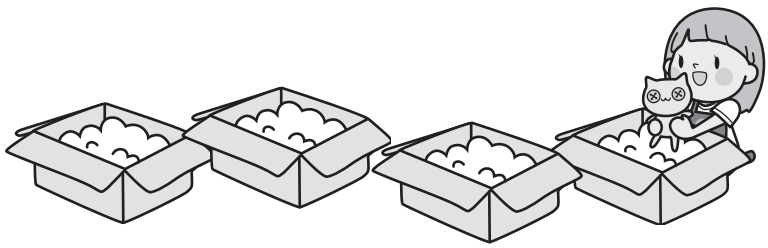
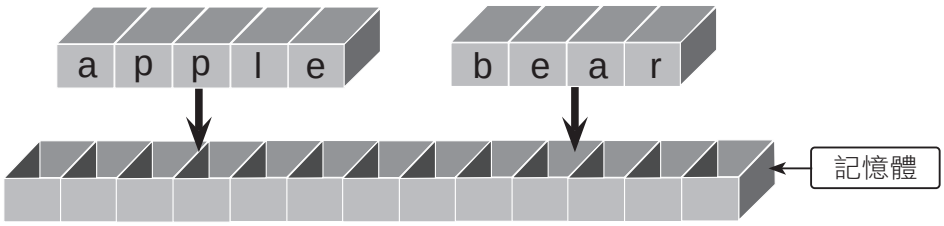
2.1 變數

「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。



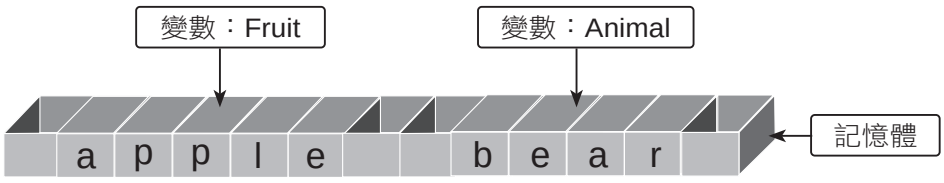
2.1.1 認識變數

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
```

2.1.4 註解

註解的用途是做為程式的說明。程式撰寫者當然了解自己程式碼的流程，但是其他使用者要理解程式在做些什麼事就比較困難，因此加入註解可幫助其他使用者了解程式。即使是程式撰寫者，在年代久遠後也常忘了當初撰寫程式的流程，註解可以讓程式撰寫者快速回憶程式的用途。

單行註解

Python 可在程式碼中加入「#」做為單行註解，使用方式有兩種：第一種是位於程式列起始處，該行程式都不會執行，例如：

```
#fruit 變數為最喜歡吃的水果名稱
fruit = "香蕉"
```

此種方式註解就佔了一行程式，會讓程式看起來變得較龐大。第二種是位於程式列後方，「#」號後的程式碼不執行，例如：

```
fruit = "香蕉"   #fruit 變數為最喜歡吃的水果名稱
```

多行註解

如果有連續多行程式需要註解，為每行程式都加上「#」符號非常麻煩，所以可以在註解的區塊前後加入三個單引號 (''') 或三個雙引號 (""") 作為多行註解。許多程式設計者會使用多行註解來說明程式用途、作者等，例如：

```
"""
本程式可計算使用者 BMI 值提供使用者參考。
使用者輸入身高及體重後會顯示 BMI 值。
設計者:文淵閣工作室
"""
```

註解是為了讓觀看程式碼的人能夠快速了解程式碼的目的、功能及使用方式，同時也幫助自己記錄程式發展的過程，因此建議在撰寫程式時盡可能為程式碼加上註解，養成撰寫程式的好習慣。

3.1 Python 程式碼縮排

程式語言以縮排方式表示是一組相同的程式區塊。

大部分語言如 C、Java 等，都是以一對大括號「{}」來表示程式區塊，例如：

```
if (score>=60) {
   grade = "及格";
}
sum = sum + score
```

程式區塊

下一列程式

3.1.1 Python 程式碼縮排格式

Python 語言以冒號「:」及縮排來表示程式區塊，縮排建議使用 4 個空白鍵，例如：

```
if score>=60:
    grade = "及格"
sum = sum + score
```

程式區塊

下一列程式

4 個空白鍵縮排

在 Python 中建議的縮排方式是用 4 個空白鍵，但許多人卻習慣使用 Tab 鍵，在不同的編輯器讀取時可能就會產生不一樣的效果。

3.1.2 絕對不要混用 Tab 鍵和空白鍵

其實只要以相同的 Tab 鍵或相同字元的空白鍵整齊排列，即可達到同一程式區塊程式碼縮排的效果，但同一個程式區塊中絕對不要混用 Tab 鍵和空白鍵，官方建議以 4 個空白鍵做為縮排。

混用 Tab 鍵和空白鍵來縮排的程式碼，應該轉成只用空白鍵。在呼叫 Python 直譯器時加上「-t」選項，它會對混用 Tab 鍵和空白鍵的程式發出警告。若使用「-tt」選項，則會發出錯誤。

如果您使用的是 Spyder 或 Jupyter NoteBook 編輯器，可以按 4 個空白鍵，即使用 Tab 鍵也會自動轉換為 4 個空白鍵，避免這個問題。

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：

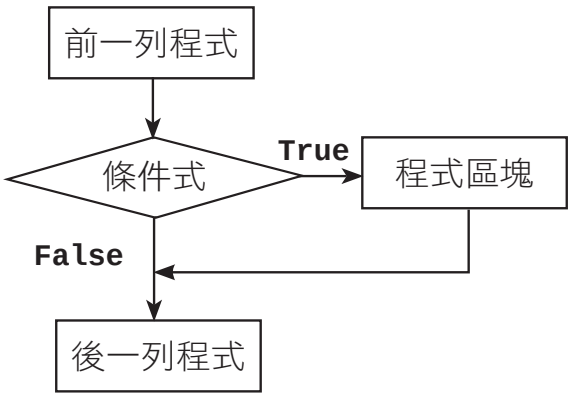
```
if 條件式：
    程式區塊
```

「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：

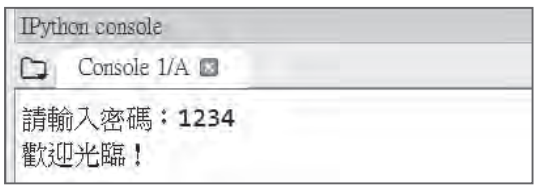
```
if 條件式 : 程式碼
```

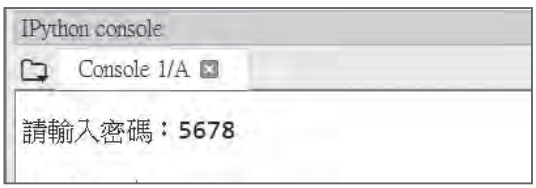
以下是單向判斷式流程控制的流程圖：



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>)





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）

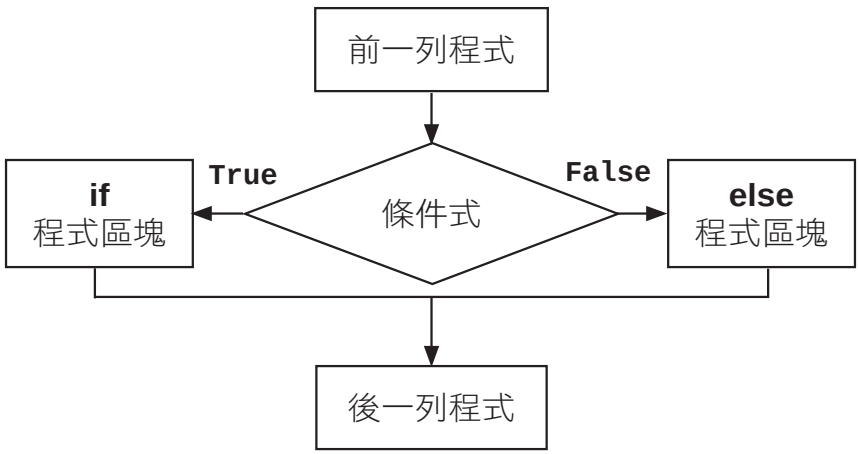
感覺上「if」語法並不完整，因為如果條件式成立就執行程式區塊內的內容，如果條件式不成立也應該做某些事來告知使用者。例如密碼驗證時，若密碼錯誤應顯示訊息告知使用者，此時就可使用「if…else…」雙向判斷式。

「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
```

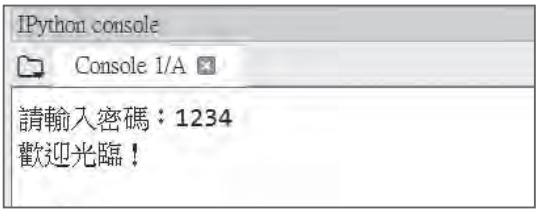
當條件式為 True 時，會執行 if 後的程式區塊一；當條件式為 False 時，會執行 else 後的程式區塊二，程式區塊中可以是一列或多列程式碼，如果程式區塊中的程式碼只有一列，可以合併為一列。

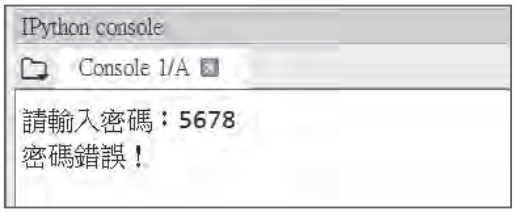
以下是雙向判斷式流程控制的流程圖：



範例實作：進階密碼判斷

小杰程式設計的功力進步許多，現在他改進了通關密碼程式，如果訪客輸入的密碼正確（1234），會顯示「歡迎光臨！」；如果訪客輸入的密碼錯誤，則會顯示「密碼錯誤！」。(<password2.py>)





程式碼：ch03\password2.py

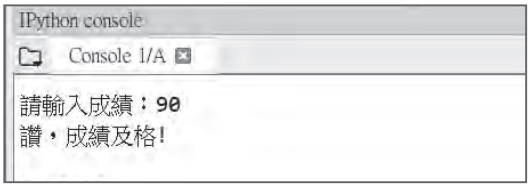
```
1  pw = input("請輸入密碼:")
2  if pw=="1234":
3      print("歡迎光臨!")
4  else:
5      print("密碼錯誤!")
```

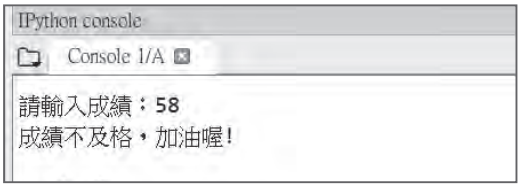
程式說明

- 2-3　若輸入的密碼正確，就執行第 3 列程式，顯示歡迎訊息。
- 4-5　若輸入的密碼錯誤，就執行第 5 列程式，顯示密碼錯誤訊息。注意第 4 列要由開頭處輸入「else:」。

延伸練習

資訊小楷模阿梅幫老師設計一個程式，讓老師輸入學生的成績，若學生成績大於等於 60 分，顯示「讚，成績及格!」，否則顯示「成績不及格，加油喔!」。(<score.py>)





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：

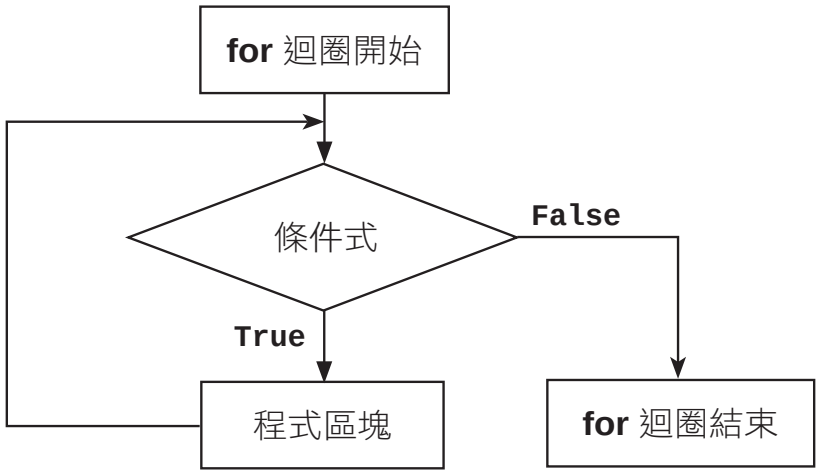
```
for 變數 in 數列：
    程式區塊
```

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：

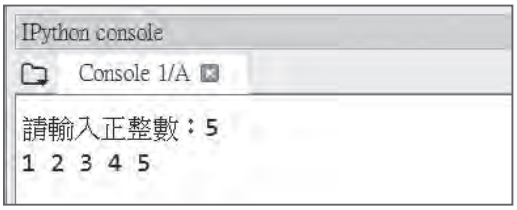


使用 range 函式可以設定 for 迴圈的執行次數，例如要列印全班成績，若班上有 30 位同學，列印程式碼為 (注意第 2 個參數終止值是 31)：

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

範例實作：顯示正整數數列

叮叮利用 range 函式，設計一個簡易的數列，使用者只要輸入一個正整數，程式就會顯示由 1 到該整數的整數數列。(<numshow.py>)



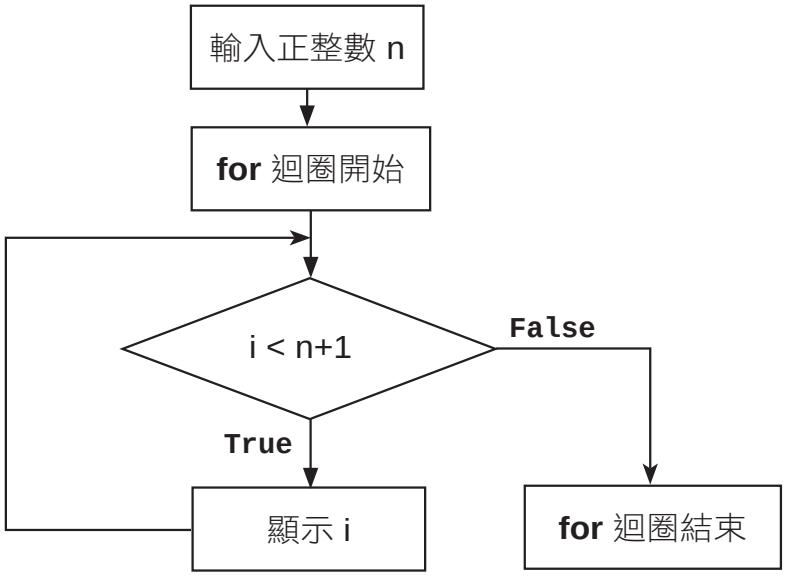
程式碼：ch04\numshow.py

```
1 n = int(input("請輸入正整數:"))
2 for i in range(1, n+1):
3     print(i,end=" ")
```

程式說明

- 1　取得輸入資料並轉為整數。
- 2-3　以迴圈顯示 1~n 的正整數列，數值之間以空白字元分隔。注意第 2 列程式第 2 個參數需用「n+1」。

上面範例的執行過程：



▲ 顯示正整數數列流程圖

6.1 字典基本操作

還記得怎麼查國語字典嗎？以查「開」字為例，先由部首目錄找到部首「門」在字典中的位置，剩下來的「幵」筆畫為 4 畫，再於「門部首」 4 畫的地方就能找到「開」字。

Python 中「字典」資料型態與國語字典結構類似，其元素是以「鍵 - 值」對方式儲存，運作方式為利用「鍵」來取得「值」。

6.1.1 建立字典

串列資料依序排列，若要取得串列內特定資料，必須知道其在串列中的位置，例如一個水果價格的串列：

```
list1 = [20, 50, 30]   # 分別為香蕉、蘋果、橘子的價格
```

若要得知蘋果的價格，就要知道蘋果價格是串列第 2 個元素，再使用「list1[1]」取出蘋果價格，是不是很不方便呢？

字典的結構也與串列類似，其元素是以「鍵 - 值」對方式儲存，這樣就可使用「鍵」來取得「值」。有多種方式可以建立字典，第一種方式為將元素置於一對大括號「{}」中，其語法為：

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

字串、整數、浮點數等皆可做為「鍵」，但以字串做為「鍵」的情況最多。

例如將前述水果價格串列建立為字典型態：

```
dict1 = {"香蕉":20, "蘋果":50, "橘子":30}
```

建立字典的第二種方式是使用 dict 函式，再將鍵 - 值對置於中括號「[]」中，語法為：

```
字典名稱 = dict([[ 鍵 1, 值 1], [ 鍵 2, 值 2], ……])
```

例如：

```
dict2 = dict([["香蕉",20], ["蘋果",50], ["橘子",30]])
```

建立字典的第三種方式也是使用 dict 函式， 只要將鍵與值以等號連接起來即可，語法為：

```
字典名稱 = dict( 鍵 1= 值 1, 鍵 2= 值 2, ……)
```

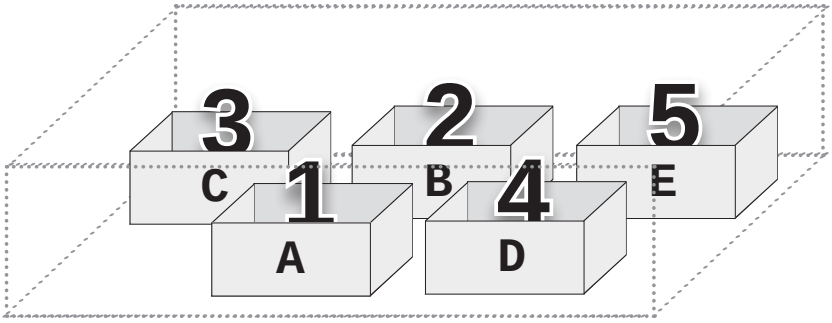
例如：

```
dict3 = dict( 香蕉 =20, 蘋果 =50, 橘子 =30)
```

第三種建立字典的方式相當簡潔，但特別注意此種方式建立的字典「鍵」不能使用數值，否則執行時會產生錯誤。

6.1.2 字典取值

可以將字典想像成一個箱子，箱子中許多盒子，每個盒子都貼上標籤，標籤寫著盒子的名稱 (鍵)，盒子內則裝著指定的物品 (值)，與串列最大的不同在於串列元素在記憶體中是依序排列，而字典元素則是隨意放置，沒有一定順序。



基本取值方式

既然字典元素沒有一定順序，那要如何取得字典元素值呢？其實很簡單，只要依據標籤 (鍵) 找到存放物品的盒子，就能取得盒子內的物品 (值)。取得字典元素值的方法是以「鍵」做為索引來取得「值」，語法為：

```
字典名稱 [ 鍵 ]
```

例如：

```
dict1 = {" 香蕉 ":20, " 蘋果 ":50, " 橘子 ":30}
print(dict1[" 蘋果 "])  #50
```

當字典的鍵重複時

字典是使用「鍵」做為索引來取得「值」，所以「鍵」必須是唯一，而「值」可以重覆。如果「鍵」重覆的話，則前面的「鍵」會被覆蓋，只有最後的「鍵」有效，例如：

```
dict2 = {"香蕉":20, "蘋果":50, "橘子":30, "香蕉":25}
print(dict2["香蕉"])   #25, 「"香蕉":20」被覆蓋
```

當字典的鍵不存在時

元素在字典中的排列順序是隨機的，與設定順序不一定相同，例如：

```
dict1 = {"香蕉":20, "蘋果":50, "橘子":30}
print(dict1)   # 結果:{"蘋果":50, "香蕉":20, "橘子":30}
```

由於元素在字典中的排列順序是隨機的，所以不能以位置數值做為索引。另外，若輸入的「鍵」不存在也會產生錯誤，例如：

```
dict1 = {"香蕉":20, "蘋果":50, "橘子":30}
print(dict1[0])          # 錯誤
print(dict1["鳳梨"])     # 錯誤
```

此種字典取值方式當「鍵」不存在時會因錯誤而讓程式中斷，因此 Python 另外提供了 get 方法可以取得字典元素值，即使「鍵」不存在也不會產生錯誤，語法為：

```
字典名稱.get(鍵[, 預設值])
```

預設值可有可無。根據是否有傳入預設值及「鍵」是否存在可分為四種情形：

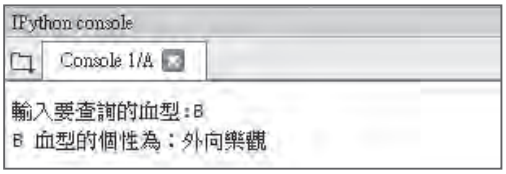
預設值狀況	「鍵」是否存在	返回值
沒有傳入預設值	「鍵」存在	返回鍵對應的值
	「鍵」不存在	返回 None
有傳入預設值	「鍵」存在	返回鍵對應的值
	「鍵」不存在	返回預設值

當「鍵」不存在時會傳回 None 或預設值，程式執行時不會產生錯誤。例如：

```
dict1 = {"香蕉":20, "蘋果":50, "橘子":30}
print(dict1.get("蘋果"))          #50
print(dict1.get("鳳梨"))          #None
print(dict1.get("蘋果", 80))      #50
print(dict1.get("鳳梨", 80))      #80
```

範例實作：血型個性查詢

不同血型的人具有不同的個性：設計程式建立 4 筆字典資料：「鍵」為血型，「值」為個性。使用者輸入血型後，若血型存在，就顯示該血型的個性，如果血型不存在，則顯示沒有該血型的訊息。(<dictget.py>)



程式碼：ch06\dictget.py

```
1 dict1 = {"A":"內向穩重", "B":"外向樂觀", "O":"堅強自信",
  "AB":"聰明自然"}
2 name = input("輸入要查詢的血型:")
3 blood = dict1.get(name)
4 if blood == None:
5     print("沒有「" + name + "」血型！")
6 else:
7     print(name + " 血型的個性為:" + str(dict1[name]))
```

程式說明

- 1　建立 4 個血型及個性的字典。
- 2　讓使用者輸入血型。
- 3　以 get 取得個性。
- 4-5　若血型不存在就顯示沒有該血型訊息。
- 6-7　若血型存在就顯示該血型的個性。

8.3 搜尋

資料搜尋是串列另一個最常使用的功能。搜尋資料常用的搜尋方法有循序搜尋和二分搜尋，循序搜尋是依序逐一搜尋；使用二分搜尋則可以提昇速度，但程式較複雜，且搜尋前資料必須先進行排序。

8.3.1 循序搜尋

循序搜尋是從串列中第一個串列元素開始，依序逐一搜尋，方法很簡單，但是缺點是較沒有效率。

循序搜尋的演算法

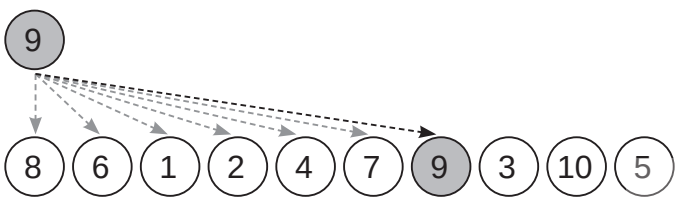
循序搜尋資料可以不用排序，演算法的運作如下：

1. 從串列中第一個串列元素開始搜尋。
2. 如果找到目標，結束搜尋。
3. 如果沒有找到目標，繼續搜尋下一個串列元素，直到串列元素全部搜尋完為止。

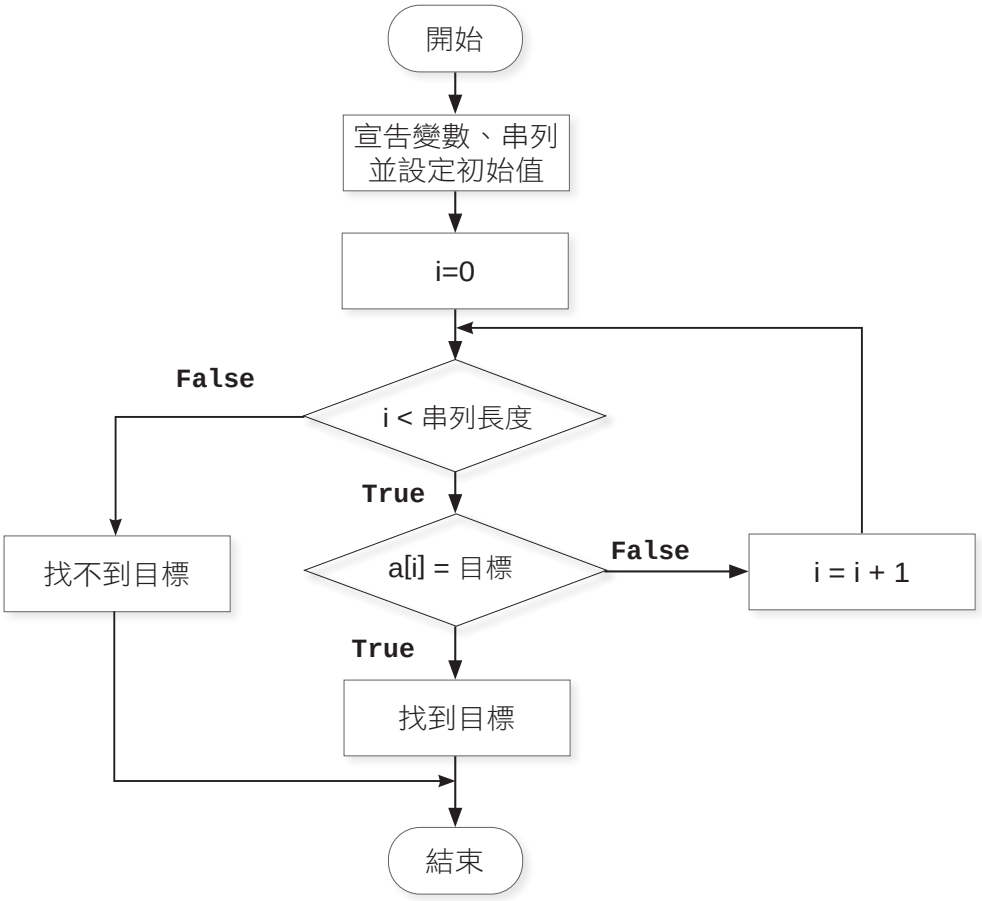
循序搜尋是從第一個串列元素開始，依序逐一搜尋，如果串列元素有 n 個，循序搜尋最快一次就可以搜尋到，最慢則需 n 次才能搜尋到。程式中刻意加入「比對次數」訊息，目的是要讓使用者了解實際的搜尋次數（實際應用程式中可將此部分移除）：觀察執行結果的第二項，當查詢資料不存在時，循序搜尋須從頭到尾搜尋一次，本範例中有 8 筆資料，所以顯示比對 8 次。實際應用時資料動輒數十萬筆，一次搜尋就要比對數十萬次，非常沒有效率，將造成系統很重的負擔，且搜尋時間很長！

循序搜尋的運作實例

例如有一序列的值是 8, 6, 1, 2, 4, 7, 9, 3, 10, 5，若想要在其中找出 9 這個值所在的位置，循序搜尋的過程如下，要找到第 7 次才能找到：



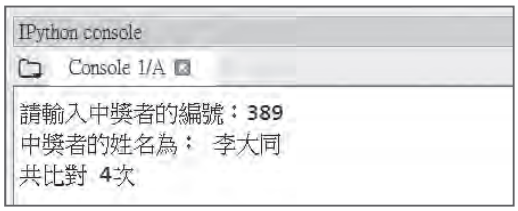
循序搜尋的流程圖

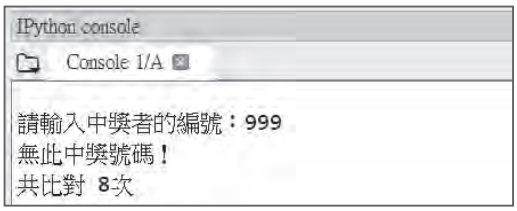


▲ 循序搜尋流程圖

範例實作：中獎者姓名 (循序搜尋)

百貨公司舉辦週年抽獎活動，將顧客的抽獎編號及姓名分別儲存於串列中，使用者輸入編號，程式會搜尋出該編號的姓名並顯示；若查詢不到也會顯示無此編號的訊息。(<sequential.py>)





程式碼：ch08\sequential.py

```
1   num=[256,731,943,389,142,645,829,945]
2   name=["林小虎","王中森","邵木淼","李大同","陳子孔",
          "鄭美麗","曾溫柔","錢來多"]
3   no = int(input("請輸入中獎者的編號:"))
4
5   IsFound=False
6   for i in range(len(name)):   #逐一比對搜尋
7       if (num[i]==no):         #號碼相符
8           IsFound=True         #設旗標為 True
9           break                #結束比對
10
11  if (IsFound==True):
12      print("中獎者的姓名為:",name[i])
13  else:
14      print("無此中獎號碼!")
15  print("共比對 %d 次 " %(i+1))
```

程式說明

- 1-2　分別建立編號及姓名的對應串列。
- 3　輸入中獎者的編號 no。
- 5　宣告 IsFound 並預設為 False，如果在 6-9 列的搜尋有找到查詢資料，就設 IsFound=True。在程式設計中，這個觀念稱為旗標，也就是如果有找到就將設旗標 IsFound=True，否則 IsFound 的值為 False，最後判斷旗標 IsFound 就可得知資料是否有找到。
- 6-9　逐一比對資料是否相符。
- 9　如果找到查詢資料就離開 for 迴圈。
- 11-14　根據判斷旗標 IsFound 以得知資料是否有找到來顯示訊息。
- 15　因為 i 的值是由 0 開始，所以比對次數是將其加 1 才是真正的比對次數。