Chapter 4

函式

/ 蔣宗哲、江政杰

隨著程式碼的規模越來越大，我們的程式功能也越來越豐富；但同時也變得越來越難維護和除錯。本章將介紹程式語言中的重要元件 -- 函式，撰寫函式是邁向中型程式的必要過程，本章分為五節，內容包括：函式的基本概念與語法、函式與變數的可用點、預設引數和關鍵字引數、函式作為引數以及程式模組。本章以臺北市自行車竊案資料為分析標的，學習完本章後，讀者將能對多欄位的文字檔資料進行統計、查詢和排序等分析。

4-1 函式初探

到目前為止，我們在不知不覺中學會了使用許多 Python 的內建函式，例如關於輸入與輸出的 input() 與 print() 函式以及關於統計的 sum() 與 min() 函式。函式可以幫忙程式員用簡短、易懂且有效率的方式來撰寫程式。這一章就讓我們來學習如何撰寫自己的函式。

什麼是函式？

函式將一段程式碼包裝成一個單元，程式員呼叫函式以執行該段程式碼。該段程式碼會共同完成一個明白、特定的功能。Python 的函式可說是由三個部份來組成：

1. 名稱：以一個清楚易懂的名稱來代表該段程式碼的功能，例如 print。
2. 參數列：接收使用該函式的程式員所提供的資料，例如 print() 函式接收要列印的資料內容。
3. 主體：用以實際完成功能的程式碼。主體中還可以回傳計算後的結果。

讓我們以程式 E4-1-1 來說明。程式第 01 行以 def 開頭，即代表要開始撰寫一個函式，len 代表函式名稱，小括號裡面的 data 稱為參數，會接受呼叫 len() 函式時傳遞的資料，最後還有一個冒號。這一行中，粗體的文字是固定的語法格式，只有函式名稱以及小括號內的內容是可以變動的。接下去的第 02-05 行內縮的程式碼，就是 len() 這個函式的主體。第 02-04 行的內容讀者應該已很熟悉，其作用為走訪 data 並以 ans 累計 data 中的資料個數。第 05 行是一個在函式內才能執行的 return 敘述，表示在此結束函式並將 ans 回傳給函式的呼叫者。程式第 06 行沒有內縮了，表示它不是函式的一部份，此行建立了列表 d。第 07 行呼叫 print() 函式來列印資料，我們把 len(d) 作為要列印的資料，此處便是呼叫了 len() 函式，並將 d 傳遞給 len() 函式的 data 變數。len() 函式執行其主體後，會回傳列表的資料個數 5，然後 print() 便會列印出 5。

E4-1-1.py: 函式的定義與呼叫

```
01  def len(data):
02      ans = 0
03      for e in data:
04          ans += 1
05      return ans
06  d = [1, 2, 10, 4, 88]
07  print(len(d))
```

執行結果：

```
5
```

讀者第一次閱讀此程式可能會感到比較吃力，不用擔心。繼續閱讀本章的過程中，我們會反覆說明與理解函式的各個部份，便會越來越理解了。

和 if、for 句型一樣，函式裡的程式碼要內縮。

撰寫函式的三種型式

前一個段落我們以完整的型式來介紹函式，此處讓我們再以三個範例來示範函式的三種型式。

- 基本型（無傳入傳出）：最簡單的函式型式是空參數列，也沒有回傳資料，如下列程式碼第 01–02 行定義了函式 hello()，其功能是列印固定的文字 Hello Python。當程式員呼叫了函式 hello()，便會列印 Hello Python。

```
01  def hello():
02      print('Hello Python')
03  hello()
```

執行結果：

```
Hello Python
```

- 傳入資料：當函式的參數列（即小括號內）不為空白時，代表呼叫函式的程式員可以傳入資料以客製化函式的行為表現。例如下列程式碼第 01–02 行定義了函式 echo()，以參數 word 接受一個函式外傳入的資料，然後列印 Hello 加上傳入的資料。當程式第 03 行呼叫 echo('NTNU') 時，傳入字串 'NTNU'，使得該次呼叫列印 Hello NTNU；第 04 行呼叫 echo('Python')，傳入字串 'Python'，使得該次呼叫列印 Hello Python。由此我們知道，當函式具有參數列時，可以增加函式的彈性。

```
01 def echo(word):
02     print('Hello ' + word)
03 echo('NTNU')
04 echo('Python')
```

執行結果：

```
Hello NTNU
Hello Python
```

- 傳入傳出：最完整的函式型態就是有傳入資料，也有傳出資料，如下列程式範例。第 01–02 行定義函式 sum3()，參數列有三個參數，可以接收函式外部傳入三個資料。函式會傳回三個資料相加的結果。因此，第 03 行呼叫 sum3() 時傳入 10、9 和 8，此時 sum3() 的 a、b 和 c 會依序指稱 10、9 和 8。所以第 02 行回傳了三個整數相加的結果 27，使得 print() 列印出 27。第 04 行傳入三個字串，此時 sum3() 回傳三個字串相加的結果並由 print() 印出。

```
01 def sum3(a, b, c):
02     return a+b+c
03 print(sum3(10, 9, 8))
04 print(sum3('Hi ', 'Python', '!'))
```

執行結果：

```
27
Hi Python!
```

函式的進入與返回

在程式碼中，如果出現了函式定義，程式的流程會有什麼改變呢？首先，沒有被呼叫的函式就不會有作用。例如下面的程式碼中，第 03-04 行定義了函式 f()；然而，整個五行的程式碼中，並沒有任一處呼叫 f()。因此，整個程式執行完畢，不會執行第 04 行的 print() 敘述。

```
01 print('第一行')
02 print('第二行')
03 def f():
04     print('沒人用我，就不會印出這行')
05 print('第三行')
```

```
執行結果：
第一行
第二行
第三行
```

當函式被呼叫，程式會進入到函式內部去執行其程式碼；執行完函式的程式碼或者執行了 return 敘述後，會結束函式並回到呼叫點繼續執行。以下列程式為例：首先會執行第 01 行。接下來會跳過第 02-04 行的 echo() 函式定義。執行第 05 行時，將呼叫 echo() 函式，並傳入字串 '1'。接著程式流程便會走到第 03 行，然後是第 04 行，結束 echo() 函式後回到第 05 行，繼續往下執行第 06 行。第 07 行再一次呼叫 echo()，這次傳入字串 '2'，程式再一次走到第 03 與 04 行。這次結束後，將會返回到第 07 行，然後往下執行第 08 行。

E4-1-2.py: 函式的執行流程

```
01  print('開始')
02  def echo(word):
03      print('現在在 echo() 函式裡面')
04      print('Hello ' + word)
05  echo('1')
06  print('------')
07  echo('2')
08  print('結束')
```

執行結果：

```
開始
現在在 echo() 函式裡面
Hello 1
------
現在在 echo() 函式裡面
Hello 2
結束
```

return 的用途

本節的最後，讓我們來談一下 return 敘述。首先，return 代表要結束函式並返回呼叫點，所以 return 敘述一定要放在函式定義中。在函式中，只要執行到 return 敘述，就會立刻結束函式。以下面的程式為例，第 01-04 行定義了函式 f()，而第 05 行呼叫了 f()。呼叫後，程式將執行第 02 行，接著是第 03 行。由於第 03 行是 return 敘述，所以函式在這裡結束，也就不會執行到第 04 行。（本範例純粹是為了說明 return 會結束函式，正常情況下我們不會寫出永遠執行不到的程式碼。）

```
01  def f():
02      print('進來 f()')
03      return
04      print('這一行跑不到')
05  f()
```

執行結果：

```
進來 f()
```

除了單純地結束函式，return 敘述也可以回傳資料，以下我們將看三個範例。首先，程式 E4-1-3 示範如何傳回單一個數值。第 01-05 行定義函式 avg()，參數列有一個參數 data 接受一個資料，從第 02 與 03 行來看，可以看出 data 預計是要接受一個數值列表。若列表的長度不為零，return 回傳列表中數值的平均；否則，return 回傳 0。第 06 行呼叫 avg() 時傳入空列表，因此 avg() 將執行第 05 行回傳 0，致使 print() 列印出 0。第 07 行呼叫 avg() 時傳入列表 [1, 3, 5]，因此 avg() 將執行第 03 行，呼叫 sum() 和 len()，並將 [1, 3, 5] 傳入這兩個函式。我們知道 sum() 和 len() 的功能是計算列表中數值總和以及數值個數，因此知道 sum() 將回傳 9 而 len() 將回傳 3 到第 03 行。接著，第 03 行再將 9/3 回傳到第 07 行，致使 print() 列印出 3.0。

E4-1-3.py: 以 return 傳回一個數值。

```
01 def avg(data):
02     if len(data)!=0:
03         return sum(data)/len(data)
04     else:
05         return 0
06 print(avg([]))
07 print(avg([1, 3, 5]))
```

執行結果：

```
0
3.0
```

return 也可以傳回多個值（以元組 tuple 型態回傳）。例如程式 E4-1-4 的第 02 行就回傳 max(data) 和 min(data) 兩個值。第 03 行呼叫 max_min() 時傳入列表 [3, 4, 1, 2]，即 max_min() 的 data 代表 [3, 4, 1, 2]；因此第 02 行的 max() 回傳 4 而 min() 傳回 1，而 return 敘述就會回傳 4 和 1 到第 03 行。第 03 行使用分解賦值的語法，讓 M 指稱 4 而 m 指稱 1。最後，第 04 行列印出 4 和 1。

E4-1-4.py：以 return 傳回多個數值

```
01  def max_min(data):
02      return max(data), min(data)
03  M, m = max_min([3, 4, 1, 2])
04  print(M, m)
```

執行結果：

```
4 1
```

最後，讓我們看看程式 E4-1-5 示範回傳列表。第 01-02 行定義函式 remove_zeros()，第 02 行使用第 3-3 節曾講過的列表推導式從 data 中取出不為 0 的值建立一個新的列表，然後以 return 回傳。第 03 行呼叫 remove_zeros()，傳入列表 [3, 2, 0, 3, 9, 0] 給第 01 行的 data。第 02 行回傳 [3, 2, 3, 9] 給第 03 行的 result。第 04 行再呼叫 print() 函式列印 result。

E4-1-5.py：以 return 傳回列表

```
01  def remove_zeros(data):
02      return [e for e in data if e!=0]
03  result = remove_zeros([3, 2, 0, 3, 9, 0])
04  print(result)
```

執行結果：

```
[3, 2, 3, 9]
```

如果函式 f() 沒有回傳值，a = f() 會令 a 的值為 None。
試試這行：print(print())。

隨堂練習

4.1.1 自行車竊案統計

- 自本書附件取得 TaipeiCityBikeCrime.txt 以及 E4-1-6.py。
- 程式 E4-1-6 中的 GetCrimeData() 函式可以傳入一個檔案名稱，並將檔案從第二行起讀成二維陣列傳回。
  - 請基於程式 E4-1-6，新增 GetYearCount() 函式，並在 main() 函式中呼叫 GetYearCount() 以查詢某一年度（如民國 106 年）的自行車竊案數。
  - 讀者可以撰寫更多的函式來查詢特定月份（如 8 月份）與特定行政區（如松山區）的自行車竊案數。

E4-1-6.py：隨堂練習 4.1.1

```
01  def GetCrimeData(filename):
02      data = []
03      with open(filename, encoding='UTF-8') as f:
04          f.readline()
05          for line in f:
06              data.append(line.split())
07      return data
08  def GetYear(record):
09      return int(record[2][:3])
10  def main():
11      data = GetCrimeData('TaipeiCityBikeCrime.txt')
12      print('第一筆竊案發生在', GetYear(data[0]) , '年')
13      qyear = int(input('請輸入欲查詢年份'))
14      print(qyear, '年總案件數=', GetYearCount(data, qyear))
15
16  main()
```

4-2 函式與變數的可用點

函式的可用點

在我們開始撰寫自己的函式之後，可能會發生無法呼叫函式的錯誤。例如執行下面左方的程式時，會出現 `NameError: name 'f' is not defined`，該錯誤訊息表示 `f` 這個名稱尚未定義。會發生這個錯誤，是因為程式由上往下執行到呼叫點，也就是第 01 行時，尚未看過函式 `f()` 的定義，因此不知道有 `f()` 的存在。把呼叫 `f()` 的程式碼移到第 03 行，如右方程式所示，在由上往下執行到第 03 行這個呼叫點前已經看過 `f()` 的定義，就可以成功呼叫 `f()` 了。

01
02
03
	`f()`
	`def f():`
	`  print('f()')`
		01
		02
		03
			`def f():`
			`  print('f()')`
			`f()`

發生 NameError		輸出 f()	

事實上，Python 的「見過」與否是以執行流程來判定，不是單純以呼叫點和函式定義的位置前後來判定。以下面的程式為例，程式由上而下一直到第 08 行才真正開始執行，此處呼叫函式 `f()`，隨即跳到第 02 行，接著第 03 行呼叫 `g()`。雖然第 03 行的呼叫寫在第 05-06 行的函式 `g()` 定義之前，但由於程式已經一路走到第 08 行才跳回第 02 行，所以已經看過 `g()` 的定義了。因此，第 03 行可以成功呼叫 `g()`。

```
01 def f():
02     print('f()')
03     g()
04
05 def g():
06     print('g()')
07
08 f()
```

Python 程式中常見一種組織函式的方式，是定義一個主函式（可稱為 main()，但也可以定成其它名稱），將程式所有要作的任務寫在主函式中，然後把呼叫主函式的敘述寫在程式碼的最下面一行。如此一來，在呼叫主函式之前必定會看過所有函式定義，也就不必擔心會發生無法呼叫函式的情況了。

程式 E4-2-1 便是示範這種常見的組織方式。儘管 main() 函式定義中呼叫 f()（第 02 行）和呼叫 g()（第 04 行）的敘述都在 f() 和 g() 的定義之前，但因為 main() 函式是在第 13 行呼叫，因此在程式從第 13 行跳到第 02 行時，已經看過第 06–08 行的 f() 定義以及第 10–11 行的 g() 定義，是以第 02 行和第 04 行的呼叫都會成功。

E4-2-1.py: 定義主函式來呼叫其它函式

```
01  def main():
02      f()
03      print('.')
04      g()
05
06  def f():
07      print('f()')
08      g()
09
10  def g():
11      print('g()')
12
13  main()
```

執行結果：

```
f()
g()
.
g()
```

有些語言要求 define higher，而 Python 要求 define earlier。

變數的可用點

程式中不只是呼叫函式可能發生 NameError，使用變數也可能發生。例如下面兩個程式片段，左方程式第 01 行先定義了變數 b，第 02 行再使用 b 定義 a，沒有問題。右方程式第 01 行便想要用變數 b 來定義 a，但此時仍未定義過 b，因此就造成了 NameError。

01 02	b = 3 a = b	01 02	a = b b = 3
沒有錯誤		NameError: name 'b' is not defined.	

變數定義與否，會和程式流程有關。例如下面的程式碼中，第 01 行 if 的條件式為 True，因此執行第 02 行定義了變數 a，沒有執行第 04 行定義變數 b。這便造成第 06 行發生 b 未定義的 NameError。

```
01 if True:
02     a = 3
03 else:
04     b = 4
05 print(a)
06 print(b)
```

定義在函式外的變數稱為全域變數，函式內部可以取用全域變數。例如下面兩個程式片段，函式 f() 中都可以順利取用到全域變數 i 來列印。

01
02
03
04
	i = 9
	def f():
	    print(i)
	f()
		01
		02
		03
		04
			def f():
			    print(i)
			i = 9
			f()

列印 9		列印 9	

讀者可能對上述右方的程式感到疑惑，事實上，這個程式之所以可行的道理和前面我們談到函式是否可用的道理相同；我們看的是程式執行的過程，而不是程式碼的位置。上述右方的程式雖然第 02 行 print(i) 在第 03 行 i = 9 之前，但是程式執行的流程是先執行第 03 行的定義，接著第 04 行呼叫 f()，然後才作到第 02 行的 print(i)。所以，在執行 print(i) 時，變數 i 已經定義過了。

下面再用一個程式說明「流程」而非「位置」決定結果。第 01 行定義變數 i 為 9，緊接著第 04 行將 i 改為 8，然後第 05 行呼叫 f()，程式跳到第 03 行執行 print(i)，因此列印的結果為 8。不要因為第 03 行放在第 01 行之後、第 04 行之前，而誤以為 print(i) 會印出 9 喔。

```
01 i = 9
02 def f():
03     print(i)
04 i = 8
05 f()
```

定義在函式內的變數稱為區域變數，之所以稱為區域，是因為區域變數只能在它的定義式所在的函式區域使用。下列三個程式片段中，第 04 行 print(fi) 取用 fi 都會造成 NameError。

```
01 def f():
02     fi = 9
03
04 print(fi)
```

```
01 def f(fi):
02     print(fi)
03 f(9)
04 print(fi)
```

```
01 def f():
02     fi = 9
03 def g():
04     print(fi)
05 g()
```

與其將「區域變數的存取範圍僅在所屬函式中」視為一種限制，不如將它視為一種保護機制。因為有這樣的保護，當區域變數的數值有誤而需要對程式除錯時，我們可以專注於該函式定義的程式碼就好，不需要把整個程式檔的每一行都檢查一次。

學到這裡，我們讓讀者來看看下面這個程式。這個程式定義了 GetPrice() 函式，其作用是根據物品的數量和單價來計算總價，並且在數量為 10 個以上的時候將總價打九折。請讀者思考看看，這個程式最後會輸出什麼呢？

```
01 def GetPrice():
02     price = num*unit_price
03     if num>=10:
04         price *= 0.9
05
06 price = 0
07 num = 10
08 unit_price = 8
09 GetPrice()
10 print(price)
```

答案是 0，您答對了嗎？關鍵在於這個程式定義了兩個 price 變數。第 02 行定義了 GetPrice() 函式內的區域變數 price，而第 06 行定義了全域變數 price。GetPrice() 函式內操作的都是區域變數 price（所以也沒有改變全域變數 price 所指稱的值），而第 10 行取用到的則是全域變數 price，這個變數自始至終都是 0。

雖然有方法可以在函式內部修改全域變數，我們建議透過參數列將數值傳遞進函式內，再以 return 敘述將運算結果傳回呼叫端。這樣的作法可以在閱讀程式的時候更清楚理解資料的流動。依據這樣的建議，我們將上述程式改寫如程式 E4-2-2。首先，我們在第 01 行 GetPrice() 函式的參數列中增加了兩個參數 n 和 p 來接受外部傳入的數量和單價。函式內我們定義了區域變數 price 來計算總價，在第 05 行將 price 傳回。函式之外，我們在第 09 行呼叫 GetPrice() 時傳入 num 和 unit_price。這樣讓閱讀程式的人一眼就知道 GetPrice() 的運算結果會和 num 與 unit_price 有關。這一行定義了全域變數 price 並以 GetPrice() 的回傳值來設定它。程式第 10 行列印全域變數 price 的值。

E4-2-2.py: 將資料透過參數列傳入，透過 **return** 敘述回傳

```
01 def GetPrice(n, p):
02     price = n*p
03     if n>=10:
04         price *= 0.9
05     return price
06
07 num = 10
08 unit_price = 8
09 price = GetPrice(num, unit_price)
10 print(price)
```

最後讓我們再看一個對比的例子。下面程式碼定義了 Adjust() 函式，其作用是計算以分數 score 開根號乘以 10。左方的程式碼「誤以為」能直接修改全域變數，但其實第 01 行的 score 參數是屬於 Adjust() 函式中的區域變數，第 02 行修改的是這個區域變數，而不是第 04 行定義的全域變數。因此，

第 05 行呼叫完 Adjust() 後，全域變數的值仍為 36。右方的程式碼則是利用函式的回傳值修改了全域變數，達到目的。

01	def Adjust(score):	01	def Adjust(score):
02	score = score**0.5*10	02	return score**0.5*10
03		03	
04	score = 36	04	score = 36
05	Adjust(score)	05	score = Adjust(score)
06	print(score)	06	print(score)
列印 36		列印 60.0	

隨堂練習

4.2.1 任務：自行車竊案統計 (II)

- 前一節的隨堂練習中，我們撰寫了程式查詢特定年份、月份和行政區的竊案總數。
- 由於只能查詢單一行政區的竊案數，如果想要掌握所有行政區的竊案數（例如想知道哪個行政區的竊案數最多），就必須先知道所有行政區的名稱並多次呼叫函式，這相當麻煩。
- 這個練習希望讀者能撰寫出一個函式 GetPlaceStat()，一次就能統計所有行政區的竊案數。

E4-2-3.py：隨堂練習 4.2.1

```
def GetCrimeData(filename):
    data = []
    with open(filename, encoding='UTF-8') as f:
        f.readline()
        for line in f:
            data.append(line.split())
    return data

def GetPlace(record):
    return record[4][3:6]

def GetPlaceStat(data):

def main():
    data = GetCrimeData('TaipeiCityBikeCrime.txt')
    results = GetPlaceStat(data)
    print(results)

main()
```

4-3 預設引數與關鍵字引數

前一節我們看了好幾個將資料傳遞到函式的範例，相信讀者已經逐漸熟悉資料傳遞的過程：在呼叫函式時放置引數（被傳遞者），引數被複製到參數（接受者），然後函式內部再使用參數來運算。本節我們將介紹兩個傳遞資料的進階語法：預設引數值和關鍵字引數。

預設引數值

在呼叫函式時，如果傳入的引數個數不足（比參數個數少），程式執行時便會發生錯誤。例如下面的程式片段，在執行到 05 行的時候，會產生 TypeError 錯誤，錯誤訊息指出缺少了一個需要的引數給參數 b。

```
01  def f(a, b):
02      print(a, b)
03
04  f(1, 2)
05  f(1)
```

```
執行結果：
1 2
TypeError: f() missing 1 required positional argument: 'b'.
```

所謂「預設引數值」，指的是在參數列為參數設定預設值；當呼叫函式者未提供足夠的引數值時，便會以預設值作為引數。以下列程式為例，第 01 行未設定參數 a 的預設值，但設定了參數 b 的預設值。在第 03 行呼叫者傳遞了兩個引數值 1 和 2，因此參數 a 和 b 分別指稱了 1 和 2 這兩個整數值，在畫面上會輸出 1 2。然而，在第 04 行呼叫者只傳遞了一個引數值 1，此時引數和參數會由左往右配對，因此參數 a 指稱整數 1，而參數 b 沒有引數，將使用預設值，也就是 0，畫面上輸出了 1 0。第 05 行呼叫沒有傳遞任何引數，由於 a 並沒有預設值，因此產生了錯誤。

```
01  def f(a, b = 0):
02      print(a, b)
03  f(1, 2)
04  f(1)
05  f()
```

```
執行結果：
1 2
1 0
TypeError: f() missing 1 required positional argument: 'a'
```

在設定預設引數值時，要注意只能由右往左設定，也就是不能發生左邊的參數有預設值但右邊的參數沒有的情況。例如下面的程式碼會造成語法錯誤（Nondefault argument follows default argument）。

```
01  def f(a = 0, b):
02      print(a, b)
```

關鍵字引數

當我們傳遞多個引數時，引數和參數的配對是依位置次序由左往右配對；當引數數量越來越多時，就可能發生引數位置錯誤，但如果程式還是能執行，程式員就不一定會發現（需要仔細查看執行結果才會發現）。以程式 E4-3-1 為例，第 01-02 行定義了函式 `f()`。第 03 行呼叫 `f()`，將 180 傳給 `height`，78 傳給 `weight`，程式也順利印出「身高 180 體重 78」。但若如第 04 行的呼叫，不小心將兩個引數顛倒了，就會導致「身高 78 體重 180」這樣不合理的狀況。

關鍵字引數可以幫助我們避免上述傳遞順序不慎而引起的錯誤。例如第 05 行，我們可以明確指定參數 `weight` 為 60，參數 `height` 為 170。注意，此時的順序不會影響，不管先寫 `weight=60` 還是 `height=170` 都可以。我們也可以一部份的引數是依位置，一部份的引數是以關鍵字方式來傳遞，如第 06 行所示。但關鍵字引數只能出現在位置引數之後，所以第 07 行會產生錯誤「positional argument follows keyword argument」。

E4-3-1.py: 使用關鍵字引數的範例

```
def f(height, weight):
    print('身高', height, '體重', weight)
f(180, 78)
f(78, 180)
f(weight=60, height=170)
f(90, weight=15)
f(height=3, 5)   # 關鍵字引數只能出現在位置引數之後

def g(a, b, c):
    print(a, b, c)
g(1, 2, 3)
g(1, c=3, b=2)
```

執行結果：

```
身高 180 體重 78
身高 78 體重 180
身高 170 體重 60
身高 90 體重 15
1 2 3
1 2 3
```

程式 E4-3-2 示範了運用預設引數值的多種呼叫函式的可能性，讀者們可以先不要看下面的執行結果，自行推論出結果，再對照是否正確。

E4-3-2.py: 使用預設引數值的範例

```
def f(a=1, b=2, c=3):
    print(a, b, c)
f(9, 8)
f(9)
f(a=9)
f(b=8)
f(c=7, a=9)
```

執行結果：

```
9 8 3
9 2 3
9 2 3
1 8 3
9 2 7
```

過去我們在呼叫 Python 內建函式時，其實就大量依賴了預設引數值。例如 print() 函式的參數 sep 的預設值是空白字元 ' '，而參數 end 的預設值是換行字元 '\n'。在程式 E4-3-3 中，第 01 行的呼叫未傳遞引數給 sep 和 end，因此採用預設值，該呼叫會列印 1 2 3 並換到下一行。第 02 行的呼叫傳遞了 ',' 給 sep，但 end 仍是預設值 '\n'，該呼叫會列印 1,2,3 並換到下一行。第 03 行則是傳遞 'X' 給 end，但 sep 是預設值 ' '，因此列印 1 2 3X，沒有換行。第 04 行則同時指定了 sep 和 end，因此會印出 1@2@3.。函式 sorted() 有一個參數 reverse，預設為 False。當我們沒有傳遞引數給 reverse，sorted() 將資料由小到大排序；若我們將 reverse 設為 True，則 sorted() 會將資料由大到小排序。

E4-3-3.py： 內建函式的關鍵字引數呼叫以及預設引數值

```
01 print(1, 2, 3)
02 print(1, 2, 3, sep=',')
03 print(1, 2, 3, end='X')
04 print(1, 2, 3, sep='@', end='.')
05 print()
06 print(sorted([9, 1, 4]))
07 print(sorted([9, 1, 4], reverse=True))
```

執行結果：

```
1 2 3
1,2,3
1 2 3X1@2@3.
[1, 4, 9]
[9, 4, 1]
```

4-4 函式作為引數

透過將程式碼需操作的資料設為參數，然後由呼叫函式的程式員將資料以引數傳入，可以提升程式碼的彈性與重用性。例如我們可以傳入各種不同的資料給 print() 去列印，傳入各個不同的列表給 sorted() 去排序。本節我們將介紹一個有點酷的概念，就是把函式作為引數來傳遞給另一個引數。是不是有點難想像？沒關係，我們先從把函式當成變數來學起。

函式如變數使用

在 Python 中，函式可以像個變數來使用。程式 E4-4-1 的第 01-02 和 03-04 行分別定義了函式 f() 和 g()。第 05 行我們令變數 ff 指稱函式 f，因此第 06 行我們可以將 ff 當成一個函式來呼叫 ff(4)，此時等同於呼叫 f(4)。第 07 行建立了一個列表 funcs，其中有兩個元素 f 和 g。第 08 行的 for 句型會依序從 funcs 中取出 f 和 g，令 e 指稱它們，然後透過 e(7) 來呼叫 f(7) 和 g(7)。

E4-4-1.py: 函式如變數的使用範例

```
01  def f(i):
02      print('f(', i, ')', sep='')
03  def g(j):
04      print('g(', j, ')', sep='')
05  ff = f
06  ff(4)
07  funcs = [f, g]
08  for e in funcs:
09      e(7)
```

執行結果：

```
f(4)
f(7)
g(7)
```

將函式作為引數

讓我們作一個更有意義的範例：老師以兩種調分函式處理成績。程式 E4-4-2 中第 01-09 行定義了三個函式 sqrt10()、add10() 和 process()。函式 sqrt10() 會接受一個數值，然後回傳該數值開根號乘以 10 的結果，函式 add10() 會接受一個數值，然後回傳該數值加 10 的結果。函式 process() 有兩個參數，data 代表一個數值列表而 f 代表一個函式。第 07 行設定索引值 i，而第 08 行則將 data[i] 設為 f(data[i]) 回傳的結果。這裡的重點是，函式 f 是可以由呼叫 process() 的程式員來決定的，不需要固定寫死在程式碼中。因此，當第 11 行傳入 sqrt10，process() 便將所有數值都開根號乘以 10，而當第 12 行傳入 add10，process() 便將所有數值都加 10。未來如果還有新的調分函式，process() 仍然可以重覆使用，豈不妙哉！

E4-4-2.py: 函式作為引數的範例

```
01  def sqrt10(s):
02      return int((s**0.5)*10)
03  def add10(s):
04      return s+10
05
06  def process(data, f):
07      for i in range(len(data)):
08          data[i] = f(data[i])
09      return data
10
11  print(process([36, 49, 64], sqrt10))
12  print(process([36, 49, 64], add10))
```

執行結果：

```
[60, 70, 80]
[46, 59, 74]
```

許多 Python 內建函式都將關鍵的操作開放成為參數，讓呼叫者有更大的彈性與操控權。以下我們就以 max() 和 sorted() 為例來說明能傳入函式有多麼方便。

內建函式 max() 的函式參數 key

內建函式 max() 有一個參數 key，在 max() 內部會被呼叫作為比較資料的基準。以程式 E4-4-3 為例，第 01-04 行定義了兩個函式，last() 會接受一個數值並回傳該數的個位數，而 last2sum() 會接受一個數值並回傳該數最後兩位數字的和。

第 07 行沒有傳入任何函式，此時 max() 會以資料的原值來比較，並回傳 data 列表中的最大值，即為 61。第 08 行傳入 last 給 key，此時 max() 會以每個資料傳入 last() 的結果來比較，因此最後會回傳個位數最大的資料，也就是 25（data 中五個整數的個位數分別為 2、4、2、1 和 5。）。第 09 行傳入 last2sum 給 key，此時 max() 會回傳十位數和個位數相加最大的值，結果為 44（其兩位數字相加為 8，大於另外四個數的十位和個位數的和。）

E4-4-3.py: 傳入自訂函式作為 max() 的 key

```
01  def last(v):
02      return v%10
03  def last2sum(v):
04      return v//10 + v%10
05
06  data = [32, 44, 52, 61, 25]
07  print(max(data))
08  print(max(data, key=last))
09  print(max(data, key=last2sum))
```

執行結果：

```
61
25
44
```

程式 E4-4-4 是一個處理多維度資料的範例。已知學生資料是由姓名、身高和體重三個資料構成的列表，函式 GetHeight() 回傳列表的第二項 v[1]，即為身高；而函式 GetWeight() 回傳列表的第三項 v[2]，即為體重。

第 07 行將 GetHeight 傳給 max() 的 key，因此 max() 會以每個列表的第二項（身高）來比較，並回傳該項最大的列表，也就是回傳身高最高的學

生資料，其結果為 ['李四', 190, 80]（190 大於 180 和 175）。而第 08 行將 GetWeight 傳給 max() 的 key，使得 max() 回傳體重最重的學生資料，結果為 ['王五', 175, 90]（90 大於 75 和 80）。

E4-4-4.py: 指定列表欄位作為 max() 的 key

```
01 def GetHeight(v):
02     return v[1]
03 def GetWeight(v):
04     return v[2]
05
06 data = [['張三', 180, 75], ['李四', 190, 80], ['王五', 175, 90]]
07 print(max(data, key=GetHeight))
08 print(max(data, key=GetWeight))
```

執行結果：

```
['李四', 190, 80]
['王五', 175, 90]
```

內建函式 sorted() 的函式參數 key

函式 sorted() 和 max() 一樣，也有一個參數叫 key，作用也類似，決定 sorted() 排序時會以什麼數值來排序。程式 E4-4-5 的第 03 行沒有傳入函式給 sorted() 的 key，因此以原值來排序，得到由小到大的結果 [3, 65, 74, 82, 91, 100]。第 04 行傳入 last 作為 key，因此 sorted() 會以個位數來排序，使得結果變成 [100, 91, 82, 3, 74, 65]（其個位數是 0, 1, 2, 3, 4, 5）。

E4-4-5.py: 傳入自訂函式作為 sorted() 的 key

```
01 def last(v):
02     return v%10
03 print(sorted([100, 91, 82, 3, 74, 65]))
04 print(sorted([100, 91, 82, 3, 74, 65], key=last))
```

執行結果：

```
[3, 65, 74, 82, 91, 100]
[100, 91, 82, 3, 74, 65]
```

程式 E4-4-6 的目的是要示範 sorted() 在排序多維度資料時，預設（第 04 行的呼叫）會先比較第一項，若第一項相同，才會再比第二項，依此類推。第 05 行傳入 mykey，因此會以第二項來排序（由小到大為 82、88、90、95、100）。

E4-4-6.py: 指定列表欄位以 sorted() 進行排序

```
01  def mykey(v):
02      return v[1]
03  data =[[1, 100], [4, 90], [3, 88], [2, 95], [4,82]]
04  print(sorted(data))
05  print(sorted(data, key=mykey))
```

執行結果：

```
[[1, 100], [2, 95], [3, 88], [4, 82], [4, 90]]
[[4, 82], [3, 88], [4, 90], [2, 95], [1, 100]]
```

程式 E4-4-7 示範了如何以兩個以上的資料來決定排序。函式 mykey() 回傳了列表的第二項和第三項，因此第 07 行將 mykey 傳入 sorted() 的 key 後，sorted() 便會先比列表的第二項，若相同再比第三項。由列印的結果可以看出，排序結果確實先依第二個項目（身高）排序，若遇到身高相同（趙六和王五都是 175），再依第三個項目（體重）排序。

E4-4-7.py: 指定多個列表欄位以 sorted() 進行排序

```
01  def mykey(v):
02      return v[1], v[2]
03  data = [['張三', 180, 75],
04          ['李四', 190, 100],
05          ['王五', 175, 90],
06          ['趙六', 175, 85]]
07  print(sorted(data, key=mykey))
```

執行結果：

```
[['趙六', 175, 85], ['王五', 175, 90], ['張三', 180, 75],
['李四', 190, 100]]
```

Lambda

將函式作為引數傳遞給另一個函式可大大擴展了函式的彈性，但撰寫程式碼還是有一點點不方便，因為我們需要額外再寫一個完整的函式定義（例如程式 E4-4-7 我們要寫出第 01-02 行）。Python 提供了所謂 lambda 的一行無具名函式，讓我們可以更直觀地將函式作為引數。

舉例來說，程式 E4-4-3 可利用 lambda 簡寫成下列程式。對照之下，我們可以了解 `lambda x: x%10` 的意思就如同一個函式藉由參數 `x` 接收傳入的資料，然後回傳 `x%10` 的結果。我們可以把 lambda 想成是這個無具名函式暫時的名字，lambda 和冒號之間放的是參數（名稱可以自訂，不一定要是 `x`），而冒號後面的運算就是回傳的結果。就文法上，要注意這裡不需要像函式用小括號把參數包起來，也不需要寫 `return`。

```
01 data = [32, 44, 52, 61, 25]
02 print(max(data, key=lambda x: x%10))
03 print(max(data, key=lambda x: x//10 + x%10))
```

程式 E4-4-4 和 E4-4-6 可使用 lambda 簡寫如下：

```
01 data = [['張三', 180, 75], ['李四', 190, 80], ['王五', 175, 90]]
02 print(max(data, key=lambda x: x[1]))
03 print(max(data, key=lambda x: x[2]))
```

```
01 data = [[1, 100], [4, 90], [3, 88], [2, 95], [4,82]]
02 print(sorted(data, key=lambda x: x[1]))
```

隨堂練習

4.4.1 任務：自行車竊案統計 (III)

- 前一節的隨堂練習我們希望讀者撰寫出一個函式 GetPlaceStat()，一次就能統計所有行政區的竊案數。
- 如果我們要修改上述函式，改為統計所有月份的竊案數，應該如何作？如何改為統計各年份？是否可透過一個函式來控制被統計的欄位？
- 修改前一節的函式為 GetStat()，除了有一個參數接收資料，新增一個參數接受欄位函式。

```
01  def GetStat(data, keyf):
02      results = []
03      for record in data:
04          key = keyf(record)
05          [略]
06      return results
07
08  def main():
09      data = GetCrimeData('TaipeiCityBikeCrime.txt')
10      print(GetStat(data, GetPlace))
11      print(GetStat(data, GetYear))
```

4.4.2 任務：自行車竊案統計 (IV)

- 前一小題的函式 GetStat() 會回傳各指定欄位的竊案次數，其回傳資料是一個二維列表，其中每個一維列表的第一個項目是欄位，第二個項目是竊案數。
- 參照程式 E4-4-6，運用 sorted() 函式，搭配傳入適當的函式，回答下列問題：
  1. 竊案數前三多的行政區？
  2. 竊案數前三多的月份？
  3. 竊案數前三多的時刻？
- 嘗試改寫程式，以 lambda 來傳入函式。

4-5 程式模組

你都怎麼組織管理你的資料呢？大家都很熟悉將資料分門別類放在不同檔案甚或資料夾（目錄）中吧！當你撰寫的程式碼越來越多，把它們分門別類放在不同檔案中，通常也有助於管理。將程式碼分拆在不同檔案的好處包括了：

1. 將性質相近的函式放在同個檔案，方便找尋和分享。
2. 將程式碼分散在多個檔案，方便修改、測試與抽換。
3. 將程式碼分散在多個檔案，方便團隊工作，團隊中的每個人可以各自專注在自己的程式檔案，不必擔心修改到別人的程式碼。

本章前面四節我們已經學習到如何撰寫函式和呼叫函式，當函式和呼叫的程式碼不在同一個程式檔案時，又該如何處理呢？讓我們透過一個範例來學習。假設我想要計算一個列表的平均值，但我不知道該如何撰寫程式碼。我的好友大明說他手邊有一個這樣的函式可以分享給我。他傳了一個檔案 *lib1.py*，裡面是他寫好的 `mean()` 函式程式碼。此時，我需要在我的程式碼檔案 *main1.py* 中寫一行導入程式的程式碼，如程式 main1 的第 01 行所示。有了這一行之後，我就可以在程式中呼叫 lib1 中任何的函式了。例如 main1 程式第 02 行我以 `lib1.mean()` 來呼叫。

main1.py

```
01  import lib1
02  print(lib1.mean([3, 5, 8]))
```

lib1.py

```
01  def mean(data, prec=2):
02      if data:
03          return round(sum(data)/len(data),prec)
04      else:
05          return 0
```

我們將一個 *XYZ.py* 檔案稱為一個模組 XYZ，import XYZ 的作用就是帶入該模組的程式碼。如果沒有以 import 帶入、忘了寫模組名稱、或是寫錯模組名稱（如以下三個範例），都會導致錯誤喔！

```
01  print(lib1.mean([3, 5, 8]))
NameError: name 'lib1' is not defined.
```

```
01  print(mean([3, 5, 8]))
NameError: name 'mean' is not defined.
```

```
01  import libb
02  print(mean([3, 5, 8]))
ModuleNotFoundError: No module named 'libb'
```

import 敘述有兩種變化，我們先看第一種：加上 from。下列的範例中，程式 main2 第 02 行的意思是從 lib2 模組中帶入函式 f 和 g。有了這一行，之後便可以直接以函式 f() 和 g() 來呼叫，不必再寫出模組 lib2 的名字了。第 05 行的 lib2. 是可寫可不寫，但第 06 行的 lib2. 就必須寫了。

lib2.py

```
01  def f():
02      print('f()')
03  def g():
04      print('g()')
05  def h():
06      print('h()')
```

main2.py

```
01  import lib2
02  from lib2 import f, g
03  f()
04  g()
05  lib2.f()
06  lib2.h()
```

執行結果：main2.py

```
f()
g()
f()
h()
```

下面程式碼的 01 行示範一個偷懶的寫法，import * 的意思就是把所有該模組的函式一口氣帶入。這種寫法雖然方便，但也增加了兩個模組間函式或變數名稱相同而造成的名稱衝突問題；因此，在使用時要特別留心。

lib2.py

```
01 def f():
02     print('f()')
03 def g():
04     print('g()')
```

main2b.py

```
01 from lib import *
02 f()
03 g()
```

執行結果：main2b.py

```
f()
g()
```

接續前一種 import 敘述的變化，我們還可以用 as 來指定模組或函式的代稱。通常我們會用來縮短名稱或解決名稱衝突的問題。以下第一個範例中我們以 lib 代稱原有的模組名稱 this_is_a_lib_with_very_long_name。

```
01 import this_is_a_lib_with_very_long_name as lib
02 lib.f()
03 lib.g()
```

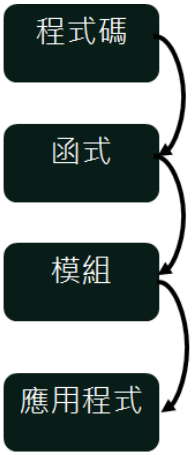
這個用法也適用於函式名稱，如下例。

```
01 from lib import a_func_with_a_very_long_name as f
02 f()
```

如果我的程式碼中已有一個 sort() 函式，我可以帶入另一個模組的 sort() 並以另一個名稱 FastSort() 來指稱它。

```
01 from lib import sort as FastSort
02 FastSort()
```

小結



當你發現有些程式碼經常被使用或是想要隱藏實作細節，便可將完成一特定功能的程式碼片段包裝成一個函式。該函式的參數代表使用者能控制此函式的彈性與權力。當你發現多個函式具有相似性、相關性或相互輔助性時，便可將這些函式歸納在一個模組（.py 檔案）。當你手邊有許多好用的模組，你就可以輕鬆地取用其中需要的函式，組成出你想要的應用程式。

讀者可能覺得「我才剛開始學寫程式，有可能寫出那麼多的程式碼，還把它們分成多個檔案嗎？」也許目前讀者們所寫的程式規模確實還不會大到需要區分模組，但是，讀者有很大的機會去使用它人的模組，例如下一章談到的圖表繪製主題，就需要帶入 matplotlib 套件的 pyplot 模組。因此，本節所介紹的 import 敘述還是很有用的喔！