# Python技巧

**iterables**

根據Python標準文檔，Iterable的概念如下：

一種能夠一次返回一個成員的物件。

iterables的示例包括：

* 所有序列類型（如list、str和tuple）
* 一些非序列類型，如dict、檔物件以及類的實現中定義了\_\_iter\_\_()方法

Iterables是一個需要我們牢記的概念，因為接下來我們展示的許多技巧都使用itertools包。

itertools模組提供了一些函數，用於接收Iterable物件，而不僅僅是列印逐個對象。

## 1.

在工作學習中，我們經常會需要使用一個簡單的函數來實現從一個list來生成新的list, set或dict.此時我們就會用到iterables概念。

舉例來說：

生成List:

names = ['John', 'Bard', 'Jessica' 'Andres']

lower\_names = [name.lower() for name in names]

生成Set:

names = ['John', 'Bard', 'Jessica' 'Andres']

lower\_names = {

name.lower() for name in names}

生成Dict:

names = ['John', 'Bard', 'Jessica' 'Andres']

lower\_names = {

name:name.lower() for name in names}

個人建議：

僅當for語句、函式呼叫和方法調用的數量較少時使用。

## 2.

有時，我們需要獲得兩個清單物件之間的所有可能組合。

我們首先想到的實現可能如下：

l1 = [1, 2, 3]

l2 = [4, 5, 6]

combinations = []

for e1 in l1:

for e2 in l2:

combinations.append((e1, e2))

或者簡化一下，如下：

combinations = [(e1, e2) for e1 in l1 for e2 in l1]

上述實現已經很簡潔了，但標準庫itertools提供product函數，從而提供了相同的結果。如下所示：

from itertools import product

l1 = [1, 2, 3]

l2 = [4, 5, 6]

combinatios = product(l1, l2)

## 3.

假設有一個元素清單，我們需要在每對相鄰元素之間比較或應用一些操作，這有時稱為2個元素的滑動視窗。我們可以採用以下方式：

from itertools import tee

from typing import Iterable

def window2(iterable: Iterable):

it, offset = tee(iter(iterable))

next(offset)

return zip(it, offset)

l = [1, 2, 3, 4, 5, 6]

dd = window2(l)

for a in dd:

print(a)

運行結果如下：

(1, 2)

(2, 3)

(3, 4)

(4, 5)

(5, 6)

## 4.

有時，我們會需要一個類來存儲資訊，但是如果我們覺得創建一個類並定義其\_\_init\_\_()函數太麻煩時，我們不妨選擇使用dataclass。如下所示：

from dataclasses import dataclass

@dataclass

class Person:

name: str

age: int

address: str

上述代碼創建了一個具有預設構造函數的類，該類以與聲明相同的順序接收相應欄位的賦值。

person = Person(name='John', age=12, address='nanjing street')

dataclass的另一個優點是，預設情況下，會生成特殊方法，如\_\_str\_\_、 **repr** 、\_\_eq\_\_等。關於dataclass的更多用法，可以 [***參考官網***](https://link.zhihu.com/?target=https%3A//docs.python.org/3/library/dataclasses.html) 。

值得一提的是我們在類中聲明的成員變數的類型注釋（str、int等）並不強制在構造函數中傳遞的值屬於這種類型。也就是說dataclasses構造物件時並不執行資料類型的檢查。

## 5.

我們有時希望將一個物件上的操作視為tuple上的操作，一種選擇是使用collections.namedtuple,但也存在更類似於dataclass的實現。如下：

from typing import NamedTuple

class Coordinate(NamedTuple):

x: int

y: int

上述定義了一個標準的類可以被當做tuple來使用，如下：

coordinate = Coordinate(10, 15)

coordinate.x == coordinate[0] // True

coordinate.y == coordinate[1] // True

## 6.

假如我們有一個dataclass，需要驗證輸入資料是否符合類型注釋。在這種情況下，安裝協力廠商套裝軟體pydantic並將

*from dataclasses import dataclass* 替換為 *from pydantic.dataclasses import dataclass* 即可，如下：

from pydantic.dataclasses import dataclass

@dataclass

class Person:

name: str

age: int

address: str

這將生成一個類，該類具有根據成員變數聲明的類型進行輸入資料的解析和類型驗證。 *Pydantic* 在運行時強制執行類型提示，並在資料無效時提供友好的錯誤提醒。

## 7.

在某些情況下，我們需要生成一些容器中元素頻率的基本統計資訊。在這種情況下，您可以使用標準結構Counter來接收iterable並根據元素的頻率生成相應的統計資訊。

from collections import Counter

l = [1, 1, 2, 3, 4, 4]

frequencys = Counter(l)

print(frequencys[1]) // Ouput: 2

print(frequencys[2]) // Ouput: 1

print(frequencys[2323]) // Ouput: 0

Counter也提供了一些其他方法，比如如most\_common，用於檢索最常見的元素。

## 8.

如果我們相對兩個list中的元素對做相應的函數處理，我們最容易想到的方法如下：

l1 = [1, 2, 3]

l2 = [4, 5, 6]

for (e1, e2) in zip(l1, l2):

f(e1, e2)

但是使用函數map可以讓代碼更加簡潔一些。

l1 = [1, 2, 3]

l2 = [4, 5, 6]

map(f, l1, l2)

## 9.

有時候我們需要從一個list中隨機選擇一個元素，此時我們使用random.choice.如下所示：

from random import choice

l = [1, 2, 3]

random = choice(l)

如果我們需要隨機選擇多個元素呢？當然是使用random.choices.

from random import choices

l = [1, 2, 3, 4, 5]

random\_elements = choices(l, k=3)

上述代碼中的參數k為我們隨機選擇元素的個數。