Python 的 50+ 練習：資料科學學習手冊

> *讓程式更簡潔的 Python 技巧*

數據交點 | 郭耀仁 yaojenkuo@datainpoint.com

這個章節會登場的保留字與函數

- `lambda` 保留字。
- `reversed()` 函數。
- `enumerate()` 函數。
- `zip()` 函數。
- `map()` 函數。
- `filter()` 函數。

Comprehension

什麼是 Comprehension

> *Comprehension 指的是依賴可迭代類別來建立* `list` *、* `dict` *或者* `set` *的語法，Python 2 推出 List comprehension，Python 3 推出 Dictionary comprehension 以及 Set comprehension。*

來源：https://python-3-patterns-idioms-test.readthedocs.io/en/latest/

依賴可迭代類別建立 `list` 的傳統方式

```
empty_list = list()
for element in iterable:
    ...
    empty_list.append(...)
```

In [1]:
```
primes = [2, 3, 5, 7, 11]
squared_primes = list()
for prime in primes:
    squared_primes.append(prime**2)
squared_primes
```

Out[1]:
```
[4, 9, 25, 49, 121]
```

以 `List comprehension` 建立 `list`

```
list_comprehension = [... for element in iterable]
```

In [2]:
```
squared_primes = list()
for prime in primes:
    squared_primes.append(prime**2)
squared_primes
```

Out[2]:
```
[4, 9, 25, 49, 121]
```

In [3]:
```
squared_primes = [prime**2 for prime in primes]
squared_primes
```

Out[3]:
```
[4, 9, 25, 49, 121]
```

在 `List comprehension` 中加入 `if` 條件敘述

```
list_comprehension = [... for element in iterable if CONDITION]
```

In [4]:
```
squared_odd_primes = list()
for prime in primes:
    if prime % 2 == 1:
        squared_odd_primes.append(prime**2)
squared_odd_primes
```

Out[4]:
```
[9, 25, 49, 121]
```

In [5]:
```
squared_odd_primes = [prime**2 for prime in primes if prime % 2 == 1]
squared_odd_primes
```

Out[5]:
```
[9, 25, 49, 121]
```

在 `List comprehension` 中加入 `if...else...` 條件敘述

```
list_comprehension = [... if CONDITION else ... for element in iterable]
```

In [6]:
```
even_or_odd = list()
for prime in primes:
    if prime % 2 == 0:
        even_or_odd.append("Even")
    else:
        even_or_odd.append("Odd")
even_or_odd
```

Out[6]:
```
['Even', 'Odd', 'Odd', 'Odd', 'Odd']
```

In [7]:
```
even_or_odd = ["Even" if prime % 2 == 0 else "Odd" for prime in primes]
even_or_odd
```

Out[7]:
```
['Even', 'Odd', 'Odd', 'Odd', 'Odd']
```

以 `Set comprehension` 建立 `set`

```
set_comprehension = {... for element in iterable}
```

In [8]:

```
squared_primes = {prime**2 for prime in primes}
print(squared_primes)
print(type(squared_primes))
```

```
{4, 9, 49, 121, 25}
<class 'set'>
```

以 `Dictionary comprehension` 建立 `dict`

```
dict_comprehension = {key: value for element in iterable}
```

In [9]:

```
squared_primes = {prime: prime**2 for prime in primes}
print(squared_primes)
print(type(squared_primes))
```

```
{2: 4, 3: 9, 5: 25, 7: 49, 11: 121}
<class 'dict'>
```

為什麼沒有 `Tuple comprehension`

In [10]:

```
primes = [2, 3, 5, 7, 11]
squared_primes = (prime**2 for prime in primes)
print(squared_primes)
print(type(squared_primes))
```

```
<generator object <genexpr> at 0x7ff0708a3190>
<class 'generator'>
```

Generator

什麼是 Generator

Generator 與 Comprehension 相似，不同的地方在於 Comprehension 所產出的是資料結構所儲存的資料，Generator 則是儲存資料的產生規則，必須經過像是類別實例化為物件的處理過程，才會將資料儲存到指定的資料結構中。若是以料理來比喻，Comprehension 的輸出就像是最後上桌的菜餚，而 Generator 的輸出則是該菜餚的食譜。

Generator 的三個特性

1. 具備儲存與計算的效率性。
2. 沒有顯示外觀。
3. 實例化為物件的次數僅有一次。

In [11]:

```
squared_primes = (prime**2 for prime in primes)
print(list(squared_primes))
print(list(squared_primes))
```

```
[4, 9, 25, 49, 121]
[]
```

為什麼要介紹 Generator

- ~~故意要混淆大家的腦袋 -_-~~
- 很多內建函數的輸出都具備 Generator 的特性。

輸出具有 Generator 特性的內建函數

- 迭代器函數
    - `reversed()`
    - `enumerate()`
    - `zip()`
- 函數型函數
    - `map()`
    - `filter()`

迭代器函數

什麼是迭代器函數

> *迭代器函數（Iterator functions）是能夠輸出為可迭代類別的函數，可迭代類別具有與函數同名的類別名稱，又被稱為迭代器（Iterator），具備有 Generator 的特性。*

來源：https://docs.python.org/3/glossary.html

`reversed()` 函數

- 將可迭代類別順序反轉。
- 輸出的類別為 `reversed` 具有 Generator 特性。

In [12]:
```
rev = reversed("Avengers")
print(rev)
print(type(rev))
print(list(rev))
print(list(rev))
```

```
<reversed object at 0x7ff07087de80>
<class 'reversed'>
['s', 'r', 'e', 'g', 'n', 'e', 'v', 'A']
[]
```

In [13]:
```
for character in reversed("Avengers"):
    print(character)
```

```
s
r
e
g
n
e
v
A
```

`enumerate()` 函數

- 將可迭代類別的「索引」與「元素」以 `tuple` 包括讓迴圈能同時走訪。
- 輸出的類別為 `enumerate` 具有 Generator 特性。

In [14]:
```
avenger_movies = ["The Avengers", "Avengers: Age of Ultron", "Avengers: Infinity War", "Avengers: Endgame
enum = enumerate(avenger_movies)
print(enum)
print(type(enum))
print(list(enum))
print(list(enum))
```

```
<enumerate object at 0x7ff0708a7540>
<class 'enumerate'>
[(0, 'The Avengers'), (1, 'Avengers: Age of Ultron'), (2, 'Avengers: Infinity Wa
r'), (3, 'Avengers: Endgame')]
[]
```

In [15]:
```
avenger_movies = ["The Avengers", "Avengers: Age of Ultron", "Avengers: Infinity War", "Avengers: Endgame
for index, movie in enumerate(avenger_movies):
    print(f"{index} | {movie}")
```

```
0 | The Avengers
1 | Avengers: Age of Ultron
2 | Avengers: Infinity War
3 | Avengers: Endgame
```

`zip()` 函數

- 將多個可迭代類別的「元素」以 `tuple` 包括讓迴圈能同時走訪。
- 輸出的類別為 `zip` 具有 Generator 特性。

In [16]:
```
avenger_movies = ["The Avengers", "Avengers: Age of Ultron", "Avengers: Infinity War", "Avengers: Endgame
release_years = [2012, 2015, 2018, 2019]
zipped = zip(release_years, avenger_movies)
print(zipped)
print(type(zipped))
print(list(zipped))
print(list(zipped))
```

```
<zip object at 0x7ff0708a7140>
<class 'zip'>
[(2012, 'The Avengers'), (2015, 'Avengers: Age of Ultron'), (2018, 'Avengers: In
finity War'), (2019, 'Avengers: Endgame')]
[]
```

In [17]:
```
avenger_movies = ["The Avengers", "Avengers: Age of Ultron", "Avengers: Infinity War", "Avengers: Endgame
release_years = [2012, 2015, 2018, 2019]
for year, movie in zip(release_years, avenger_movies):
    print(f"{movie} was released in {year}.")
```

```
The Avengers was released in 2012.
Avengers: Age of Ultron was released in 2015.
Avengers: Infinity War was released in 2018.
Avengers: Endgame was released in 2019.
```

函數型函數

什麼是函數型函數

函數型函數（Functiona functions）亦稱為高階函數，是能夠將可迭代類別中的元素作為參數依序輸入給指定函數的特殊函數。

```
def function():
    ...
    return ...

functional_function(function, iterable)
```

`map()` 函數

- 將可迭代類別中的元素作為參數依序傳給指定函數。
- 輸出的類別為 `map` 具有 Generator 特性。
- 可以省去撰寫迴圈的作法。

傳統對可迭代類別中的元素使用函數的作法

In [18]:

```
def square(x):
    return x**2

primes = [2, 3, 5, 7, 11]
for prime in primes:
    print(square(prime))
```

```
4
9
25
49
121
```

透過 `map()` 函數對可迭代類別中的元素使用函數的作法

In [19]:

```
def square(x):
    return x**2

primes = [2, 3, 5, 7, 11]
mapped = map(square, primes)
print(mapped)
print(type(mapped))
print(list(mapped))
print(list(mapped))
```

```
<map object at 0x7ff0708acbe0>
<class 'map'>
[4, 9, 25, 49, 121]
[]
```

`filter()` 函數

- 將可迭代類別中的元素作為參數依序傳給指定函數，保留函數回傳值為 `True` 的輸出。
- 輸出的類別為 `filter` 具有 Generator 特性。
- 可以省去撰寫迴圈與條件敘述的作法。

傳統對可迭代類別中的元素使用函數與條件敘述的作法

In [20]:

```
def my_favorite_avenger_movie(x):
    return x == "The Avengers"

avenger_movies = ["The Avengers", "Avengers: Age of Ultron", "Avengers: Infinity War", "Avengers: Endgame
for movie in avenger_movies:
    if my_favorite_avenger_movie(movie):
        print(movie)
```

```
The Avengers
```

透過 `filter()` 對可迭代類別中的元素使用函數與條件敘述的作法

In [21]:

```
def my_favorite_avenger_movie(x):
    return x == "The Avengers"

avenger_movies = ["The Avengers", "Avengers: Age of Ultron", "Avengers: Infinity War", "Avengers: Endgame
filtered = filter(my_favorite_avenger_movie, avenger_movies)
print(filtered)
print(type(filtered))
print(list(filtered))
print(list(filtered))
```

```
<filter object at 0x7ff0708ac9a0>
<class 'filter'>
['The Avengers']
[]
```

以 Lambda 敘述搭配函數型函數

Lambda 敘述也稱為匿名函數，不同於使用 `def` 與 `return` 保留字所組成的結構來定義函數，Lambda 可以在不指定函數名稱的情況下「同時」定義並使用，並且只有在定義的當下被使用。

傳統的函數型函數作法

```
# The traditional way
def function():
    ...
    return ...

functional_function(function, iterable)
```

In [22]:

```
def square(x):
    return x**2
def my_favorite_avenger_movie(x):
    return x == "The Avengers"

primes = [2, 3, 5, 7, 11]
avenger_movies = ["The Avengers", "Avengers: Age of Ultron", "Avengers: Infinity War", "Avengers: Endgame
print(list(map(square, primes)))
print(list(filter(my_favorite_avenger_movie, avenger_movies)))
```

```
[4, 9, 25, 49, 121]
['The Avengers']
```

搭配 Lamda 敘述的函數型函數作法

```
# The Lambda expression way
functional_function(lambda INPUTS: OUTPUTS, iterable)
```

In [23]:

```
primes = [2, 3, 5, 7, 11]
avenger_movies = ["The Avengers", "Avengers: Age of Ultron", "Avengers: Infinity War", "Avengers: Endgame
print(list(map(lambda x: x**2, primes)))
print(list(filter(lambda x: x == "The Avengers", avenger_movies)))
```

```
[4, 9, 25, 49, 121]
['The Avengers']
```

重點統整

- Comprehension 指的是依賴可迭代類別來建立 `list`、`set` 或者 `dict` 的語法。
- Comprehension 可以結合 `if` 與 `if...else...` 敘述。
- Comprehension 所產出的是資料結構所儲存的資料，Generator 則是儲存資料的產生規則。

重點統整（續）

- Generator 的三個特性
    - 具備儲存與計算的效率性。
    - 沒有顯示外觀。
    - 實例化為物件的次數僅有一次。
- 輸出具有 Generator 特性的內建函數
    - 迭代器函數：`reversed()`、`enumerate()`、`zip()`
    - 函數型函數：`map()`、`filter()`
- Lamda 敘述常用來搭配函數型函數使用。