Chapter 3

成績資料庫

/ 蔣宗哲、江政杰

相對於人腦，電腦的其中一個優勢是能持續且快速地重覆運算。本章將介紹程式語言的第二個重要句型：重覆（迴圈）句型 (for)，同時也介紹重覆句型最常搭配的資料結構 -- 列表 (list)。本章分為七個小節，內容較多，但是以成績資料庫作為應用主軸，循序漸進，初學的讀者若感到些許困難，建議放慢腳步，重覆練習。學習完前三章後，讀者將具備基礎程式能力，已可處理大量、結構化的數值型態資料，進行簡易的統計分析（如求平均、極值、標準差），並能對資料排序與查詢。

3-1 列表 (list) 與迴圈

資料型態與操作

前面兩章的程式裡，我們運用到三種資料型態，分別為整數型態 (int) 、小數或稱浮點數型態 (float) 以及字串 (str) 型態。在程式裡，我們可以使用 type() 函式得知一個資料或變數的型態，如程式 E3-1-1 所示。

E3-1-1.py： type() 函式

```
01 print(type(3))
02 print(type(3.14))
03 print(type('NTNU'))
04 a = 3
05 print(type(a))
06 a = 3.14
07 print(type(a))
```

執行結果：

```
<class 'int'>
<class 'float'>
<class 'str'>
<class 'int'>
<class 'float'>
```

type() 函式可以取得資料的型態。

不同資料型態在電腦內部的儲存方式不同，佔用的記憶體大小也可能不同，但對一般程式員來說，最顯而易見的差異是它們支援的操作不同，即使同一種操作，其結果也不同。例如整數可以作除法，但字串就沒有除法；整數和字串都可以作乘法，但整數乘法 3*2 和字串乘法 '3'*2 就分別得到整數 6 和字串 '33' 這樣兩個不同的結果。

Python 的字串型態除了支援兩個字串相加以及字串和整數相乘，還支援一個稱作 split() 的操作。在程式中，我們會在字串的後面以一個句點再接 split() 來表示要對字串作此操作，如程式 E3-1-2 所示。

E3-1-2.py：字串的 split() 操作

```
01  s = 'NTNU is great!'
02  words = s.split()
03  print(type(words))
04  print(words)
```

執行結果：

```
<class 'list'>
['NTNU', 'is', 'great!']
```

這個程式中，第 01 行以變數 s 代表 'NTNU is great!' 這個字串。第 02 行以變數 words 代表字串 s 執行 split() 的結果。第 03 行列印出 words 的型態，而第 04 行列印出 words 的內容。從執行結果我們可以推論出：

1. 對字串進行 split() 操作會得到一種 list 型態的資料。
2. 對字串進行 split() 操作，會以空白字元分隔出多個字串，並以 list 型態儲存之。

⚠ 不是任何資料型態都有 split() 操作。

從實驗了解字串的 split() 操作

學習程式的過程中，我們經常可以透過實驗來幫助我們理解程式，例如我們可以用多個不同內容的字串來測驗 split() 的操作結果。在程式 E3-1-3 中，我們可以測試：

1. 字串中含有多個空白字元的 split() 結果
2. 字串中沒有任何空白字元的 split() 結果
3. 字串中含有其它看似分隔字元的 split() 結果

E3-1-3.py：字串的 split() 實驗

```
01 print('NTNU      is great!'.split())
02 print('abcde'.split())
03 print('123, 456, 789'.split())
04 print('123,456,789'.split())
```

執行結果：

```
['NTNU', 'is', 'great!']
['abcde']
['123,', '456,', '789']
['123,456,789']
```

從結果我們可以發現，split() 分隔出來的字串中不會含有空白字元。而如果原字串中沒有空白字元，split() 還是可以運作，只是會得到和原字串相同的結果。

取得列表中的資料

從字串中經由 split() 分離出一或多個字串，成為一個字串的列表後，我們該如何取得指定的資料呢？列表型態支援下標運算子 []，以從 0 開始的整數取用列表中的個別資料。在程式 E3-1-4 中，第 03 行直接列印整個字串列表，會以一對中括號將內容包起來。第 04 行是以下標運算子取用列表的第「一」個元素（索引值是 0，或稱為 0 號元素），然後列印出來，因此畫面列印出 2018 這個字串。第 05 行將字串列表中的第一和第二個元素相加（還記得字串相加嗎？），因此會得到 '2018'+'9'，也就是 '20189' 這個字串。第 06 行和第 05 行的差異是第 06 行把字串轉換成整數型態後才相加。

E3-1-4.py：列表的下標運算子

```
01  s = '2018 9 26'
02  num = s.split()
03  print(num)
04  print(num[0])
05  print(num[0]+num[1])
06  print(int(num[0])+int(num[1]))
```

執行結果：

```
['2018', '9', '26']
2018
20189
2027
```

從實驗了解列表的索引值

剛剛說到實驗是寫程式過程中很有趣的一部份，我們再來實驗看看列表的索引值應該如何使用。程式 E3-1-5 裡我們嘗試了幾件事：

1. 第 05 行我們使用了過大的索引值：第 03 行的結果讓我們知道 `num` 列表中有 3 個元素，因此列表中的三個元素應該分別是 `num[0]`、`num[1]` 和 `num[2]`。也就是說，`num[3]` 是不存在的，這也使得程式發出 IndexError 這個錯誤。
2. 第 06 行和第 07 行我們使用字串和小數作為索引值：結果發現這兩種型態不能作為列表的索引值，程式發出 TypeError 這個錯誤。
3. 第 08 和第 09 行我們使用負值作為索引值：結果發現列表的索引值可以是負的，它的結果是從列表的尾端開始取用，`num[-1]` 代表最後一個元素，`num[-3]` 代表倒數第三個元素。第 10 行和第 05 行類似，都是因為索引值超出元素個數而發生了錯誤。

E3-1-5.py：列表的下標運算子實驗

```
s = '2018 9 26'
num = s.split()
print(num)
print(num[0])
#print(num[3])   #IndexError
#print(num['first']) #TypeError
#print(num[1.5]) #TypeError
print(num[-1])
print(num[-3])
#print(num[-4])   #IndexError
```

執行結果：

```
['2018', '9', '26']
2018
26
2018
```

小結

本節中，我們學會了下列知識：

1. 字串有一種操作叫 split()，split() 會將原字串以空白分割成很多個新字串，並把這些新字串以一個列表來儲存。
2. 列表是一個可儲存多個資料的容器。（本節中我們只存放字串，事實上列表也可以存放整數和小數，每個資料可以有不同的型態。）
3. 列表可用下標運算子搭配索引值來取出其中一個資料，索引值必須是整數值，而且數值範圍必須符合列表中的資料個數。

3-2 逐一取用列表元素：for 句型

在學習 3-2 節以前，如果我們的程式要從使用者端讀入三個整數，需要呼叫三次 input()，每次讀入一個整數。現在，我們學會了字串的 split() 以及列表的相關知識，我們可以讓使用者一次輸入三個整數了！（別忘了輸入時必須以空白隔開三個整數。）

```
01  num = input('請輸入三個整數 > ').split()
02  ans = int(num[0])+int(num[1])+int(num[2])
03  print(ans)
```

上面這三行程式可以讓使用者一次輸入三個整數，然後計算三數總和，最後列印出總和。雖然比原本作三次 input() 簡潔，但 .....，如果是四個數、五個數、甚至五十個數，該怎麼辦呢？如果事先不知道有幾個數，又該怎麼辦呢？

for 句型

Python 語言有一個 for … in … 句型，可以依序取出列表中所有元素。我們先看個程式範例。在程式 E3-2-1 中，第 01 行將使用者輸入的字串 split() 成一個字串列表 num。第 03 行的 for 句型會逐一取出 num 中的 num[0], num[1], ... 直到列表結尾，每次取出一個元素，會以變數 e 代表之，然後執行第 04 行的程式碼。注意到 04 行之於 03 行的內縮格式嗎？這和前一章我們學習 if 句型的格式相同，for 句型下內縮的程式片段，會對每個列表元素執行一次。

E3-2-1.py：以 **for** 句型計算總和

```
01  num = input(' 請輸入任意個整數 > ').split()
02  ans = 0
03  for e in num:
04      ans = ans + int(e)
05  print(ans)
```

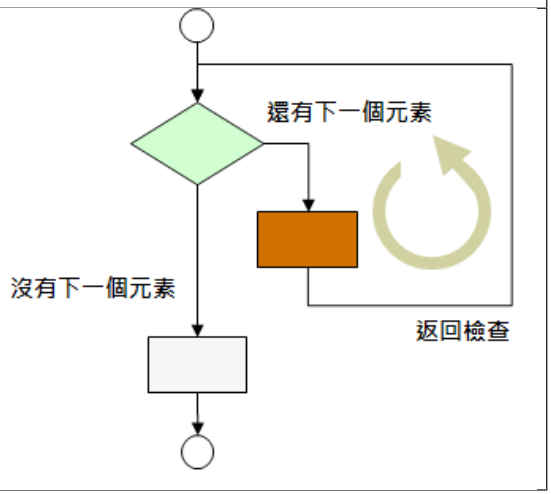
執行結果 1：	執行結果 2：
請輸入任意個整數 > 1 2 9 12	請輸入任意個整數 > 1 3 4 5 13

當使用者輸入字串 '1 2 9'，split() 會產生列表 ['1', '2', '9']，以 num 代表之。接著第 03 行會讓 e 代表 num[0]，進入第 04 行，計算 ans，得到結果整數 1。 依據 for 句型的控制流程，由於 num 列表還沒有結束，所以第 03 行會再讓 e 代表 num[1]，再進入第 04 行，計算 ans 得到結果 3。同理，再讓 e 代表 num[2]，計算 ans 得到 3+9 = 12。最後，第 05 行列印出 12。

下面我們以流程圖對照程式碼來幫助理解。我們可以把 for … in … 的這一行想像成之前學過的 if 句型的條件式，如果列表裡還有元素，就會對這個元素執行 for 句型下面內縮的程式片段。和 if 句型不同的是，if 句型作完內縮片段之後就會繼續往下執行，但 for 句型在作完內縮片段後，還會再去檢查列表內還有沒有元素。也就是這個「還會再去檢查」，使得 for 句型的執行流程形成一個反覆操作的過程，在程式員的術語中，我們稱這裡有一個「迴圈」。

語法說明：**for**

```
for 變數 in 列表：
  動作 1
  動作 2
非重覆動作
```



到目前為止，我們已經學會了程式語言中最重要的兩種句型 – 選擇句型 (if) 和重覆句型 (for)。活用這兩個句型，就會作到非常多的事情。舉例來說，程式 E3-2-2 可以幫我們計算多個數字中有幾個大於等於 60，實務上的應用，可以幫老師計算有幾個學生的考試分數及格。

E3-2-2.py：以 **for** 句型和 **if** 句型計算及格人數

```
num = input(' 請輸入數值 > ').split()
ans = 0
for e in num:
    if int(e)>=60:
        ans += 1
print(ans)
```

執行結果：	執行結果：
請輸入數值 > 59 60 61 9 100 3	請輸入數值 > 90 80 70 60 4

程式 E3-2-3 可以幫我們計算多個數值中的偶數值的和。

E3-2-3.py：以 for 句型和 if 句型計算偶數值的和

```
num = input(' 請輸入數值 > ').split()
ans = 0
for e in num:
    e = int(e)
    if e%2==0:
        ans += e
print(ans)
```

執行結果：	執行結果：
請輸入數值 > 1 2 3 4 8 14	請輸入數值 > 1 3 5 7 9 0

隨堂練習

3.2.1 運用所學，修改以下程式，使其具備以下功能：

1. 使用者在一行輸入任意個整數分數；
2. 輸出分數的個數，總和以及平均；
3. 只記錄介於 0-100 分的分數；
4. 如果沒有任何合法分數，個數、總和與平均皆為 0。

```
num = input(' 請輸入任意個整數 > ').split()
ans = 0
count = 0
for e in num:
    ans = ans + int(e)
    count = count + 1
print(' 個數 =', count, ' ，總和 = ',ans, ' ，平均 =', ans/count)
```

♦ 思考程式流程

目標	程式
在一行輸入任意個整數分數	使用 input() 與 split() 得到資料列表
輸出分數的個數，總和以及平均	使用 for 句型檢視列表中的資料 使用 int() 將字串轉換成整數值 使用兩個變數累計資料個數和總和
只記錄介於 0-100 分的分數	使用 if 句型判斷數值是否符合
輸出分數的個數，總和以及平均	使用 print() 列印結果 使用 if 句型判斷合法分數個數是否大於零

小結

學習完本節後，你已經能撰寫程式處理許多工作，快給自己拍拍手鼓勵一下！

能力	具體作法
向使用者取得大量數據	input()
以易處理的方式儲存數據	字串 split() 與列表 list
以簡潔的程式檢視數據	for 句型
對數據進行檢核與分類	if 句型，關係運算，邏輯運算
（根據分類）進行運算	算術運算
顯示結果	print()

3-3 自建列表與內建函式

在前一小節的隨堂練習中（什麼？你還沒寫完，那還不快去寫！），我們整合運用迴圈控制和選擇控制來計算總和與平均。在實務上，計算平均與極值、搜尋與排序，通常會透過內建函式來完成。截至目前，我們已學過 print() 和 input() 兩個函式，你應該會預期，Python 語言有一些現成的函式，例如下列程式中，我們試圖呼叫 sum() 函式。當程式執行到第 02 行時，程式發出一個 TypeError 錯誤訊息，告訴我們整數和字串無法相加。因此，我們知道 Python 語言中確實存在 sum() 函式，sum() 函式期待的是一個整數列表，而 num 是一個字串列表，因此 sum() 無法處理。

```
01  num = input('請輸入任意個整數 > ').split()
02  s = sum(num)
```

再看另一個程式，我們試圖呼叫 min() 函式。這個程式可以順利執行。我們輸入兩組測試資料，發現第一組測試資料結果符合預期，但第二組結果卻告訴我們 78 比 9 還要小，這是怎麼回事呢？

```
01  num = input('請輸入任意個整數 > ').split()
02  minimum = min(num)
03  print(minimum)
```

執行結果：	執行結果：
請輸入任意個整數 > 9 6 8 6	請輸入任意個整數 > 78 9 78

在前一個程式裡，我們意識到 num 是一個字串列表（所以 sum() 無法順利運算）。我們預期 min() 函式的功用是找出列表中的最小元素，此時因為列表中的元素是字串，所以這時候 min() 找出的是最小的字串。

這裡讓我們暫時跳開列表的處理，談一下字元與字串的大小關係。先看看比較單一字元常用的規則（事實上是基於 ASCII 碼來比較的）：

1. 數字字元由小到大為 '0', '1', '2', …, '9'。
2. 大寫英文字母字元由小到大為 'A', 'B', 'C', …, 'Z'。
3. 小寫英文字母字元由小到大為 'a', 'b', 'c', …, 'z'。
4. 任一數字字元小於任一大寫英文字母字元，任一大寫英文字母字元小於任一小寫英文字母字元。

看完上面的說明，回頭看一下隨堂練習 2.4.1 的文字比較，是不是就更清楚了？上面是單一字元的比較原則，如果是兩個字串在比較時，會依序從各自的第一個字元開始比較，如果兩個字元相異，則以這兩個字元比較大小（如 `'Ant'` 小於 `'Bat'`）；如果兩個字元相同，則比下一個字元（如 `'Candy'` 小於 `'Cat'`）。若字元兩兩相同，直到某一個字串已經結束了，則視另一個字串的狀態決定關係：如果另一個字串還沒結束，則該字串比較大（如 `'Dad'` 小於 `'Daddy'`）；如果另一個字串也結束了，則兩個字串相等。

好了，現在我們了解字串的大小關係，讓我們再看一次前頁的第二個程式：為什麼 `min()` 說 78 小於 9 呢？其實，因為我們傳給 `min()` 函式的是字串列表 `['78', '9']`，所以 `min()` 說的是 `'78'` 小於 `'9'`，這樣一切就合理了。

學會字串的大小關係是很不錯啦，可是這沒有解決我們的問題啊 – 要怎麼樣運用內建函式 `sum()` 與 `min()` 來計算一堆數值的總和與最小值呢？答案很明白，我們需要將字串列表轉換成整數列表。喔！我知道你在想什麼。很遺憾，下面程式的第 02 行會造成 TypeError 錯誤，因為 Python 沒辦法把一個字串列表轉換成「一個」整數值。我們必須將字串列表中的字串一個一個轉成整數才行。

```
01  str = input('請輸入任意個整數 > ').split()
02  num = int(str) #TypeError
```

從字串列表建立整數列表，方法 1：append()

列表型態有一種操作叫 append()，它可以將一個元素加入到列表的尾端。程式 E3-3-1 的第 03 行先建立一個空的列表 num。接著第 04 行運用 for 句型逐一取出 str 字串列表的每一個字串，令 e 代表該字串，對 e 執行第 05 行。第 05 行中，首先將字串 e 轉換成整數，然後再以列表的 append() 操作加入列表。當第 04-05 行的整個 for 句子執行完畢，第 06 行列印出 num 的結果，我們可以看到 num 現在是一個整數列表了（對比第 02 行的輸出）。最令人興奮的是，第 07 行列印了 sum() 計算 num 列表中數值總和，這個總和就是我們希望計算的整數值 80+90+70 的正確結果。第 08 行也藉助 min() 函式求出三個整數值中的最小值了。

E3-3-1.py：列表的 append() 操作

```
01  s = input('請輸入任意個整數 > ').split()
02  print(s)
03  num = []
04  for e in s:
05      num.append(int(e))
06  print(num)
07  print(sum(num))
08  print(min(num))
```

執行結果：

```
請輸入任意個整數 > 80 90 70
['80', '90', '70']
[80, 90, 70]
240
70
```

從字串列表建立整數列表，方法 2：+ 運算子

除了用 append() 將單一個元素加到列表尾端，列表型態也支援兩個列表的 + 法運算，此運算會把兩個列表的元素合併起來，並產生一個新的列表（原本的左列表並不會受到影響）。

程式 E3-3-2 的第 01-02 行建立了列表 a 和 b。第 03 行令 c 代表 a+b 的結果：將 [1, 2, 3] 和 [4] 合併，形成新列表 [1, 2, 3, 4]。注意程式第 04 行列印三個列表 a、b 和 c 的結果：a 仍為 [1, 2, 3]，沒有改變。如果是使用前一種方法，以 a.append(4) 來添加元素 4 的話，a 就會有所改變，變成 [1, 2, 3, 4] 了。

E3-3-2.py：列表的 + 運算子

```
01 a = [1, 2, 3]
02 b = [4]
03 c = a + b
04 print(a, b, c)
```

執行結果：

```
[1, 2, 3] [4] [1, 2, 3, 4]
```

如果以 + 運算子來改寫程式 E3-3-1，第 04-05 行會寫成：

```
04 for e in s:
05     num = num + [int(e)]
```

append() 和 + 運算子有類似的功用，但還是有些差異。我們以下面兩個程式片段來比較它們。左邊的程式中，當我們執行第 03 行時，a.append(b) 是把 b 當成一個元素來加入列表 a，因此我們可以看到第 04 行列印出來的 a 當中有四個元素，最後一個是列表 [4, 5]。相比之外，右邊的程式在執行第 03 行時，是取出兩個列表內的元素（列表 a 有 3 個整數元素，列表 b 有 2 個整數元素）再組合成一個新的列表。所以第 04 行列印的 a 會有五個元素。

append()

```
01 a = [1, 2, 3]
02 b = [4, 5]
03 a.append(b)
04 print(a)
05 print(b)
```

執行結果：

```
[1, 2, 3, [4, 5]]
[4, 5]
```

+ 運算子

```
01 a = [1, 2, 3]
02 b = [4, 5]
03 a = a + b
04 print(a)
05 print(b)
```

執行結果：

```
[1, 2, 3, 4, 5]
[4, 5]
```

從字串列表建立整數列表，方法 3：列表推導 (list comprehension)

列表推導是 Python 語言建立列表的的一種具 Python 風格的語法，乍看之下有點複雜，但是習慣之後就會喜歡上它的簡潔。再者，許多 Python 程式都會用這種方法來建立列表，所以希望讀者還是能儘量了解。

列表推導式可將一個列表中的所有元素逐一取出並作運算，然後以這些運算結果建立另一新列表。程式 E3-3-3 的第 01 行建立列表 a，內含 1、12 和 43 三個整數元素。第 02 行是列表推導式，逐一取出 1、12 和 43，並以這三個數加 1 後的結果構成新的列表 [2, 13, 44]。程式 E3-3-4 也很類似，不過是以列表 a 中的三個整數的平方來構成新的列表 b，所以 b 的內容是 [1^2, 2^2, 3^2]。

語法說明：
[運算結果 for 元素 in 列表]

E3-3-3.py: 列表推導

```
01 a = [1, 12, 43]
02 b = [e+1 for e in a]
03 print(a)
04 print(b)
```

E3-3-4.py: 列表推導

```
01 a = [1, 2, 3]
02 b = [e**2 for e in a]
03 print(a)
04 print(b)
```

執行結果：

```
[1, 12, 43]
[2, 13, 44]
```

執行結果：

```
[1, 2, 3]
[1, 4, 9]
```

程式 E3-3-1 的第 03 行之後以列表推導式可改寫成程式 E3-3-5 所示。

E3-3-5.py: 列表推導

```
03 num = [int(e) for e in s]
04 print(num)
05 print(sum(num))
06 print(min(num))
```

列表推導式中也可以加入 if 條件式來篩選資料，如下面兩個程式碼所示。兩個程式目的相同，都是以列表 a 中的奇數建立新的列表 b。左邊的程式是傳統的寫法，右邊的程式則是列表推導的寫法。

```
a = [1, 2, 4, 7, 9]
b = []
for e in a:
    if e%2==1:
        b.append(e)
```

```
a = [1, 2, 4, 7, 9]
b = [e for e in a if e%2==1]
```

呼叫內建函式

這一節講了這麼多，無非是希望將原本從使用者手上取得的文字輸入轉換成數值的列表，才好運用 Python 內建的函式。表 3-3-1 條列了常用數值處理函式及使用範例。

表 3-3-1：常用的內建數值處理函式

函式名稱	簡述	函式用法
abs()	絕對值	abs(-3) 回傳 3； abs(-3.14) 回傳 3.14。
round()	四捨五入	round(3.4) 回傳 3； round(3.5) 回傳 4； round(3.44, 1) 回傳 3.4； round(3.45, 1) 回傳 3.5。
len()	資料量	len([]) 回傳 0； len([90, 3, 56]) 回傳 3。
min()	最小值	min([9, 3, 2, 1, 5]) 回傳 1； min(['bac', 'ab']) 回傳 'ab'。
max()	最大值	max([9, 3, 2, 1, 5]) 回傳 9； max(['bac', 'ab']) 回傳 'bac'。
sum()	總和	sum([]) 回傳 0； sum([5, 1, 3]) 回傳 9。
sorted()	排序	a = [8, 1, 3] b = sorted(a) print(a, b) 列印 [8, 1, 3],[1, 3, 8]。

隨堂練習

3.3.1 撰寫一個程式，讀入全班成績，過濾非法分數和離群分數，然後輸出最低分、中位數和最高分。

- 過濾條件一：只留下介於 0 到 100 分的分數。
- 過濾條件二：只留下和平均值差距三個標準差之內的分數。
- 注意：平均值和標準差之計算，必須先過濾掉非法分數。
- 平均值公式 (μ)：總和 / 個數
- 標準差公式 (SD)：$\mathrm{SD} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2}\mathrm{SD} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2}$
- 思路：
  (1) 先寫下整個資料處理步驟與流程（大流程寫好後，再慢慢拆解）。
  (2) 再寫下每步驟需要使用到的程式關鍵字。
  (3) 最後再開始寫程式。
- 測試程式
  (1) 測試的時間點依你對程式的掌握度而定。
  (2) 你覺得程式有點長了，就可以測試該段程式碼的正確性。

3-4 列表切片

現在我們已經很熟悉以索引值和下標運算子取用列表中的單一元素了。如果我們想要拿出列表中的某「一段」元素，同樣可以透過下標運算子來完成所謂的「切片」(slice)。

程式 E3-4-1 的第 02 行示範了切片的基本語法：在下標運算子中以一個冒號隔開起始索引值及結束索引值。要特別注意，切片取出來的不會包括結束索引值的那個元素，只會取到前一個元素。因此第 02 行的 `data[1:3]` 只會取到 data[1] 和 data[2] 這兩個元素（不包含 data[3]）。第 03 行令 `c` 代表取出的 `data[0:2]`。第 04-05 行的目的是要表示對 `c` 列表作修改並不會修改到原本的 `data` 列表。

E3-4-1.py: 列表切片基本型式

```
01  data = [1, 2, 4, 8]
02  print(data[1:3])
03  c = data[0:2]
04  c[0] = 30
05  print(data, c)
```

執行結果：

```
[2, 4]
[1, 2, 4, 8] [30, 2]
```

切片的開始索引值和結束索引值都可以省略。若省略開始索引值，代表從列表的第一個元素開始，如程式 E3-4-2 第 02 行從頭開始取用三個元素。若省略結束索引值，代表取到列表的最後一個元素（含最後一個元素）為止，如第 03 行，從第三個元素開始取到最後。如果兩個都省略，則表示複製一份列表內容，如第 05 行。同樣要注意，第 05 行創造出來的 `c` 列表和原本 `data` 列表如今是兩個不同的列表了。最後，兩個索引值都可以是負值，如第 04 行從倒數第三個元素開始取用。

E3-4-2.py：列表切片省略起始或 / 和結束索引值

```
01  data = [1, 2, 4, 8]
02  print(data[:3])
03  print(data[2:])
04  print(data[-3:])
05  c = data[:]
06  c[0] = 9
07  print(data, c)
```

執行結果：

```
[1, 2, 4]
[4, 8]
[2, 4, 8]
[1, 2, 4, 8] [9, 2, 4, 8]
```

切片操作還有一個變化型式，就是指定取用的間隔。程式 E3-4-3a 中示範正值間隔，第 02 行從索引值 0 開始取出，每 2 個取一次（也就是取到 data[0]、data[2]、data[4] 和 data[6]。）第 03 行和第 02 行的差異在於從索引值 1 開始取出，所以取到的是 data[1]、data[3]、data[5] 和 data[7]。第 04 行和第 05 行指定了結束索引值，請記得只取到結束索引值的前一個元素。

程式 E3-4-3b 示範了負值間隔。由於是負值，所以取用的索引值會變小，也就是從尾向頭取用的意思。程式第 02 行是從最後 data[7] 往前逐一取用到 data[0]。這一行是倒轉列表的標準寫法。第 03 行從尾向頭每隔 2 個取一次。第 04 行有指定開始索引值和結束索引值，由索引值 2 無法遞減到索引值 6，所以得到一個空列表。第 05 行從索引值 6 遞減到索引值 2，因為不包含索引值 2 的元素，所以會逐一取出 data[6]、data[5]、data[4] 和 data[3]。第 06 行的從索引值 -1 遞減至 -5，每次減 2，取出 data[-1] 和 data[-3]（不包含 data[-5]）。

E3-4-3a.py: 列表切片正值間隔

```
data = [11, 22, 33, 44, 55, 66, 77, 88]
print(data[::2])
print(data[1::2])
print(data[1:6:2])
print(data[1:7:2])
```

E3-4-3b.py: 列表切片負值間隔

```
data = [11, 22, 33, 44, 55, 66, 77, 88]
print(data[::-1])
print(data[::-2])
print(data[2:6:-1])
print(data[6:2:-1])
print(data[-1:-5:-2])
```

執行結果：

```
[11, 33, 55, 77]
[22, 44, 66, 88]
[22, 44, 66]
[22, 44, 66]
```

執行結果：

```
[88, 77, 66, 55, 44, 33, 22, 11]
[88, 66, 44, 22]
[]
[77, 66, 55, 44]
[88, 66]
```

3-5 range()、索引值走訪與搜尋

雖然我們已經學會用「for ... in 列表」句型逐一取出列表元素，也學會搭配切片取出列表中的一部份元素，不過有些時候我們會想要透過索引值來逐一取用列表元素。有了索引值，我們知道正在取用第幾個元素，也可以藉由目前元素的索引值來計算其它元素的索引值。例如我們想要比較兩個鄰近元素是否相等，若已知目前元素的索引值為 i，則前一個元素的索引值是 i-1，我們可以拿出 [i] 和 [i-1] 來比較。

以索引值走訪：range( )

呼叫 range(n) 可以幫我們產生 0, 1, 2, ..., (n-1) 的整數值，程式 E3-5-1 示範了使用 for 句型加上 range() 來逐一取出列表元素的標準寫法。第 01 行以列表推導式將使用者輸入建立成一個整數列表 data。第 02 行先以 len() 函式求得 data 的元素個數，然後傳入 range() 產生 0, 1, 2, ... 的整數值。假設使用者輸入五個整數值，則第二行 len(data) 會是 5，for 句型會讓 i 依序代表 0, 1, 2, 3, 和 4。這些值會被當作列表索引值，在第 03 行列印出 data 中的每一個元素。

E3-5-1.py: 以 range() 產生索引值並走訪列表

```
01 data = [int(e) for e in input().split()]
02 for i in range(len(data)):
03     print(data[i])
```

執行結果：

```
10 20 30 40 55
10
20
30
40
55
```

range() 函式的小括號內可以傳入一個、兩個或三個整數值，其意義和我們在前一節切片操作中的三個數值相同 – 分別代表起始值、終止值「的下一個值」、以及間隔值。程式 E3-5-2 示範了如何以 range() 函式產生連續有規律變化的整數值。print() 函式中，end=' ' 的意思是要求 print() 列印 i 後不要以預設的換行字元作結束，改以空白字元作結束，如此一來可以把所有的 i 值印在同一行。

E3-5-2.py: range() 的呼叫範例

```
01  # 列印 0, 1, 2, 3, 4
02  for i in range(5):
03      print(i, end=' ')
04  print()
05
06  # 列印 3, 4, 5, 6
07  for i in range(3, 7):
08      print(i, end=' ')
09  print()
10
11  # 列印 10, 12, ..., 20
12  for i in range(10, 21, 2):
13      print(i, end=' ')
14  print()
15
16  # 列印 -10, -9, ..., -1
17  for i in range(-10, 0, 1):
18      print(i, end=' ')
19  print()
```

執行結果：

```
0 1 2 3 4
3 4 5 6
10 12 14 16 18 20
-10 -9 -8 -7 -6 -5 -4 -3 -2 -1
```

列表的搜尋

在列表中搜尋指定的資料是很常見的工作，Python 很貼心地提供了 in 運算子，讓我們可以輕易地判斷資料是否存在於列表中。如果想判斷不存在，加個 not 也可輕鬆搞定。

```
scores = [int(e) for e in input().split()]
if 100 in scores:
    print('有人滿分')
```

```
scores = [int(e) for e in input().split()]
if 100 not in scores:
    print('無人滿分')
```

執行結果 1：	執行結果 2：
90 100 80 70 30 有人滿分	90 60 80 70 30 無人滿分

如果想要計算有幾個符合查詢的值，Python 的列表型態提供了 count() 函式，可以如下面程式來使用。

```
scores = [int(e) for e in input().split()]
print('有', scores.count(100), '人滿分')
```

執行結果：
100 80 90 70 40 50 100 80 有 2 人滿分

Python 的列表型態還提供了 index() 函式，可以用來查詢某個資料在列表中的索引值。程式 E3-5-3 為一個活動報名的應用，使用者先依報名順序輸入報名者的姓名；輸入後便可查詢某人是否有報名，若有報名，則輸出他 / 她的報名順位，若無報名，則輸出查無報名資料。要注意，使用 index() 函式時，如果查詢的資料不在列表中，程式會發出錯誤訊息。因此，我們通常會先以 in 運算子確認資料存在，才呼叫 index() 函式來取得該資料的索引值。

E3-5-3.py:in 運算子與列表 index() 函式的結合應用

```
01 names = input().split()
02 query = input()
03 if query in names:
04     print(query, '的報名序號為', names.index(query)+1)
05 else:
06     print('查無報名資料')
```

執行結果 1：	執行結果 2：
請輸入報名資料 > 張三 李四 王五 趙六 請輸入欲查詢姓名 > 王五 王五 的報名序號為 3	請輸入報名資料 > 張三 李四 請輸入欲查詢姓名 > 王五 查無報名資料

range() 函式還有一個用途是以指定次數重覆執行程式片段。例如下面的程式利用第 01 行的 for 搭配 range(3) 來執行第 02 行的程式碼三次。簡單來說，range() 傳入的數值就是重覆執行的次數。此處 range(3) 所產生的 0, 1, 2 數值並不會被拿來運算，所以慣例上我們會用 _ 作為變數名稱，示意閱讀程式的人此處的 range() 純粹是為了控制迴圈次數而已。

```
01 for _ in range(3):
02     print('很重要所以說三遍')
```

執行結果：
很重要所以說三遍 很重要所以說三遍 很重要所以說三遍

3-6 分解賦值

列表有一個很有趣的分解賦值操作，讓我們可以很方便地用具名變數指稱其中的每個元素。下列程式碼的第 01 行將列表 [10, 20, 30] 的三個資料分別以 x、y 和 z 來指稱。要注意，等號左邊的變數個數和右邊的列表資料個數必須相等，否則程式就會產生 ValueError: not enough values to unpack 或 ValueError: too many values to unpack 的錯誤訊息。（讀者可以自行嘗試刪除第 01 行的 ,z 或者 ,30 來觀察錯誤訊息。）

```
01  x, y, z = [10, 20, 30]
02  print(x, y, z)
```

執行結果：

```
10 20 30
```

分解賦值常用來將輸入的資料賦予個別名稱。下面的兩段程式碼執行後有相同的結果，但第二段程式碼中我們明確地指定了三個輸入資料的變數名稱，可以提升程式的可讀性。

```
01  data = [
02  int(e) for e in input('輸入身高、體重和年齡>').split()]
03  print('你的年齡是', data[2])
```

```
01  height, width, age = [
02  int(e) for e in input('輸入身高、體重和年齡>').split()]
03  print('你的年齡是', age)
```

執行結果：

```
輸入身高、體重和年齡>180 80 20
你的年齡是 20
```

在 Python 程式中，若要交換兩個變數，標準寫法如程式 E3-6-1 第 04 行所示。事實上，這一行便用到分解賦值的操作，只不過這時候等號右邊的 y, x 會被視為一個元組 (tuple)。元組和列表相似，不過元組是不可修改的。此處我們就不繼續討論元組型態了。

E3-6-1.py: 交換兩個變數

```
x = 10
y = 20
print(x, y)
x, y = y, x
print(x, y)
```

執行結果：

```
10 20
20 10
```

程式 E3-6-2 結合運用了 Python 內建的 `min()` 和 `max()` 函式、列表的 `index()` 函式，以及本節所介紹的分解賦值操作，其功能為交換列表中最小值和最大值。讀者應可看出執行結果的兩行輸出中，最小值 10 和最大值 60 已互換位置。

E3-6-2.py: 交換列表中的最小與最大值

```
data = [40, 20, 50, 10, 60, 30]
print(data)
i = data.index(min(data))
j = data.index(max(data))
data[i], data[j] = data[j], data[i]
print(data)
```

執行結果：

```
[40, 20, 50, 10, 60, 30]
[40, 20, 50, 60, 10, 30]
```

3-7 多維度列表

假如今天我們的程式要儲存一個 40 人班級的期中考分數，相信讀者已經能很直覺地以列表來儲存，並能從列表中以索引值取出任一學生的分數。（當然，我們一定都記得 1 號學生的分數是存放在索引值 0，而非索引值 1，的位置。）

那麼，如果這學期不只一次考試呢？我們該如何存放，以便未來仍然可以很方便地取出第 i 號學生的第 j 次考試分數呢？由於我們有兩個索引值（學生號碼及考試次別），我們需要讓列表也同樣擁有兩個索引值 – 這就要用到本節所談的多維度列表了。

程式 E3-7-1 第 01 行建立了二維列表 scores。事實上，建立二維列表的方式與建立一維列表無異，同樣以是一對中括號把資料包起來，只不過一維列表中的資料是單一數值，而此處二維列表中的資料也是列表，讀者可以看到最外圍的中括號裡面又有三對中括號。第 02 行列印出 3 和 2，代表 scores 這個二維列表中有三個資料，而 scores 的第一個資料 scores[0] 中含有 2 個資料（即 100 和 80）。第 03 行展示了如何以兩個索引值取用二維列表中的資料，我們可以看到 scores[0][0] 表示先取出 scores 中的第一個一維列表，再取出該一維列表的第一個資料，也就是 100；依此類推，scores[1][1] 即為第二個一維列表的第二個資料 90。第 04 行展示我們可以將二維列表中取出的一維列表傳入之前介紹過的函式來處理，此處 sum() 回傳 60+50 的結果。

E3-7-1.py：二維列表的建立與取用

```
01  scores = [[100, 80], [70, 90], [60, 50]]
02  print(len(scores), len(scores[0]))
03  print(scores[0][0], scores[1][1])
04  print(sum(scores[2]))
```

執行結果：

```
3 2
100 90
110
```

程式 E3-7-2 示範了從使用者輸入來建立二維列表的方法。首先，第 01 行建立一個空的列表。第 02 行表示第 03-04 行要重覆 3 次。第 03 行將使用者輸入以 input() 函式讀入後拆解為字串列表，然後再以列表推導式建立成整數列表，最後指定給 stu 變數。第 04 行則將 stu 這個一維整數列表附加到 scores 尾部。由於 stu 本身是個一維列表，scores 就會變成二維列表了。注意這裡使用者每次輸入的分數個數即使不同，scores 仍然能夠儲存。第 05 行列印出 scores 的內容，讀者可觀察一下 Python 以何種格式列印二維列表。程式第 06 行判斷 scores[2] 是否含有超過 1 個分數，若有，第 07 行輸出 scores[2][1]；若無，則第 09 行為 scores[2] 增加一個 0 分的資料。第 10 行再次列印 scores 的內容。

E3-7-2.py： 二維列表的建立與取用

```
01  scores = []
02  for _ in range(3):
03      stu = [int(e) for e in input().split()]
04      scores.append(stu)
05  print(scores)
06  if len(scores[2])>1:
07      print(scores[2][1])
08  else:
09      scores[2].append(0)
10  print(scores)
```

執行結果 1：	執行結果 2：
100 90 70 60 80 [[100, 90], [70], [60, 80]] 80 [[100, 90], [70], [60, 80]]	100 90 70 60 [[100, 90], [70], [60]] [[100, 90], [70], [60, 0]]

我們經常以 for 迴圈搭配 range() 產生索引值來走訪一維列表的所有資料，由於二維列表有兩個索引值，可想而言，我們需要兩個 for 迴圈來控制這兩個索引值 – 而且，這兩個迴圈是巢狀關係。

讓我們先理解巢狀 for 迴圈的執行流程與結果。下列左邊的程式碼有兩個獨立的 for 迴圈。第一個 for 迴圈會讓程式第 02 行執行三次，列印 0、1 和 2；第二個 for 迴圈執行後列印 * 0 和 * 1，然後程式結束。右邊的程式碼則是將第二個 for 迴圈置入第一個 for 迴圈中，形成巢狀迴圈。此時，第一個（外圍）for 迴圈會讓程式第 02-04 行都執行三次。藉由這樣的巢狀迴圈，我們能創造出 (0, 0)、(0, 1)、(1, 0)、(1, 1)、(2, 0)、(2, 1) 這樣六個組合。讀者們發現了嗎？這樣的組合，恰好可以用來走訪二維列表！

兩個獨立的 **for** 迴圈

```
01 for i in range(3):
02     print(i)
03 for j in range(2):
04     print('*',j)
```

巢狀的 **for** 迴圈

```
01 for i in range(3):
02     print(i)
03     for j in range(2):
04         print('*',j)
```

執行結果：

```
0
1
2
* 0
* 1
```

執行結果：

```
0
* 0
* 1
1
* 0
* 1
2
* 0
* 1
```

程式 E3-7-3 綜合了前述幾個程式範例所講授的概念與語法。首先，第 01-04 行取得使用者輸入的三個學生的考試分數。第 06 行和第 08 行形成巢狀迴圈，第 06 行的 i 代表學生的索引值（第幾個學生），所以 range() 傳入的是二維列表 scores 的長度；而第 08 行的 j 代表分數的索引值（第幾個分數），所以 range() 傳入 scores[i] 的長度，即學生 i 的分數的個數。

第 07-10 行的目的是計算學生 i 的分數 scores[i] 中有幾個是 60 以上（通過考試），因此以第 08 行的迴圈來產生分數的索引值 j，然後依序取出 scores[i][j] 並判斷是否及格，若及格，則 pa 的值加 1。最後，第 11-12 行顯示每位學生的考試通過次數。

E3-7-3.py： 以雙層迴圈與索引值走訪二維列表

```
scores = []
for _ in range(3):
    stu = [int(e) for e in input().split()]
    scores.append(stu)

for i in range(len(scores)):
    pa = 0
    for j in range(len(scores[i])):
        if scores[i][j]>=60:
            pa += 1
    print('學生', i+1, '的', len(scores[i]),
    '次考試通過', pa, '次')
```

執行結果：

```
100 80 90
60 70 50
50 40 60 30
學生 1 的 3 次考試通過 3 次
學生 2 的 3 次考試通過 2 次
學生 3 的 4 次考試通過 1 次
```

隨堂練習

3.7.1. 撰寫一個程式，讀入五位學生的成績（成績數量不定），每位學生的成績在一行，輸出五位學生的四次成績（由高到低）和平均分數，以二維表格呈現。

- 若某位學生的成績數量少於四次，不足的成績補零分。
- 若某位學生的成績數量多於四次，只採計最高分的四次。

執行結果：範例一

```
100 90 80 70
60 50 40 30
30 30 30 30
20 30 40 50
91 92 93 94
100 90 80 70 85.0
60 50 40 30 45.0
30 30 30 30 30.0
50 40 30 20 35.0
94 93 92 91 92.5
```

輸入

輸出

執行結果：範例二

```
100 90 80 70
80 80 80 80
100 100
100 0 10 20 90 90 100

100 90 80 70 85.0
80 80 80 80 80.0
100 100 0 0 50.0
100 100 90 90 95.0
0 0 0 0 0.0
```

輸入

輸出

（參考答案）

```
NumStudens = 5
NumCountedScores = 4

data = []
for _ in range(NumStudens):
    data.append([int(e) for e in input().split()])

for stu in data: # 對每位學生成績 stu
    # 將 stu 由小到大排序，反轉，取出前 NCS 個
    stu = sorted(stu)[::-1][:NumCountedScores]
    # 不足 NCS 個分數補 0 分
    for _ in range(NumCountedScores-len(stu)):
        stu.append(0)
    # 列印分數與平均
    for sc in stu:
        print(sc, end=' ')
    print(sum(stu)/len(stu))
print()
```

排序的部份也可以利用 sorted() 的參數 reverse 來由大到小排序，sorted(stu, reverse=True)。
另外，如果不要保留原始資料的順序，也可以直接排序，stu.sort(reverse=True)。