Python 的 50+ 練習：資料科學學習手冊

> *使用資料結構類別管理資料*

數據交點 | 郭耀仁 yaojenkuo@datainpoint.com

這個章節會登場的保留字與函數

- `del` 保留字。
- `len()` 函數。
- `list()` 函數。
- `tuple()` 函數。
- `dict()` 函數。
- `set()` 函數。

關於資料結構

什麼是資料結構

> *資料結構是電腦儲存、組織以及取得資料的機制，可以透過程式語言所提供的類別與自行定義類別實現，一個設計良好的資料結構，會在盡可能使用較少的時間與空間資源下，支援各種程式執行。*

來源：https://en.wikipedia.org/wiki/Data_structure

為什麼需要資料結構

- 在資料科學家日常的工作任務中，資料處理佔有相當高的比例。
- 需要有一個機制能夠協助他們輸入、處理最後輸出資料。
- 這個「機制」就是**資料結構**。
- 適當地選擇資料結構，讓資料科學家能夠有效率地儲存與取得資料。
- 就像是將食物放置在冰箱、衣服放置在衣櫥、鞋子放置在鞋櫃。

Python 內建的四個資料結構類別

1. `list`
2. `tuple`
3. `dict` (dictionary)
4. `set`

基礎的 `list` 類別

`list`

- `list` 是一種「有序」且能夠「更新」的資料結構。
- `list` 可以透過「逗號」`,` 分隔值與「中括號」`[]` 形成。

命名物件作為 `list` 類別的實例

In [1]:
```
primes = [2, 3, 5, 7, 11]
type(primes)
```

Out[1]:
```
list
```

使用內建函數 `len()` 得知一個 `list` 中有幾個資料值

In [2]:
```
len(primes)
```

Out[2]:
```
5
```

從一個 `list` 中取出資料的方式有兩種

1. **indexing** 指的是從一個 `list` 中取出特定位置的單個資料值。
2. **slicing** 指的是從一個 `list` 中擷取特定片段。

indexing `[index]`

`list` 採用兩個方向的索引機制：

1. 由左至右：「從 0 開始」的索引機制。
2. 由右至左：「從 -1 開始」的索引機制。

In [3]:

```
print(primes[0])  # the first element
print(primes[1])  # the second element
print(primes[-1]) # the last element
print(primes[-2]) # the second last element
```

```
2
3
11
7
```

slicing `[start:stop:step]`

除了可以取出特定位置的單個資料值，`list` 也支援擷取特定片段，藉此獲得一個較短長度 `list` 的語法。

- `start` 起始位置（包含）。
- `stop` 終止位置（排除）。
- `step` 間隔。

In [4]:

```
print(primes[0:3:1])            # slicing the first 3 elements
print(primes[-3:len(primes):1]) # slicing the last 3 elements
print(primes[0:len(primes):2])  # slicing every second element
```

```
[2, 3, 5]
[5, 7, 11]
[2, 5, 11]
```

slicing 如果沒有指定 `start:stop:step` 就採用預設值

- `start` 起始位置（包含）預設為 `0`。
- `stop` 終止位置（排除）預設為 `list` 的長度。
- `step` 間隔預設為 `1`。

In [5]:

```
print(primes[:3])  # slicing the first 3 elements
print(primes[-3:]) # slicing the last 3 elements
print(primes[::2]) # slicing every second element
```

```
[2, 3, 5]
[5, 7, 11]
[2, 5, 11]
```

`str` 也適用 indexing 與 slicing

In [6]:

```
luke_skywalker = "Luke Skywalker"
print(luke_skywalker[0])
print(luke_skywalker[-1])
print(luke_skywalker[:4])
print(luke_skywalker[5:])
```

```
L
r
Luke
Skywalker
```

更新 `list` 中的資料值

In [7]:

```
print(primes)   # before update
primes[-1] = 13 # update
print(primes)   # after update
```

```
[2, 3, 5, 7, 11]
[2, 3, 5, 7, 13]
```

也可以透過 `list` 的多種方法更新

- `list.append()`
- `list.pop()`
- `list.remove()`
- `list.insert()`
- `list.sort()`
- …等。

使用 `help()` 函數查詢 `list` 方法的說明

In [8]:
```
help(primes.append)
```

```
Help on built-in function append:

append(object, /) method of builtins.list instance
    Append object to the end of the list.
```

在小括號中按 `Shift - Tab` 查詢方法的說明

In [9]:
```
#primes.append()
```

`list` 支援的運算符

- `+` 運算符：連接 lists
- `*` 運算符：複製 `list` 中的元素。

In [10]:

```
primes = [2, 3, 5, 7, 11]
primes_to_concatenate = [13, 17, 19, 23, 29]
primes + primes_to_concatenate
```

Out[10]:

```
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29]
```

In [11]:

```
print(primes * 2)
print(primes_to_concatenate * 3)
```

```
[2, 3, 5, 7, 11, 2, 3, 5, 7, 11]
[13, 17, 19, 23, 29, 13, 17, 19, 23, 29, 13, 17, 19, 23, 29]
```

兩種函數的使用形式

在應用 `list` 的多種方法之前，先注意函數的兩種使用形式

1. 對物件應用函數，語法為 `function(object)`
2. 使用附屬於物件的函數，稱為使用物件的方法，語法為 `object.method()`

除了使用函數與方法的差異，也要留意物件更新的兩種方式

1. 將更新的結果回傳（Return），物件本身維持不變。
2. 更新物件本身並回傳 `None`

兩種更新方式的範例：排序一個 `list`

1. 使用內建函數 `sorted()` 採取將更新的結果回傳（Return），物件本身維持不變的更新方式。
2. 使用 `list.sort()` 採取更新物件本身並回傳 `None` 的更新方式。

In [12]:

```
unsorted_primes = [11, 5, 7, 2, 3]
sorted_primes = sorted(unsorted_primes)
print(unsorted_primes)
print(sorted_primes)
```

```
[11, 5, 7, 2, 3]
[2, 3, 5, 7, 11]
```

In [13]:

```
unsorted_primes = [11, 5, 7, 2, 3]
sorted_primes = unsorted_primes.sort()
print(unsorted_primes)
print(sorted_primes)
```

```
[2, 3, 5, 7, 11]
None
```

如何判斷兩種更新方式

- 詳細閱讀函數與方法的文件。
- 注意函數與方法使用後儲存格是否有 Out 顯示。

詳細閱讀函數與方法的說明

內建函數 `sorted()`：**Return** a new list containing all items from the iterable in ascending order.

In [14]:
```
help(sorted)
```

```
Help on built-in function sorted in module builtins:

sorted(iterable, /, *, key=None, reverse=False)
    Return a new list containing all items from the iterable in ascending order.

    A custom key function can be supplied to customize the sort order, and the
    reverse flag can be set to request the result in descending order.
```

詳細閱讀函數與方法的說明（續）

`list.sort()`：Sort the list in ascending order and **return None**.

In [15]:
```
help(unsorted_primes.sort)
```

```
Help on built-in function sort:

sort(*, key=None, reverse=False) method of builtins.list instance
    Sort the list in ascending order and return None.

    The sort is in-place (i.e. the list itself is modified) and stable (i.e. the
    order of two equal elements is maintained).

    If a key function is given, apply it once to each list item and sort them,
    ascending or descending, according to their function values.

    The reverse flag can be set to sort in descending order.
```

注意函數與方法使用後儲存格是否有 Out 顯示

In [16]:
```
unsorted_primes = [11, 5, 7, 2, 3]
sorted(unsorted_primes)
```

Out[16]:
```
[2, 3, 5, 7, 11]
```

In [17]:
```
unsorted_primes.sort()
```

兩種更新方式並不一定是「擇一」也可能都有

In [18]:

```
primes = [2, 3, 5, 7, 11]
the_last_element = primes.pop()
print(primes)
print(the_last_element)
```

```
[2, 3, 5, 7]
11
```

釐清兩種更新方式是至關重要的

- 雖然是利用 `list` 作為範例，但不管任何的資料或者資料結構都適用。
- 多數的函數更新機制都是回傳（Return），但並不是絕對。
- 不論函數或者方法，都要詳細閱讀文件或注意使用後儲存格是否有 Out 顯示。

不能更新資料的 `tuple` 類別

`tuple`

- `tuple` 是一種「有序」且「不能夠更新」的資料結構。
- `tuple` 可以透過「逗號」`,` 分隔值與「小括號」`()` 形成。

命名物件作為 `tuple` 類別的實例

In [19]:
```
primes = (2, 3, 5, 7, 11)
type(primes)
```

Out[19]:
```
tuple
```

使用內建函數 `len()` 得知一個 `tuple` 中有幾個資料值

In [20]:
```
len(primes)
```

Out[20]:
```
5
```

`tuple` 在許多地方的表現與 `list` 相同，像是 indexing 以及 slicing

In [21]:
```
print(primes[0])   # the first element
print(primes[:3])  # slicing the first 3 elements
```

```
2
(2, 3, 5)
```

`tuple` 與 `list` 最大的不同點，在於 `tuple`「不能夠更新」的特性

- 以更新 `list` 的語法更新 `tuple` 會產生錯誤。
- `try…except…` 是例外處理的語法，之後在「使用流程控制管理程式區塊的執行」章節會說明。

In [22]:
```
try:
    primes[-1] = 13
except TypeError as error_message:
    print(error_message)
```

```
'tuple' object does not support item assignment
```

在 Python 中有眾多的應用場景會遭遇到 `tuple`

- 自行定義函數預設「多個輸出」的資料結構。
- `dict.items()` 輸出。
- 函數的彈性參數 `*args`
- ...等。

自行定義函數預設「多個輸出」的資料結構

In [23]:

```
def retrieve_the_first_and_last_element(x):
    first_element = x[0]
    last_element = x[-1]
    return first_element, last_element # did not specify returning a tuple

primes = [2, 3, 5, 7, 11]
print(retrieve_the_first_and_last_element(primes))
print(type(retrieve_the_first_and_last_element(primes)))
```

```
(2, 11)
<class 'tuple'>
```

命名兩個物件作為自行定義函數的多個輸出

In [24]:

```
first, last = retrieve_the_first_and_last_element(primes)
print(first)
print(last)
```

```
2
11
```

前述寫法比下列寫法來得精簡

In [25]:
```
two_outputs = retrieve_the_first_and_last_element(primes)
first = two_outputs[0]
last = two_outputs[1]
print(first)
print(last)
```

```
2
11
```

以「鍵」查找「值」的 `dict` 類別

`dict`

- `dict` 是一種使用「鍵值對應」關係的資料結構。
- `dict` 可以透過「逗號」`,`、「鍵值對應」`key: value` 與「大括號」`{}` 形成。

In [26]:

```
the_shawshank_redemption = {
    "title": "The Shawshank Redemption",
    "year": 1995,
    "rating": 9.3,
    "director": "Frank Darabont"
}
type(the_shawshank_redemption)
```

Out[26]:

```
dict
```

使用內建函數 `len()` 得知一個 `dict` 中有幾組鍵值對應

In [27]:
```
len(the_shawshank_redemption)
```

Out[27]:
```
4
```

`dict` 採用以「鍵」取「值」的索引機制

```
dict["key"]
```

In [28]:

```
print(the_shawshank_redemption["title"])
print(the_shawshank_redemption["year"])
print(the_shawshank_redemption["rating"])
print(the_shawshank_redemption["director"])
```

```
The Shawshank Redemption
1995
9.3
Frank Darabont
```

用來檢視 `dict` 的三個方法

1. `dict.keys()` 檢視鍵。
2. `dict.values()` 檢視值。
3. `dict.items()` 檢視鍵值對應。

In [29]:

```
print(the_shawshank_redemption.keys())
print(the_shawshank_redemption.values())
print(the_shawshank_redemption.items())  # (key, value) as a tuple
```

```
dict_keys(['title', 'year', 'rating', 'director'])
dict_values(['The Shawshank Redemption', 1995, 9.3, 'Frank Darabont'])
dict_items([('title', 'The Shawshank Redemption'), ('year', 1995), ('rating', 9.
3), ('director', 'Frank Darabont')])
```

新增 `dict` 中的鍵值對應

In [30]:
```
the_shawshank_redemption['lead_actors'] = ['Tim Robbins', 'Morgan Freeman']
the_shawshank_redemption
```

Out[30]:
```
{'title': 'The Shawshank Redemption',
 'year': 1995,
 'rating': 9.3,
 'director': 'Frank Darabont',
 'lead_actors': ['Tim Robbins', 'Morgan Freeman']}
```

使用 `del` 保留字刪除 `dict` 中的鍵值對應

In [31]:
```
del the_shawshank_redemption['lead_actors']
the_shawshank_redemption
```

Out[31]:
```
{'title': 'The Shawshank Redemption',
 'year': 1995,
 'rating': 9.3,
 'director': 'Frank Darabont'}
```

使用 `dict.pop(key)` 方法指定「鍵」將「值」拋出

In [32]:
```
the_shawshank_redemption.pop("director")
```

Out[32]:
```
'Frank Darabont'
```

指定 `dict` 的「鍵」更新「值」

In [33]:
```
the_shawshank_redemption["year"] = 1994
the_shawshank_redemption
```

Out[33]:
```
{'title': 'The Shawshank Redemption', 'year': 1994, 'rating': 9.3}
```

能做集合運算的 `set` 類別

`set`

- `set` 是一種「無序」、儲存「獨一值」並且能夠進行「集合運算」的資料結構。
- `set` 可以透過「逗號」`,` 分隔值與「大括號」`{}` 形成。

In [34]:
```
primes = {2, 3, 5, 7, 7}  # 7 is duplicated
odds = {1, 3, 5, 7, 9, 9} # 9 is duplicated
print(type(primes))
print(type(odds))
```

```
<class 'set'>
<class 'set'>
```

使用內建函數 `len()` 得知一個 `set` 中有幾個獨一值

In [35]:

```
print(len(primes))
print(primes)
print(len(odds))
print(odds)
```

```
4
{2, 3, 5, 7}
5
{1, 3, 5, 7, 9}
```

`set` 重要的兩個特性

1. 不支援 indexing
2. 支援集合運算（Set operations）。

`set` 不支援 indexing

In [36]:
```
try:
    primes[0]
except TypeError as error_message:
    print(error_message)
```

```
'set' object is not subscriptable
```

`set` 支援集合運算

- 使用集合運算符（Set operators）。
- 使用 `set` 類別的方法。

常用的集合運算

- 交集。
- 聯集。
- 差集。
- 對稱差集。

使用 `&` 交集運算符或者 `set.intersection()` 方法

In [37]:

```
print(primes & odds)             # with an operator
print(primes.intersection(odds)) # equivalently with a method
```

```
{3, 5, 7}
{3, 5, 7}
```

使用 `|` 聯集運算符或者 `set.union()` 方法

In [38]:

```
print(primes | odds)      # with an operator
print(primes.union(odds)) # equivalently with a method
```

```
{1, 2, 3, 5, 7, 9}
{1, 2, 3, 5, 7, 9}
```

使用 `-` 差集運算符或者 `set.difference()` 方法

In [39]:

```
print(primes - odds)           # with an operator
print(primes.difference(odds)) # equivalently with a method
print(odds - primes)           # with an operator
print(odds.difference(primes)) # equivalently with a method
```

```
{2}
{2}
{1, 9}
{1, 9}
```

使用 `^` 對稱差集運算符或者 `set.symmetric_difference()` 方法

In [40]:

```
print(primes ^ odds)                      # with an operator
print(primes.symmetric_difference(odds))  # equivalently with a method
```

```
{1, 2, 9}
{1, 2, 9}
```

資料結構的判斷與轉換

資料結構類別判斷的兩種方式

1. 將 `type()` 函數的輸出與資料結構類別名稱比較。
2. 透過內建函數 `isinstance()` 判斷。

資料結構類別可以透過內建函數 `type()` 判斷

將 `type()` 函數的輸出與資料結構類別名稱比較。

In [41]:

```
primes_list = [2, 3, 5, 7, 11]
primes_tuple = (2, 3, 5, 7, 11)
print(type(primes_list) == list)
print(type(primes_tuple) == list)
```

```
True
False
```

In [42]:

```
primes_tuple = (2, 3, 5, 7, 11)
primes_set = {2, 3, 5, 7, 11}
print(type(primes_tuple) == tuple)
print(type(primes_set) == tuple)
```

```
True
False
```

資料結構類別可以透過內建函數 `isinstance()` 判斷

`isinstance(x, classinfo)` 函數判斷輸入物件 `x` 是否為某個資料結構類別的實例，其中 `classinfo` 參數輸入欲判斷的資料結構類別名稱。

In [43]:

```
primes_list = [2, 3, 5, 7, 11]
primes_tuple = (2, 3, 5, 7, 11)
print(isinstance(primes_list, list))
print(isinstance(primes_tuple, list))
```

```
True
False
```

In [44]:

```
primes_tuple = (2, 3, 5, 7, 11)
primes_set = {2, 3, 5, 7, 11}
print(isinstance(primes_tuple, tuple))
print(isinstance(primes_set, tuple))
```

```
True
False
```

資料結構類別可以透過與其同名的內建函數轉換

- `list()` 可以將輸入資料結構轉換為 `list`。
- `tuple()` 可以將輸入資料結構轉換為 `tuple`。
- `dict()` 可以將輸入資料結構轉換為 `dict`。
- `set()` 可以將輸入資料結構轉換為 `set`。

總結三種括號的應用場景

截至目前為止，我們已經看過了三種括號的應用場景

1. 小括號 `()`
2. 中括號 `[]`
3. 大括號 `{}`

小括號 `()` 的應用場景

- 對物件應用函數：`function(object)`
- 運算的優先順序。
- 使用物件的方法：`object.method()`
- 形成 `tuple` 資料結構類別。

中括號 `[]` 的應用場景

- 形成 `list` 資料結構類別。
- 從不同資料結構類別 indexing/slicing 資料。

大括號 `{}` 的應用場景

- 與 `str.format()` 以及 f-string 語法搭配。
- 形成 `dict` 資料結構類別。
- 形成 `set` 資料結構類別。

Python 資料結構類別具備有威力的「複合性」

複合性指的是資料結構類別中能夠包含異質的資料類別以及不同的資料結構類別，在資料處理上相當有優勢。

- 資料類別：`int` / `float` / `str` / `bool` / `NoneType`
- 資料結構類別：`list` / `tuple` / `dict` / `set`

重點統整

- 適當地選擇資料結構，讓資料科學家能夠有效率地儲存與取得資料。
- Python 內建的四個資料結構類別
    - `list`
    - `tuple`
    - `dict` (dictionary)
    - `set`

重點統整（續）

- `list` / `tuple` 採用兩個方向的索引機制：
    - 由左至右：「從 0 開始」的索引機制。
    - 由右至左：「從 -1 開始」的索引機制。
- `tuple` 是一種不能更新資料的的資料結構。
- `dict` 是一種使用「鍵值對應」關係的資料結構。
- `set` 是一種支援集合運算的資料結構。
- 總結三種括號的應用場景。