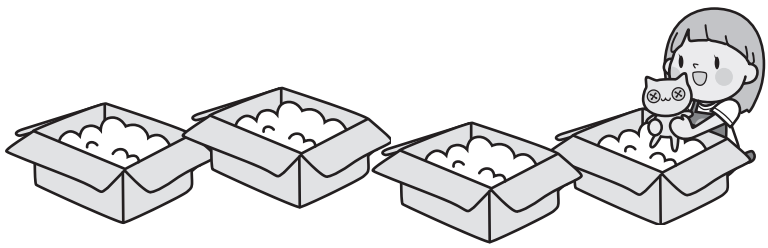
Chapter

02

變數與運算式

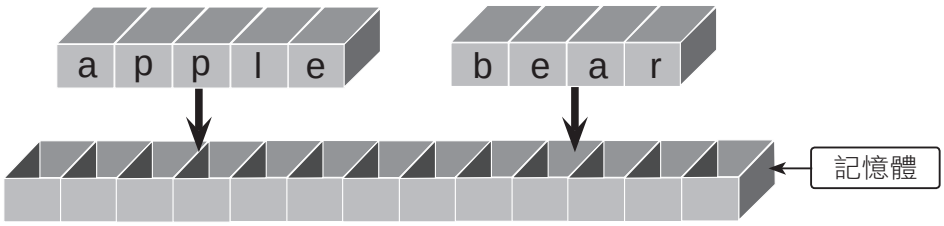
2.1 變數

「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。



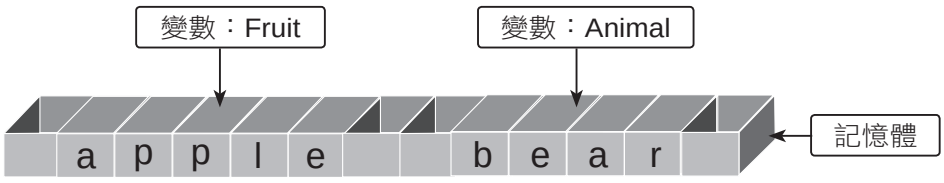
2.1.1 認識變數

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
```

2.1.3 變數命名規則

為變數命名必須遵守一定規則，否則在程式執行時會產生錯誤。Python 變數的命名規則為：

- 變數名稱的第一個字母必須是大小寫字母、_、中文。
- 只能由大小寫字母、數字、_、中文組成變數名稱。
- 英文字母大小寫視為不同變數名稱。
- 變數名稱不能與 Python 內建的保留字相同。Python 的保留字有 ：

acos	and	array	asin	assert	atan
break	class	close	continue	cos	Data
def	del	e	elif	else	except
exec	exp	fabs	float	finally	floor
for	from	global	if	import	in
input	int	is	lambda	log	log10
not	open	or	pass	pi	print
raise	range	return	sin	sqrt	tan
try	type	while	write	zeros	

雖然 Python 3.x 的變數名稱支援中文，但建議最好不要使用中文做為變數命名，不但在撰寫程式時輸入麻煩，而且會降低程式的可攜性。

下表是一些錯誤變數名稱的範例實作：

屬性	說明
7eleven	第一個字元不能是數字
George&Mary	不能包含特殊字元「&」
George Mary	不能包含空白字元
if	Python 的保留字

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
==	運算式 1 是否等於運算式 2	(6+9==2+13) (8+9==2+13)	True False
!=	運算式 1 是否不等於運算式 2	(8+9!=2+13) (6+9!=2+13)	True False
>	運算式 1 是否大於運算式 2	(8+9>2+13) (6+9>2+13)	True False
<	運算式 1 是否小於運算式 2	(5+9<2+13) (8+9<2+13)	True False
>=	運算式 1 是否大於或等於運算式 2	(6+9>=2+13) (3+9>=2+13)	True False
<=	運算式 1 是否小於或等於運算式 2	(3+9<=2+13) (8+9<=2+13)	True False

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。

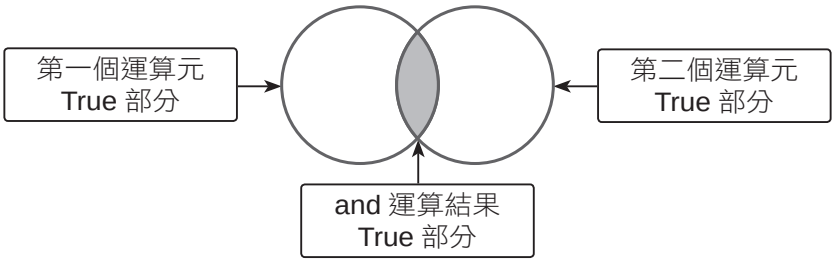
2.4.3 邏輯運算子

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。

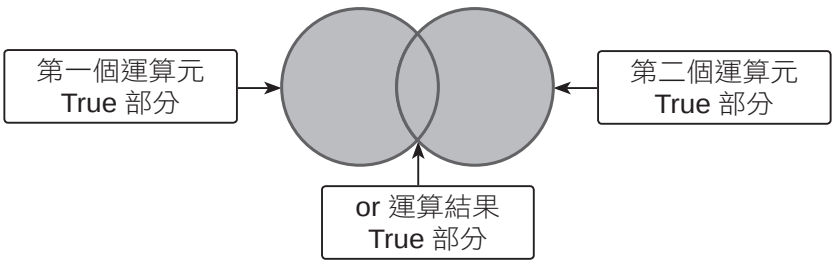
運算子	意義	範例	範例結果
not	傳回與原來比較結果相反的值，即比較結果是 True，就傳回 False；比較結果是 False，就傳回 True。	not(3>5) not(5>3)	True False
and	只有兩個運算元的比較結果都是 True 時，才傳回 True，其餘情況皆傳回 False。	(5>3) and (9>6) (5>3) and (9<6) (5<3) and (9>6) (5<3) and (9<6)	True False False False

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：

```
i += 3   #即 i = i + 3
i -= 3   #即 i = i – 3
```

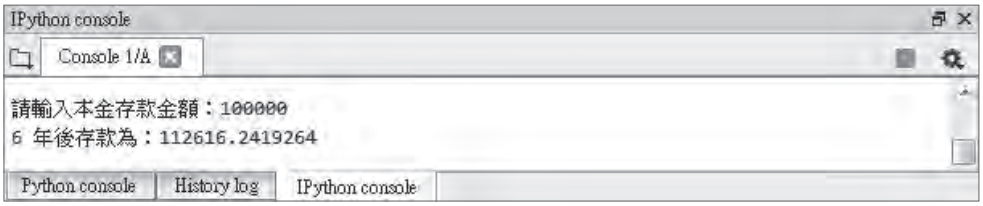
複合指定運算子同時做了「執行運算」及「指定」兩件工件。

下表是以i 變數值為10 來計算範例結果：

運算子	意義	範例	範例結果
+=	相加後再指定給原變數	i += 5	15
-=	相減後再指定給原變數	i -= 5	5
*=	相乘後再指定給原變	i *= 5	50
/=	相除後再指定給原變數	i /= 5	2
%=	相除得到餘數後再指定給原變數	i %= 5	0
//=	相除得到整除商數後再指定給原變數	i //= 5	2
**=	做指數運算後再指定給原變數	i **= 3	1000

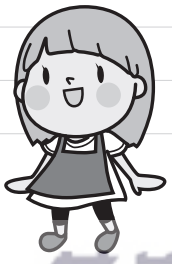
範例實作：計算複利本金

複利公式為：本金*(1+利率)^{年}，目前存款年利率為 2%，以複利計算。阿輝在銀行有一筆存款，設計程式讓阿輝輸入存款本金後，以複合指定運算子計算 6 年後的本金。(<complex.py>)



程式碼：ch02\complex.py

```
1 deposit = int(input("請輸入本金存款金額:"))
2 times = 1.02 ** 6
3 deposit *= times
4 print("6 年後存款為:" + str(deposit))
```



Chapter

03

判斷式

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，縮排為 1 個 Tab 鍵或 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

Tab 鍵或 4 個空白鍵

Python 最常用的縮排方式是只用空白鍵，第二常用的方式是只用 Tab 鍵。

每一層的縮排一般使用 4 個空白鍵，在相當舊的程式碼中，為了一致也可以繼續使用 Tab 鍵。

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。

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，然後再依決策結果從事不同的事件，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：

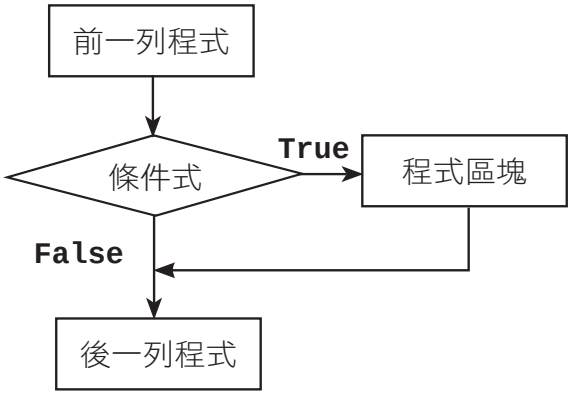
```
if (條件式):
    程式區塊
```

「(條件式)」的括號可移除，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：

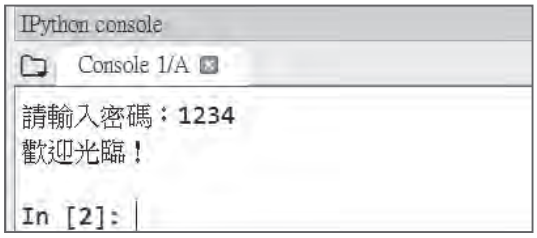
```
if (條件式):  程式碼
```

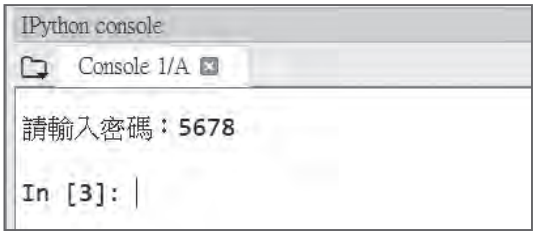
以下是單向判斷式流程控制的流程圖：



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）

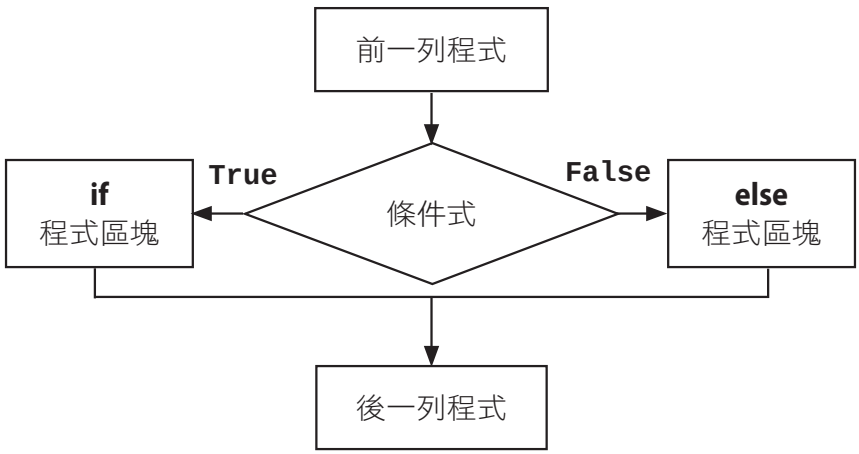
感覺上「if」語法並不完整，因為如果條件式成立就執行程式區塊內的內容，如果條件式不成立也應該做某些事來告知使用者。例如密碼驗證時，若密碼錯誤應顯示訊息告知使用者，此時就可使用「if…else…」雙向判斷式。

「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
```

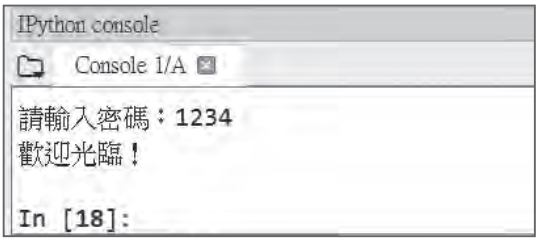
當條件式為 True 時，會執行 if 後的程式區塊一；當條件式為 False 時，會執行 else 後的程式區塊二，程式區塊中可以是一列或多列程式碼，如果程式區塊中的程式碼只有一列，可以合併為一列。

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

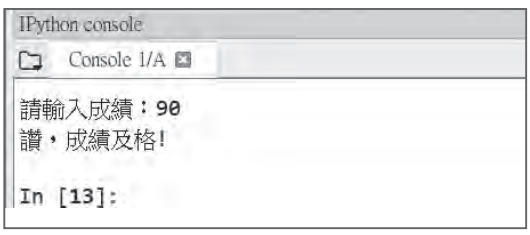
```
1  pw = input("請輸入密碼:")
2 if(pw=="1234"):
3     print("歡迎光臨!")
4 else:
5     print("密碼錯誤!")
```

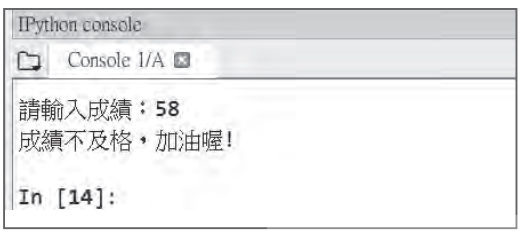
程式說明

- 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>)





Chapter

07

函式與套件

7.4.3 產生整數或浮點數的亂數函式

randint 函式

randint 函式的功能是由指定範圍產生一個整數亂數，語法為：

```
亂數套件別名 .randint( 起始值 , 終止值 )
```

執行後會產生一個在起始值（含）和終止值（含）之間的整數亂數，注意產生的亂數可能是起始值或終止值，例如：

```
import random as r
for i in range(0,5):   # 執行 5 次，產生 5 個整數亂數
    print(r.randint(1,10), end=",")   #9,8,1,10,4,
```

上例中，1 與 10 都是可能產生的亂數。

randrange 函式

randrange 函式的功能與 randint 雷同，也是產生一個整數亂數，只是其多了一個遞增值，語法為：

```
亂數套件別名 .randrange( 起始值 , 終止值 [, 遞增值 ])
```

執行後會產生一個在起始值（含）和終止值（不含）之間，且每次增加遞增值的整數亂數，遞增值值可有可無，遞增值的預設值為 1。特別注意產生的亂數可能是起始值，但不包含終止值，例如：

```
import random as r
for i in range(0,5):   # 執行 5 次，產生 5 個整數亂數
    print(r.randrange(0,12,2), end=",")   #8,0,10,6,6,
```

由於從 0 開始，每次遞增 2，且不包含 12（終止值），所以產生的亂數是「0、2、4、6、8、10」六個數其中之一。

random 函式

random 函式的功能是產生一個 0 到 1 之間的浮點數亂數，語法為：

```
亂數套件別名 .random()
```

例如：

```
import random as r
print(r.random())  #0.5236730771512399
```

uniform 函式

uniform 函式的功能是產生一個指定範圍的浮點數亂數，語法為：

```
亂數套件別名 .uniform( 起始值 , 終止值 )
```

執行後會產生一個在起始值和終止值之間的整數亂數，例如：

```
import random as r
print(r.uniform(3,10))  #6.063374013178429
```

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>)

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：

```
亂數套件別名 .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 as r
for i in range(0,5):   # 執行 5 次，產生 5 個整數亂數
    print(r.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 as r
for i in range(0,5):   # 執行 5 次，產生 5 個整數亂數
    print(r.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：

```
亂數套件別名 .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 as r
print(r.sample("abcdefg",3))   #['f', 'b', 'g']
print(r.sample([1,2,3,4,5,6,7],3))   #[3, 1, 4]
```

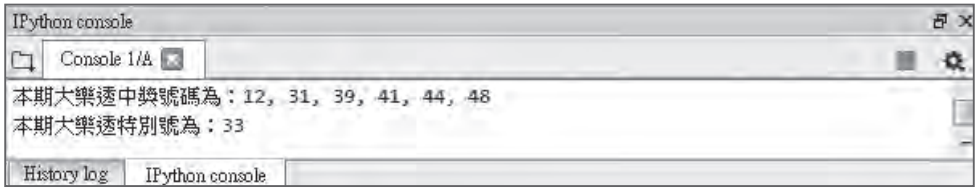
需注意「數量」參數的值不能大於字串長度或串列元素個數，也不能是負數，否則執行時會產生錯誤，例如：

```
import random as r
print(r.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 as r
 2
 3 list1 = r.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(0,6):
 8     if i == 5:    print(str(list1[i]))
 9     else:    print(str(list1[i]), end=", ")
10 print("本期大樂透特別號為：" + str(special))
```

Appendix

A

MTA Python 程式設計
核心能力國際認證模擬試題

MTA 98-381
Introduction to Programming Using Python

4. 你設計了一個電影票收費的函式，票價的規則如下：
   - 5 歲以下 = 免費入場
   - 5 歲及以上的學生 = 60 元
   - 5 歲到 17 歲但不是學生 = 120 元
   - 17 歲以上但不是學生 = 180 元

   你要如何完成這段程式碼？回答時，在回答區選擇適當的程式碼片段。

```
def ticket_fee(age, school):
    fee = 0
    ___(1)___
        fee = 60
    ___(2)___
        ___(3)___
            fee = 120
        else:
            fee = 180
    return fee
```

```
( )(1)  A. if age >= 5 and school == True:
        B. if age >= 5 and school == False:
        C. if age <= 17
( )(2)  A. if age >= 5 and school == True:
        B. if age >= 5 and school == False:
        C. if age <= 17
( )(3)  A. if age >= 5 and school == True:
        B. if age >= 5 and school == False:
        C. if age <= 17
```

5. 你設計一個 Python 程式來檢查使用者輸入的數字是是 1 位數，2 位數還是 2 位數以上，其中規定輸入的值必須是正整數。你要如何完成這段程式碼？

```
num = int(input("請輸入一個正整數： "))
digits = "0"
if num > 0:
    ___(1)___
        digits = "1"
```