CHAPTER

5

綜合應用一

綜合應用一

在前面章節的範例與練習題中，相信大家已經發現串列好用之處。除了能容納不同資料型別的元素，也有依靠索引值與迭代的存取方式，更有串列綜合運算以及多個方便的操作函式。透過串列不僅能輕易撰寫解決問題的程式，而且也能讓程式碼更加簡潔。儘管串列好用，但其執行速度也比較慢，當處理的資料量一大就很容易感覺到差異。同時，在應對某些情況時使用串列需要轉換邏輯思維，並不直覺。Python 除了串列，還有三個常見能儲存資料的容器，分別是元組（tuple）、字典（dictionary）與集合（set），將在本章一一介紹。

在程式執行的過程中，難免有機會發生錯誤，原因可能是使用者的輸入不合法、檔案不存在、資料型別不符或甚至是網路斷線等。在程式執行期間發生錯誤時會引發例外（exception），若沒有規劃好相應的處理機制，程式很可能因此中斷或發生不可預期的情況。試想，當不小心輸入錯誤時，會希望程式直接中斷，亦或是提醒輸入有誤可再重新輸入呢？倘若客戶直接看到系統給的錯誤訊息，對產品品質的信心也會大打折扣。為了避免上述窘境，在程式開發過程中就要考慮到當錯誤發生時，程式要給出哪些適當的反應，並據此撰寫例外處理（exception handling）。除了不讓程式輕易中斷外，也能友善地提醒並引導使用者進行操作。

5-1 元組、字典與集合

串列、元組、字典與集合都是 Python 常用的資料結構（data type），它們各自具有不同的特性和用途。初學 Python 時大多由串列入手，在熟練一段時間後，當撰寫程式過程中需要存放資料時，第一時間想到的就是串列。可是隨著接觸的需求越來越多元，單單只倚賴串列難免會有力不從心的感覺。畢竟靠一招半式就想闖蕩江湖，有些不切實際了。

5-1-1 元組

語法

元組（tuple）的語法如下：

```
元組名稱 = (元素 1, 元素 2, 元素 3, …)
```

元組和串列一樣都是有順序的序列結構，但有以下幾點不同：

- 元組在建立後不能修改元素值。
- 元組使用小括號，但串列使用中括號。
- 如果元組內只有一個元素，後方必須加上逗點，多個元素就不用。

範例程式：

```
1  tup1 = ()          # 建立一個空元組，也可用 tuple()
2  tup2 = ("皮卡丘", "小火龍", "傑尼龜")
3  print(type(tup2))
4  print(tup2[1])
5
6  tup = tup2 + ("可達鴨",)
7  lst = list(tup)  # 利用串列間接修改元組內容
8  lst[0] = "雷丘"
9  tup = tuple(lst)
10 print(tup)
```

輸出結果：

```
<class ´tuple´>
小火龍
(´雷丘´, ´小火龍´, ´傑尼龜´, ´可達鴨´)
```

元組在使用上與串列頗為相似，若建立時沒有指派儲存值，則必須先宣告，如同第 1 行使用一對小括號或是內建函式 tuple()來進行；而雖然建立時用小括號，存取時也和串列一樣將索引值放在中括號內，如第 4 行般。第 6 行以類似字串的結合方式，透過「+」號結合兩個元組，這裡還有一個要特別注意的地方，若元組內只有一個元素，該元素後面要加上逗點。接著第 9 ~ 11 行則是以串列為媒介，間接修改內容後再轉回元組。相較於串列，雖然元組在使用上有不少限制，可是也有讀取速度快、資料更安全等優點，而 Python 提供的內建和第三方函式，其回傳值也大多是元組結構，以確保資料不會被變更。順帶一提，數學上所謂的元組是指有限個元素所組成的有序序列，若元組有 n 個元素，通常稱為 n 元組（n-tuple），例如 (2, 7, 4, 1, 7) 即為 5 元組。

5-1-2 字典

語法

字典（dictionary）的語法如下：

```
字典名稱 = {鍵 1: 值 1, 鍵 2: 值 2, …}
```

由字典的語法可以看到，其元素是以「鍵:值」對的方式儲存，可透過以鍵（key）為索引值來存取字典裡對應的值（value），而「鍵」可以是數字或字串，只要不重複就行了，至於「值」可以是前面學過的數字、字串、串列，甚至是字典也行。若把串列的索引值看成鍵，那麼串列也可視為一種鍵皆為數字的特殊字典，但是與串列相比，字典有以下幾點不同：

- 字典使用大括號，而串列使用中括號。
- 元素在字典內沒有順序（與設定順序無關），但串列是有序結構。

範例程式：

```
1  dct = {}          # 建立一個空字典，也可用 dict()
2  hp_dct = {"皮卡丘":90, "小火龍":80, "傑尼龜":75}
3  print(type(hp_dct))
4  print(hp_dct)
5  print("小火龍 HP =", hp_dct["小火龍"])
6
7  if "傑尼龜" in hp_dct:
8      hp_dct["傑尼龜"] = 85     # 修改元素內容
9  else:
10     hp_dct["傑尼龜"] = 85     # 新增元素內容
11
12 print("傑尼龜 HP =", hp_dct["傑尼龜"])
```

輸出結果：

```
<class 'dict'>
{'皮卡丘': 90, '小火龍': 80, '傑尼龜': 75}
小火龍 HP = 80
傑尼龜 HP = 85
```

與串列、元祖相仿，若建立字典時沒有同時指派儲存值，則需要以一對大括號或內建函式 dict()進行宣告，而存取方式則是把「鍵」放在中括號內，以取出對應字典元素的「值」，如第 5 行那般。接著是第 7 行的 if 敘述，之所以這樣寫是因為想區分出究竟是修改（第 8 行）抑或是新增（第 10 行）元素，所以才先確認這個鍵是否有在字典內，避免不小心修改到元素內容。至於刪除字典內容，有下列三種方式：

範例程式：

```
1  dct = {"皮卡丘":90, "小火龍":80, "傑尼龜":75}
2
3  del dct["皮卡丘"]     # 刪除字典內的特定元素
4  print(dct)
5  dct.clear()          # 刪除字典內所有元素
6  print(dct)
7  del dct              # 刪除字典變數
8  print(dct)
```

輸出結果：

```
{'小火龍': 80, '傑尼龜': 75}
{}
NameError: name 'dct' is not defined
```

如果只是想單純取得字典的所有鍵，其實可以簡單地透過 for 迴圈，把字典當成可迭代物件來一一拜訪。例如：

範例程式：

```
1  dct = {"皮卡丘":90, "小火龍":80, "傑尼龜":75}
2
3  for key in dct:
4      print(f"key:{key} -> value:{dct[key]}")
```

輸出結果：

```
key:皮卡丘 -> value:90
key:小火龍 -> value:80
```

```
key:傑尼龜 -> value:75
```

Python 對字典提供的操作方法如下表所示，表中的 dct = {'A':3, 'B':5}：

方法	意義	範例	變數 n 的值
len(dct)	取得字典元素個數	n = len(dct)	2
copy()	複製字典	n = dct.copy()	{'A':3, 'B':5}
in / not in	檢查鍵是否在字典內	'X' in dct	False
items()	取出所有元素的「鍵:值」	n = dct.items()	[('A':3), ('B':5)]
keys()	取出所有元素的「鍵」	n = dct.keys()	['A', 'B']
values()	取出所有元素的「值」	n = dct.values()	[3, 5]
pop()	取得「鍵」對應的「值」，並刪除該「鍵:值」。	n = dct.pop('B')	5
get(鍵, 值)	取得「鍵」對應的「值」，若該「鍵」不存在，則回傳參數內的值。	n = dct.get('B', 9)	5
		n = dct.get('X', 9)	9
Setdefault (鍵, 值)	若「鍵」不存在，則以參數的「鍵:值」建立新元素。	dct.setdefault ('A', 9)	{'A':3, 'B':5}
		dct.setdefault ('X', 9)	{'A':3, 'B':5, 'X':9}

範例程式：

```
1  dct = {'X':90, 'a':80, 'A':75, 'x':65}
2
3  for key, val in dct.items():
4      print(f"[{key}]->{val},", end=' ')
5
6  print("\n對「鍵」排序：", end=' ')
7  for key in sorted(dct):
8      print(f"[{key}]->{dct[key]},", end=' ')
9
10 print("\n對「值」排序：", end=' ')
11 for key in sorted(dct, key=dct.get):
12     print(f"[{key}]->{dct[key]},", end=' ')
```

輸出結果：

```
[X]->90, [a]->80, [A]->75, [x]->65,
對「鍵」排序： [A]->75, [X]->90, [a]->80, [x]->65,
對「值」排序： [x]->65, [A]->75, [a]->80, [X]->90,
```

透過 items()方法可在 for 迴圈內逐步取出所有「鍵:值」元素，而之前也提到字典內的元素並沒有順序，所以這個範例也展示分別以「鍵」和「值」來排序的作法。要排序字典的「鍵」直接利用 sorted()函式即可，但要以「值」作為排序的依據就得麻煩些，要搭配 get()方法來進行，如第 11 行。此外，若要將多個字典合併成一個，可以透過底下範例的兩個方法，但需要注意同一個「鍵」的「值」會被覆蓋掉：

範例程式：

```
1  dct1 = {"皮卡丘":90, "小火龍":80, "傑尼龜":75}
2  dct2 = {"雷丘":120, "傑尼龜":90}
3  dct1.update(dct2)
4  print(dct1)
5
6  dct3 = {"皮卡丘":10, "小火龍":20}
7  # 「**」會將字典拆解為元素，再由大括號組合，需注意相同「鍵」的覆蓋
8  dct = {**dct1, **dct2, **dct3}
9  print(dct)
```

輸出結果：

```
{'皮卡丘': 90, '小火龍': 80, '傑尼龜': 90, '雷丘': 120}
{'皮卡丘': 10, '小火龍': 20, '傑尼龜': 90, '雷丘': 120}
```

CHAPTER

6

綜合應用二

綜合應用二

在我們對程式語言還不嫻熟，開發經驗也還不足的時候，一旦遇到稍微複雜或是大型應用程式的開發需求時，難免手忙腳亂。好不容易將問題拆解成數個待實現的功能，卻又千頭萬緒不知從何做起。而 Python 最為人津津樂道的特色之一就是大量功能覆蓋眾多領域的模組（module）與套件（package），能滿足各式各樣應用的開發需求。將這些模組與套件想像成許多積木，可以快速取用、組合以實現各種功能，不僅增加開發效能，也讓程式碼更容易維護。因此，本章將介紹如何匯入模組來使用。

此外，我們在 Chapter 3 學習過函式，能協助將重複或有特別定義的程式片段，拆開成容易管理且獨立的程式片段。有提高程式可讀性、容易除錯與維護等優點，而採用這種模組化開發也有利於團隊分工。因此，本章繼續介紹 Python 函式的參數傳遞模式與匿名函式。

6-1 模組與套件

事實上，模組也是一個 Python 程式的檔案，裡面定義了一些資料、函式與類別。要使用模組所提供的功能時，得先匯入（import）該模組；而套件可以想成是一堆模組的集合，能提供更全面的功能，使用前也一樣要先匯入。

6-1-1 匯入模組

語法

匯入模組與套件的方式一樣，語法如下：

```
import 模組或套件名稱
```

範例程式：

```
1  import random          # 匯入亂數模組
2
3  print(random.random())          # 在0~1間隨機取一個浮點數
4  print(random.randint(1, 10))    # 在1~10間隨機取一個整數
5  print(random.uniform(1, 10))    # 在1~10間隨機取一個浮點數
```

輸出結果：

```
0.9465516603981697
9
1.0139022503442041
```

模組內通常有許多函式提供我們挑選，在匯入之後就能透過內建函式 dir() 列出模組內所有函式，也可用 help() 查閱更詳細的使用說明。上述範例在第 1 行匯入亂數模組 random 後，接著就能使用模組提供的函式產生符合我們需求的亂數。有些模組的名稱很長，每次使用時都要輸入名稱也很麻煩。因此，有幾種比較簡便的使用方式，例如單獨匯入模組中要使用的函式名稱，或是透過萬用字元「*」直接匯入模組的所有函式，但不推薦這種 all in 的匯入方式，因為容易與其他模組同名稱的函式相衝突。

```
from random import randint, uniform
from random import *
```

這種 from…import 的用法雖然免去使用模組函式時要先輸入模組名稱的麻煩，但也需要對該模組有相當程度的了解，且使用時因為沒有標註模組名稱，也容易誤會或與自己撰寫的同名稱函式相混在一起。另一個作法則是幫模組取一個簡短的別名，以此在後續的程式碼中替代模組名稱。例如：

範例程式：

```
1 import random as rd
2
3 print(rd.random())
4 print(rd.choice([1, 2, 3, 4, 5]))      # 隨機挑1個號碼
5 print(rd.sample([1, 2, 3, 4, 5], 3))   # 隨機挑3個不重複號碼
```

輸出結果：

```
0.3753755382924715
5
[3, 4, 1]
```

再舉一個實用的模組為例，有些問題可以透過暴力窮舉所有可能性，再逐一檢驗排查。例如只使用 1 ~ 4 個數字組合成兩位數，且個位數要大於十位數。這類排列組合問題可透過兩層巢狀迴圈（因為是兩位數）列舉所有可能性並檢驗之，可是若題

目再複雜些，使用多層巢狀迴圈容易降低可讀性與執行效率。Python 有一個內建模組 itertools 可協助輕鬆產生所有排列組合，例如：

範例程式：

```
1  import itertools as it
2
3  for x in it.permutations(range(1,5), 2):  # 窮舉所有排列
4      if x[0] < x[1]:
5          print(x[0]*10 + x[1], end=' ')
6
7  print("\n === 底下是所有組合 ===")
8  print(list(it.combinations(range(1,5), 2)))
```

輸出結果：

```
12 13 14 23 24 34
=== 底下是所有組合 ===
[(1, 2), (1, 3), (1, 4), (2, 3), (2, 4), (3, 4)]
```

這個範例的第 3 行先窮舉所有兩位數的排列可能，再以 for 迴圈逐一探訪並檢驗是否滿足題目要求，而第 8 行則是窮舉所有組合，即 12 與 21 是同一筆，本章後面的習題可找到相關題目來做練習。

此外，既然模組也是 Python 檔案，除了使用內建與第三方模組外，也能自己建立模組打包共用的程式，再由其他程式匯入使用。此時，匯入模組使用的目錄底下會多一個名為「__pycache__」的資料夾，內有「.pyc」檔案。這是 Python 透過解譯器（interpreter）對程式碼逐行翻譯成「位元組碼」（byte code）的結果，有助於提供更靈活架構與跨平台能力，也能提升載入模組的速度。

6-1-2 使用套件

如果把模組比喻成工具，套件就像工具箱一樣，裡面擺放數個模組。原則上，只要該目錄底下有包含「__init__.py」檔案，就會被當作 Python 的套件。相較於內建模組與套件會在安裝 Python 時也一併安裝，第三方套件（third-party package）則需要手動安裝後才能匯入使用。常見是在命令提示字元的視窗內，透過 pip 指令進行安裝，例如：

```
pip list                  # 列出目前安裝的套件與版本
pip show 套件名稱          # 查詢已經安裝的套件資訊
pip install 套件名稱       # 安裝指定套件
pip uninstall 套件名稱     # 移除指定套件
```

當我們透過 pip 指令來安裝套件時，預設是從 PyPI 網站下載套件。PyPI（the Python Package Index）是官方的第三方套件儲存庫，類似 Google Play 和 App Store 般。迄今為止，PyPI（https://pypi.org/）已收錄了近五十萬個專案，涵蓋眾多應用領域，在程式開發過程中所需要的功能，幾乎都能在這裡找到適合的套件來協助。以下列出幾個常見的第三方套件與其功能：

- NumPy：提供陣列與平行處理能力，許多重量級數據科學的相關套件都奠基在 numpy 的基礎上開發。
- SciPy：基於 numpy 開發而成，主要用於科學計算。
- Pandas：進行數據處理，能大幅簡化複雜的數據操作過程。
- PyGame：多媒體與遊戲開發。
- Statsmodels：統計分析。
- Scikit-learn：機器學習。
- TensorFlows、PyTorch：兩個主流的深度學習（deep learning）框架。

底下我們來看 NumPy 這個著名的第三方套件，它提供陣列與大量操作陣列的函式，同時也能處理傅立葉轉換、線性代數、多項式等科學計算。陣列（array）是一種能存放多筆資料的結構，與 Python 串列類似的地方是存放在陣列裡的資料也稱為元素（element），且一樣是透過索引值（index）來存取元素；然而，最明顯的差異在於陣列只能存放同型別的元素，Python 的串列則沒有這個限制。

NumPy 提供的陣列型別稱為 ndarray（n-dimensional array），其建立方式通常是使用 array()方法轉換串列而來。ndarray 三個最重要的屬性如下：

- ndarray.ndim：陣列的維度，例如 2 代表是二維陣列。
- ndarray.shape：陣列的形狀，以元組的結構儲存，每個整數代表該維度的元素個數，例如(3, 2)意謂第一維度有 3 個元素，而第二維度有 2 個元素。
- ndarray.dtype：陣列的元素型別，可以是 Python 內建的 int、bool、str 等型別，也可以是 NumPy 提供的 int32、float64 等型別。

範例程式：

```
1  import numpy as np
2
3  arr = np.array([1, 2, 3, 4, 5])
4  print(arr)
5  print("ndim =", arr.ndim)
6  print("shape =", arr.shape)
7  print("dtype =", arr.dtype)
8
9  print(np.arange(0, 2, 0.3))
10 print(np.linspace(0, 2, 5))
```

輸出結果：

```
[1 2 3 4 5]
ndim = 1
shape = (5,)
dtype = int32
[0.  0.3 0.6 0.9 1.2 1.5 1.8]
[0.  0.5 1.  1.5 2. ]
```

這個範例除了透過 array()方法轉換串列成陣列，也使用 NumPy 提供的 arrange()與 linspace()方法建立陣列，其中前者的用法類似 range()，同樣可指定起始、中止及間隔值，而後者的前兩個參數也是起始及中止值，最後一個參數則是在範圍內均勻產生的數量點。陣列可搭配條件式建立所謂的遮罩（mask），可用來選取陣列內的目標元素，例如：

範例程式：

```
1  import numpy as np
2
3  score = np.array([60, 11, 33, 70, 22, 80])
4  print("原始分數", score)
5  print("遮罩", score < 60)
6
7  score[score < 60] += 10     # 只選出低於60分的來加分
8  print("修改分數", score)
```

輸出結果：

```
原始分數 [60 11 33 70 22 80]
遮罩 [False  True  True False  True False]
修改分數 [60 21 43 70 32 80]
```

二維陣列（two-dimensional array）的操作方式與二維串列相仿，一樣是透過列索引與行索引來存取元素，其建立方式可由二維串列轉換而來，也可用 reshape() 方法改變一維陣列的形狀而成。例如：

範例程式：

```
1  import numpy as np
2
3  mat = np.array([[1, 2, 3], [4, 5, 6]])
4  print(mat)
5  print('='*10)
6  mat = np.array(range(1, 7)).reshape(2, 3)
7  print(mat)
```

輸出結果：

```
[[1 2 3]
 [4 5 6]]
==========
[[1 2 3]
 [4 5 6]]
```

矩陣（matrix）以二維陣列來表示後，就能進行一些矩陣基本運算，例如：

範例程式：

```
1  import numpy as np
2  mat_A = np.array(range(1, 7)).reshape(2, 3)
3  mat_add = mat_A + mat_A     # 矩陣對應項相加
4  print(mat_add)
5  mat_mul = mat_A * mat_A     # 矩陣對應項相乘
6  print(mat_mul)
```

輸出結果：

```
[[ 2  4  6]
 [ 8 10 12]]
[[ 1  4  9]
 [16 25 36]]
```

要注意的是上述範例裡第 5 行的矩陣相乘是矩陣內對應元素的相乘，因此只要相乘的兩個矩陣有同樣形狀即可進行。若是想得到線性代數裡的矩陣相乘結果，可使用 dot() 方法，此時若因為矩陣形狀限制而無法相乘，會得到「ValueError」錯誤訊息。因此，執行矩陣相乘的程式碼也常被放在 try 敘述內，以便於在引發錯誤時能捕捉到系統拋出的例外。

範例程式：

```
1  import numpy as np
2
3  try:
4      mat_A = np.array(range(1, 7)).reshape(2, 3)
5      mat_B = np.array(range(1, 7)).reshape(3, 2)
6      print(np.dot(mat_A, mat_B))
7      print(np.dot(mat_A, mat_A))
8  except:
9      print("矩陣無法相乘")
```

輸出結果：

```
[[22 28]
 [49 64]]
矩陣無法相乘
```

這個範例在第 4 與 5 行分別建立形狀為 2×3 與 3×2 兩個矩陣，這兩個矩陣相乘可得到 2×2 的矩陣，但第 7 行因為相乘的兩個矩陣形狀不對，故而進行例外處理，顯示無法相乘的錯誤訊息。做為許多重量級應用（如數據分析、機器學習、深度學習等）的基礎套件，NumPy 不僅支援高維度的陣列與矩陣運算，也具備大量的數學與統計函式庫以及平行處理的能力，可到 NumPy 官網（https://numpy.org/）查閱更多功能。

CHAPTER

7

綜合應用三

綜合應用三

我們在前面的章節已經知道把資料儲存在檔案內，比較能夠長久地保存，也知道如何進行開啟、讀取、寫入以及關閉檔案等操作。這樣已經能妥善地處理單一檔案，可是對於處理一個目錄下的所有檔案仍舊是力不從心。所幸 Python 有一些內建模組能協助管理目錄，且配合各種篩選方式能輕易取得符合條件的檔案列表，接著方能逐一處理目標檔案。因此，本章將介紹一些常見管理與拜訪目錄內所有檔案的模組與方法。

另一方面，當我們要透過電腦解決問題前，必須以電腦能理解的模式來描述問題，而資料結構（data structure）就是描述資料的方法，也是電腦內儲存資料的基本架構。資料結構不僅涉及儲存的資料與儲存方法，同時也考慮到存取資料的方式，能讓程式的執行速度加快，並降低佔用的計算資源（如 CPU、記憶體等）。著名的瑞士計算機科學家尼克勞斯・維爾特（Niklaus Wirth）曾在 1984 年因發展數個程式語言而獲圖靈獎（Turing Award），他寫過一本書的書名是『Algorithms + Data Structures = Programs』（演算法+資料結構=程式），這已經是計算機科學的名句。一個程式應該包括對「資料」與「操作」的描述，其中前者是指在程式中要指定資料的類型和資料的組織形式，亦即前面提到的資料結構；而後者則意指操作步驟，也就是演算法。本章將以簡單易懂的方式，介紹在資料結構裡最基本的堆疊（stack）與佇列（queue）兩個結構。

7-1 目錄管理

我們已經熟悉單一檔案的開啟、讀取及寫入等處理動作，但面對多個檔案，目前除了手動一個個利用程式處理外，似乎也沒有比較方便的做法。然而，許多實務應用的第一步大多是從眾多檔案裡讀取資料，這裡涉及目錄管理與檔案操作的各種技巧，能將整個操作過程自動化。底下我們介紹 Python 提供的 glob 與 os 兩個實用模組，以便於操作檔案與目錄。

7-1-1 檔案搜尋：glob 模組

若想快速地取得指定條件的檔案名稱，可使用 glob 模組來進行。這是一個簡單的檔案搜尋模組，除了能明確指定檔案名稱外，也可以搭配正規表示法（regular expression）來描述指定條件，比方說用萬用字元「*」代表所有可能性。舉例如下：

範例程式：

```
1  from glob import glob
2
3  files = glob('*.csv') + glob('*05.*')
4  print(type(files))
5
6  for f in files:
7      print(f)
```

資料夾內容：

輸出結果：

```
<class 'list'>
1.csv
2.csv
3.csv
FBAFC0002-05.png
FBAFC0003-05.jpg
```

這個範例的第 3 行描述要取出檔案名稱的條件，包括副檔名為 csv 以及檔名內有「05.」的所有檔案，而 glob()方法會把符合的檔名放在串列內。因此，第 6 行可透過 for 迴圈逐一瀏覽並處理目標檔案。

7-1-2 檔案操作：os 模組

如果要處理的目標檔案不是單純放置在單一資料夾內，而分散在許多資料夾，甚至是涉及數個不同目錄，可以考慮使用 os 模組。這個模組提供取得工作目錄、目錄與檔案操作、執行作業系統命令等，功能包山包海，但使用起來也較繁瑣。例如：

範例程式：

```
1  import os
2
3  print(os.getcwd())          # 取得目前的工作目錄
4
5  file = "myFile.txt"
6  if os.path.exists(file):    # 檢查檔案是否存在
7      os.remove(file)         # 移除指定檔案
8  else:
9      print(file + "檔案不存在")
10
11 os.system("cls")            # 執行作業系統命令，清除螢幕
12 os.system("mkdir myDir")    # 建立 myDir 目錄
```

輸出結果：

```
C:\Users\Ian\VS_Code
```

若要更細緻地處理檔案路徑與名稱、取得檔案各種屬性等功能，可使用 os.path 模組，其提供的方法整理如下：

	os.path 的方法	說明
檢查（True/False）	exists()	檔案或路徑是否存在
	isabs()	是否為絕對路徑
	isfile()	指定路徑是否為檔案
	isdir()	指定路徑是否為目錄
處理檔案路徑	abspath()	傳回絕對路徑
	basename()	傳回指定字串最後的檔案或路徑名稱
	dirname()	傳回指定檔案的絕對路徑
	split()	分割路徑成 dirname 和 basename
	splitdrive()	分割路徑成磁碟機與路徑名稱
	join()	合併路徑與檔名為完整路徑
處理檔案	getsize()	傳回檔案大小
	getctime()	傳回檔案建立時間
	getatime()	傳回檔案最後的訪問時間

範例程式：

```
1  import os.path as op
2
3  file = op.abspath("os_path.py")
4  print("檔案絕對路徑：", file)
5
6  if op.exists(file):
7      f_path, f_name = op.split(file)
8      print("檔案路徑：", f_path)
9      print("檔案名稱：", f_name)
10     print("檔案大小：", op.getsize(file), "bytes")
11
12 print("組合路徑：", op.join(f_path, f_name))
```

輸出結果：

```
檔案絕對路徑： C:\Users\Ian\VS_Code\myDir\os_path.py
檔案路徑： C:\Users\Ian\VS_Code\myDir
檔案名稱： os_path.py
檔案大小： 341 bytes
組合路徑： C:\Users\Ian\VS_Code\myDir\os_path.py
```

在使用 Python 開發處理檔案的程式時，時常會需要列舉目錄內的所有檔案名稱，然後再透過迴圈逐一處理之。此時，可使用 os.lisrdir 模組取得檔案列表，而 os.walk 模組更是能以遞迴（recursive）方式列出特定路徑下的所有子目錄與檔案。比方有一個檔案目錄如圖 7-1-1 所示，則列出所有目錄與檔案的程式可以這樣寫：

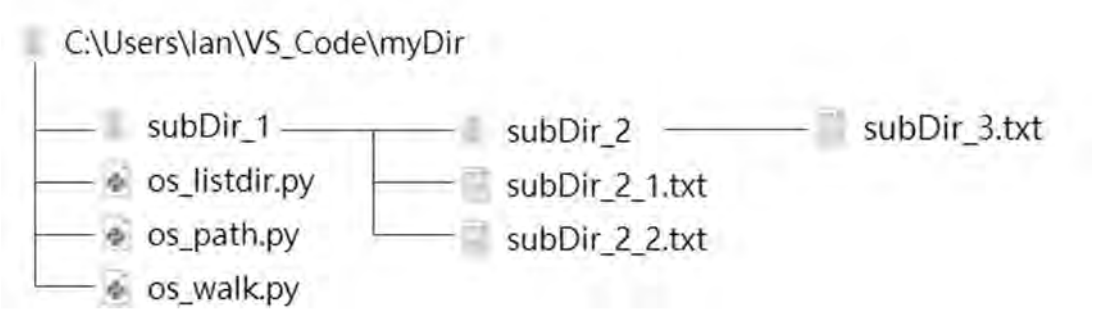

圖 7-1-1 電腦內的一個檔案目錄

範例程式：

```
import os.path as op
from os import listdir

my_path = r"C:\Users\Ian\VS_Code\myDir"
files = listdir(my_path)       # 取得所有檔案與目錄名稱

for f in files:
    full_path = op.join(my_path, f)

    if op.isfile(full_path):
        print("檔案：", f)
    elif op.isdir(full_path):
        print("目錄：", f)
```

輸出結果：

```
檔案： os_listdir.py
檔案： os_path.py
檔案： os_walk.py
目錄： subDir_1
```

os.walk()方法會回傳一個包含三個元素的元組，分別是「目前路徑」、「資料夾串列」以及「檔案串列」，且由於是採用遞迴方式處理，功能強大可是程式也不太好理解。同樣以圖 7-1-1 的目錄結構為例來撰寫底下範例：

範例程式：

```
import os.path as op
from os import walk

my_path = r"C:\Users\Ian\VS_Code\myDir"

# 遞迴列出所有目錄與檔案
for root, dirs, files in walk(my_path):
    print("路徑：", root)
    print("└目錄：", dirs)
    print("└檔案：", files)
```

輸出結果：

```
路徑： C:\Users\Ian\VS_Code\myDir
└目錄： ['subDir_1']
└檔案： ['os_listdir.py', 'os_path.py', 'os_walk.py']
路徑： C:\Users\Ian\VS_Code\myDir\subDir_1
└目錄： ['subDir_2']
└檔案： ['subDir_2_1.txt', 'subDir_2_2.txt']
路徑： C:\Users\Ian\VS_Code\myDir\subDir_1\subDir_2
└目錄： []
└檔案： ['subDir_3.txt']
```

倘若要取得目錄下所有檔案的絕對路徑，讓程式能逐一處理的話，可以如底下範例這樣改寫，搭配 os.path.basename()能進一步取出絕對路徑最後的檔案名稱，以便於繼續篩選目標檔案。

範例程式：

```
1  import os.path as op
2  from os import walk
3
4  my_path = r"C:\Users\Ian\VS_Code\myDir"
5
6  for root, dirs, files in walk(my_path):
7      for f in files:
8          full_path = op.join(root, f)
9          print(full_path)
```

輸出結果：

```
C:\Users\Ian\VS_Code\myDir\os_listdir.py
C:\Users\Ian\VS_Code\myDir\os_path.py
C:\Users\Ian\VS_Code\myDir\os_walk.py
C:\Users\Ian\VS_Code\myDir\subDir_1\subDir_2_1.txt
C:\Users\Ian\VS_Code\myDir\subDir_1\subDir_2_2.txt
C:\Users\Ian\VS_Code\myDir\subDir_1\subDir_2\subDir_3.txt
```