Python Coding Style

Author: Davis Wang
Date: 2020.12.03
System Version: v1.0.2.0
Document Version: v1.0

DOCUMENT STATUS SHEET

Author

Position	Name	Contact Phone Ext.
Engineer	Davis.Wang	#1137

Revision History

Version	Issue date	Author/Editor	Description/Summary of Changes
[1.0.0.0]	2020.02.13	Davis.Wang	First release.
[1.0.1.0]	2020.03.24	Davis.Wang	Modified Blank Lines
[1.0.2.0]	2020.12.03	Davis.Wang	Correct misspelling

Reviewed By

Version	Issue Date	Name	Position	Review Date

TABLE OF CONTENTS

1 STYLE RULES

1.1 Semicolons

不要使用分號結束句子，也不要使用分號讓兩行字寫在同一行

Yes:

```
do_something1()
do_something2()
```

No:

```
do_something1();do_something2();
```

1.2 Line length

單行不要超過 80 個字元

以下情況例外:

- Import 宣告
- 網址
- 路徑
- 註解內換行會造成理解困難

Yes:

```
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'):

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

No:

```
# See details at
# http://www.example.com/us/developer/documentation/api/content/\
# v2.0/csv_file_name_extension_full_specification.html
```

1.3 Parentheses

謹慎使用括號。

在判斷式及回傳值時，不要使用括號

Yes:

```
if x:
    bar()

if not x:
    bar()

return foo
```

No:

```
if (x):
    bar()

if not(x):
    bar()

return (foo)
```

當使用數個變數的組合時，可加括號或不加。

Yes:

```
return spam, beans
return (spam, beans)
```

1.4 Indentation

使用 4 格 Space 進行縮排。不要使用 Tab 也不要混合使用 Tab 和 Space。單行因為太長需要換行時，應對齊被包裹的參數；或是在第一行括號後面清空，使用 4 格 Space 進行縮排。

Yes:

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

# 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)
meal = (
    spam,
    beans)
```

```
# 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)
meal = (spam,
    beans)

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

# No hanging indent in a dictionary
foo = {
    long_dictionary_key:
    long_dictionary_value,
    ...
}
```

1.5 Trailing Comma

當序列內容換行時，最後一個項目後面也需要加逗號。沒有換行時則不加逗號。

Yes:

```
golomb3 = [0, 1, 3]
golomb4 = [
    0,
    1,
    4,
    6,
]
```

No:

```
golomb4 = [
    0,
    1,
    4,
    6
]
```

1.6 Blank Lines

最高階的定義之間使用兩行空行隔開。

Yes:

```
class Animal():


    def __init__(self, name):
        self.name = name


class People():


    def __init__(self, name):
        self.name = name


def do_something():
    return something
```

Class 內的 method 定義之間以使用兩行空行隔開。Class 宣告的第一行和第一個 method 定義之間也均使用兩行空行隔開。

Yes:

```
class Animal():


    def __init__(self, name):
        self.name = name

    def do_something():
        return something
```

Method 內部則由使用者自行判斷何時使用一行空行間隔。

Yes:

```
def deposit(self, amount):
    if amount <= 0:
        raise ValueError('must be positive')

    self.balance += amount
```

1.7 Whitespace

括號內的前後都不要留空格。

Yes:

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

No:

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

逗號前不留空白。

Yes:

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

x, y = y, x
```

No:

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

任何括號前不要留空格

Yes:

```
spam(1)

dict['key'] = list[index]
```

No:

```
spam (1)

dict ['key'] = list [index]
```

以下 operator 前後都要留空格。(=, ==, <, >, !=, <>, <=, >=, in, not in, is, is not, and, or, not, +, -, *, /)

Yes:

```
x == 1
```

No:

```
x<1
```

以下 operator 可自行判斷。(//, %, **, @)

當傳遞參數時，等號前後不要留空格。除非有型別註解存在則要留空白。

Yes:

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

No:

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

連續行之間不要特別對齊符號，那是維護上的負擔。

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,
}
```

1.8 Comments and Docstrings

1.8.1 Docstrings

所有文件字串均使用 “”” 來做識別。

Yes:

```
""" blablablabla
blablabla
blablabla
"""
```

1.8.2 Modules

每份檔案的開頭都必須要有文件字串描述內容及其用法。

Yes:

```
"""A one line summary of the module or program, terminated by a
period.

Leave one blank line.  The rest of this docstring should contain
an overall description of the module or program.  Optionally, it
may also contain a brief description of exported classes and
functions and/or usage examples.

  Typical usage example:

  foo = ClassFoo()
  bar = foo.FunctionBar()
"""
```

1.8.3 Functions and Methods

所有的 Function 和 Method 都必須要有文件字串，除非符合以下三點則不須:

- 外部看不到
- 非常簡短
- 明顯

文件字串必須包含足夠的資訊，避免還要看完 code 才知道該如何使用。內容必須撰寫地像描述而非撰寫地像命令。

Yes:

```
"""Fetches rows from a Bigtable."""
```

No:

```
"""Fetch rows from a Bigtable."""
```

如果 method 是覆寫來的，可以簡單地告知讀者去看被覆寫的 method 文件字串。

Yes:

```
"""See base class."""
```

當然如果覆寫的 method 本質上有別於被覆寫的 method，或是有一些不同的地方，則必須把敘述寫在覆寫的 method 的文件字串內。

每一個 function 的文件字串都必須包含特定的部分。每一個部分都使用冒號來區別並且使用 4 格空格做縮排。每部分之間都使用一行空行隔開。

Args:

使用各自名稱表列出來。每一個名稱後面使用一個冒號和一個空格接續其描述的內容。若是單行太長一樣必須要換行。每一個參數的描述必須要包含可以使用的型別。

Returns:

必須描述回傳的型別和意義。如果沒有回傳值，此部分可以不寫。

Raises:

表列出所有對應 error 介面的描述。

Yes:

```
def fetch_bigtable_rows(big_table, keys, other_silly_variable=None)
    """Fetches rows from a Bigtable.

    Retrieves rows pertaining to the given keys from the Table
    instance represented by big_table. Silly things may happen if
    other_silly_variable is not None.

    Args:
        big_table: An open Bigtable Table instance.
        keys: A sequence of strings representing the key of each
            table row to fetch.
        other_silly_variable: Another optional variable, that has
            a much longer name than the other args, and which
            does nothing.

    Returns:
        A dict mapping keys to the corresponding table row data
        fetched. Each row is represented as a tuple of strings.
        For example:

        {'Serak': ('Rigel VII', 'Preparer'),
         'Zim': ('Irk', 'Invader'),
         'Lrrr': ('Omicron Persei 8', 'Emperor')}

        If a key from the keys argument is missing from the
        dictionary, then that row was not found in the table.

    Raises:
        IOError: An error occurred accessing the bigtable.Table
            object.
    """
```

1.8.4 Classes

假如有公開的 Attributes 就必須條列在 *Attributes* 的部分。格式與 function 的 *Args* 部分相同。

Yes:

```
class SampleClass(object):
    """Summary of class here.

    Longer class information....
    Longer class information....

    Attributes:
        likes_spam: A boolean indicating if we like SPAM or not.
        eggs: An integer count of the eggs we have laid.
    """

    def __init__(self, likes_spam=False):
        """Inits SampleClass with blah."""
        self.likes_spam = likes_spam
        self.eggs = 0

    def public_method(self):
        """Performs operation blah."""
```

1.8.5 Blocks & Inline Comments

假設有需要在 code review 時解釋，就把相關的解釋註解寫下來。複雜的操作就把解釋註解寫在前面。若只是該行無法馬上看懂，把註解寫在該行的最後即可，並且至少間隔 2 格空格。

Yes:

```
# We use a weighted dictionary search to find out where i is in
# the array.  We extrapolate position based on the largest num
# in the array and the array size and then do binary search to
# get the exact number.

if i & (i-1) == 0:  # True if i is 0 or a power of 2.
```

不要去描述 code 本身。

No:

```
# Now go through the b array and make sure whenever i occurs
# the next element is i+1
```

1.8.6 Punctuation, Spelling, and Grammar

請注意標點符號、拼字和文法的使用。

1.9 Classes

若沒有特別繼承一個 Base Class，請明確寫上繼承 object。Class 內的 Class 一樣要遵照此規則。

此動作是為了在 Python 2 裡面確保 properties 正常運作和在 Python 3 裡面避免不正常的行為，並且也會自行定義特別的 method，包含 __new__, __init__, __delattr__, __getattribute__, __setattr__, __hash__, __repr__, and __str__。

Yes:

```
class SampleClass(object):
        pass


class OuterClass(object):

    class InnerClass(object):
            pass


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

No:

```
class SampleClass:
        pass


class OuterClass:

    class InnerClass:
            pass
```

1.10 Strings

使用 format 或是 % operator，就算變數都是字串。

Yes:

```
x = a + b
x = '%s, %s!' % (imperative, expletive)
x = '{}, {}'.format(first, second)
x = 'name: %s; score: %d' % (name, n)
x = 'name: {}; score: {}'.format(name, n)
x = f'name: {name}; score: {n}'  # Python 3.6+
```

No:

```
x = '%s%s' % (a, b)  # use + in this case
x = '{}{}'.format(a, b)  # use + in this case
x = first + ', ' + second
x = 'name: ' + name + '; score: ' + str(n)
```

避免在迴圈內，使用 + 或 += 累加字串。此種方式會造成創造多餘的物件，並且花費的時間會是指數型的成長。

使用 append 和 join 來取代。

Yes:

```
items = ['<table>']

for last_name, first_name in employee_list:
    items.append('<tr><td>%s, %s</td></tr>' % (last_name, first_nam

items.append('</table>')

employee_table = ''.join(items)
```

No:

```
employee_table = '<table>'

for last_name, first_name in employee_list:
    employee_table += '<tr><td>%s, %s</td></tr>' % (last_name, firs

employee_table += '</table>'
```

在一份檔案內統一使用引號的方式。

Yes:

```
Python('Why are you hiding your eyes?')
Gollum("I'm scared of lint errors.")
Narrator('"Good!" thought a happy Python reviewer.')
```

No:

```
Python("Why are you hiding your eyes?")
Gollum('The lint. It burns. It burns us.')
Gollum("Always the great lint. Watching. Watching.")
```

當字串是多行時，請使用 “”” 。若字串內容可接受每行前面有空格，請遵照縮排規則。若不可接受每行前面有空格，請使用 format 或是使用 textwrap。

Yes:

```
long_string = """This is fine if your use case can accept
    extraneous leading spaces."""

long_string = ("And this is fine if you can not accept\n" +
               "extraneous leading spaces.")

long_string = ("And this too is fine if you can not accept\n"
               "extraneous leading spaces.")

import textwrap

long_string = textwrap.dedent("""\
    This is also fine, because textwrap.dedent()
    will collapse common leading spaces in each line.""")
```

No:

```
long_string = """This is pretty ugly.
Don't do this.
"""
```

1.11 Files and Sockets

在使用 file 或 socket 時，必須在不使用時確保其被關閉，以避免系統資源被咬住。可以使用 with。

Yes:

```
with open("hello.txt") as hello_file:
    for line in hello_file:
        print(line)

import contextlib

with contextlib.closing(urllib.urlopen("http://www.python.org/")) as
    for line in front_page:
        print(line)
```

1.12 TODO Comments

請使用 TODO 符號來做暫時性、短期和可再優化的註解。

Yes:

```
# TODO(kl@gmail.com): Use a "*" here for string repetition.
# TODO(Zeke) Change this to use relations.
```

1.13 Imports formatting

Import 請分行。

Yes:

```
import os
import sys
```

No:

```
import os, sys
```

Import 應放置在一份檔案的最上方，在 Module 註解和文件字串後面，但在所有全域變數或是常數前面。

所有 Import 應被分類成以下順序撰寫。

- Future 相關 import 宣告

```
from __future__ import absolute_import
from __future__ import division
from __future__ import print_function
```

- 載入標準函式庫

```
import sys
```

- 載入第三方 module 或 package

```
import tensorflow as tf
```

- 載入 sub-package

```
from otherproject.ai import mind
```

每一類別的 import 都應按照字母順序排列。

Yes:

```
import collections
import queue
import sys

from absl import app
from absl import flags
import bs4
import cryptography
import tensorflow as tf

from book.genres import scifi
from otherproject.ai import body
from otherproject.ai import mind
from otherproject.ai import soul
```

1.14 Statements

當宣告只有一行，而且內容也只有一行，可以放置在同一行，但必須不超過單行長度限制。

如果使用 try except 或是 if 需要使用 else 時，則不可放置在同一行。

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)
```

1.15 Accessors

存取時，若是要存取瑣碎的東西，則直接使用 public variable，以避免執行 function calls 所花費的時間。

如果稍微需要一點處理，則透過 property 的方式來維持用法上的合理性。如果處理的過程過於複雜，或是花費的時間比執行 function calls 的時間來的多許多，才使用 function calls。

1.16 Naming

module_name, package_name, ClassName, method_name, ExceptionName, function_name, GLOBAL_CONSTANT_NAME, global_var_name, instance_var_name, function_parameter_name, local_var_name.

function name、variable name 和 filenames 應該要稍微描述式地命名，避免縮寫。

1.16.1 Names to Avoid

- 除非是 counters 或 iterators，不然不要用單一字元命名
- 在 package/module 名稱內不要使用 dash (-)

1.16.2 Naming Conventions

- “Internal” 意指 module 內，或是 class 內 protected 或 private
- 名稱前方加入一個下底線會支援某些保護來保護 variable 或 function。例如若是執行 (from module import *) 會無法看到那些 variable 和 function。名稱前方加入兩個下底線會讓其他人無法任意存取該

variable 或 function (透過 Python name mangling 的機制)。但以上並非真的無法存取，只是需要透過更麻煩的方式，以此達到某種程度的保護。

- 放置有關聯的 class 和 top-level function 在同一個 module。

1.16.3 File Naming

副檔名一定是 .py，並且不要使用 dash (-)。

1.16.4 Guidelines derived from Guido's Recommendations

Type	Public	Internal
Packages	`lower_with_under`	
Modules	`lower_with_under`	`_lower_with_under`
Classes	`CapWords`	`_CapWords`
Exceptions	`CapWords`	
Functions	`lower_with_under()`	`_lower_with_under()`
Global/Class Constants	`CAPS_WITH_UNDER`	`_CAPS_WITH_UNDER`
Global/Class Variables	`lower_with_under`	`_lower_with_under`
Instance Variables	`lower_with_under`	`_lower_with_under` (protected)
Method Names	`lower_with_under()`	`_lower_with_under()` (protected)
Function/Method Parameters	`lower_with_under`	
Local Variables	`lower_with_under`	

1.17 Main

當一份檔案是被設計為可以直接被執行的，執行的內容應該被放在 main() 裡面，並且要加上 if __name__ == '__main__'。避免在被 import 的時候被執行。

Yes:

```
def main():
    ...

if __name__ == '__main__':
    main()
```

1.18 Function length

盡量小且聚焦。

若一個 function 超過 40 行，請思考是不是可以拆開。

2 REFERENCES

- Google Python Style Guide: http://google.github.io/styleguide/pyguide.html#33-parentheses