

Python PEP8

Index

1. 코드 레이아웃
2. 문자열 인용문
3. 표현식 명령문 공백
4. 기타 권장 사항
5. 후행 싹표를 사용해야 하는 경우
6. 주석
7. 네이밍

1. 코드 레이아웃

1. 들여쓰기 규칙 1
2. 들여쓰기 규칙 2
3. Jupyter Notebook에 autopep8 기능 추가
4. 최대 글자수
5. 연산자 전후의 줄바꿈
6. 공백라인
7. 코드 문자에 대한 인코딩
8. imports

1.1 들여쓰기 규칙 1

- 들여쓰기의 기본 단위는 스페이스 4칸을 사용
- `tab`보다는 `space`가 더 선호
- 괄호 안에 코드가 연속적으로 있는 경우 괄호에서 수직정렬
- 괄호 바로 다음부터 개행되는 경우 4칸의 들여쓰기를 사용
- 함수에서 인수가 개행되는 경우 인수를 구분하기 위해 4칸의 들여쓰기를 추가

In []:

괄호 안에 코드가 연속적으로 있는 경우 괄호에서 수직정렬

```
foo = long_function_name(var_one, var_two,  
                          var_three, var_four)
```

괄호 바로 다음부터 개행되는 경우 4칸의 들여쓰기를 사용

```
foo =  
    long_function_name( va  
r_one, var_two,  
var_three, var_four)
```

함수에서 인수가 개행되는 경우 인수를 구분하기 위해 4칸의 들여쓰기를 추가

```
def  
    long_function_name( var_o  
ne, var_two, var_three,  
var_four):  
    print(var_one)
```

1.2 들여쓰기 규칙 2

- 조건문의 조건의 코드의 경우는 수직 정렬 및 추가 4칸 들여쓰기 모두 사용이 가능
-),]의 기호같은 경우에는 마지막줄에 사용될때 들여쓰기를 해도 되고 안해도 됨

```

In [ ]: # 조건문의 조건의 코드의 경우는 수직 정렬 및 추가 4칸 들여쓰기 모두 사용이 가능
if (this_is_one_thing and
    that_is_another_thing):
    do_something()

# autopep8에서는 아래에 있는 코드의 규칙을 따름
if (this_is_one_thing and
    that_is_another_thing):
    do_something()

# ), ]의 기호같은 경우에는 마지막줄에 사용될때 들여쓰기를 해도 되고 안해도 됨
my_list =
[ 1,2, 3,
  4, 5, 6,
]

result =
    some_function_that_takes_arguments( 'a',
        'b', 'c',
    ) 'd', 'e', 'f',

```

1.3 Jupyter Notebook에 autopep8 기능 추가

- install nbextensionsconhg
 - `$ conda install -c conda-forge jupyter_contrib_nbextensions`
 - `jupyter notebook server restart`
- install autopep8
 - `$ conda install -c conda-forge autopep8`
- jupyter notebook 메뉴에서 Edit > nbextensions config 선택
- autopep8 체크
- 단축키 확인
 - 단일셀 `^ + L`
 - 모든셀 `^ + ␣ + L`

1.4 최대 라인의 코드 글자수

- 하나의 라인에 사용하는 코드의 최대 글자수는 **79**자
- 하나의 라인에 사용하는 **docstring** 및 **comment**의 최대 글자수는 **72**자
- 해당 글자수가 넘어가면 들여쓰기 규칙에 맞도록 코드의 줄을 바꿔서 사용
- 암시적인 줄바꿈이 안되는 코드에는 \를 사용하여 줄바꿈을 사용

```
In [ ]: # 암시적인 줄바꿈이 안되는 코드에는 `\\`를 사용하여 줄바꿈을 사용
with open('/path/to/some/file/you/want/to/read') as file_1,
    \\ open('/path/to/some/file/being/written', 'w') as
    file_2:
    file_2.write(file_1.read())
```

1.5 연산자 전후의 줄바꿈

- 연산자가 있는 부분에서 줄바꿈을 할 때에는 연산자를 줄바꿈하는 아래 라인에 작성

```
In [ ]: # 잘못된 작성된 예
income = (gross_wages +
          taxable_interest +
          (dividends - qualified_dividends) -
          ira_deduction -
          student_loan_interest)

# 잘 작성된 예
income = (gross_wages
          + taxable_interest
          + (dividends - qualified_dividends)
          - ira_deduction
          - student_loan_interest)
```

1.6 공백 라인

- 함수와 클래스 사이는 두줄의 공백을 사용
- 클래스 내 함수 사이에는 한줄 공백 사용

```
In [ ]: def test(msg):  
        print("echo", msg)  
        # 함수와 클래스 사이는 두줄의 공백을 사용  
  
        class Test:  
  
            # 클래스 내 함수 사이에는 한줄 공백 사용  
            def echo(self, msg):  
                print("Test.echo", msg)  
  
        def echo(self, msg):  
            return msg
```

1.7 코드 문자에 대한 인코딩

- python2에서는 ASCII 사용
- python3에서는 UTF-8 사용
- 인코딩 선언을 사용 하면 안됨

인코딩 선언은 파이썬 코드 파일의 최상단에 아래와 같은 코드를 추가

```
■ #-*- coding: utf-8 -*-  
■
```

1.8 Imports

- 여러개의 모듈을 **import** 하는 경우에는 별도의 라인에 작성
- **from**을 사용하여 **import** 하는 경우에는 한줄에 여러 모듈의 **import**가 가능
- **import**는 파일의 가장 위에 모듈의 주석이나 **docstring** 뒤에 위치
- 표준라이브러리, **third-party** 라이브러리, 로컬 어플리케이션 순으로 가져와야 하며 종류별로 한 줄의 공백을 두어야 함
- 상대경로 보다는 절대경로로 가져오는것을 권장하지만 절대경로가 너무 길어지면 상대경로로 가져오는것을 권장

In []:

```
# 잘못된 예
import sys, os

# 잘된 예
import os
import sys

from subprocess import Popen, PIPE
```

2. 문자열 인용문

- 작은 따옴표와 큰따옴표로 묶은 문자열이 동일
- 문자열로 작은 따옴표나 큰따옴표를 표현할때 \의 사용을 지양 -> 가독성 향상
- 멀티 라인의 문자열은 [PEP257 \(https://www.python.org/dev/peps/pep-0257/\)](https://www.python.org/dev/peps/pep-0257/)을 따름

```
In [ ]: # 문자열로 작은 따옴표나 큰따옴표를 표현할때 \'\' 의 사용을 지양-> 가독성 향상  
"python 'jupyter notebook'"  
'python \'jupyter notebook\''
```

3. 표현식 명령문의 공백

- 괄호와 대괄호에 붙어있는 첫번째 문자와 공백을 사용하지 않음
- 닫힘 문자의 앞에는 공백을 사용하지 않음
- , ; : 앞에는 공백을 사용하지 않음
- 대입 연산자를 이용한 변수 선언시 대입 연산자 앞뒤로 한칸의 공백을 사용

In []:

```
# 괄호와 대괄호에 붙어있는 첫번째 문자와 공백을 사용하지 않음
# Yes
spam(ham[1], {eggs: 2})
# No
spam( ham[ 1], { eggs: 2 } )

# 닫힘 문자의 앞에는 공백을 사용하지 않음
# Yes
foo = (0,)
# No
bar = (0, )

# , ; : 앞에는 공백을 사용하지 않음
# Yes
if x == 4:
    print x, y
    x, y = y, x
# No
if x == 4:
    print x, y
    x, y = y, x

# 대입 연산자를 이용한 변수 선언시 대입 연산자 앞뒤로 한칸의 공백을 사용
# Yes
x = 1
y = 2
long_variable = 3
# No
x  = 1
y  = 2
var = 3
```

4. 기타 권장 사항

- 연산자 우선순위가 낮은 연산자에 공백을 추가
- 키워드 아규먼트나 디폴트 파라미터를 작성할때 = 앞뒤로 공백을 사용하지 않음
- 인수의 주석을 연결하는 경우 = 앞뒤로 공백을 사용

In []:

```
# 연산자 우선순위가 낮은 연산자에 공백을 추가
i = i + 1
submitted += 1
x = x*2 - 1
hypot2 = x*x + y*y
c = (a+b) * (a-b)

# 키워드 아규먼트나 디폴트 파라미터를 작성할때 = 앞뒤로 공백을 사용하지 않음
def complex(real, imag=0.0):
    return magic(r=real, i=imag)

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

# 인수의 주석을 연결하는 경우 = 앞뒤로 공백을 사용
# Yes
def munge(sep: AnyStr = None): ...
def munge(input: AnyStr, sep: AnyStr = None, limit=1000): ...
# No
def munge(input: AnyStr=None): ...
def munge(input: AnyStr, limit = 1000): ...
```

5. 후행 쉽표를 사용해야 하는 경우

- 후행 쉽표를 사용할때는 괄호로 묶어 줌
- 여러줄로 데이터를 작성할때 마지막 쉽표를 붙여주는게 좋지만 한줄로 작성될때는 붙이지 않음


```

In [ ]: # 후행 쉽표를 사용할때는 괄호로 묶어줌
# Yes
(1,2,)
# No
1,2,

# 여러줄로 데이터를 작성할때 마지막 쉽표를 붙여주는게 좋지만 한줄로 작성될때는 붙이지 않음
# Yes
FILES = [
    'setup.cfg',
    'tox.ini',
]
initialize(FILES,
           error=True,
           )
# No
FILES = ['setup.cfg', 'tox.ini',]
initialize(FILES, error=True,)

```

6. 주석

- 주석은 완전한 문장
- 마지막 문장을 제외하고 문장의 마침표 뒤에 두개의 공백을 사용
- 되도록 영어로 작성

7. 네이밍

- l, O, I의 단일 변수명은 사용하지 않음
 - 글꼴에 따라 숫자 0, 1과 구분이 힘들
- 패키지 모듈 네이밍
 - 짧은 소문자를 사용
 - C, C++ 모듈명 앞에는 _를 사용
- 클래스 네이밍
 - CapWord 규칙을 사용
- 변수, 함수 네이밍
 - 소문자를 사용하고 _로 구분
- 예외처리 네이밍
 - 클래스의 규칙을 따르며 접미사로 **Error**를 사용
- 클래스의 메서드
 - 인스턴스 메서드
 - 첫번째 파라미터에 **self**를 사용
 - 클래스 메서드
 - 첫번째 파라미터에 **cls**를 사용
 - 스테틱 메서드
 - 첫번째 파라미터에 꼭 넣어줘야하는 인자가 없음
- 상수 네이밍
 - 상수(변하지 않는 데이터)는 대문자와 _의 조합을 사용

```
In [ ]: # 패키지 모듈 네이밍
import numpy
import _cextention

# 변수 함수 네이밍
jupyter_notebook = 10
def python_function():
    pass

# 예외 처리 네이밍
class BigNumberError(Exception):
    pass

# 클래스의 메서드
class CustomClass:

    # instance method
    def instance_method(self, a, b):
        return a + b

    # classmethod
    @classmethod
    def class_method(cls, a, b):
        return a + b

    # staticmethod
    @staticmethod
    def static_method(a, b):
        return a + b

# 상수 네이밍
MAX_SPEED = 100
```