Python Coding Style Guide for KMU-CS Software Project II

About This Document

- 이 문서는, 국민대학교 소프트웨어학부 "소프트웨어 프로젝트 II" 과목의 코딩 실습에서 이용할 목적으로 작성되었음
- 대부분의 내용은 공개되어 있는 "Google Python Style Guide" (https://google.github.io/styleguide/pyguide.html) 문서를 참고하여 만들어졌음
- version 0.1 (2017-08-14), 이시윤
- version 0.2 (2017-08-21), 이시윤

Python Language Rules

- Imports
  - `import` 문장은 패키지 (package) 와 모듈 (module) 에 대해서만 사용한다.
  - 모듈 내에 포함된 객체를 표현할 때에는 `x.obj` (`x` 는 모듈, `obj` 는 그 모듈 내에서 정의된 객체) 와 같이 표현한다.
- Packages
  - 모듈을 `import` 할 때에는 해당 모듈에 대한 완전한 경로명을 이용한다.

    ```
    # 완전한 경로명을 이용
    import sound.effects.echo
    # 모듈 이름만을 사용하려는 경우 (선호)
    from sound.effects import echo
    ```
- Global variables
  - 전역 변수 (global variable) 의 사용은 가급적 피한다.
  - 예외:
    - 스크립트에 대한 디폴트 옵션을 나타낼 때
    - 모듈 수준에서의 상수를 정의할 때 (예: `PI = 3.14159`): 상수의 이름은 밑줄 (underscore; `_`) 을 포함할 수 있는, 대문자로만 이루어진 이름을 이용한다.
- List comprehensions
  - 단순한 (쉽게 이해되는) 경우에 한하여 사용한다.
  - Yes:

```
result = []
for x in range(10):
    for y in range(5):
        if x * y > 10:
            result.append((x, y))

for x in xrange(5):
    for y in xrange(5):
        if x != y:
            for z in xrange(5):
                if y != z:
                    yield (x, y, z)

return ((x, complicated_transform(x))
         for x in long_generator_function(parameter)
         if x is not None)

squares = [x * x for x in range(10)]

eat(jelly_bean for jelly_bean in jelly_beans
    if jelly_bean.color == 'black')
```

- No:

```
result = [(x, y) for x in range(10) for y in range(5) if x * y > 10]

return ((x, y, z)
        for x in xrange(5)
        for y in xrange(5)
        if x != y
        for z in xrange(5)
        if y != z)
```

- Lambda functions
  - 한 줄에 표현 가능한 경우에 대해서만 사용한다.
  - 경우에 따라 읽고 이해하기가 어렵기 때문에, 복잡한 경우에 대해서는 가급적 사용하지 않도록 한다.
- Ternary conditional expressions
  - 한 줄에 표현 가능한 단순한 경우에 대해서만 사용한다 (예: `x = 1 if cond else 2`).
  - 복잡해서 읽고 이해하기가 어려울 것이 예상되는 경우에 대해서는 `if` 문장을 대신 사용한다.
- True/False evaluations
  - 가능한 경우, 묵시적으로 `False` 로 간주되는 객체들을 이용한다 (예: `0`, `None`, `[]`, `{}`, `''` 등).

- 예: `if foo != []:` 로 표현하는 대신에 `if foo:` 로 표현하는 것을 선호
- `None` 과의 비교에 있어서는 `==` 또는 `!=` 를 사용하지 않고 `is` 또는 `is not` 을 사용한다.
- 어떠한 경우에도 Boolean 값을 `False` 와 `==` 를 이용하여 비교하지 않는다. 대신 `if not x:` 와 같은 구문을 사용한다.
- 문자열, 리스트, 순서쌍 등이 비어 있는지를 판단할 때에는 길이를 이용하기보다는 빈 객체가 묵시적으로 `False` 로 취급된다는 사실을 이용한다. 예를 들어, `if len(seq):` 보다는 `if seq:` 를, `if len(seq) == 0:` 보다는 `if not seq:` 를 이용한다.

Python Style Rules

- Semicolons
  - 행의 마지막을 세미콜론 (semicolon) 으로 끝내거나, 두 문장을 한 행에 표현하기 위하여 세미콜론으로 병치하지 않는다.
- Line length
  - 한 행의 최대 길이는 80 글자로 제한한다.
  - 예외:
    - 긴 import 문장
    - 주석에 포함된 URL
  - 한 행을 두 행 이상에 나누어 표현하기 위하여 역슬래시 (backslash; `\`) 를 이용하지 않는다.
  - 예:

```
foo_bar(self, width, height, color='black', design=None, x='foo',
        emphasis=None, highlight=0)

if (width == 0 and height == 0 and
    color == 'red' and emphasis == 'strong'):
```

    - 문자열 상수 (string literal) 가 한 행에 표현 불가능한 경우에는 두 행 이상에 나누어 표현하되, 이 문자열 상수의 앞과 뒤를 괄호로 묶어서 잘 드러나도록 표현한다.

```
x = ('This will build a very long long '
     'long long long long long long string.')
```

- Parentheses
  - 불필요한 괄호를 가급적 사용하지 않는다.
  - Yes:

```
if foo:
    bar()
while x:
    x = bar()
if x and y:
    bar()
if not x:
    bar()
return foo
for (x, y) in dict.items():
    ...
```

- No:

```
if (x):
    bar()
if not (x):
    bar()
return (foo)
```

- Indentation
  - 코드 블록 (code block) 의 들여쓰기는 4 개의 공백 (white space) 문자를 이용한다.
  - 탭 (tab) 은 이용하지 않는다. 더욱이, 탭과 공백을 섞어서 이용하지 않는다.
  - 코드의 이해를 돕기 위하여, 줄바꿈을 포함한 표현에서는 세로로 칸을 맞추어 쓰는 것을 권장한다.
  - Yes:

```
# Aligned with opening delimiter
foo = long_function_name(var_one, var_two,
                         var_three, var_four)

# Aligned with opening delimiter in a dictionary
foo = {
    long_dictionary_key: value1 +
                         value2,
    ...
}

# 4-space hanging indent; nothing on first line
foo = long_function_name(
    var_one, var_two, var_three,
    var_four)

# 4-space hanging indent in a dictionary
foo = {
    long_dictionary_key:
        long_dictionary_value,
    ...
}
```

- No:

```
# Stuff on first line forbidden
foo = long_function_name(var_one, var_two,
    var_three, var_four)

# 2-space hanging indent forbidden
foo = long_function_name(
  var_one, var_two, var_three,
  var_four)
```

- Blank lines
  - 최상위 수준 정의 (top-level definition) 의 사이에는 두 줄을 비우고, 메서드 (method) 정의 사이에는 한 줄을 비운다.
    - 최상위 수준에서 정의된 클래스 및 함수에 대해서는 시작 이전에 두 개의 빈 줄을 둔다.
    - 최상위 수준에서 정의된 클래스에 포함된 메서드의 시작 이전에는 한 개의 빈 줄을 둔다.
    - 함수나 메서드 안에서는 읽는 데 도움이 되고 필요하다고 판단되는 경우에 대해서 한 개의 빈줄을 두는 것을 허용한다.
- White spaces
  - 구두점 및 괄호 등의 주변에 두는 공백 문자는 상식을 따른다.

- 괄호, 대괄호 (brackets; `[]`), 중괄호 (braces; `{}`) 의 안쪽에 불필요한 공백을 두지 않는다.
  - Yes:

```
spam(ham[1], {eggs: 2}, [])
```

  - No:

```
spam( ham[ 1 ], { eggs: 2 }, [ ] )
```

- 콤마 (`,`), 세미콜론 (`;`), 콜론 (`:`) 의 앞에는 공백을 두지 않고, 뒤에는 하나의 공백을 둔다.
  - Yes:

```
if x == 4:
    print x, y
x, y = y, x
```

  - No:

```
if x == 4 :
    print x , y
x , y = y , x
```

- 함수의 인자 나열, 인덱싱 (indexing), 슬라이싱 (slicing) 에 이용되는 괄호/대괄호 열기 앞에는 공백을 두지 않는다.
  - Yes:

```
spam(1)
dict['key'] = list[index]
```

  - No:

```
spam (1)
dict ['key'] = list [index]
```

- 이항 연산자 (binary operator) 의 앞/뒤에는 각각 하나씩의 공백을 둔다.
  - 대입 연산자 (`=`)
  - 비교 연산자 (`==`, `<`, `>`, `!=`, `<>`, `<=`, `>=`, `in`, `not in`, `is`, `is not`)
  - 논리 연산자 (`and`, `or`, `not`)
  - Yes:

```
x == 1
```

- No:

```
x<1
```

- 키워드 인자 (keyword argument) 또는 디폴트 인자 값 (default parameter value) 을 표현할 때에는 `=` 의 앞과 뒤에 공백을 두지 않는다.

  - Yes:

```
def complex(real, imag=0.0):
    return magic(r=real, i=imag)
```

  - No:

```
    def complex(real, imag = 0.0):
        return magic(r = real, i = imag)
```

- 공백 문자를 이용해서 서로 다른 행들 사이에 세로 맞춤을 하지 않는다.

  - Yes:

```
foo = 1000    # comment
long_name = 2    # comment that should not be aligned

dictionary = {
    'foo': 1,
    'long_name': 2,
}
```

  - No:

```
foo          = 1000    # comment
long_name    = 2       # comment that should not be aligned

dictionary = {
    'foo'          : 1,
    'long_name'    : 2,
}
```

- Classes

  - 클래스가 다른 베이스 클래스를 상속하지 않는 경우, 명시적으로 `object` 를 상속함을 표현한다. 중첩된 클래스 (nested class) 에 대해서도 해당한다.

    - Yes:

```
class SampleClass(object):
    pass

class OuterClass(object):

    class InnerClass(object):
        pass

class ChildClass(ParentClass):
    """Explicitly inherits from anoter class already."""
```

    - No:

```
class SampleClass:
    pass

class OuterClass:

    class InnerClass:
        pass
```

- Imports formatting
    - 서로 다른 모듈에 대한 `import` 문장은 개별 행을 차지하도록 한다.
        - Yes:

```
import os
import sys
```

        - No:

```
import os, sys
```

    - `import` 는 항상 모듈 파일의 첫 부분에 위치시키도록 하고, 일반적으로 아래의 순서를 지키도록 한다.

        1. 표준 라이브러리에 포함된 모듈들
        2. 외부 라이브러리 모듈들
        3. 프로젝트 내에서 구현된 모듈들

- Statements
    - 한 행에는 한 문장만을 쓰도록 한다.
    - 단, 읽기에 유리해지는 경우, 조건 판단과 그 조건에 해당하여 실행할 문장을 (한 행에 표현될 수 있는 경우에 한

하여) 동일한 행에 위치시킬 수 있다.

- 한 행에 조건과 실행 문장을 표현하는 것은, `else` 절을 가지는 `if` 문장에 대해서는 허용하지 않는다.
  - Yes:

```
if foo: bar(foo)
```

  - No:

```
if foo: bar(foo)
else:   baz(foo)

try:                    bar(foo)
except ValueError:    baz(foo)

try:
    bar(foo)
except ValueError:    baz(foo)
```

- Naming
  - 일반적으로 통용되는 Python 객체의 이름 붙이기 규칙은 아래와 같다. 단, 이것을 엄격히 따르지는 않기로 한다.
    - Packages: `lower_with_under`
    - Modules: `lower_with_under`
    - Classes: `CapWords`
    - Exceptions: `CapWords`
    - Functions: `lower_with_under()`
    - Global/class constants: `CAPS_WITH_UNDER`
    - Global/class variables: `lower_with_under`
    - Instance variables: `lower_with_under`
    - Method names: `lower_with_under()`
    - Function/method parameters: `lower_with_under`
    - Local variables: `lower_with_under`

Revision History

- 2017-08-14, 처음 버전 작성 (ver. 0.1)
- 2017-08-21, 초기 코멘트 반영 (ver. 0.2)