Appendix A

PEP8 Python 程式碼撰寫風格

本節摘錄自 PEP 8（Python Enhancement Proposal 8），是由 Python 之父 Guido van Rossum 和 Barry Warsaw 所撰寫，內容說明了 Python 偏好的程式碼撰寫風格。您應該盡可能遵循 PEP 8 的建議，讓您的程式更符合 Python 風格。

程式碼風格的一致性

第 4 章提到 Guido 的重要見解是：『程式碼的讀取頻率遠高於撰寫』，所以遵照 PEP 8 的準則可以提高程式碼的可讀性，讓不同人寫出來的 Python 程式碼都能保持一致性的風格。

雖然依照 PEP 8 的準則保持一致性很重要，但是相較之下同一個專案內的一致性較為重要，而同一個模組或函式內的一致性才是最重要的。所以並不一定要完全依照 PEP 8，請自行作出最佳判斷。

以下是打破 PEP 8 準則的兩個充分理由：

- 應用 PEP 8 準則反而會降低程式碼的可讀性。
- 也許是出於歷史原因，為了與前後的程式碼保持一致時，也會打破這個規則（雖然這也是一個清理別人爛攤子的機會）。

縮排與最大行寬

建議每一級縮排使用 4 個空格，縮排時 Tab 和空格切勿混用。

將程式碼的行寬不應超過 80 個字元，對於長文字區塊（docstring 或註解），建議將長度限制為 72 個字元。

長行首選的續行方法是在小括號、中括號、和大括號內使用 Python 隱含的續行語法。如有必要，可以在表達式前後添加一對額外的小括號，但有時使用反斜線看起來會更好。

斷行

定義函式時若參數過多時，斷行縮排的建議樣式如下：

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

不建議這樣寫(**編註：**這並非語法錯誤，只是會影響可讀性所以不建議)：

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

呼叫函式時如果參數過多，斷行縮排的建議樣式如下：

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

或

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

不建議這樣寫：

Bad

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

if 的條件式過長時，可以如下斷行縮排：

```
if (this_is_one_thing and
    that_is_another_thing):
    do_something()

if (this_is_one_thing and
    that_is_another_thing):
    # Since both conditions are true, we can frobnicate.
    do_something()

if (this_is_one_thing
        and that_is_another_thing):
    do_something()
```

定義 list 或 tuple 時，元素過多的話請如下斷行縮排：

```
my_list = [
    1, 2, 3,
    4, 5, 6,
    ]
result = some_function_that_takes_arguments(
    'a', 'b', 'c',
    'd', 'e', 'f',
    )
```

或者

```
my_list = [
    1, 2, 3,
    4, 5, 6,
]
result = some_function_that_takes_arguments(
    'a', 'b', 'c',
    'd', 'e', 'f',
)
```

斷行時，算符應該和被運算的運算元在一起：


```
income = (gross_wages
          + taxable_interest
          + (dividends - qualified_dividends)
          - ira_deduction
          - student_loan_interest)
```

不建議這樣寫：


```
income = (gross_wages +
          taxable_interest +
          (dividends - qualified_dividends) -
          ira_deduction -
          student_loan_interest)
```

空行

使用兩個空行分隔函式和類別定義。類別中的 method 定義由單一空行分隔。

可將類似功能的函式定義放在一起變成一組，不同組之間兩個以上的空行分隔。

在函式中可使用空行分隔不同的邏輯區塊，但使用時要儘量保守。

import

每個匯入指令通常應單獨寫在一行，例如：


```
import os
import sys
```

不要像以下這樣把它們放在同一行：


```
import sys, os
```

不過可以這樣寫是 OK 的：

```
from subprocess import Popen, PIPE
```

import 應該放在檔案的頂端，匯入時應按照以下順序分組，每組之間要空一行：

1 標準函式庫

2 第三方函式庫

3 特定應用的函式庫

表達式和敘述中的空格

括號與字元之間避免空格：

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

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

逗號、分號、或冒號左邊避免空格：

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

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

呼叫函數時，名稱與括號之間避免空格：

```
spam(1)
```

```
spam (1)
```

索引或切片的括號左邊避免空格：

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

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

變數賦值不要因為對齊而添加多個空格：

```
x = 1
y = 2
long_variable = 3
```

```
x             = 1
y             = 2
long_variable = 3
```

算符與空格

通用原則是算符左右兩邊各放一個空格：

```
i = i + 1
submitted += 1
x = x * 2 - 1
hypot2 = x * x + y * y
c = (a + b) * (a - b)
```

Bad

```
i=i+1
submitted +=1
x = x*2 - 1
hypot2 = x*x + y*y
c = (a+b) * (a-b)
```

= 用於函式的參數或引數時不要有空格：

Good

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

Bad

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

複合敘述

通常不鼓勵使用複合述（多條述寫在同一行）：

Good

```
if foo == 'blah':
    do_blah_thing()
do_one()
do_two()
do_three()
```

Bad

```
if foo == 'blah': do_blah_thing()
do_one(); do_two(); do_three()
for x in lst: total += x
   while t < 10: t = delay()
```

雖然有時可以將 if / for / while 與一小段程式碼放在同一行上，但是不要跟多多子句述放在同一行，並且要避免把像這樣長的多子句述寫在不同行！

絕對不要這樣寫：

Bad

```
if foo == 'blah': do_blah_thing()
else: do_non_blah_thing()
try: something()
finally: cleanup()
do_one(); do_two(); do_three(long, argument,
                             list, like, this)
if foo == 'blah': one(); two(); three()
```