# 예제로 살펴보는 PEP20

**출처** https://artifex.org/~hblanks/talks/2011/pep20_by_example.pdf

**한글로 옮긴이** https://www.linkedin.com/in/danielyounghokim/

# 아름다운 것이 추한 것보다 낫다.
## Beautiful is better than ugly.

**Ugly**

```
halve_evens_only = lambda nums: map(lambda i: i/2, filter(lambda i: not i%2, nums))
```

**Beautiful**

```
def halve_evens_only(nums):
    return [i/2 for i in nums if not i % 2]
```

# 명시적인 것이 암시적인 것보다 낫다.
## Explicit is better than implicit.

**Implicit**

```
def load():
    from menagerie.cat.models import *
    from menagerie.dog.models import *
    from menagerie.mouse.models import *
```

---

**Explicit**

```
def load():
    from menagerie.models import cat as cat_models
    from menagerie.models import dog as dog_models
    from menagerie.models import mouse as mouse_models
```

# 단순한 것이 복합적인 것보다 낫다.
## Simple is better than complex.

```
measurements = [
    {'weight': 392.3, 'color': 'purple', 'temperature': 33.4},
    {'weight': 34.0, 'color': 'green', 'temperature': -3.1},
    ]
```

**Complex**

```
def store(measurements):
    import sqlalchemy
    import sqlalchemy.types as sqltypes

    db = sqlalchemy.create_engine('sqlite:///measurements.db')
    db.echo = False
    metadata = sqlalchemy.MetaData(db)
    table = sqlalchemy.Table('measurements', metadata,
        sqlalchemy.Column('id', sqltypes.Integer, primary_key=True),
        sqlalchemy.Column('weight', sqltypes.Float),
        sqlalchemy.Column('temperature', sqltypes.Float),
        sqlalchemy.Column('color', sqltypes.String(32)),
        )
    table.create(checkfirst=True)

    for measurement in measurements:
        i = table.insert()
        i.execute(**measurement)
```

**Simple**

```
def store(measurements):
    import json
    with open('measurements.json', 'w') as f:
        f.write(json.dumps(measurements))
```

# 복합적인 것이 얽힌 것보다 낫다.
## Complex is better than complicated.

### Complicated

```
def store(measurements):
    import sqlalchemy
    import sqlalchemy.types as sqltypes

    db = create_engine(
        'mysql://user:password@localhost/db?charset=utf8&use_unicode=1')
    db.echo = False
    metadata = sqlalchemy.MetaData(db)
    table = sqlalchemy.Table('measurements', metadata,
        sqlalchemy.Column('id', sqltypes.Integer, primary_key=True),
        sqlalchemy.Column('weight', sqltypes.Float),
        sqlalchemy.Column('temperature', sqltypes.Float),
        sqlalchemy.Column('color', sqltypes.String(32)),
        )
    table.create(checkfirst=True)

    for measurement in measurements:
        i = table.insert()
        i.execute(**measurement)
```

### Complex

```
def store(measurements):
    import MySQLdb
    db = MySQLdb.connect(user='user', passwd="password", host='localhost', db="db")

    c = db.cursor()
    c.execute("""
        CREATE TABLE IF NOT EXISTS measurements
          id int(11) NOT NULL auto_increment,
          weight float,
          temperature float,
          color varchar(32)
          PRIMARY KEY id
          ENGINE=InnoDB CHARSET=utf8
          """)

    insert_sql = (
        "INSERT INTO measurements (weight, temperature, color) "
        "VALUES (%s, %s, %s)")

    for measurement in measurements:
        c.execute(insert_sql,
            (measurement['weight'], measurement['temperature'], measurement['color'])
            )
```

# 평평한 것이 중첩된 것보다 낫다.
## Flat is better than nested.

**Nested**

```
def identify(animal):
    if animal.is_vertebrate():
        noise = animal.poke()
        if noise == 'moo':
            return 'cow'
        elif noise == 'woof':
            return 'dog'
    else:
        if animal.is_multicellular():
            return 'Bug!'
        else:
            if animal.is_fungus():
                return 'Yeast'
            else:
                return 'Amoeba'
```

**Flat**

```
def identify(animal):
    if animal.is_vertebrate():
        return identify_vertebrate()
    else:
        return identify_invertebrate()

def identify_vertebrate(animal):
    noise = animal.poke()
    if noise == 'moo':
        return 'cow'
    elif noise == 'woof':
        return 'dog'

def identify_invertebrate(animal):
    if animal.is_multicellular():
        return 'Bug!'
    else:
        if animal.is_fungus():
            return 'Yeast'
        else:
            return 'Amoeba'
```

# 성긴 것이 빽빽한 것보다 낫다.
## Sparse is better than dense.

**Dense**

```
def process(response):
    selector = lxml.cssselect.CSSSelector('#main > div.text')
    lx = lxml.html.fromstring(response.body)
    title = lx.find('./head/title').text
    links = [a.attrib['href'] for a in lx.find('./a') if 'href' in a.attrib]
    for link in links:
        yield Request(url=link)
    divs = selector(lx)
    if divs: yield Item(utils.lx_to_text(divs[0]))
```

**Sparse**

```
def process(response):
    lx = lxml.html.fromstring(response.body)

    title = lx.find('./head/title').text

    links = [a.attrib['href'] for a in lx.find('./a') if 'href' in a.attrib]
    for link in links:
        yield Request(url=link)

    selector = lxml.cssselect.CSSSelector('#main > div.text')
    divs = selector(lx)
    if divs:
        bodytext = utils.lx_to_text(divs[0])
        yield Item(bodytext)
```

# 가독성이 중요하다.
## Readability counts.

```python
def factorial(n):
    """
    Return the factorial of n, an exact integer >= 0.

    >>> [factorial(n) for n in range(6)]
    [1, 1, 2, 6, 24, 120]

    >>> factorial(30)
    265252859812191058636308480000000L

    >>> factorial(-1)
    Traceback (most recent call last):
        ...
    ValueError: n must be >= 0
    """
    pass

if __name__ == '__main__' and '--test' in sys.argv:
    import doctest
    doctest.testmod()
```

```python
import unittest

def factorial(n):
    pass

class FactorialTests(unittest.TestCase):
    def test_ints(self):
        self.assertEqual(
            [factorial(n) for n in range(6)], [1, 1, 2, 6, 24, 120])

    def test_long(self):
        self.assertEqual(
            factorial(30), 265252859812191058636308480000000L)

    def test_negative_error(self):
        with self.assertRaises(ValueError):
            factorial(-1)

if __name__ == '__main__' and '--test' in sys.argv:
    unittest.main()
```

# 실용성이 순수함보다 중요하더라도
# 특수한 경우는 규칙을 깰만큼 특별하지 않다.
# Special cases aren't special enough to break the rules.
# Although practicality beats purity.

```
def make_counter():
    i = 0
    def count():
        """ Increments a count and returns it. """
        i += 1
        return i
    return count

count = make_counter()
assert hasattr(count, '__name__') # No anonymous functions!
assert hasattr(count, '__doc__')


assert float('0.20000000000000007') == 1.1 - 0.9 # (this is platform dependent)
assert 0.2 != 1.1 - 0.9 # Not special enough to break the rules of floating pt.
assert float(repr(1.1 - 0.9)) == 1.1 - 0.9
```

```
def make_adder(addend):
    return lambda i: i + addend # But lambdas, once in a while, are practical.

assert str(1.1 - 0.9) == '0.2' # as may be rounding off floating point errors
assert round(0.2, 15) == round(1.1 - 0.9, 15)
```

명시적으로 조용히 하게 만들지 않았다면
오류들은 절대로 조용하게 넘어가면 안 된다.
Errors should never pass silently
Unless explicitly silenced.

```
try:
    import json
except ImportError:
    try:
        import simplejson as json
    except:
        print 'Unable to find json module!'
        raise
```

애매함과 마주할 땐, 추측하려는 유혹을 떨쳐라.

**In the face of ambiguity, refuse the temptation to guess.**

```
def process(response):
    db.store(url, response.body)
```

---

```
def process(response):
    charset = detect_charset(response)
    db.store(url, response.body.decode(charset))
```

여러분이 네덜란드 사람[1]이 아니라면 이런 방식이 처음엔 분명하지 않을 것이지만,
특정 일을 할 때는 한 가지 방법만으로 해야 한다.

**There should be one, and preferably only one way to do it.**
**Although that way may not be obvious at first unless you're Dutch.**

```
def fibonacci_generator():
    prior, current = 0, 1
    while current < 100:
        yield prior + current
        prior, current = current, current + prior

sequences = [
    range(20),
    {'foo': 1, 'fie': 2},
    fibonacci_generator(),
    (5, 3, 3)
    ]

for sequence in sequences:
    for item in sequence: # all sequences iterate the same way
        pass
```

1 파이썬 창시자(Guido van Rossum)는 네덜란드 사람.

종종 결코 하지 않는 것이 *지금 당장*하는 것보다 낫기도 하지만
지금 하는 것이 결코 하지 않는 것보다 낫다.

Now is better than never.

Although never is often better than *right* now.

```
def obsolete_func():
    raise PendingDeprecationWarning

def deprecated_func():
    raise DeprecationWarning
```

구현한 것을 설명하기 어렵다면, 안 좋은 구현이다.
구현한 것을 설명하기 쉽다면, 좋은 구현일 것이다.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.

```
def hard():

    # Example 1
    try:
        import twisted
        help(twisted) # (this may not be as hard as I think, though)
    except:
        pass

    # Example 2
    import xml.dom.minidom
    document = xml.dom.minidom.parseString(
        '''<menagerie><cat>Fluffers</cat><cat>Cisco</cat></menagerie>''')
    menagerie = document.childNodes[0]
    for node in menagerie.childNodes:
        if node.childNodes[0].nodeValue== 'Cisco' and node.tagName == 'cat':
            return node
```

```
def easy(maybe):

    # Example 1
    try:
        import gevent
        help(gevent)
    except:
        pass

    # Example 2
    import lxml
    menagerie = lxml.etree.fromstring(
        '''<menagerie><cat>Fluffers</cat><cat>Cisco</cat></menagerie>''')
    for pet in menagerie.find('./cat'):
        if pet.text == 'Cisco':
            return pet
```