Chapter 2

電腦的決策

/ 蔣宗哲、江政杰

有一個古早的廣告詞說「電腦也會撿土豆」，意思是電腦能根據不同情況作出不同的回應動作。本章將介紹程式語言的兩大句型之一：選擇句型 (`if`)，分為四小節，內容包括：基本二分法句型、多分法句型、巢狀句型以及邏輯運算子。學習完本章後，讀者將具備能力把日常生活中常見的流程圖轉譯成 Python 程式，並可進行資料的比較和分類。

2-1 如果句型 if...

平均的危機

前一章我們學會了撰寫 Python 程式進行簡單的數值運算，數值資料分析最常見的是計算資料的個數、極值（最大 / 小值）、平均值等等。學習程式撰寫最好的方法就是多練習寫程式，所以在本章的開始，我們先撰寫一個求平均的程式當練習，順便複習一下基本語法，以下是一個參考程式：

E2-1-1.py:

```
01  print('計算平均的程式')
02  total = int(input('請輸入總和 > '))
03  count = int(input('請輸入個數 > '))
04  print('答案是', total/count)
```

對於一個程式員，撰寫出完整的程式只是第一步，更重要的下一步是測試並確認程式運作無誤。最簡單的測試方式是輸入一些資料，觀察程式的反應是否如預期。

執行結果：測試案例 1

```
計算平均的程式
請輸入總和 > 100
請輸入個數 > 5
答案是 20.0
```

執行結果：測試案例 2

```
計算平均的程式
請輸入總和 > 42
請輸入個數 > -8
答案是 -5.25
```

執行結果：測試案例 3

```
計算平均的程式
請輸入總和 > 100
請輸入個數 > 0
Traceback (most recent call last):
     File "divide by zero.py", line 4, in <module>
       print('答案是', total/count)
   ZeroDivisionError: division by zero
```

上述三個測試案例中，案例 1 是一般人通常會測試的「正常資料」，案例 2 和案例 3 則出現異常的輸入，包含負的個數，甚至是 0 作為個數。當程式試圖計算除以 0 的結果時，程式便會出現錯誤訊息 ZeroDivisonError，代表這是一個不應該發生的運算。既然這是一個不該發生的運算，我們是否能修改程式

來避免呢？我們希望的程式流程是這樣的：

1. 使用者輸入總和與個數。
2. 判斷個數：

   2.1 如果個數為正時，計算平均。

   2.2 如果個數不為正時，提示使用者輸入錯誤。

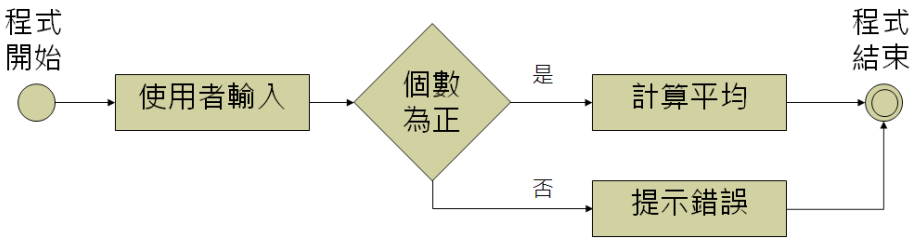


圖 2-1 程式流程圖 (平均的危機)

美好的如果

大多數的程式語言都有個「如果」的句型。

- 以中文為例：

  如果個數為正，計算並列印平均；

  否則，提示使用者輸入錯誤。

- 以英文為例：

  If the count is positive, calculate and print the average;

  otherwise, show an error message.

而在 Python 語言中，也有「如果」的句型。

在 `if` 句型中，程式會依據條件判斷式的真偽結果，選擇不同的程式片段來執行，因此 `if` 句型也常被稱為是一種「選擇控制」的句型。

語法說明：**if ... else ...**
if 條件判斷式：（別忘記冒號歐！） 　　條件成立會作的動作（可以不只一行） else: 　　條件不成立會作的動作（可以不只一行）

以下面的程式片段來舉例，如果 count>0 這個條件式是真的（或說條件式成立），也就是 count 值為正，程式便會執行第 02 行；如果條件式是假的（或說不成立），程式便執行第 04 行。

```
01 if count>0:
02     print('答案是', total/count)
03 else:
04     print('無法計算 - 個數需為正值')
```

將上述的程式片段整合至前面求取平均的程式碼中，我們便可以避開計算個數不為正時的平均值，如下程式範例：

```
01 print('計算平均的程式')
02 total = int(input('請輸入總和 > '))
03 count = int(input('請輸入個數 > '))
04 if count>0:
05     print('答案是', total/count)
06 else:
07     print('無法計算 - 個數需為正值')
```

要注意，Python 對於「如果」句型有嚴格的排版格式規定，若不符合規定，會出現 expected an indented block 的語法錯誤。簡單來說，被 if/else 控制的程式片段需要往內縮，並且要內縮一樣的空白數，常見的慣例是內縮四個空白。現今的程式編輯器都有自動內縮的功能，一般情況下，程式員不需要刻意去調整縮排。

格式錯誤範例 1

```
01 if a>0:
02 print(a, '是正值')
03 else:
04 print(a, '不是正值')
```

格式錯誤範例 2

```
01 if a>0:
02 print(a, '是正值')
03   print('這行格式錯了')
04 else:
05 print(a, '不是正值')
```

若條件不成立時不需要執行特定的運算，可以省略掉句型中的 else 部份。例如下面的程式執行到第 05 行時會判斷 school 變數的內容，如果是「臺師大」便會執行第 06 行；若 school 內容不是「臺師大」，則程式會跳過第 06 行，往下執行第 07 行。

E2-1-2.py:

```
01 print('薩莉亞餐廳的程式')
02 school = input('請輸入就讀學校 > ')
03 count = int(input('請輸入人數 > '))
04 price = count*100
05 if school == '臺師大':
06     price = price*0.9 #師大人打九折
07 print('總價', int(price), '元')
```

執行結果 1：	執行結果 2：
薩莉亞餐廳的程式 請輸入就讀學校 > 臺師大 請輸入人數 > 10 總價 900 元	薩莉亞餐廳的程式 請輸入就讀學校 > 某大 請輸入人數 > 10 總價 1000 元

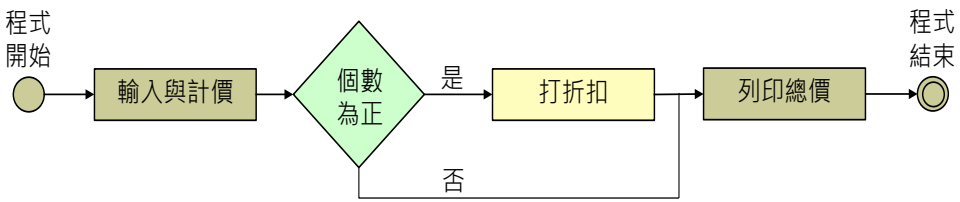
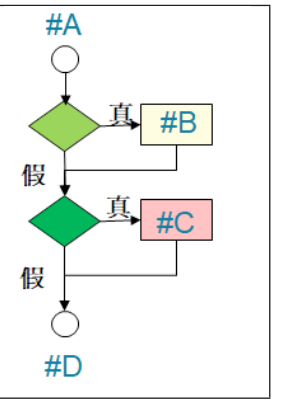


圖 2-2 程式流程圖 (省略 else)

如果句型的寓意

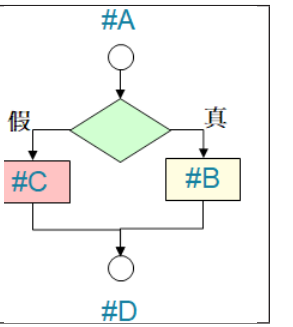
在撰寫 if 句型時，有些人會寫出類似下面的程式碼。這個程式使用了兩個獨立的 if 句子，第一個會在 count 值大於 0 時執行動作 B，第二個會在 count 值小於等於 0 時執行動作 C。雖然這個程式片段在功能面上和本頁下方使用一個 if 句子的程式片段相同，但程式可讀性卻不如下方的程式來得好。

```
01  # action A
02  if count > 0:
03      # action B
04  if count <= 0:
05      # action C
06  # action D
```



一般來說，兩個獨立 if 句子的流程圖如上面右圖，它代表著程式有四種可能的執行流程：AD、ABD、ACD 和 ABCD。本例的程式中因為兩個 if 句子的條件式有互補的關係，使得程式僅剩下 ABD 和 ACD 兩種執行流程。然而，閱讀上方程式碼的條件式並理解它們的關係，會增加閱讀程式的負擔。

```
01  # action A
02  if count > 0:
03      # action B
04  else:
05      # action C
06  # action D
```



相對而言，本頁下方的程式碼，在程式員一看到 if … else … 之後，便能立刻掌握這是一個「二分法」的選擇控制，要嘛執行動作 B，要嘛執行動作 C，沒有第三種流程了。

布林值

前面說到 if 句型會依「條件式」的真假來選擇程式片段，在 Python 程式中，所謂的真與假，可用布林 (boolean) 型態（bool） 的特定值來表示：True 代表真，False 代表假。以下列程式為例，第一個 if 句子的條件式明白寫了 True，因此程式會執行第 02 行，列印出 1 True；第二個 if 句子的條件式明白寫了 False，因此程式會執行第 09 行，列印出 2 False。

E2-1-3.py:

```
01  if True:
02      print('1 True')
03  else:
04      print('1 False')
05
06  if False:
07      print('2 True')
08  else:
09      print('2 False')
```

執行結果：

```
1 True
2 False
```

關係運算子

上面的程式只是為了舉例說明 if 句型是依條件式運算結果為 True 或 False 來選擇程式流程，我們不會真的寫出 E2-1-3 這樣的程式。（那個程式中我們已經確知程式的執行流程，可以只留下第 02 和 09 行，並刪除掉其它程式碼。）一般 if 句子的條件式會是一或多個關係運算來構成。表 2-1 列出 Python 的關係運算子 (operator) 以及運算範例。

表 2-1：Python 的關係運算子

意義	關係運算子	結果為真 True	結果為假 False
相等	==	3==3	3==4
不相等	!=	3!=4	4!=4
大於	>	4>3	3>4
大於等於	>=	4>=4	3>=4
小於	<	3<4	4<3
小於等於	<=	3<=4	5<=4

在目前我們所學過的運算子當中，關係運算子的運算優先度最低，也就是它們會最後計算：「次方」高於「乘 / 除 / 餘」高於「加 / 減」高於「關係運算」。

Python 程式碼	運算次序	運算結果
3*3==9	(3*3)==9	True
2+3>1+4	(2+3)>(1+4)	False

隨堂練習

2.1.1　請問以下程式碼的輸出為何？

```
x, y = 11, 20
print('A')
if 100 > 3:
    print('B')
if 100 < 200:
    print('C')
if x/2 > 5:
    print('D')
print('E')
if x+y < 40:
    print('F')
if x*2 == y:
    print('G')
if x <= y:
    print('H')
if x+3*y >= x*3+y:
    print('I')
if x != y:
    print('J')
```

解答：

執行結果：

```
A
B
C
D
E
F
H
I
J
```

2.1.2　修改「求平均」程式以達下述功能：給予使用者一次輸入錯誤（即輸入不為正的個數）的機會，當使用者連續錯誤兩次時，程式便結束。

2-2 多分法

如果不只兩種可能？

在前一節「求平均」的程式中，我們依據個數「為正值」與「不為正值」將程式流程分成兩個路線。如果我們將依據一個數值「為正」、「為 0」和「為負」分成三個執行路線，該怎麼作呢？

Python 的 if 句型除了 if 和 else 區塊，還有一個 elif（是的，就是 else 加 if）區塊。我們先看一個例子來學習它的用法：

E2-2-1.py:

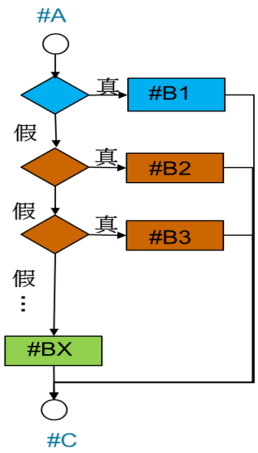
```
01  v = int(input('請輸入一個整數值 > '))
02  if v>0:
03      print('正值')
04  elif v==0:
05      print('零')
06  else:
07      print('負值')
08  print('程式結束')
```

上述程式在第 02 行會判斷 v 的值是否大於 0：如果是，會執行第 03 行，然後就執行第 08 行了。是的，請注意，當程式執行其中一個被控制的區塊，就不可能再進入其它區塊了，也就是在 if … elif 的語法中，我們必須謹慎安排條件判斷的執行順序，放在前面的條件一旦符合為真，在同一組 if … elif 後面的條件就不會被檢驗。如果 v 的值沒有大於 0，程式會檢查第 04 行的條件：如果 v 的值為 0，程式會依序執行第 05 和 08 行。如果 v 的值為負，則程式會在第 02 和 04 行的條件式都為假的情況下，跳到第 06 行，然後執行第 07、08 行。

執行結果 1：	執行結果 2：	執行結果 3：
請輸入一個整數值 > 3 正值 程式結束	請輸入一個整數值 > -4 負值 程式結束	請輸入一個整數值 > 0 零 程式結束

語法說明：**if ... elif ... else ...**

```
#A
if condition1:
    #B1
elif condition2:
    #B2
elif condition3:
    #B3
...
else:
    #BX
#C
```



上表描述了 if 的完整句型以及執行流程。總結來說，一個 if 的句子會以 if 作為開頭，接著有 0 個以上的 elif 區塊，最後會有 0 或 1 個 else 區塊。當句子只有 if 區塊的時候，控制的是某段程式碼的執行與否（就像一個開關，開就執行，關就不執行）；當句子有 if 和 else 區塊時，表現的是一種二分法；而再加上 elif 區塊後，便可以表現多分法。

接下來呈現兩個常見的多分法型式，E2-2-2 是單值多分法，以變數是否等於特定值來控制程式流程，用的是相等（==）關係運算子，而 E2-2-3 是區間多分法，以變數值是否落在一段區間內來控制程式碼，用的是大於等於（>=）關係運算子。

E2-2-2.py:

```
grade = int(input('你的年級>'))
if grade == 1:
    print('一年級')
elif grade == 2:
    print('二年級')
elif grade == 3:
    print('三年級')
elif grade == 4:
    print('四年級')
else:
    print('大大級')
```

⚠ 雖然 if 句型不一定要有 else 的部份，但建議保留此部份作為檢查時的最後一道關卡，以便發現預期之外的變數值（如輸入 5）。

E2-2-3.py:

```
01 price = 10
02 number = int(input())
03 if number>=12:
04     price = price*number*0.9   #12人以上
05 elif number>=6:
06     price = price*number*0.95   #6~11人
07 elif number>=2:
08     price = price*number*0.99   #2~5人
09 else:
10     price = price*number   #1人
11 print(number, price)
```

⚠ 注意「由上而下」的判斷順序，當程式執行第 05 行的條件式判斷時，代表 number 沒有大於等於 12，即 number 小於 12。

隨堂練習

2.2.1　多分法練習：請撰寫一個程式，使用者輸入一個分數，請依以下區間輸出等第。

分數區間	等第
90~100	A+
85~89	A
80~84	A-
77~79	B+
73~76	B
70~72	B-

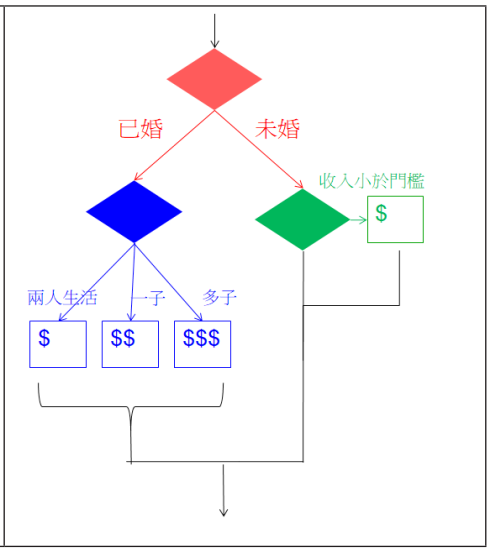
2.2.2　多分法練習：請撰寫一個程式，使用者輸入中文星期幾，程式輸出對應的英文字。

中文輸入	英文輸出
星期日	Sunday
星期一	Monday
星期二	Tuesday
星期三	Wednesday
星期四	Thursday
星期五	Friday
星期六	Saturday

2-3 巢狀控制

前一節的尾聲我們說到 if 句型可以表現出「開 / 關」、「二分法」和「多分法」這些「單層」的控制流程。事實上，在 if/elif/else 的程式區塊裡面，我們仍然可以繼續使用 if 句型，這就構成了所謂「巢狀 (nested)」控制。我們以下面的調薪程式片段來講解：

```
01  if married:
02      if num_kids == 0:
03          bonus = 1000
04      elif num_kids == 1:
05          bonus = 2000
06      else:
07          bonus = 4000
08  else:
09      if salary <= 50000:
10          bonus = 500
```



在這段程式碼中，首先第 01 行的 if 和第 08 行的 else 構成一個以婚姻狀態 (married 變數) 來控制的二分法。而這個二分法的 if 區塊（已婚狀態）中，我們再使用 if … elif … else 句型表現一個以小孩人數 (num_kids) 來控制的三分法，當小孩人數為 0、1 和 2 人以上時，分別加薪 1000、2000 和 4000 元。外層二分法的 else 區塊（未婚狀態）中，則以一個簡單的 if 句子控制，當薪資不高於 50000 元時，加薪 500 元。讀者可以對照右邊的流程圖以清楚理解左邊程式碼的執行流程。

隨堂練習

2.3.1　下面的程式碼可以判斷三個變數 A, B, C 中的最小值，請畫出其流程圖。

```
if A < B:
    if A < C:
        m = A
    else:
        m = C
else:
    if B < C:
        m = B
    else:
        m = C
print(' 最小值為 ',m)
```

2.3.2　這個程式共有幾種可能的執行流程（以 A/B/C/D 區塊來看）？

```
print(' 計算平均的程式 ')
total = int(input(' 請輸入總和 > '))
count = int(input(' 請輸入個數 > '))      #A
if count>0:
    print(' 答案是 ', total/count)        #B
else:
    count = int(input(' 個數須為正值，請輸入個數 > '))
    if count>0:
       print(' 答案是 ', total/count)
    else:
       print(' 無法計算 - 個數需為正值 ')   #D
```

解答：

編號	執行流程	測試案例
1	AC	依序輸入 100 3
2	ABC	依序輸入 100 0 3
3	ABD	依序輸入 100 0 0

2-4 邏輯運算子

在我們繼續學習更多程式知識前，我們先看看讀者是否已完全理解巢狀控制的概念，請看下列程式片段：說出三組 (month, day) 的數值可以分別讓程式執行第 03、05 和 07 行。

```
01  if month == 8:
02      if day == 15:
03          print('中秋節快樂!')
04      else:
05          print('平常日')
06  else:
07      print('平常日')
```

執行第 03 行	執行第 05 行	執行第 07 行
month: 8 day: 15	month:8 day: 13	month: 7 day: 1

建議讀者在撰寫或閱讀程式碼時，都能常作這樣的練習，有助於增強對於程式流程的掌握，也可提高測試與除錯的效率。好，現在我們進入本節主題：上述程式碼能否以更精簡、易懂的方式來撰寫呢？答案當然是可以。Python 語言中有三個邏輯運算子：而且 (and)、或者 (or) 及非 (not)，前兩個是二元運算子，運算時需要左、右兩個布林值作為運算元；而第三個是一元運算子，運算時僅需右邊一個布林值作為運算元。

邏輯運算子 and

因為 and 運算是二元運算，左右兩邊的運算元都是布林值（還記得布林值只有 False 和 True 兩種嗎？），因此 and 運算共有四種運算組合。以表 2-4-1 的第一列為例，當值 a 為 False 而值 b 為 False 時，a and b 這個邏輯運算的結果是 False。從這四種運算組合來看，我們可以得到一個簡單的規則，只有當兩個運算元皆為真 (True) 時，and 運算的結果才會為真，其餘的運算結果都是假 – 這也符合我們對「而且」這個詞的理解。

表 2-4-1: and 運算子的運算結果

a	b	a and b
False	False	False
False	True	False
True	False	False
True	True	True

邏輯運算子可以幫助我們簡化 if 句型，例如前頁的程式可改寫如下：

```
if month == 8 and day == 15:
    print('中秋節快樂!')
else:
    print('平常日')
```

這個程式從巢狀結構簡化成了單層結果，在閱讀時也更容易理解。建議使用者以前頁的三種 (month, day) 數值代入 and 運算，然後以 and 運算結果決定 if 句子的執行流程，並比對兩個程式片段的功能是否完全相同（答案是「完全相同」）。

邏輯運算子 or

第二個邏輯運算子是「或者」，它也是二元運算子，表 2-4-2 列出它的四種運算組合及結果。我們同樣可以歸納出一條簡單的法則，只要有一個運算元為真，or 運算的結果就為真 – 這也符合我們對「或者」這個詞的理解。

表 2-4-2: or 運算子的運算結果

a	b	a or b
False	False	False
False	True	True
True	False	True
True	True	True

下列右方程式片段展示了如何運用 or 運算來簡化左方程式的結構。

```
01 if age <= 12:
02     price *= 0.9
03 elif age >= 65:
04     price *= 0.9
05 else:
06     price -= 50
```

```
01 if age <= 12 or age >= 65:
02     price *= 0.9
03 else:
04     price -= 50
```

邏輯運算子 not

最後一個邏輯運算子是「非」，它是個一元運算子。簡單來說，它會把真變假，把假變真，如同表 2-4-3 所示。

表 2-4-3: not 運算子的運算結果

a	not a
False	True
True	False

邏輯運算子的運算優先次序低於關係運算；其中「not 運算」高於「or 運算」高於「and 運算」。下列程式片段中，第 01 行的條件式會先運算兩個相等關係運算 (==)，然後運算 not，最後再運算 and。因此，這個程式片段無法在正確的日期列印出「中秋節快樂!」。如同我們能以小括號改變算術運算的順序，例如 (2+3)*6 中先運算加法 (+) 才運算乘法 (*)，我們也能在這個程式中加上一對小括號來改正它，這裡就留給讀者去思考囉！

錯誤的範例

```
01 if not month == 8 and day == 15:
02     print('平常日')
03 else:
04     print('中秋節快樂!')
```

隨堂練習

2.4.1 請使用 Python 執行下列的運算，執行前先思考想想各邏輯運算式結果式 Ture 或是 False? 如果結果跟你想的不一樣，可以嘗試將邏輯判斷子左右兩邊的運算式分別執行看結果是甚麼。

(1) 9>0

(2) '9'>'0'

(3) '2'+'9' < '8'

(4) 2+9<8

(5) 'a'<='A'

(6) 10<5*3 and 5<3+1

(7) 10<5*3 or 5<3+1

(8) 10%3==1

(9) 10%3!=1

(10) 10 < '10'

(參考答案)

(1). True (2). True (3). True (4). False (5). False

(6). False (7). True (8). True (9). False (10). 出現錯誤