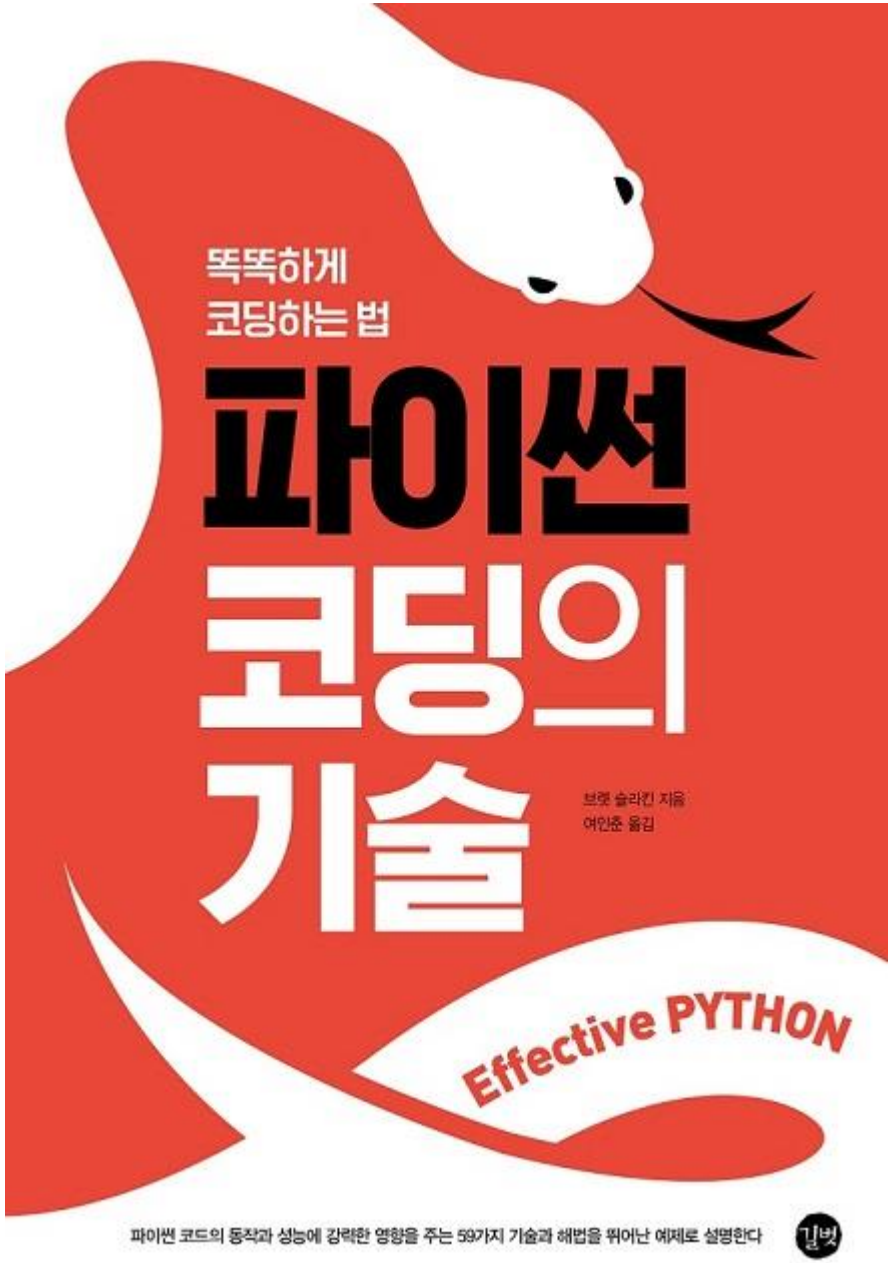
Effective python

강서연

2020. 01. 14



Chapter 2. *function*

- 다른 프로그래밍 언어에서 제공하는 기능과 비슷하지만 그 외의 파이썬에만 있는 기능.
- 부가기능은 함수의 목적을 더 분명하게 함.
- 불필요한 요소를 제거하고 호출자의 의도를 명료하게 보여주며, 찾기 어려운 미묘한 버그를 줄일 수 있음.

BetterWay14. None을 반환하기 보다는 예외를 일으키자

- None을 반환하도록 작성한다면

```python
# Example 1
def divide(a, b):
    try:
        return a / b
    except ZeroDivisionError:
        return None

assert divide(4, 2) == 2
assert divide(0, 1) == 0
assert divide(3, 6) == 0.5
assert divide(1, 0) == None
```

```python
# Example 2
x, y = 1, 0
result = divide(x, y)
if result is None:
    print('Invalid inputs')
else:
    print('Result is %.1f' % result)


# Example 3
x, y = 0, 5
result = divide(x, y)
if not result:
    print('Invalid inputs')  # This is wrong!
else:
    assert False
```

  - 빈 문자열, 빈 리스트, 0 이 모두 암시적으로 false로 평가됨 (betterWay4. 에서..)
  - None이 아닌 경우에도 false로 검사될 수 있어서 문제가 됨

BetterWay14. None을 반환하기 보다는 예외를 일으키자

- 개선 1. 반환 값을 두 개로 나눠서 튜플에 담기
  - (작업 성공 여부, 계산된 실제 결과)

```
# Example 4
def divide(a, b):
    try:
        return True, a / b
    except ZeroDivisionError:
        return False, None


# Example 5
x, y = 5, 0
success, result = divide(x, y)
if not success:
    print('Invalid inputs')
```

  - But 호출자가 튜플의 첫 번째 부분을 쉽게 무시할 수 있음.

(파이썬에서 사용하지 않을 변수에 붙이는 관례인 밑줄 변수 이름을 사용해서)

➔ None 반환 만큼이나 나쁨

```
# Example 6
x, y = 5, 0
_, result = divide(x, y)
if not result:
    print('Invalid inputs')  # This is right

x, y = 0, 5
_, result = divide(x, y)
if not result:
    print('Invalid inputs')  # This is wrong
```

BetterWay14. None을 반환하기 보다는 예외를 일으키자

- 개선2. 절대로 None을 반환하지 않기
  - 대신에 호출하는 쪽에 예외 일으켜서 호출하는 쪽에서 예외를 처리하도록 함

```python
# Example 7
def divide(a, b):
    try:
        return a / b
    except ZeroDivisionError as e:
        raise ValueError('Invalid inputs') from e
```

```python
# Example 8
x, y = 5, 2
try:
    result = divide(x, y)
except ValueError:
    print('Invalid inputs')
else:
    print('Result is %.1f' % result)
```

BetterWay15.클로저가 변수 스코프와 상호작용하는 방법을 알자

- 숫자 리스트를 정렬할 때 특정 그룹의 숫자들이 먼저 오도록 정렬

- *key* 매개 변수의 값은 단일 인자를 취하고 정렬 목적으로 사용할 키를 반환하는 함수여야 함

```
# Example 1
def sort_priority(values, group):
    def helper(x):
        if x in group:
            return (0, x)
        return (1, x)
    values.sort(key=helper)
```

```
# Example 2
numbers = [8, 3, 1, 2, 5, 4, 7, 6]
group = {2, 3, 5, 7}
sort_priority(numbers, group)
print(numbers)
```

```
[2, 3, 5, 7, 1, 4, 6, 8]
```

BetterWay15.클로저가 변수 스코프와 상호작용하는 방법을 알자

- 동작하는 이유

1. 클로저 closure : 자신이 정의된 스코프에 있는 변수를 참조하는 함수
    1. Helper함수가 sort_priprity의 group인수에 접근할 수 있게 해줌

2. 함수는 파이썬에서 first-class object임.
    1. 함수를 직접 참조하고, 변수에 할당하고, 다른 함수의 인수로 전달하고, 표현식과 if문 등에서 비교할 수 있음

3. 파이썬에서 튜플을 비교하는 특정한 규칙
    1. 인덱스 0으로 아이템을 비교하고 그 다음으로 인덱스 1, 다음은 인덱스 2와 같이 진행
    2. 이 때문에 Helper 클로저의 반환값이 정렬 순서를 분리된 두 그룹으로 나뉘게 함

```python
# Example 1
def sort_priority(values, group):
    def helper(x):
        if x in group:
            return (0, x)
        return (1, x)
    values.sort(key=helper)


# Example 2
numbers = [8, 3, 1, 2, 5, 4, 7, 6]
group = {2, 3, 5, 7}
sort_priority(numbers, group)
print(numbers)
```

BetterWay15.클로저가 변수 스코프와 상호작용하는 방법을 알자

- 이전의 함수에서 우선순위가 높은 아이템을 발견했는지 여부 반환하도록 한다면
  - Boolean 타입 변수를 추가해서 동작시킨다면?

```
# 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


# Example 4
found = sort_priority2(numbers, group)
print('Found:', found)
print(numbers)
```

```
Found: False
[2, 3, 5, 7, 1, 4, 6, 8]
```

←제대로 된 결과X

BetterWay15.클로저가 변수 스코프와 상호작용하는 방법을 알자

- 결과가 잘못된 이유 / 파이썬 scope 탐색 순서
  1. 현재 함수의 스코프
  2. 감싸고 있는 스코프
  3. 코드를 포함하고 있는 모듈의 스코프 (전역 스코프라고도 함)
  4. Len이나 str같은 함수를 담고 있는 내장 스코프

  찾지 못하면 NameError 예외 발생

- 변수 값 할당 시
  - 변수가 이미 스코프에 정의되어 있다면 값 새로 얻음
  - 현재 스코프에 존재하지 않으면 새로운 변수 정의로 취급

    새 변수의 스코프는 할당된 함수

```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


# Example 4
found = sort_priority2(numbers, group)
print('Found:', found)
print(numbers)
```

←새 변수

BetterWay15.클로저가 변수 스코프와 상호작용하는 방법을 알자

- Python3에 클로저에서 데이터 얻어오는 문법
- nonlocal : 특정 변수 이름에 할당 할 때 스코프 탐색이 일어나야 함을 나타냄
  - 모듈 수준의 스코프까지는 탐색할 수 없음(전역변수의 오염을 피하기 위해)

  - 클로저에서 데이터를 다른 스코프에 할당하는 시점 알아보기 쉽게 해줌
  - 변수 할당이 모듈 스코프에 직접 들어가게 하는 global문을 보완

  - But 간단한 함수 이외에는 사용하지 말아야 함. 함수가 길어지면 변수 할당과 멀어져서 이해하기 어려움

```python
# Example 6
def sort_priority3(numbers, group):
    found = False
    def helper(x):
        nonlocal found
        if x in group:
            found = True
            return (0, x)
        return (1, x)
    numbers.sort(key=helper)
    return found
```

```python
# Example 7
found = sort_priority3(numbers, group)
print('Found:', found)
print(numbers)
```

```
Found: True
[2, 3, 5, 7, 1, 4, 6, 8]
```

BetterWay15.클로저가 변수 스코프와 상호작용하는 방법을 알자

- Python3

- 복잡해지면 헬퍼 클래스로 상태를 감싸는 방법 이용

```
# Example 8
class Sorter(object):
    def __init__(self, group):
        self.group = group
        self.found = False

    def __call__(self, x):
        if x in self.group:
            self.found = True
            return (0, x)
        return (1, x)

sorter = Sorter(group)
numbers.sort(key=sorter)
assert sorter.found is True
print('Found:', found)
print(numbers)
```

```
Found: True
[2, 3, 5, 7, 1, 4, 6, 8]
```

BetterWay15.클로저가 변수 스코프와 상호작용하는 방법을 알자

- Python2 의 스코프
- Nonlocal 지원X / 깔끔하진 않지만 다른 방법으로..

- Found변수가 어디서 참조되었는지 상위 스코프로 탐색하는 거 이용

```
# Example 9
def sort_priority(numbers, group):
    found = [False]
    def helper(x):
        if x in group:
            found[0] = True
            return (0, x)
        return (1, x)
    numbers.sort(key=helper)
    return found[0]

numbers = [8, 3, 1, 2, 5, 4, 7, 6]
group = set([2, 3, 5, 7])
found = sort_priority(numbers, group)
print('Found:', found)
print(numbers)
```

BetterWay16.리스트를 반환하는 대신 제너레이터를 고려하자

- 일련의 결과를 생성하는 함수
- 가장 간단한 방법 -> 아이템의 리스트 반환

  but 두가지 문제가 있음

문제1. 코드 복잡, 깔끔하지 않음

문제2. 반환하기 전에 모든 결과 리스트에 저장해야 해서 메모리 문제

```
# Example 1
def index_words(text):
    result = []
    if text:
        result.append(0)
    for index, letter in enumerate(text):
        if letter == ' ':
            result.append(index + 1)
    return result
```

문자열에서 단어의 인덱스 출력하는 예제→

```
# Example 2
address = 'Four score and seven years ago...'
address = 'Four score and seven years ago our fathers
result = index_words(address)
print(result[:3])
```

```
[0, 5, 11]
```

BetterWay16.리스트를 반환하는 대신 제너레이터를 고려하자

앞의 예제와 동일한 동작하는 함수를 제너레이터로 구현

제너레이터 호출로 반환되는 이터레이터를 내장함수 list에 전달하여 list로 변환

```
# Example 3
def index_words_iter(text):
    if text:
        yield 0
    for index, letter in enumerate(text):
        if letter == ' ':
            yield index + 1


# Example 4
result = list(index_words_iter(address))
print(result[:3])
```

제너레이터 사용할 때 주의

반환되는 이터레이터에 상태가 있고 재사용할 수 없다는 사실을 호출하는 쪽에서 알아야 함

BetterWay17.인수를 순회할 때는 방어적으로 하자

- 리스트를 여러 번 순회해야 할 경우

```python
# Example 1
def normalize(numbers):
    total = sum(numbers)
    result = []
    for value in numbers:
        percent = 100 * value / total
        result.append(percent)
    return result


# Example 2
visits = [15, 35, 80]
percentages = normalize(visits)
print(percentages)
```

```
[11.538461538461538, 26.923076923076923, 61.53846153846154]
```

```python
# Example 3
path = 'my_numbers.txt'
with open(path, 'w') as f:
    for i in (15, 35, 80):
        f.write('%d\n' % i)

def read_visits(data_path):
    with open(data_path) as f:
        for line in f:
            yield int(line)


# Example 4
it = read_visits('my_numbers.txt')
percentages = normalize(it)
print(percentages)
```

```
[]
```

← 파일에서 데이터 읽도록 확장했을 때.

Sum에서 한 번 순회하고 끝나버림. 이터레이터가 결과를 한 번만 생성하기 때문에 이미 stopIteration 예외를 일으킨 이터레이터나 제너레이터를 순회해도 결과를 얻을 수 없음.

BetterWay17. 인수를 순회할 때는 방어적으로 하자

- 이터레이터를 소진했는지 알 수 없어서 문제. 입력 이터레이터를 방어적으로 복사하여 해결.

```
# Example 6
def normalize_copy(numbers):  numbers: <class 'list'>: [15, 35, 80]
    numbers = list(numbers)  # Copy the iterator
    total = sum(numbers)  total: 130
    result = []  result: <class 'list'>: [11.538461538461538, 26.923076923076923, 61.53846153846154]
    for value in numbers:  value: 80
        percent = 100 * value / total  percent: 61.53846153846154
        result.append(percent)
    return result


# Example 7
it = read_visits('my_numbers.txt')
percentages = normalize_copy(it)
print(percentages)


[11.538461538461538, 26.923076923076923, 61.53846153846154]
```

BetterWay17. 인수를 순회할 때는 방어적으로 하자

- But 입력받은 이터레이터 콘텐츠의 복사본이 클 수도 있음

  ➔ 호출 될 때마다 새 이터레이터를 반환하는 함수를 받게 만들어서 해결

```python
# Example 8
def normalize_func(get_iter):
    total = sum(get_iter())    # New iterator
    result = []
    for value in get_iter():   # New iterator
        percent = 100 * value / total
        result.append(percent)
    return result


# Example 9
percentages = normalize_func(lambda: read_visits(path))
print(percentages)
```

```python
def read_visits(data_path):
    with open(data_path) as f:
        for line in f:
            yield int(line)
```

BetterWay17.인수를 순회할 때는 방어적으로 하자

- 앞의 예제보다 더 좋은 방법, 이터레이터 프로토콜을 구현한 새 컨테이너 클래스
  - 이터레이터 프로토콜은 파이썬의 for 루프와 관련 표현식이 컨테이너 타입의 콘텐츠를 탐색하는 방법을 나타냄.
    - 파이썬의 for x in foo 같은 문장을 만나면 실제로는 iter(foo)를 호출함.
    - 그러면 내장함수는 iter라는 특별한 메서드인 foo.__iter__를 호출.
    - __iter__메서드는 이터네이터 객체를 반환해야 함.
    - 마지막으로 for루프는 이터레이터를 모두 소진할 때까지 이터레이터 객체에 내장 함수 next를 계속 호출

단점 : 입력 데이터를 여러 번 읽음

```python
# Example 10
class ReadVisit(object):
    def __init__(self, data_path):
        self.data_path = data_path

    def __iter__(self):
        with open(self.data_path) as f:
            for line in f:
                yield int(line)
```

```python
# Example 11
visits = ReadVisit(path)
percentages = normalize(visits)
print(percentages)
```

```python
# Example 1
def normalize(numbers):
    total = sum(numbers)
    result = []
    for value in numbers:
        percent = 100 * value / total
        result.append(percent)
    return result
```

BetterWay17. 인수를 순회할 때는 방어적으로 하자

- 파라미터가 단순한 이터레이터가 아님을 보장하는 함수 작성
  - 프로토콜에 따르면 내장함수 iter에 이터레이터를 넘기면 이터레이터 자체가 반환됨. But iter에 컨테이너 타입을 넘기면 매번 새로운 이터레이터 객체가 반환됨.
  - 입력 이터레이터 전체를 복사하고 싶지는 않지만, 입력 데이터를 여러 번 순회해야 할 때 사용

```python
# Example 12
def normalize_defensive(numbers):
    if iter(numbers) is iter(numbers):  # An iterator -- bad!
        raise TypeError('Must supply a container')
    total = sum(numbers)
    result = []
    for value in numbers:
        percent = 100 * value / total
        result.append(percent)
    return result
```

BetterWay18. 가변 위치 인수로 깔끔하게 보이게 하자

- 선택적 위치인수 (*args)를 받게 만들면 함수 호출을 더 명확하게 할 수 있음

- 로그를 남기는 예제
  - 로그에 넘길 값이 없을 때 빈 리스트를 넘기기 불편하므로..

```
# Example 1
def log(message, values):
    if not values:
        print(message)
    else:
        values_str = ', '.join(str(x) for x in values)
        print('%s: %s' % (message, values_str))

log('My numbers are', [1, 2])
log('Hi there', [])
```

```
My numbers are: 1, 2
Hi there
```

```
# Example 2
def log(message, *values):  # The only difference
    if not values:
        print(message)
    else:
        values_str = ', '.join(str(x) for x in values)
        print('%s: %s' % (message, values_str))

log('My numbers are', 1, 2)
log('Hi there')  # Much better
```

```
# Example 3
favorites = [7, 33, 99]
log('Favorite colors', *favorites)
```

BetterWay18. 가변 위치 인수로 깔끔하게 보이게 하자

- 가변개수의 위치 인수를 받는 방법의 문제점 두가지

1. 가변 인수가 함수에 전달되기 전에 항상 튜플로 변환됨
    1. 함수를 호출하는 쪽에서 제너레이터에 *연산자를 쓰면 제너레이터가 모두 소진될때까지 순회 ➔ 메모리 문제
    2. *args는 입력의 수가 적을 때 쓰기 좋음
    3. 많은 리터럴이나 변수 이름을 한꺼번에 넘기는 함수 호출에 이상적임.

```
# Example 4
def my_generator():
    for i in range(10):
        yield i

def my_func(*args):  args: <class 'tuple'>: (0, 1, 2, 3, 4, 5, 6, 7, 8, 9)
    print(args)

it = my_generator()
my_func(*it)
```

BetterWay18. 가변 위치 인수로 깔끔하게 보이게 하자

- 가변개수의 위치 인수를 받는 방법의 문제점 두가지

2. 추후에 코드를 모두 변경하지 않고서는 새 위치 인수를 추가할 수 없음

```
# Example 5
def log(sequence, message, *values):
    if not values:
        print('%s: %s' % (sequence, message))
    else:
        values_str = ', '.join(str(x) for x in values)
        print('%s: %s: %s' % (sequence, message, values_str))

log(1, 'Favorites', 7, 33)      # New usage is OK
log('Favorite numbers', 7, 33)  # Old usage breaks
```

```
1: Favorites: 7, 33
Favorite numbers: 7: 33
```

BetterWay19.키워드 인수로 선택적인 동작을 제공하자

- 파이썬에서 파라미터를 **위치**로 넘길 수 있고, **키워드**로도 전달 가능

```
# Example 2
remainder(20, 7)
remainder(20, divisor=7)
remainder(number=20, divisor=7)
remainder(divisor=7, number=20)

remainder(number=20, 7)

# Ex
try:
    remainder(20, number=7)
```

Positional argument after keyword argument

- 키워드 파라미터의 이점

1. 함수호출을 더 명확하게 이해할 수 있음
2. 함수 정의 시 기본값 설정 가능
3. 기존의 호출 코드와 호환성을 유지하면서, 함수의 파라미터 확장 가능

BetterWay19.키워드 인수로 선택적인 동작을 제공하자

2. 함수 정의 시 기본값 설정 가능

    기본값이 간단할 때 잘 동작.

```python
# Example 8
def flow_rate(weight_diff, time_diff, period=1):
    return (weight_diff / time_diff) * period


# Example 9
flow_per_second = flow_rate(weight_diff, time_diff)

flow_per_hour = flow_rate(weight_diff, time_diff, period=3600)
```

BetterWay19.키워드 인수로 선택적인 동작을 제공하자

3. 기존의 호출 코드와 호환성을 유지하면서, 함수의 파라미터 확장 가능

선택적인 키워드 인수를 여전히 위치 인수로 넘길 수있음. 하지만 혼동일으킬 수 있으니 항상 키워드 이름으로만 사용하는게 좋음.

```
# Example 10
def flow_rate(weight_diff, time_diff,
              period=1, units_per_kg=1):
    return ((weight_diff * units_per_kg) / time_diff) * period


# Example 11
pounds_per_hour = flow_rate(weight_diff, time_diff,
                            period=3600, units_per_kg=2.2)
print(pounds_per_hour)
assert pounds_per_hour == 1320.0


# Example 12
pounds_per_hour = flow_rate(weight_diff, time_diff, 3600, 2.2)
print(pounds_per_hour)
assert pounds_per_hour == 1320.0
```

BetterWay20. 동적 기본 인수를 지정하려면 None과 docstring을 사용하자

- 키워드 인수의 기본값을 non-static 타입을 사용할 때

- 함수 호출한 시각을 출력하려는 예제
  - 인수가 수정 가능할 때 None 사용하는게 중요

```
def log(message, when=datetime.now()):
    print('%s: %s' % (when, message))

log('Hi there!')
sleep(0.1)
log('Hi again!')
```

```
2020-01-13 21:47:39.273366: Hi there!
2020-01-13 21:47:39.273366: Hi again!
```

```
# Example 2
def log(message, when=None):
    """Log a message with a timestamp.

    Args:
        message: Message to print.
        when: datetime of when the message occurred.
            Defaults to the present time.
    """
    when = datetime.now() if when is None else when
    print('%s: %s' % (when, message))


# Example 3
log('Hi there!')
sleep(0.1)
log('Hi again!')
```

```
2020-01-13 21:47:39.374060: Hi there!
2020-01-13 21:47:39.474083: Hi again!
```

BetterWay20. 동적 기본 인수를 지정하려면 None과 docstring을 사용하자

- 예제2. 디코딩 실패 시 빈 딕셔너리 반환하고자 할 때

```
def decode(data, default={}):  data: 'also bad'  default: <class 'dict'>: {'stuff': 5}
    try:
        return json.loads(data)
    except ValueError:
        return default


# Example 5
foo = decode('bad data')
foo['stuff'] = 5
bar = decode('also bad')
bar['meep'] = 1
print('Foo:', foo)
print('Bar:', bar)


# Example 6
assert foo is bar
```

```
# Example 7
def decode(data, default=None):
    """Load JSON data from a string.

    Args:
        data: JSON data to decode.
        default: Value to return if decoding fails.
            Defaults to an empty dictionary.
    """
    if default is None:
        default = {}
    try:
        return json.loads(data)
    except ValueError:
        return default


# Example 8
foo = decode('bad data')
foo['stuff'] = 5
bar = decode('also bad')
bar['meep'] = 1
print('Foo:', foo)
print('Bar:', bar)
```

BetterWay21. 키워드 전용 인수로 명료성을 강요하자

- 키워드 인수
  - 여전히 호출하는 쪽에서 키워드 인수로 의도를 명확히 드러내라고 강요할 방법이 없음.

```python
# Example 4
def safe_division_b(number, divisor,
                    ignore_overflow=False,
                    ignore_zero_division=False):
    try:
        return number / divisor
    except OverflowError:
        if ignore_overflow:
            return 0
        else:
            raise
    except ZeroDivisionError:
        if ignore_zero_division:
            return float('inf')
        else:
            raise


# Example 5
assert safe_division_b(1.0, 10**500, ignore_overflow=True) is 0
assert safe_division_b(1.0, 0, ignore_zero_division=True) == float('inf')


# Example 6
assert safe_division_b(1.0, 10**500, True, False) is 0
```

BetterWay21. 키워드 전용 인수로 명료성을 강요하자

- 키워드 전용 인수 / python3

```python
# Example 7
def safe_division_c(number, divisor, *,
                    ignore_overflow=False,
                    ignore_zero_division=False):
    try:
        return number / divisor
    except OverflowError:
        if ignore_overflow:
            return 0
        else:
            raise
    except ZeroDivisionError:
        if ignore_zero_division:
            return float('inf')
        else:
            raise
```

```python
# Example 8
try:
    safe_division_c(1.0, 10**500, True, False)
except:
    logging.exception('Expected')
else:
    assert False


# Example 9
safe_division_c(1.0, 0, ignore_zero_division=True)  # No exception
try:
    safe_division_c(1.0, 0)
    assert False
except ZeroDivisionError:
    pass  # Expected
```

BetterWay21. 키워드 전용 인수로 명료성을 강요하자

- Python2 키워드 전용 인수
  - 명시적 문법이 없음.
  - 인수 리스트에 **연산자 사용

```
# Example 10
def print_args(*args, **kwargs):
    print 'Positional:', args
    print 'Keyword:   ', kwargs

print_args(1, 2, foo='bar', stuff='meep')

>>>
Positional: (1, 2)
Keyword: {'foo' : 'bar', 'stuff' : 'meep'}
```

BetterWay21. 키워드 전용 인수로 명료성을 강요하자

- Python2 키워드 전용 인수

```
# Example 11
def safe_division_d(number, divisor, **kwargs):
    ignore_overflow = kwargs.pop('ignore_overflow', False)
    ignore_zero_div = kwargs.pop('ignore_zero_division', False)
    if kwargs:
        raise TypeError('Unexpected **kwargs: %r' % kwargs)
    try:
        return number / divisor
    except OverflowError:
        if ignore_overflow:
            return 0
        else:
            raise
    except ZeroDivisionError:
        if ignore_zero_div:
            return float('inf')
        else:
            raise
```

```
assert safe_division_d(1.0, 10) == 0.1
assert safe_division_d(1.0, 0, ignore_zero_division=True) == float('inf')
assert safe_division_d(1.0, 10**500, ignore_overflow=True) is 0

# Example 12
try:
    safe_division_d(1.0, 0, False, True)
except:
    logging.exception('Expected')
else:
    assert False


# Example 13
try:
    safe_division_d(0.0, 0, unexpected=True)
except:
    logging.exception('Expected')
else:
    assert False
```

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