드를 추가로 갖는 예제

```
class FrequencyList(list):
    def __init__(self, members):
        super().__init__(members)

    def frequency(self):  self: ['a', 'b', 'a', 'c', 'b', 'a']
        counts = {}  counts: <class 'dict'>: {}
        for item in self:  item: 'a'
            counts.setdefault(item, 0)
            counts[item] += 1
        return counts


# Example 2
foo = FrequencyList(['a', 'b', 'a', 'c', 'b', 'a', 'd'])
print('Length is', len(foo))
foo.pop()
print('After pop:', repr(foo))
print('Frequency:', foo.frequency())
```

```
>>>
Length is 7
After pop: ['a', 'b', 'a', 'c', 'b', 'a']
Frequency: {'a':3, 'c':1, 'b':2}
```

BetterWay28. 커스텀 컨테이너 타입은 collections.abc의 클래스를 상속받게 만들자

- Collections.abc
  - 파이썬의 내장모듈
  - 각 컨테이너 타입에 필요한 일반적인 메서드를 모두 제공하는 추상 기반 클래스들을 정의함.
  - 추상 기반 클래스들에서 상속받아 서브클래스를 만들다가 필수 메서드를 구현하지 않으면 오류발생

```
from collections.abc import Sequence

class BadType(Sequence):
    pass

foo = BadType()
```

```
Traceback (most recent call last):
  File "C:/Users/PLAS/Desktop/3_winter/006764/test.py", line 6, in <module>
    foo = BadType()
TypeError: Can't instantiate abstract class BadType with abstract methods __getitem__, __len__
```

BetterWay28. 커스텀 컨테이너 타입은 collections.abc의 클래스를 상속받게 만들자

- 쓰임새가 간단할 때는 list나 dict 같은 파이썬의 컨테이너 타입에서 직접 상속받게 하자.
- 커스텀 컨테이너 타입을 올바르게 구현하는 데 필요한 많은 메서드에 주의해야 한다.
- 커스텀 컨테이너 타입이 collections.abc에 정의된 인터페이스에서 상속받게 만들어서 클래스가 필요한 인터페이스, 동작과 일치하게 하자.

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?

- 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

BetterWay29. 게터와 세터 메서드 대신에 일반 속성을 사용하자