商管程式設計 107-2 Style Guide

2019/5/29-30

Agenda

- 簡介
- 規範
  - 註解
  - 空格與空行
  - 命名
  - 雜項
- 檢查與修正

簡介

Python's Official Coding Style Guide

PEP 8 (link)

PEP 8 是做甚麼用的？

- 一個 Python 通用、統一的 coding style 規範
  - 哪裡可以空格，哪裡不應該空格...
  - 就像是寫國文作文，標題空四格，每一段開頭空兩格

- 通用、統一的格式**非常**有助於團隊開發
  - 或者讓你一年之後還看得懂你自己寫了甚麼
  - 每個人習慣不同，格式規範就是用來統一大家的習慣的

註解

註解永遠是最重要的

如何寫好註解

- 把程式碼在做甚麼事情，**正確的**交代清楚
  - 尤其是特殊狀況，一定要補充說明
  - 修改 code 的時候，記得把動到的地方的註解一併修改
- **不要寫廢話**
  - 註解不是把程式碼用中文重寫一次
  - 這是期末報告，不是作業解答

```
for i in range(1, x + 1):   # 1 到 x (廢話)
for i in range(1, x + 1):   # 編號從 1 開始 (有意義)
```

註解的類型與用途

- 廣義的註解可以分為以下幾種：
  1. Docstring：說明「本段 code 的內容」，通常都是出現在程式碼/class/function 的第一行
  2. Block Comments：說明「以上 / 以下幾行」的程式意義
  3. In-line Comments：說明「這一行」的程式意義
  4. `Ellipsis`、`None`：填補寫到一半的東西，表示「這裡以後會 or 不會有 code」

Docstring

```
def my_func_1(a, b):
    """This is a function."""
```

- 用三個雙引號包起來；最後三個雙引號請**自己一行**
  - 除非你的 docstring 根本就只有一行，那就全部合併同一行
- 在整個檔案、函式、class 的開頭**強烈建議一定要寫**
  - 說明這個檔案、函式、class 是做甚麼用的，以及注意事項
    - 也可以記錄編輯狀況
  - 函數的 docstring 最好也要寫清楚 input 和 return
  - 內容格式自己小組統一就好

```
def my_func_2(a, b):
    """
    This is a function. It does nothing.

    Input: a (int), b (int)
    Return: None
    """
```

如何取用 docstring

```
# docstring 會自動變成 __doc__
print(my_func.__doc__)  # 雙底線
```

```
    This is a function. It does nothing.

    Input: a (int), b (int)
    Return: None
```

```
# 如果不想要當初的縮排
from inspect import cleandoc
print(cleandoc(my_func.__doc__))
```

```
This is a function. It does nothing.

Input: a (int), b (int)
Return: None
```

Block & In-Line Comment

- 使用井字號作為起始標記；井字號**後方空一格**
- Block Comment
  - 在一段程式之中，說明接下來的 code 大概在幹嘛
  - 如果有不只一段，段與段之間放一個井字號當空行
- In-Line Comment
  - 在一行 code 最後做出輔助說明，井字號**前面**至少**空兩格**
  - 不要濫用 in-line comment，寫太多反而會讓 code 很亂

Block & In-Line Comment Example

```
# 這是一段程式碼，只是示範用的，並沒有
# 甚麼特別的意義。
#
# 大家要努力寫程式喔！

a = input()
b = input()

c = a/b * 100    # 轉換成百分比
print(str(c) + '%')
```

Ellipsis and None

- `Ellipsis`：就是三個點，表示「這裡現在甚麼都沒有，但總有一天會 / 應該要寫上去」
- `None`：表示「這裡現在甚麼都沒有，也不打算有」

```
def new_func(a, b):
    ...   # 預計 6/10 完成

def old_func(a, b):
    None
```

空格與空行

縮排與換行

- 標準縮排為**四個空格**
- 行尾**不要多出空格**

```
# 一行塞不下時，下一行跟著括號垂直對齊
fib_li = [1, 1, 2, 3, 5, 8, 13, 21,
          34, 55]
my_2d_li = [[1, 2, 3, 4],
            [2, 2, 3, 4],
            [3, 2, 3, 4]]
foo = my_func(var_one, var_two,
              var_three, var_four)
```

```
# 如果覺得對齊括號太浪費空間的話，
# 可以直接全部放在下一行 + 比後面行
# 再多一層縮排進去
foo = long_function_name(
    var_one, var_two,
    var_three, var_four)
bar = 0

def very_very_long_function_name(
        var_one, var_two, var_three,
        var_four):
    print(var_one)
```

應該換行的位置

- 一行不要寫太長；適當的換行可以使程式顯得簡潔
  - 程式碼不超過 79 個半形字元、註解不超過 72 個半形字元
    - 設定：Sublime → ruler，Notepad++ → 邊緣線
  - 同一行分割成兩行以上的時候，在**逗號後面**、**運算子前面**換行 (同樣注意多一個縮排 or 垂直對齊括號)

```
# 多一個縮排
my_list = [
    1, 2, 3,
    4, 5, 6,
    ]
```

```
# 垂直對齊括號；注意要有開頭結尾的括號，中間才可以換行
income = (gross_wages
          + taxable_interest
          + (dividends - qualified_dividends)
          - ira_deduction
          - student_loan_interest)
```

空行

- 開頭 class、def：前後空兩行
- 內部的 def：前後空一行
- 分隔程式邏輯：空一行
- 空行**不要有空格 (含 docstring)**
  - 容易破壞結構 (尤其是複製貼上的時候)
    - Sublime 設定：`trim_trailing_white_space_on_save`
    - Notepad++ 設定：在「巨集」裡面

```
import os
(1)
(2)
class A():
(1)
    def inner_def_a():
        return None
(1)
    def inner_def_b():
        return None
(1)
(2)
def outer_def():
    return None
(1)
(2)
a_str = input()
```

空格排版

- 逗號、冒號、分號後面還有其他元素，空一格

```
# 注意：行尾冒號及逗號後面「沒東西」，因此不空格
for key, val in {1: True, 2: False}:
    my_tuple = (val,)  # 沒有第二個元素，所以不空格
    my_list = [1, 2, 3, 4, 5,
               6, 7, 8, 9, 0]  # 上面逗號後面沒東西！
```

- 有複雜東西的 slicing：冒號前後各一個空格

```
small_li_1 = big_li[1:my_int]  # 不用
small_li_2 = big_li[a : b+1]   # 要

# 碰到外層括號就不用 ↓
small_li_3 = big_li[: a : int(x)]
```

空格排版

- 等號、比較運算子、邏輯運算子的前後
  - `=, [+=, -=, *=, ……], ==, <, >, !=, <>, <=, >=, in, not in, is, is not, and, or, not`
- 運算子 (加減乘除等)：
  - 同一串中，優先權**最低**的要有空格
    - Ex. 加減和乘除同時出現時，加減要空格，乘除不用

```
my_int = 2*3 + 4*5 - 6/2
c = (a+b) * (a-b)
```

```
my_int = (1+2) * (1+2 * 3+4)     # NO
my_int = (1+2) * (1 + 2*3 + 4)   # OK
my_int = (1+2) * (1+2*3+4)       # OK
```

空格排版

- 不要在行內用連續空格排版
  - 例外：註解對齊時，可以有額外空格
    - 記得還是要空至少兩格
- 唯一的空格例外：函數的指定參數
  - 等號不加空白，運算子不加空白

```
my_int  = 123  # Bad

my_int_1 = 123456  # Good
my_int_2 = 789     # Good
```

```
my_li = some_li.sort(key=lambda k: k-1)      # Yes
my_li = some_li.sort(key = lambda k: k - 1)  # No
```

命名

命名規則

- 變數：`lower_case_with_underscores`
  - 常數：`UPPER_CASE_WITH_UNDERSCORES`
- 函式：`lower_case_with_underscores()`
- 類別：`CapitalizedWordsWithoutUnderscores`
- 錯誤：`CapitalizedWordsError`

特殊命名形式

- 結尾單底線_：避開跟預設關鍵字一樣的變數名稱

```
Tkinter.Toplevel(master, class_='ClassName')
```

- _開頭單底線：表示非公開
- __開頭雙底線：在 class 中表示 private
  - Private object 無法從外部直接存取

```
class MyCls:
    __private = 0

MyCls.__private  # Error
```

- __開頭結尾雙底線__：python 內建特殊功能

雜項

try … except

- 盡量不要只寫 except，請把錯誤種類寫出來，出現預期外的錯誤的時候才會發現
- try 裡面寫越少東西越好

```
try:
    ...
except ValueError:  # Good
    return None
except:             # Bad!!
    return None
```

```
# Bad
try:
    return f(a/b)
except ZeroDivisionError:
    return None
```

```
# Good
try:
    num = a/b
except ZeroDivisionError:
    return None
else:  # 沒有上述 except 的時候
    return f(num)
```

Import

- Import 順序：python 內建 → pip 安裝 → 自己的
- 同一行不要寫兩個 library

```
"""This is my program."""


import os  # 註解優先；第一順位是 import 內建
import sys, csv  # Bad! 這行是錯誤示範
from subprocess import Popen, PIPE

import matplotlib  # 第二順位是 import 第三方 (記得換行分隔)

from mypkg.sibling import example  # 第三順位是 import 自己寫的
```

其他

- 程式內標準順序 (從上到下)：docstring → `import` → class & functions → 主程式
- 請寫 `if a is None`，不要寫 `if a == None`
  - 請寫 `if a is not None`，不要寫 `if not a is None`
- 如果要寫 `class` 的比較運算子 (eg. `__lt__`)，盡量一次把六個 (`==`, `!=`, `<`, `<=`, `>`, `>=`) 全都寫出來
  - `__eq__`, `__ne__`, `__lt__`, `__le__`, `__gt__`, `__ge__`

其他

- 對於如 file 之類需要 close 的東西，盡量使用 with

```
# Bad
file = open(filename)
my_str = file.readlines()
file.close()
```

```
# Good
with open(filename) as file:
    my_str = file.readlines()
# Automatically close!
```

- 寫函數的時候，如果用 if 控制 return，一定要確定 else 裡面也有 return、或者函數最後有預設的 return

其他

- 檢查 string 的**開頭**或**結尾**是否符合某字串的時候，不要用 `str[a:]` 或 `str[:b]`，請用 `.startswith()` 或 `.endswith()`
  - 當你的 string 長度不足的時候，才不會出現 error

```
# Bad
if my_str[:3] == 'abc':
    return True
```

```
# Good
if my_str.startswith('abc'):
    return True
```

其他

- 檢查 type 的時候，請使用 `isinstance()`

```
# Bad
if type(my_obj) is MyType:
```

```
# Good
if isinstance(my_obj, MyType):
```

- 檢查 `True` / `False` 的時候，不用再 `== True`

```
# Bad
if a+b==2 is True:
```

```
# Good
if a+b == 2:
```

```
# Bad
if my_bool == False:
```

```
# Good
if not my_bool:
```

檢查與修正

自動檢查是否符合 PEP 8

- 線上工具
  - Ex. pep8online.com
- pycodestyle

```
> pip3 install pycodestyle

> pycodestyle test.py
test.py:68:11: E221 multiple spaces before operator
```

自動修正符合 PEP 8

- 線上工具
  - Ex. pythoniter.appspot.com
- autopep8

```
> pip3 install autopep8

> autopep8 --in-place test.py
```

- 自動修正工具只是輔助用，可能無法修正所有錯誤