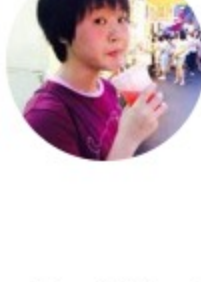


Python 的 Import 陷阱

 Rachel Liao [Follow](#)
Oct 30, 2017 · 16 min read

在脫離 Python 幼幼班準備建立稍大型的專案的時候，學習如何組織化你的 Python 專案是一大要點。Python 提供的 **module**（模組）與 **package**（套件）是建立架構的基本元件，但在 module 之間為了重複使用一些 function（函數）或 class（類別）而必須互相 **import**（匯入），使用上一個不注意就會掉入混亂的 import 陷阱。

這篇將會從基本 module 和 package 介紹起，提點基本 import 語法及 absolute import 和 relative import 的用法與差異，最後舉出幾個常見因為錯誤 import 觀念造成的錯誤。

請注意，以下只針對 Python3 進行講解與測試。

· · ·

Module與Package

基本上一個檔案就是一個 module，裡頭可以定義 function，class，和 variable。

把一個 module 想成一個檔案，那一個 package 就是一個目錄了。Package 可裝有 subpackage 和 module，讓你的專案更條理更組織化，最後一坨打包好還能分給別人使用。

先看看 module。假設有一個 module `sample_module.py` 裡頭定義了一個 function `sample_func`：

```
def sample_func():
    print('Hello!')
```

現在你在同一個目錄裡下有另一個 module `sample_module_import.py` 想要重複使用這個 function，這時可以直接從 `sample_module` import 拿取：

```
from sample_module import sample_func

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

跑 `python3 sample_module_import.py` 會得到：

```
Hello!
```

再來是 package。我們把上面兩個檔案包在一個新的目錄 `sample_package` 底下：

```
sample_package/
├── __init__.py
├── sample_module.py
└── sample_module_import.py
```

很重要的是新增那個 `__init__.py` 檔。它是空的沒關係，但一定要有，有點宣稱自己是一個 package 的味道。

這時候如果是進到 `sample_package` 裡面跑一樣的指令，那沒差。但既然都打包成 package 了，通常是在整個專案的其他地方需要用到的時候 import 它，這時候裡面的 import 就要稍微做因應。

讓我們修正一下 `sample_package/sample_module_import.py`。假設這時我們在跟 `sample_package` 同一個 folder 底下執行下面兩種指令：

```
1. python3 sample_package/sample_module_import.py
2. python3 -m sample_package.sample_module_import
```

以下幾種不同的 import 寫法，會各有什麼效果呢？

```
# 不標準的 implicit relative import 寫法 (Python 3 不支援)
from sample_module import sample_func
1. 成功印出 Hello!
2. ModuleNotFoundError. 因為 Python 3 不支援 implicit relative import (前面不加點的寫法)，故會將之當作 absolute import，但第三個例子才是正確寫法。

# 標準的 explicit relative import 寫法
from .sample_module import sample_func
1. 包含相對路徑的檔案不能直接執行，只能作為 module 被引用，所以失敗
2. 成功印出 Hello!

# 標準的 absolute import 寫法
from sample_package.sample_module import sample_func
1. 如果此層目錄位置不在 python path 中，就會失敗
2. 成功印出 Hello!
```

這邊 absolute import 和 relative import 的詳細說明請稍候。

執行指令中的 `-m` 是為了讓 Python 預先 import 你要的 package 或 module 給你，然後再執行 script。所以這時 `sample_module_import` 在跑的時候，是以 `sample_package` 為環境的，這樣那些 import 才會合理。

另外，**python path** 是 Python 查找 module 時候使用的路徑，例如 standard module 所在的目錄位置。因此在第三種寫法中，Python 會因為在 python path 中找不到 `sample_package.sample_module` 而噴 error。你可以選擇把當前目錄加到 `sys.path`，也就是 Python path（初始化自環境變數 `PYTHONPATH`），來讓 Python 搜尋得到這個 module，但這個方法很髒很難維護，最多用來 debug，其他時候強烈不建議使用。

· · ·

基本 import 語法

前面有看過了，這邊統整介紹一下。如果你想使用在其他 module 裡定義的 function、class、variable 等等，就需要在使用它們之前先進行 import。通常都會把需要 import 的 module 們列在整個檔案的最一開始，但不是必須。

語法1：**import [module]**

```
# Import 整個 `random` module
import random

# 使用 `random` module 底下的 `randint` function
print(random.randint(0, 5))
```

語法2：**from [module] import [name1, name2, ...]**

```
# 從 `random` module 裡 import 其中一個 function `randint`
from random import randint

# 不一樣的是，使用 `randint` 的時候就不需要先寫 `random` 了
print(randint(0, 5))
```

語法3：**import [module] as [new_name]**

```
# Import 整個 `random` module，
# 但這個名字可能跟其他地方有衝突，因此改名叫 `rd`
import random as rd

# 使用 `rd` 這個名稱取代原本的 `random`
print(rd.randint(0, 5))
```

語法4（不推薦）：**from [module] import ***

```
# Import 所有 `random` module 底下的東西
from random import *

# 使用 `randint` 的時候也不需要先寫 `random`
print(randint(0, 5))
```

語法4不推薦原因是容易造成名稱衝突，降低可讀性和可維護性。

Top highlight

· · ·

Absolute Import v.s. Relative Import

Python 有兩種 import 方法，**absolute import** 及 **relative import**。Absolute import 就是完整使用 module 路徑，relative import 則是使用以當前 package 為參考的相對路徑。

Relative import 的需求在於，有時候在改變專案架構的時候，裡面的 package 和 module 會拉來拉去，這時候如果這些 package 裡面使用的是 relative import 的話，他們的相對關係就不會改變，也就是不需要再一一進入 module 裡更改路徑。但因為 relative import 的路徑取決於當前 package，所以在哪裡執行就會造成不一樣的結果，一不小心又要噴一堆 error；這時 absolute import 就會減少許多困擾。

這邊參考PEP328提供的範例。Package 架構如下：

```
package
├── __init__.py
├── subpackage1
│   ├── __init__.py
│   ├── moduleX.py
│   └── moduleY.py
├── subpackage2
│   ├── __init__.py
│   └── moduleZ.py
└── moduleA.py
```

現在假設 `package/subpackage1/moduleX.py` 想要從其他 module 裡 import 一些東西，則使用下列語法（[A] 表 absolute import 範例；[R] 表 relative import 範例）：

```
# Import 同一個 package 底下的 sibling module `moduleY`
[A] from package.subpackage1 import moduleY
[R] from . import moduleY
[Error] import .moduleY

# 從同一個 package 底下的 sibling module `moduleY` 中，
# import `spam` 這個 function
[A] from package.subpackage1.moduleY import spam
[R] from .moduleY import spam

# 從隔壁 package 底下的 module `moduleZ` 中，
# import `eggs` 這個 function
[A] from package.subpackage2.moduleZ import eggs
[R] from ..subpackage2.moduleZ import eggs

# Import parent package 底下的 module `moduleA`
[A] from package import moduleA
[R] from .. import moduleA 或 from ... package import moduleA
```

要點：

- Relative import 裡，`..` 代表上一層，多幾個`.`就代表多上幾層。
- Relative import 一律採用 `from ... import ...` 語法，即使是從`.` import 也要寫 `from . import some_module` 而非 `import .some_module`。原因是 `.some_module` 這個名稱在 expression 裡無法出現。Absolute import 則無限制。