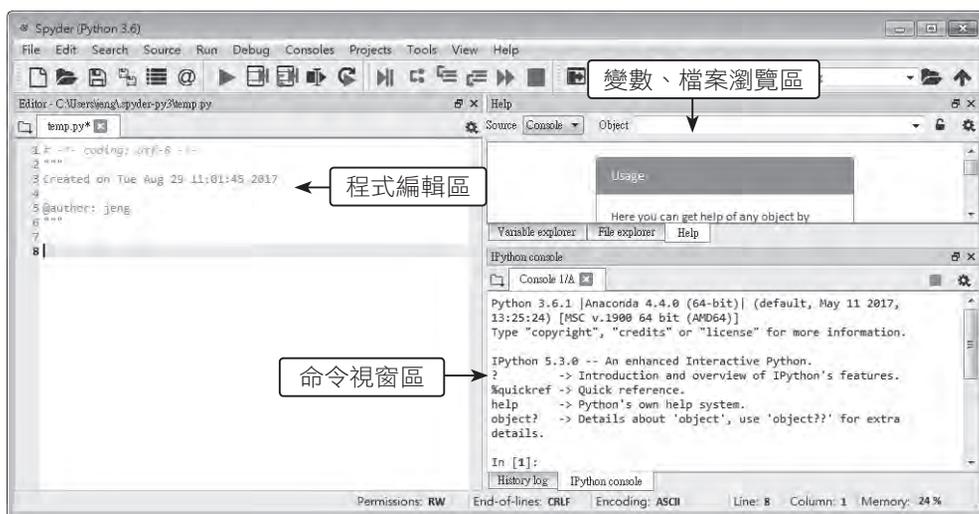


## 1.6 Spyder 編輯器

Anaconda 內建 Spyder 做為開發 Python 程式的編輯器。在 Spyder 中可以撰寫及執行 Python 程式，Spyder 還提供了簡單智慧輸入及強悍的程式除錯功能。本書所有範例預設皆使用 Spyder 進行編輯及執行。

### 1.6.1 啟動 Spyder 編輯器及檔案管理

執行 開始 \ 所有程式 \ Anaconda3 (64-bit) \ Spyder 即可開啟 Spyder 編輯器，編輯器左方為程式編輯區，可在此區撰寫程式；右上方為物件、變數、檔案瀏覽區；右下方為命令視窗區，包含 IPython 命令視窗，可在此區域用交談模式立即執行使用者輸入的 Python 程式碼；預設為 IPython 命令視窗，本書範例在此視窗顯示執行結果。

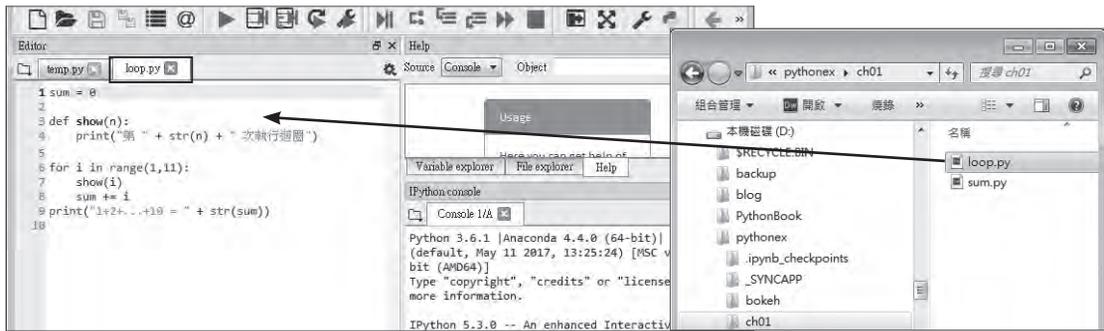


### 檔案開啟

啟動 Spyder 後，預設編輯的檔案為 <c:\users\ 電腦名稱 \.spyder-py3\temp.py>。若要建立新的 Python 程式檔，可執行 **File \ New file** 或點選工具列  鈕，撰寫程式完成後要記得存檔。

要開啟已存在的 Python 程式檔，可執行 **File \ Open** 或點選工具列  鈕，於 **Open file** 對話方塊點選檔案即可開啟。

另一個快速的方法：由檔案總管中將檔案拖曳到 Spyder 程式編輯區就會開啟該檔案。

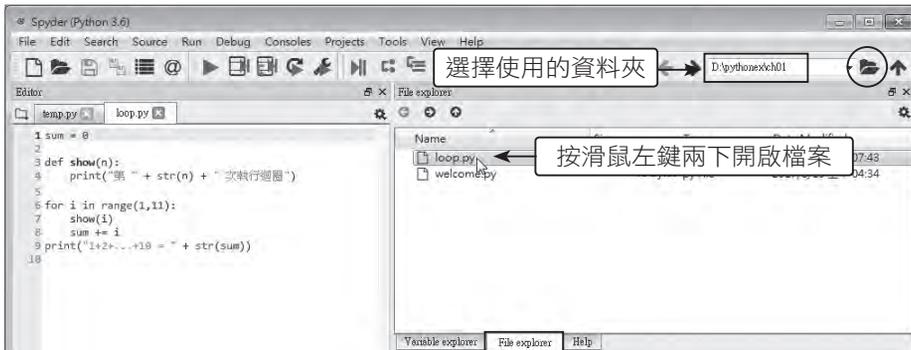


## File Explorer ( 檔案瀏覽器 ) 面板

使用者編輯的檔案通常會在同一個資料夾中，每次都要拖曳檔案到 Spyder 程式編輯區實在是一件耗時的工作。Spyder 提供 **檔案瀏覽器** 面板讓使用者管理檔案，在 **檔案瀏覽器** 面板中即可快速開啟檔案。

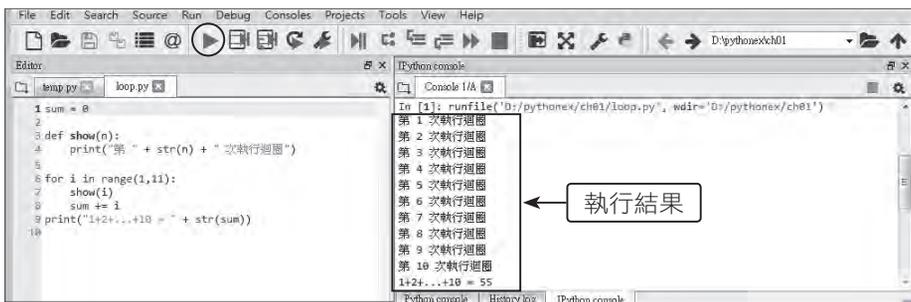
在右上方面板區點選 **File Explorer** 頁籤切換到 **檔案瀏覽器** 面板，按右上角 鈕開啟 **Select directory** 對話方塊選取資料夾後按 **選擇資料夾** 鈕。

在檔案名稱上按滑鼠左鍵兩下即可開啟檔案。



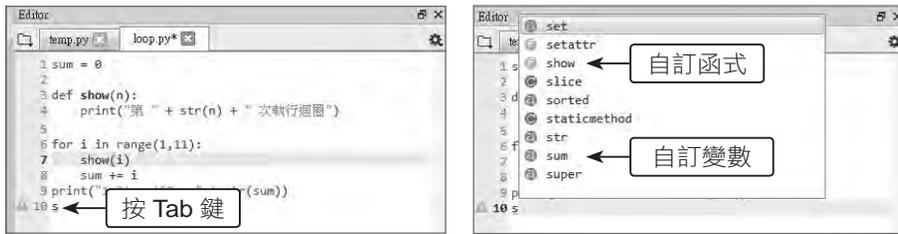
## 執行程式

執行 **Run \ Run** 或點選工具列 鈕就會執行程式，執行結果會在命令視窗區顯示。

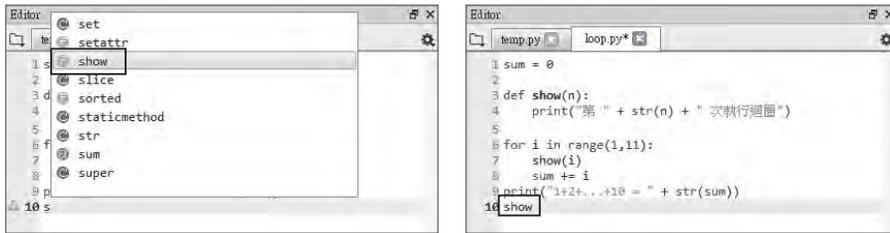


## 1.6.2 Spyder 簡易智慧輸入

Spyder 簡易智慧輸入功能與 IPython 命令視窗雷同，但操作方式比 IPython 命令視窗方便。使用者在 Spyder 程式編輯區輸入部分文字後按 **Tab** 鍵，系統會列出所有可用的項目讓使用者選取，列出項目除了內建的命令外，還包括自行定義的變數、函式、物件等。例如在 <loop.py> 輸入「s」後按 **Tab** 鍵：



使用者可按「↑」鍵或「↓」鍵移動選取項目，找到正確項目按 **Enter** 鍵就完成輸入。例如輸入「show」：



## 1.6.3 程式除錯

如何除錯，一直是程式設計師困擾的問題，如果沒有良好的除錯工具及技巧，日後當你面對較複雜的程式時就麻煩了。

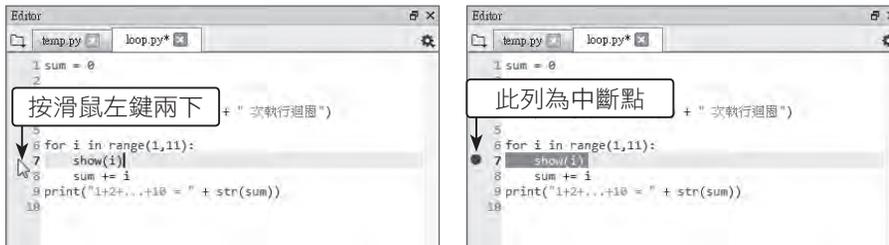
於 Spyder 輸入 Python 程式碼時，系統會隨時檢查語法是否正確，若有錯誤會在該列程式左方標示 ▲ 圖示；將滑鼠移到 ▲ 圖示片刻，會提示錯誤訊息。



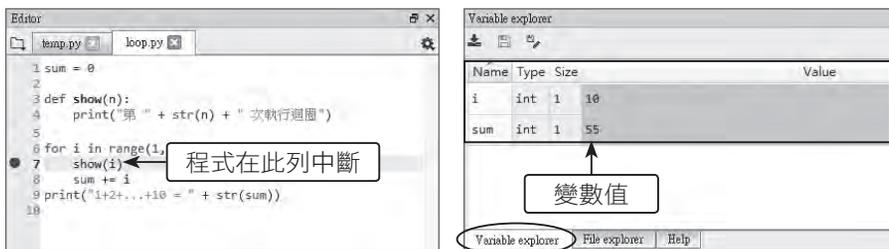


即使程式碼語法都正確，執行時仍可能發生一些無法預期的錯誤。Spyder 的除錯工具相當強大，足以應付大部分除錯狀況。

首先為程式設定中斷點：設定的方式為點選要設定中斷點的程式列，按 **F12** 鍵；或在要設定中斷點的程式列左方快速按滑鼠左鍵兩下，程式列左方會顯示紅點，表示該列為中斷點。程式中可設定多個中斷點。



以除錯模式執行程式：點選工具列 鈕會以除錯模式執行程式，程式執行到中斷點時會停止（中斷點程式列尚未執行）。於 Spyder 編輯器右上方區域點選 **Variable explorer** 頁籤，會顯示所有變數值讓使用者檢視。



除錯工具列：Spyder 除錯工具列有各種執行的方式，如單步執行、執行到下一個中斷點等，程式設計師可視需求執行，配合觀查變數值達成除錯任務。



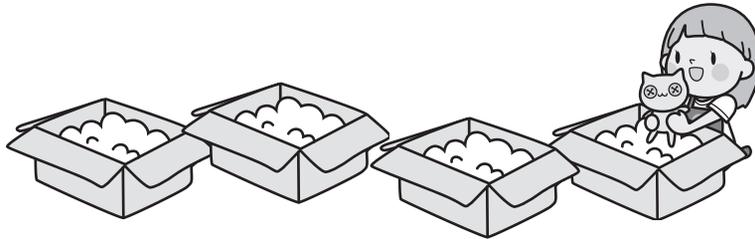
- ：以除錯方式執行程式。
- ：單步執行，不進入函式。
- ：單步執行，會進入函式。
- ：程式繼續執行，直到由函式返回或下一個中斷點才停止執行。
- ：程式繼續執行，直到下一個中斷點才停止執行。
- ：終止除錯模式回到正常模式。





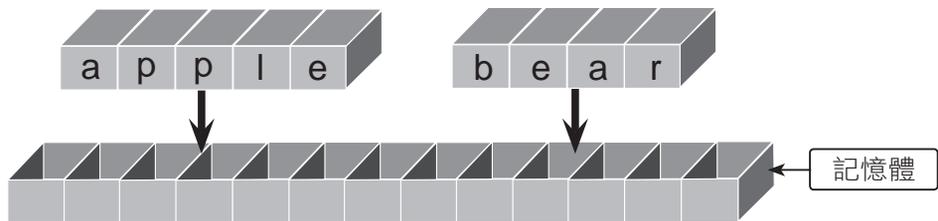
## 2.1 變數

「變數」顧名思義，是一個隨時可能改變內容的容器名稱，就像家中的收藏箱可以放入各種不同的東西。



### 2.1.1 認識變數

應用程式執行時必須先儲存許多資料等待進一步處理，例如在英文單字教學應用程式中，許多英文單字必須先儲存在電腦內，等到要使用時再將其取出。那麼電腦將這些資料儲存在哪裡呢？事實上，電腦是將資料儲存於「記憶體」中，等到需要使用特定資料時，就到記憶體中將該資料取出。



▲ 資料儲存於記憶體

當資料儲存於記憶體時，電腦會記住該記憶體的位置，以便要使用時才可以取出。但電腦的地址是一個複雜且隨機的數字，例如「65438790」，程式設計者怎麼可能會記得此地址呢？更何況有很多地址要記憶。解決的方法是給予這些地址一個有意義的名稱，取代無意義的數字地址，就可輕鬆取得電腦中的資料了！這些取代數字地址的名稱就是「變數」。



▲ 以變數取代記憶體地址

## 2.1.2 建立變數

當建立一個變數時，應用程式就會配置一塊記憶體給此變數使用，並以變數名稱做為辨識此塊記憶體的標誌，設計者就可在程式中將各種值存入該變數中。

### 新增變數

Python 變數不需宣告就可以使用，語法為：

```
變數名稱 = 變數值
```

例如建立變數 `score` 的值為 80：

```
score = 80
```

使用變數時不必指定資料型態，Python 會根據變數值設定資料型態。例如上述 `score` 變數，系統會設定其資料型態為整數 (int)。又如：

```
fruit = "香蕉" #fruit 的資料型態為字串
```

如果多個變數具有相同變數值，可以一起指定變數值，例如變數 `a`、`b`、`c` 的值皆為 20，其宣告方式為：

```
a = b = c = 20
```

也可以在同一列中指定多個變數，「變數」之間以「`,`」分隔，「值」之間也以「`,`」分隔。例如變數 `age` 的值為 18，`name` 的值為「林大山」：

```
age, name = 18, "林大山"
```

### 刪除變數

如果變數不再使用，可以將變數刪除以節省記憶體。刪除變數的語法為：

```
del 變數名稱
```

例如刪除變數 `score`：

```
del score
```



## 2.4.2 比較運算子

比較運算子會比較兩個運算式，若比較結果正確，就傳回 `True`，若比較結果錯誤，就傳回 `False`。設計者可根據比較結果，進行不同處理程序。

運算子	意義	範例	範例結果
<code>==</code>	運算式 1 是否等於運算式 2	<code>(6+9==2+13)</code> <code>(8+9==2+13)</code>	<code>True</code> <code>False</code>
<code>!=</code>	運算式 1 是否不等於運算式 2	<code>(8+9!=2+13)</code> <code>(6+9!=2+13)</code>	<code>True</code> <code>False</code>
<code>&gt;</code>	運算式 1 是否大於運算式 2	<code>(8+9&gt;2+13)</code> <code>(6+9&gt;2+13)</code>	<code>True</code> <code>False</code>
<code>&lt;</code>	運算式 1 是否小於運算式 2	<code>(5+9&lt;2+13)</code> <code>(8+9&lt;2+13)</code>	<code>True</code> <code>False</code>
<code>&gt;=</code>	運算式 1 是否大於或等於運算式 2	<code>(6+9&gt;=2+13)</code> <code>(3+9&gt;=2+13)</code>	<code>True</code> <code>False</code>
<code>&lt;=</code>	運算式 1 是否小於或等於運算式 2	<code>(3+9&lt;=2+13)</code> <code>(8+9&lt;=2+13)</code>	<code>True</code> <code>False</code>

要特別注意「`=`」及「`==`」的區別：「`=`」是將等號右方的值設定給等號左方，例如「`a=5`」，表示設定變數 `a` 的值為 5；「`==`」是判斷等號兩邊的值是否相等，例如「`a==5`」，表示判斷變數 `a` 的值是否為 5，其結果是布林值 `True`。

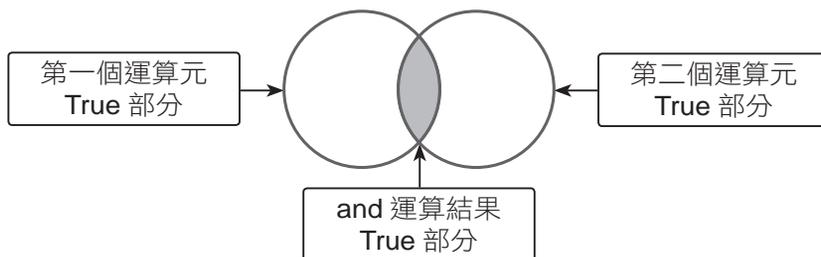
## 2.4.3 邏輯運算子

邏輯運算子通常是結合多個比較運算式來綜合得到最終比較結果，用於較複雜的比較條件。

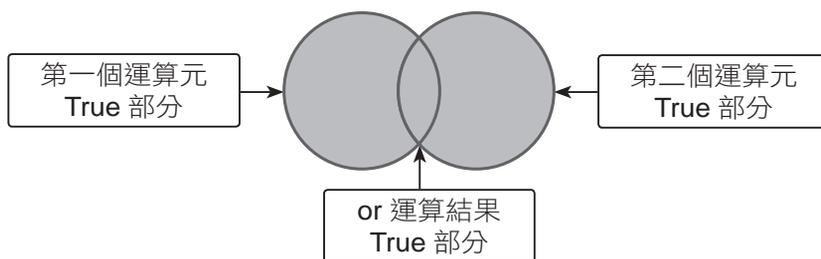
運算子	意義	範例	範例結果
<code>not</code>	傳回與原來比較結果相反的值，即比較結果是 <code>True</code> ，就傳回 <code>False</code> ；比較結果是 <code>False</code> ，就傳回 <code>True</code> 。	<code>not(3&gt;5)</code> <code>not(5&gt;3)</code>	<code>True</code> <code>False</code>
<code>and</code>	只有兩個運算元的比較結果都是 <code>True</code> 時，才傳回 <code>True</code> ，其餘情況皆傳回 <code>False</code> 。	<code>(5&gt;3) and (9&gt;6)</code> <code>(5&gt;3) and (9&lt;6)</code> <code>(5&lt;3) and (9&gt;6)</code> <code>(5&lt;3) and (9&lt;6)</code>	<code>True</code> <code>False</code> <code>False</code> <code>False</code>

運算子	意義	範例	範例結果
or	只有兩個運算元的比較結果都是 False 時，才傳回 False，其餘情況皆傳回 True。	(5>3) or (9>6) (5>3) or (9<6) (5<3) or (9>6) (5<3) or (9<6)	True True True False

「and」是兩個運算元都是 True 時其結果才是 True，相當於數學上兩個集合的交集，如下圖：



「or」是只要其中一個運算元是 True 時其結果就是 True，相當於數學上兩個集合的聯集，如下圖：



比較運算子及邏輯運算子通常會搭配判斷式使用，判斷式將在下章詳細說明。

## 2.4.4 複合指定運算子

在程式中，某些變數值常需做某種規律性改變，例如：在迴圈中需將計數變數做特定增量。一般的做法是將變數值進行運算後再指定給原來的變數，例如下面程式說明將變數  $i$  的值增加 3：

```
i = i + 3
```

這樣的寫法似乎有些累贅，因為同一個變數名稱重複寫了兩次。複合指定運算子就是為簡化此種敘述產生的運算子，將運算子置於「=」前方來取代重複的變數名稱。例如：

## 3.2 判斷式

在日常生活中，我們經常會遇到一些需要做決策的情況，然後再依決策結果進行不同的事件，例如：暑假到了，如果所有學科都及格的話，媽媽就提供經費讓自己與朋友出國旅遊；如果有某些科目當掉，暑假就要到校重修了！程式設計也一樣，常會依不同情況進行不同處理方式，這就是「判斷式」。

### 3.2.1 程式流程控制

程式的執行方式有循序式及跳躍式兩種，循序式是程式碼由上往下依序一列一列的執行，到目前為止的範例都是這種模式。程式設計也和日常生活雷同，常會遇到一些需要做決策的情況，再依決策結果執行不同的程式碼，這種方式就是跳躍式執行。

Python 流程控制命令分為兩大類：

- **判斷式**：根據關係運算或邏輯運算的條件式來判斷程式執行的流程，若條件式結果為 **True**，就執行跳躍。判斷式命令只有一個：

```
if...elif...else
```

- **迴圈**：根據關係運算或邏輯運算條件式的結果為 **True** 或 **False** 來判斷，以決定是否重複執行指定的程式。迴圈指令包括下列兩種：（迴圈將在第 4 章詳細說明）

```
for  
while
```

### 3.2.2 單向判斷式（if...）

「if...」為單向判斷式，是 if 指令中最簡單的型態，語法為：

```
if 條件式：  
    程式區塊
```

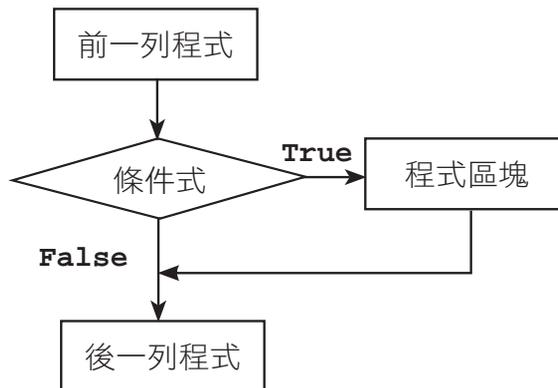
「條件式」允許加上括號，即「if (條件式):」。當條件式為 **True** 時，就會執行程式區塊的敘述；當條件式為 **False** 時，則不會執行程式區塊的敘述。



條件式可以是關係運算式，例如：「 $x > 2$ 」；也可以是邏輯運算式，例如：「 $x > 2$  or  $x < 5$ 」，如果程式區塊只有一列程式碼，也可以將兩列合併為一列，直接寫成：

```
if 條件式 : 程式碼
```

以下是單向判斷式流程控制的流程圖：



## 範例實作：密碼輸入判斷

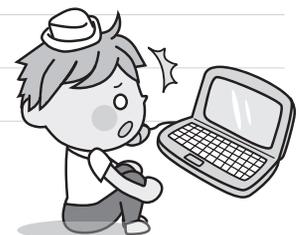
小杰設計了一個通關密碼的程式，訪客必須輸入正確密碼才能登入，如果輸入的密碼正確（1234），會顯示「歡迎光臨！」；如果輸入的密碼錯誤，則不會顯示任何訊息。（<password1.py>）

```
Python console
Console 1/A
請輸入密碼：1234
歡迎光臨！
In [2]:
```

```
Python console
Console 1/A
請輸入密碼：5678
In [3]:
```

程式碼：ch03\password1.py

```
1 pw = input("請輸入密碼：")
2 if pw=="1234":
3     print("歡迎光臨!")
```



## 程式說明

- ▣ 2-3 預設的密碼為「1234」，若輸入的密碼正確，就執行第 3 列程式列印「歡迎光臨！」訊息；若輸入的密碼錯誤就結束程式。
- ▣ 3 if 條件成立的程式區塊，必須以 Tab 鍵或空白鍵向右縮排，本例是以 4 個空白鍵做縮排。

因為此處 if 程式區塊的程式碼只有一列，所以第 2-3 列可改寫為：

```
if pw=="1234" : print("歡迎光臨!")
```

### 3.2.3 雙向判斷式 (if...else)

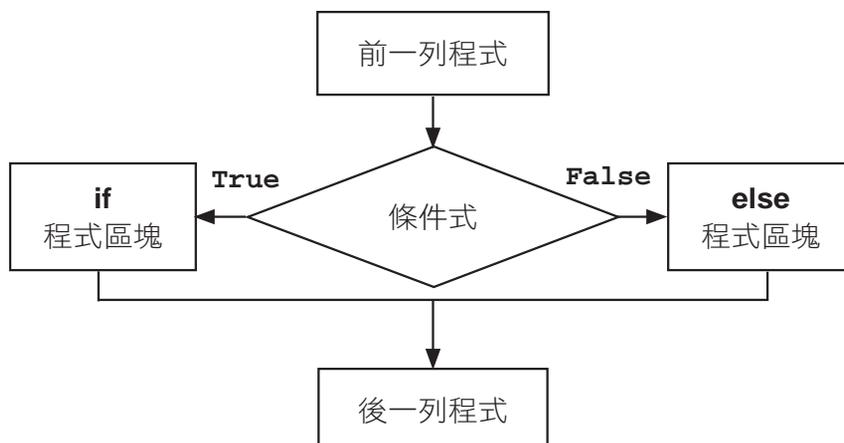
感覺上「if」語法並不完整，因為如果條件式成立就執行程式區塊內的內容，如果條件式不成立也應該做某些事來告知使用者。例如密碼驗證時，若密碼錯誤應顯示訊息告知使用者，此時就可使用「if...else...」雙向判斷式。

「if...else...」為雙向判斷式，語法為：

```
if 條件式:
    程式區塊一
else:
    程式區塊二
```

當條件式為 True 時，會執行 if 後的程式區塊一；當條件式為 False 時，會執行 else 後的程式區塊二，程式區塊中可以是一列或多列程式碼，如果程式區塊中的程式碼只有一列，可以合併為一列。

以下是雙向判斷式流程控制的流程圖：



## 4.2 for 迴圈

for 迴圈通常用於執行固定次數的迴圈，其基本語法結構為：

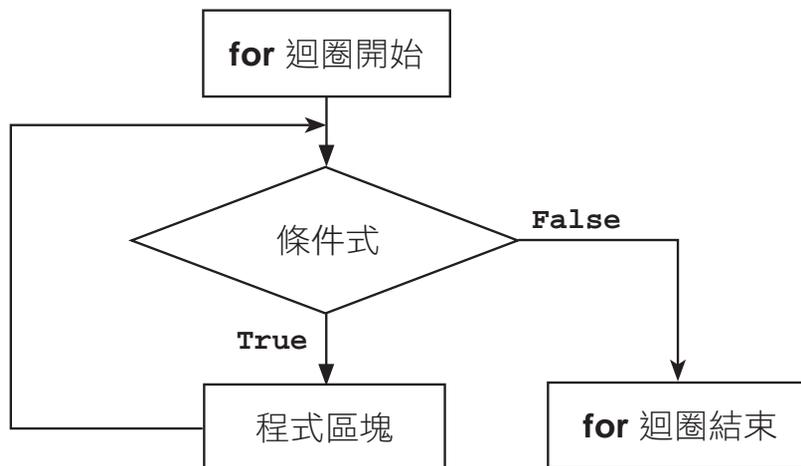
```
for 變數 in 數列：
    程式區塊
```

執行 for 迴圈時，系統會將數列的元素依序做為變數的值，每次設定變數值後就會執行「程式區塊」一次，即數列有多少個元素，就會執行多少次「程式區塊」。以實例解說：

```
1 for n in range(3):           # 產生 0,1,2 的數列
2     print(n, end=",")       # 執行結果為：0,1,2,
```

開始執行 for 迴圈時，變數 n 的值為「0」，第 2 列程式列印「0,」；然後回到第 1 列程式設定變數 n 的值為「1」，再執行第 2 列程式列印「1,」；同理回到第 1 列程式設定變數 n 的值為「2」，再執行第 2 列程式列印「2,」，數列元素都設定完畢，程式就結束迴圈。

for 迴圈的流程如下：



使用 range 函式可以設定 for 迴圈的執行次數，例如要列印全班成績，若班上有 30 位同學，列印程式碼為（注意第 2 個參數終止值是 31）：

```
for i in range(1,31):
    列印程式碼
```



## 4.2.1 巢狀 for 迴圈

與「if...elif...else」相同，for 迴圈之中也可以再包含 for 迴圈，稱為巢狀 for 迴圈。使用巢狀 for 迴圈時需特別注意執行次數的問題，因為它是各層迴圈的乘積，執行次數太多會耗費相當長時間，可能讓使用者以為電腦當機，例如：

```
n = 0
for i in range(1,10001):
    for j in range(1,10001):
        n += 1
print(n)
```

當外層迴圈及內層迴圈都是執行一萬次，則「n+=1」總共會執行一億次(10000x10000)，執行時間視 CPU 速度約需十餘秒到數十秒。

下面範例建立兩層迴圈，內層迴圈的執行次數會依外層迴圈的變數值而改變，如此可將顯示的「井」字排列成三角形。



### 範例實作：井字直角三角形

利用兩層迴圈列印「井」字，將其排列成直角三角形。(forneest.py)

```
IPython console
Console 1/A
外部第 1 次迴圈,內部執行 1 次迴圈: #
外部第 2 次迴圈,內部執行 2 次迴圈: ##
外部第 3 次迴圈,內部執行 3 次迴圈: ###
外部第 4 次迴圈,內部執行 4 次迴圈: ####
外部第 5 次迴圈,內部執行 5 次迴圈: #####
In [130]:
```



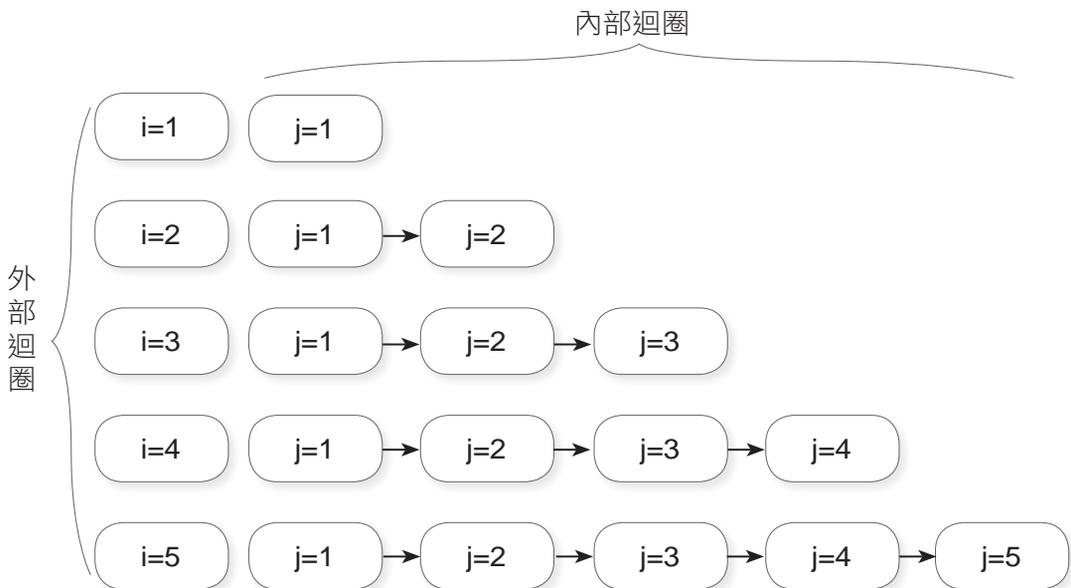
程式碼：ch04\forneest.py

```
1 for i in range(1,6): # 外部迴圈，共執行 5 次
2     print("外部第",i,"次迴圈,內部執行",i,"次迴圈: ",end="")
3     for j in range(1,i+1): # 內部迴圈
4         print("#", end="")
5     print() # 換行
```

程式說明

- 1 建立外層 for 迴圈，共執行 5 次。
- 2 顯示訊息。
- 3 建立內層 for 迴圈，執行次數由外層迴圈的 i 變數值決定（第二個參數  $j<i+1$ ），即第一次列印一個「井」字，第二次列印兩個「井」字，依此類推。
- 4 列印「井」字。
- 5 每一次外部迴圈都由新的一列開始，所以加入換行字元。

第一次執行外部迴圈時 i 的值為 1，內部迴圈只執行一次，所以列印一個「井」字；第二次執行外部迴圈時 i 的值為 2，內部迴圈需執行兩次，所以列印兩個「井」字；依此類推，直到執行完 i 等於 5 的迴圈。



▲ 井字三角形示意圖



範例實作：九九乘法表

利用兩層 for 迴圈列印九九乘法表。( <ninenine.py > )

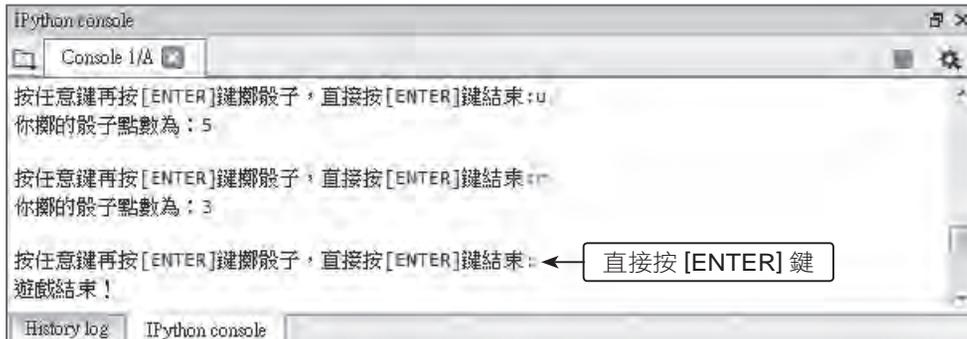
```

Console 1/A
1*1= 1  1*2= 2  1*3= 3  1*4= 4  1*5= 5  1*6= 6  1*7= 7  1*8= 8  1*9= 9
2*1= 2  2*2= 4  2*3= 6  2*4= 8  2*5=10  2*6=12  2*7=14  2*8=16  2*9=18
3*1= 3  3*2= 6  3*3= 9  3*4=12  3*5=15  3*6=18  3*7=21  3*8=24  3*9=27
4*1= 4  4*2= 8  4*3=12  4*4=16  4*5=20  4*6=24  4*7=28  4*8=32  4*9=36
5*1= 5  5*2=10  5*3=15  5*4=20  5*5=25  5*6=30  5*7=35  5*8=40  5*9=45
6*1= 6  6*2=12  6*3=18  6*4=24  6*5=30  6*6=36  6*7=42  6*8=48  6*9=54
7*1= 7  7*2=14  7*3=21  7*4=28  7*5=35  7*6=42  7*7=49  7*8=56  7*9=63
8*1= 8  8*2=16  8*3=24  8*4=32  8*5=40  8*6=48  8*7=56  8*8=64  8*9=72
9*1= 9  9*2=18  9*3=27  9*4=36  9*5=45  9*6=54  9*7=63  9*8=72  9*9=81
    
```



## 範例實作：擲骰子遊戲

阿寶想玩擲骰子遊戲，但手邊沒有骰子，設計程式讓阿寶按任意鍵再按 **Enter** 鍵擲骰子，會顯示 1 到 6 之間的整數亂數代表骰子點數，直接按 **Enter** 鍵會結束遊戲。( <randint.py> )



程式碼：ch07\randint.py

```

1 import random
2
3 while True:
4     inkey = input(" 按任意鍵再按 [ENTER] 鍵擲骰子，直接
                    按 [ENTER] 鍵結束：")
5     if len(inkey) > 0:
6         num = random.randint(1,6)
7         print(" 你擲的骰子點數為：" + str(num))
8     else:
9         print(" 遊戲結束！")
10        break

```

終於可以玩遊戲了！



### 程式說明

- 1 匯入亂數模組。
- 3-10 以無窮迴圈讓使用者擲骰子。
- 5-7 使用者按任意鍵再按 **Enter** 鍵就取得 1 到 6 之間的亂數顯示。
- 8-10 使用者直接按 **Enter** 鍵就顯示「結束遊戲」訊息並跳出迴圈。



## 延伸練習

小朋友在玩「大富翁」桌遊，遊戲者要連擲三次骰子，然後以總點數決定遊戲前進步數。遊戲時小朋友發現骰子不見了，請設計程式以亂數函式擲骰子三次，然後計算三次骰子點數的總和。(〈randint\_cl.py〉)

```
IPython console
Console 1/A
你三次擲骰子的點數為 6 2 4，總點數為：12
History log IPython console
```

## 7.4.4 隨機取得字元或串列元素

### choice 函式

choice 函式的功能是隨機取得一個字元或串列元素，語法為：

```
random.choice( 字串或串列 )
```

如果參數是字串，就隨機由字串中取得一個字元，例如：

```
import random
for i in range(5): # 執行 5 次，產生 5 個整數亂數
    print(random.choice("abcdefg"), end=",") #f,a,g,g,d,
```

如果參數是串列，就隨機由串列中取得一個元素，例如：

```
import random
for i in range(5): # 執行 5 次，產生 5 個整數亂數
    print(random.choice([1,2,3,4,5,6,7]), end=",") #1,1,2,7,6,
```

### sample 函式

sample 函式的功能與 choice 雷同，只是 sample 函式可以隨機取得多個字元或串列元素，語法為：

```
random.sample( 字串或串列, 數量 )
```



如果參數是字串，就隨機由字串中取得指定數量的字元；如果參數是串列，就隨機由串列中取得指定數量的元素，例如：

```
import random  
print(random.sample("abcdefg",3))      #['f','b','g']  
print(random.sample([1,2,3,4,5,6,7],3)) # [3,1,4]
```

需注意「數量」參數的值不能大於字串長度或串列元素個數，也不能是負數，否則執行時會產生錯誤，例如：

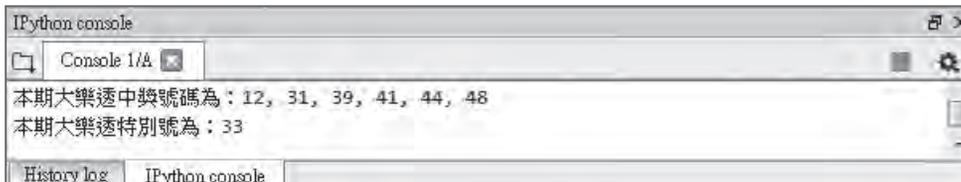
```
import random  
print(random.sample([1,2,3,4,5,6,7],8))  
# 錯誤，數量大於串列元素個數
```

`sample` 函式最重要的用途是可以由串列中取得指定數量且不重複的元素，這使得設計樂透開獎應用程式變得輕鬆愉快。



## 範例實作：大樂透中獎號碼

大樂透中獎號碼為 6 個 1 到 49 之間的數字加 1 個特別號：撰寫程式取得大樂透中獎號碼，並由小到大顯示方便對獎。( <sample.py> )



程式碼：ch07\sample.py

```
1 import random  
2  
3 list1 = random.sample(range(1,50), 7)  
4 special = list1.pop()  
5 list1.sort()  
6 print(" 本期大樂透中獎號碼為：", end="")  
7 for i in range(6):
```