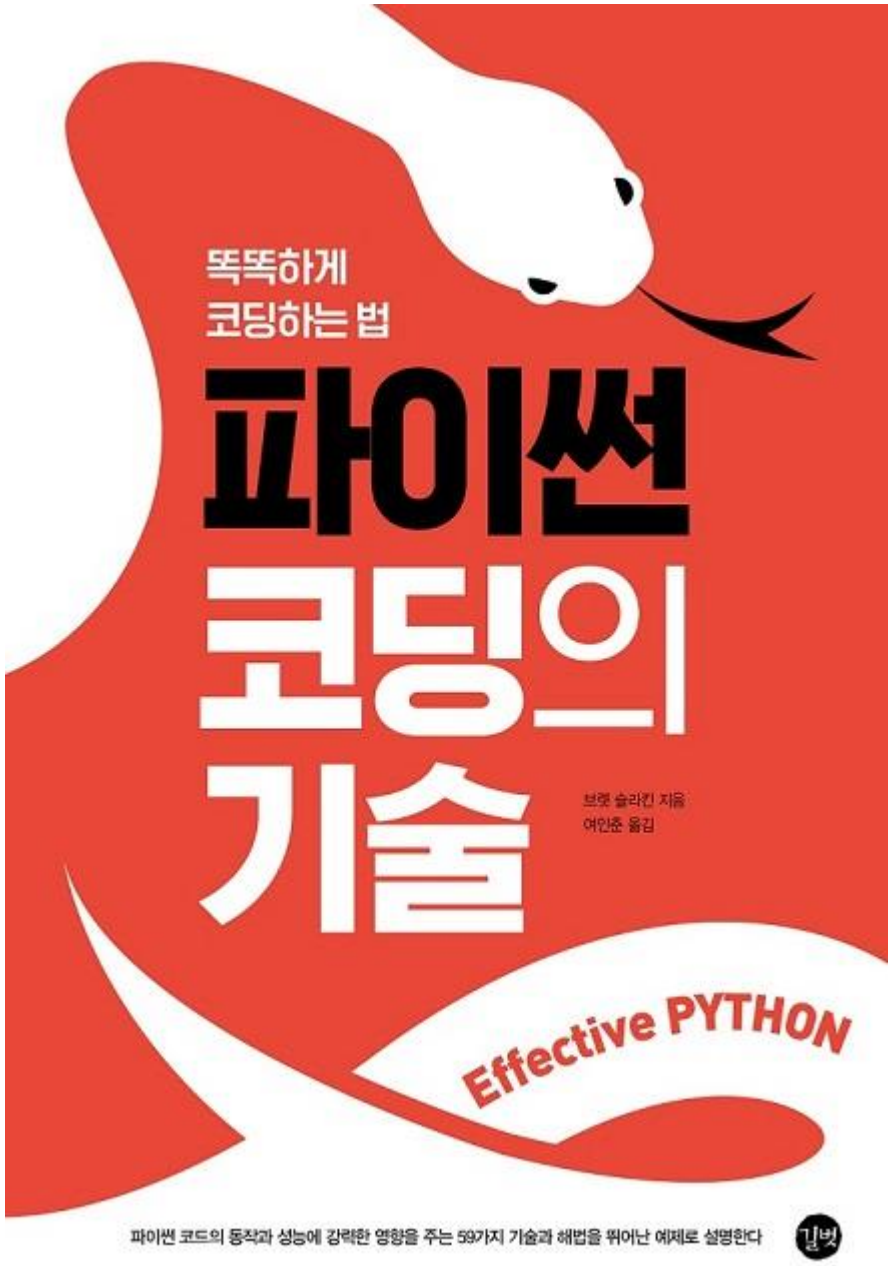
Effective python

강서연

2020. 01. 23



- 변수가 list일 때와 boolean일 때

```python
# Example 3
def sort_priority2(numbers, group):
    found = False
    def helper(x):
        if x in group:
            found = True # Seems simple
            return (0, x)
        return (1, x)
    numbers.sort(key=helper)
    return found
```

```python
# Example 3
def sort_priority2(numbers, group):
    found = [True, False]
    def helper(x):
        print(group)
        if x in group:
            found = [False,True] # Seems simple
            return (0, x)
        return (1, x)
    numbers.sort(key=helper)
    return found
```

```python
# Example 3
def sort_priority2(numbers, group):
    found = [True, False]
    def helper(x):
        if x in group:
            found[0] = False # Seems simple
            return (0, x)
        return (1, x)
    numbers.sort(key=helper)
    return found
```

- 할당이 아닌 참조할 때

```python
# Example 3
def sort_priority2(numbers, group):
    found = False
    def helper(x):
        if x in group:
            print(found) # Seems simple
            return (0, x)
        return (1, x)
    numbers.sort(key=helper)
    return found
```

```python
# Example 3
def sort_priority2(numbers, group):
    found = [True, False]
    def helper(x):
        print(group)
        if x in group:
            print(found) # Seems simple
            return (0, x)
        return (1, x)
    numbers.sort(key=helper)
    return found
```

- If 조건에 쓰인 변수 변경하려고 할 때

```python
# Example 3
def sort_priority2(numbers, group):
    found = [3,4,56]
    def helper(x):
        if x in found:
            found = {1,2}
            return (0, x)
        return (1, x)
    numbers.sort(key=helper)
    return found
```

```python
# Example 3
def sort_priority2(numbers, group):
    found = [True, False]
    def helper(x):
        print(group)
        if x in group:
            group = {1,2} # Seems simple
            return (0, x)
        return (1, x)
    numbers.sort(key=helper)
    return found
```

```
Traceback (most recent call last):
  File "C:\Users\PLAS\AppData\Local\JetBrains\Toolbox\
    main()
  File "C:\Users\PLAS\AppData\Local\JetBrains\Toolbox\
    globals = debugger.run(setup['file'], None, None,
  File "C:\Users\PLAS\AppData\Local\JetBrains\Toolbox\
    pydev_imports.execfile(file, globals, locals)  # e
  File "C:\Users\PLAS\AppData\Local\JetBrains\Toolbox\
    exec(compile(contents+"\n", file, 'exec'), glob,
  File "C:/Users/PLAS/Desktop/3_winter/006764/item_15.
    found = sort_priority2(numbers, group)
  File "C:/Users/PLAS/Desktop/3_winter/006764/item_15.
    numbers.sort(key=helper)
  File "C:/Users/PLAS/Desktop/3_winter/006764/item_15.
    if x in group:
UnboundLocalError: local variable 'group' referenced b
```

```python
# Example 3
def sort_priority2(numbers, group):
    found = [3,4,56]
    def helper(x):
        group = [1,2,3]
        if x in group:
            return (0, x)
        return (1, x)
    numbers.sort(key=helper)
    return found
```

```python
# Example 3
def sort_priority2(numbers, group):
    found = [True, False]
    def helper(x):
        print(group)
        if x in group:
            print(found) # Seems simple
            return (0, x)
        return (1, x)
    numbers.sort(key=helper)
    return found
```

Chapter 3. *클래스와 상속*

- 파이썬은 상속, 다향성, 캡슐화 같은 객체 지향 언어의 모든 기능 제공

BetterWay22. 딕셔너리와 튜플보다는 헬퍼 클래스로 관리하자

- 딕셔너리 타입 - 객체의 수명이 지속되는 동안 동적인 내부상태를 관리하는 용도로 좋음.
  - 동적 : 예상하지 못한 식별자들을 관리해야 하는 상황

- 학생마다 과목별로 성적 관리하는 예제

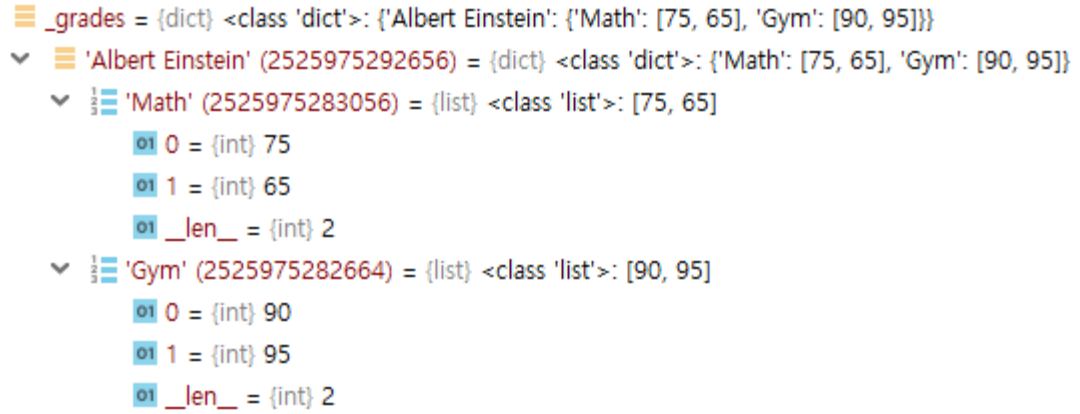
```
# Example 3
class BySubjectGradebook(object):
    def __init__(self):
        self._grades = {}

    def add_student(self, name):
        self._grades[name] = {}


# Example 4
    def report_grade(self, name, subject, grade):
        by_subject = self._grades[name]
        grade_list = by_subject.setdefault(subject, [])
        grade_list.append(grade)

    def average_grade(self, name):
        by_subject = self._grades[name]
        total, count = 0, 0
        for grades in by_subject.values():
            total += sum(grades)
            count += len(grades)
        return total / count
```

```
book.add_student('Albert Einstein')
book.report_grade('Albert Einstein', 'Math', 75)
book.report_grade('Albert Einstein', 'Math', 65)
```



BetterWay22. 딕셔너리와 튜플보다는 헬퍼 클래스로 관리하자

```python
# Example 3
class BySubjectGradebook(object):
    def __init__(self):
        self._grades = {}

    def add_student(self, name):
        self._grades[name] = {}


# Example 4
    def report_grade(self, name, subject, grade):
        by_subject = self._grades[name]
        grade_list = by_subject.setdefault(subject, [])
        grade_list.append(grade)

    def average_grade(self, name):
        by_subject = self._grades[name]
        total, count = 0, 0
        for grades in by_subject.values():
            total += sum(grades)
            count += len(grades)
        return total / count
```

```python
def countLetters(word):
    counter = {}
    for letter in word:
        counter.setdefault(letter, 0)
        counter[letter] += 1
    return counter
```

```python
from collections import defaultdict

def countLetters(word):
    counter = defaultdict(int)
    for letter in word:
        counter[letter] += 1
    return counter
```

```python
def countLetters(word):
    counter = defaultdict(lambda: 0)
    for letter in word:
        counter[letter] += 1
    return counter
```

BetterWay22. 딕셔너리와 튜플보다는 헬퍼 클래스로 관리하자

- 중간고사와 기말고사 성적 비중을 다르게 한다면
  - Add할때 튜플로 추가하면 됨
  - 평균계산할 때 너무 복잡해짐

  - 계층이 한 단계가 넘는 중첩은 피해야함.

(딕셔너리를 담은 딕셔너리는 쓰지 말아야함.)

```
# Example 6
class WeightedGradebook(object):
    def __init__(self):
        self._grades = {}

    def add_student(self, name):
        self._grades[name] = {}

    def report_grade(self, name, subject, score, weight):
        by_subject = self._grades[name]
        grade_list = by_subject.setdefault(subject, [])
        grade_list.append((score, weight))
```

```
# Example 7
    def average_grade(self, name):
        by_subject = self._grades[name]
        score_sum, score_count = 0, 0
        for subject, scores in by_subject.items():
            subject_avg, total_weight = 0, 0
            for score, weight in scores:
                subject_avg += score * weight
                total_weight += weight
            score_sum += subject_avg / total_weight
            score_count += 1
        return score_sum / score_count
```

BetterWay22. 딕셔너리와 튜플보다는 헬퍼 클래스로 관리하자

- 클래스 리팩토링
  - 성적에 튜플 사용.

  - 관례적으로 사용하지 않을 변수에 ' _ ' 사용

```
# Example 10
grades = []
grades.append((95, 0.45, 'Great job'))
grades.append((85, 0.55, 'Better next time'))
total = sum(score * weight for score, weight, _ in grades)
total_weight = sum(weight for _, weight, _ in grades)
average_grade = total / total_weight
print(average_grade)
```

제너레이터

  - 튜플의 아이템이 두개인 경우에만 사용하는게 좋음.

BetterWay22. 딕셔너리와 튜플보다는 헬퍼 클래스로 관리하자

- 작은 불변 클래스 정의
  - 위치인수나 키워드인수로 생성 가능
  - 단점 : 기본 값 설정 불가하므로 속성 많으면 쓰기 안 좋음

```
import collections
Grade = collections.namedtuple('Grade', ('score', 'weight'))
```

```
>>> # Basic example
>>> Point = namedtuple('Point', ['x', 'y'])
>>> p = Point(11, y=22)     # instantiate with positional or keyword arguments
>>> p[0] + p[1]             # indexable like the plain tuple (11, 22)
33
>>> x, y = p                # unpack like a regular tuple
>>> x, y
(11, 22)
>>> p.x + p.y               # fields also accessible by name
33
>>> p                       # readable __repr__ with a name=value style
Point(x=11, y=22)
```

BetterWay22. 딕셔너리와 튜플보다는 헬퍼 클래스로 관리하자

- 코드 길이가 더 길어졌지만 이해하기 더 명확해짐

```
class Subject(object):
    def __init__(self):
        self._grades = []

    def report_grade(self, score, weight):
        self._grades.append(Grade(score, weight))

    def average_grade(self):
        total, total_weight = 0, 0
        for grade in self._grades:
            total += grade.score * grade.weight
            total_weight += grade.weight
        return total / total_weight
```

```
# Example 13
class Student(object):
    def __init__(self):
        self._subjects = {}

    def subject(self, name):
        if name not in self._subjects:
            self._subjects[name] = Subject()
        return self._subjects[name]

    def average_grade(self):
        total, count = 0, 0
        for subject in self._subjects.values():
            total += subject.average_grade()
            count += 1
        return total / count
```

```
# Example 14
class Gradebook(object):
    def __init__(self):
        self._students = {}

    def student(self, name):
        if name not in self._students:
            self._students[name] = Student()
        return self._students[name]
```

```
# Example 15
book = Gradebook()
albert = book.student('Albert Einstein')
math = albert.subject('Math')
math.report_grade(80, 0.10)
math.report_grade(80, 0.10)
math.report_grade(70, 0.80)
gym = albert.subject('Gym')
gym.report_grade(100, 0.40)
gym.report_grade(85, 0.60)
print(albert.average_grade())
```

BetterWay23. 인터페이스가 간단하면 클래스대신 함수를 받자

- Hook : 파이썬 내장 api에서 함수를 넘겨 동작을 사용자화 하는 기능
  - 파이썬에서는 first-class function 을 갖춤. 추상클래스가 아닌 함수로 동작.

```
# Example 1
names = ['Socrates', 'Archimedes', 'Plato', 'Aristotle']
names.sort(key=lambda x: len(x))
print(names)
```

- Defaultdict 클래스의 동작을 사용자화 하는 예제
  - 결정 동작과 부작용을 분리하므로 api를 쉽게 구축하고 테스트

```
# Example 2
from collections import defaultdict

def log_missing():
    print('Key added')
    return 0


# Example 3
current = {'green': 12, 'blue': 3}
increments = [
    ('red', 5),
    ('blue', 17),
    ('orange', 9),
]
result = defaultdict(log_missing, current)
print('Before:', dict(result))
for key, amount in increments:
    result[key] += amount
print('After: ', dict(result))
```

```
Before: {'green': 12, 'blue': 3}
Key added
Key added
After:  {'green': 12, 'blue': 20, 'red': 5, 'orange': 9}
```

BetterWay23. 인터페이스가 간단하면 클래스대신 함수를 받자

- 찾을 수 없는 키의 총 개수를 센다면
  - Defaultdict는 missing 후크가 상태를 유지한다는 사실을 모르지만 결과를 얻음.
  - 클로저 안에 상태를 숨겨서 기능을 추가하기 쉬워짐

  - But 이해하기 어려움.

```
# Example 4
def increment_with_report(current, increments):
    added_count = 0

    def missing():
        nonlocal added_count  # Stateful closure
        added_count += 1
        return 0

    result = defaultdict(missing, current)
    for key, amount in increments:
        result[key] += amount

    return result, added_count
```

BetterWay23. 인터페이스가 간단하면 클래스대신 함수를 받자

- 보존할 상태를 캡슐화하는 작은 클래스 정의한다면
  - 일급함수라서 객체로 counter.missing 메서드를 직접참조해서 기본값 후크로 넘길 수 있음.

```
# Example 6
class CountMissing(object):
    def __init__(self):
        self.added = 0

    def missing(self):
        self.added += 1
        return 0


# Example 7
counter = CountMissing()
result = defaultdict(counter.missing, current)  # Method reference
for key, amount in increments:
    result[key] += amount
assert counter.added == 2
print(result)
```

BetterWay23. 인터페이스가 간단하면 클래스대신 함수를 받자

- Missing 메서드 사용보다 더 명확한 표현
  - __call__메서드 : 객체를 함수처럼 호출할 수 있게 해줌.

```python
# Example 8
class BetterCountMissing(object):
    def __init__(self):
        self.added = 0

    def __call__(self):
        self.added += 1
        return 0

counter = BetterCountMissing()
counter()
assert callable(counter)


# Example 9
counter = BetterCountMissing()
result = defaultdict(counter, current)  # Relies on __call__
for key, amount in increments:
    result[key] += amount
assert counter.added == 2
print(result)
```

BetterWay24. 객체를 범용으로 생성하려면 @classmethod 다형성을 이용하자

- Python에서는 클래스별로 생성자 __init__ 한 개만 생성가능
- 다형성 위해서 @classmethod 사용해라
  - 클래스의 다른 생성자를 정의하는 것처럼 사용할 수 있음

- @staticmethod VS @classmethod

```
class TestTestTestTest :
    num = 10

    @staticmethod
    def add (x, y) :
        return x + y + self.num    # TestTestTestTest.num 이면 가능


t = TestTestTestTest()
print t.add(1,1)
```

```
class Test :
    num = 10

    @classmethod
    def add (cls, x, y) :
        return x + y


print Test.add(1,1)
```

BetterWay24. 객체를 범용으로 생성하려면 @classmethod 다형성을 이용하자

```
class Date :

    word = 'date : '

    def __init__(self, date):
        self.date = self.word + date

    @staticmethod
    def now():
        return Date("today")


    def show(self):
        print self.date


a = Date("2016, 9, 13")
a.show()
b = Date.now()
b.show()
```

```
date : 2016, 9, 13
date : today
```

```
class KoreanDate(Date):
    word = '날짜 : '


a = KoreanDate.now()
a.show()

결과)
daate : today
```

```
class Date :

    word = 'date : '

    def __init__(self, date):
        self.date = self.word + date

    @classmethod
    def now(cls):
        return cls("today")

    def show(self):
        print self.date
```

```
class KoreanDate(Date):
    word = '날짜 : '


a = KoreanDate.now()
a.show()

결과)
날짜 : today
```

```python
# Example 3
def sort_priority2(numbers, group):
    found = False
    def helper(x):
        if x in group:
            found = True # Seems simple
            return (0, x)
        return (1, x)
    numbers.sort(key=helper)
    return found
```

```python
# Example 3
def sort_priority2(numbers, group):
    found = [3,4,56]
    def helper(x):
        if x in found:
            found = {1,2}
            return (0, x)
        return (1, x)
    numbers.sort(key=helper)
    return found
```

```python
# Example 3
def sort_priority2(numbers, group):
    found = False
    def helper(x):
        if x in group:
            print(found) # Seems simple
            return (0, x)
        return (1, x)
    numbers.sort(key=helper)
    return found
```

```python
# Example 3
def sort_priority2(numbers, group):
    found = [3,4,56]
    def helper(x):
        group = [1,2,3]
        if x in group:
            return (0, x)
        return (1, x)
    numbers.sort(key=helper)
    return found
```

```python
# Example 3
def sort_priority2(numbers, group):
    found = [True, False]
    def helper(x):
        if x in group:
            found[0] = False # Seems simple
            return (0, x)
        return (1, x)
    numbers.sort(key=helper)
    return found
```

```python
# Example 3
def sort_priority2(numbers, group):
    found = [True, False]
    def helper(x):
        print(group)
        if x in group:
            found = [False,True] # Seems simple
            return (0, x)
        return (1, x)
    numbers.sort(key=helper)
    return found
```

```python
# Example 3
def sort_priority2(numbers, group):
    found = [True, False]
    def helper(x):
        print(group)
        if x in group:
            print(found) # Seems simple
            return (0, x)
        return (1, x)
    numbers.sort(key=helper)
    return found
```

```python
# Example 3
def sort_priority2(numbers, group):
    found = [True, False]
    def helper(x):
        print(group)
        if x in group:
            group = {1,2} # Seems simple
            return (0, x)
        return (1, x)
    numbers.sort(key=helper)
    return found
```