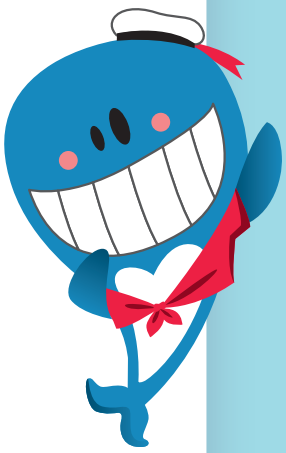
主題2
程式設計

把想法付諸實現吧！當我們腦海裡有好的「想法」卻不付諸行動的話，它只會是一個想法，不能化為實際的價值。學了資料結構和演算法後，我們接著就要開始撰寫電腦程式。

我們不必人人都成為程式設計師，但是實作電腦程式可以培養解決問題的思維，並運用這種思考方式，化解生活的難題。我們可以把生活中的問題看作程式要解決的目標，解決問題的步驟與方法則視為演算法，而透過程式設計的練習，便可以增強我們解決問題的能力。

Chapter 03　程式設計的概念

- 3-1　程式是什麼
- 3-2　程式中常見的結構
- 3-3　程式設計工具



Chapter 04　基礎程式設計實作

- 4-1　陣列資料結構實作
- 4-2　結構化程式實作
- 4-3　模組化程式實作
- 4-4　遞迴結構實作

Chapter 05　重要演算法實作

- 5-1　排序
- 5-2　搜尋

CHAPTER 03

程式設計的概念

我們撰寫程式就是下指令給電腦做事情，電腦會確實遵照每一個指令執行。那麼，設計程式有沒有什麼訣竅呢？寫文章講究起承轉合，撰寫程式雖然沒有，但也不能隨意揮灑，本章要跟大家介紹如何撰寫具有「結構性」的程式。

3-1 程式是什麼

程式（Program）包含了一連串的電腦指令，能用來指示電腦該做什麼樣的操作與運算。還記得在第2章提到的演算法嗎？如果我們能將演算法轉換成程式，電腦便能依照我們的演算法運行來解決問題（圖3-1）。

圖3-1　程式設計的概念

程式的編譯與直譯

透過程式，我們能讓電腦遵循人類的指揮來完成任務。然而，電腦被設計成只看得懂0與1這種二進位（Binary）資料，因此，最早的程式也必須使用這種0與1的表示法來撰寫，我們稱之爲**機器語言**（Machine Language），也就是給機器看的語言。

IT加油站

早期程式設計師會利用「打孔卡」（Punched Card）來撰寫程式，利用打洞與不打洞代表0或1。例如：打洞代表1，沒打洞代表0。圖3-2的打孔機就是用這種方式寫程式的範例，不僅耗時也非常費工，要讀懂一「張」寫好的程式，更是一件困難的事。

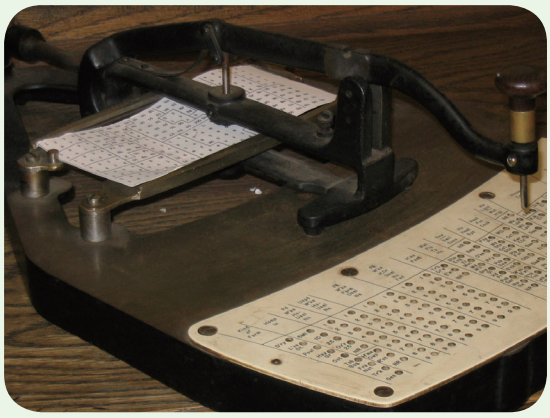


圖3-2　1880年代的打孔機
（圖片來源：維基百科）

以人類的角度來看，使用0、1來撰寫程式實在太困難了，也難以找出程式的錯誤，進而導致開發效率低落。因此，電腦科學家開發了**組合語言**（Assembly Language），使用容易記憶的英文指令來代換難懂的0與1。之後，更接近人類語言的**高階語言**（High Level Language）被開發出來，它有許多強大的功能，使得程式開發的速度大幅提升（圖3-3）。

雖然人類能夠輕易的利用高階語言來撰寫程式，這些程式仍然需要轉換成電腦看得懂的機器語言才行。組合語言與高階語言撰寫的程式碼，分別需要經過**組譯**（Assemble）與**編譯**（Compile）的過程，轉換成電腦能執行的程式檔案。

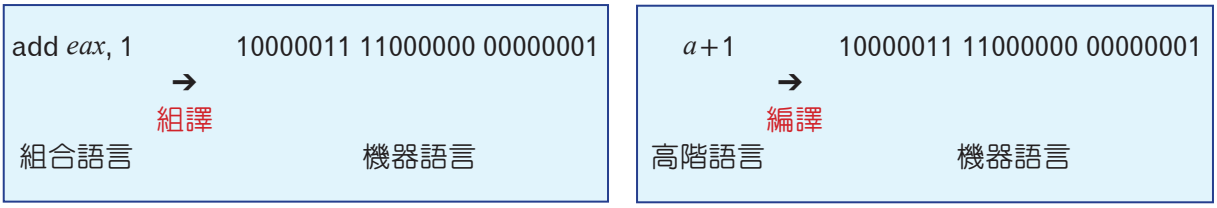


圖3-3　利用三種語言表示加法指令

高階語言指令「$a+1$」與數學課上見到的方程式非常相似，雖然組合語言指令「add *eax*, 1」沒有高階語言直覺，卻也可以從指令中的「add」看出其功能爲加法；而機器語言的指令「10000011 11000000 00000001」只有0與1，難以從指令本身看出其意義。

那麼，有沒有一種不需要編譯即可執行的高階語言呢？那就是直譯語言，它能透過**直譯器**（Interpreter），將程式碼一行一行、即時地轉換成機器語言讓電腦執行，而不必在每次程式修改後都重新將程式再編譯一次。直譯語言雖然方便，卻因爲需要在執行的當下，將程式碼轉換爲電腦看得懂的機器語言，導致執行的效率比事先編譯過的程式差（圖3-4）。

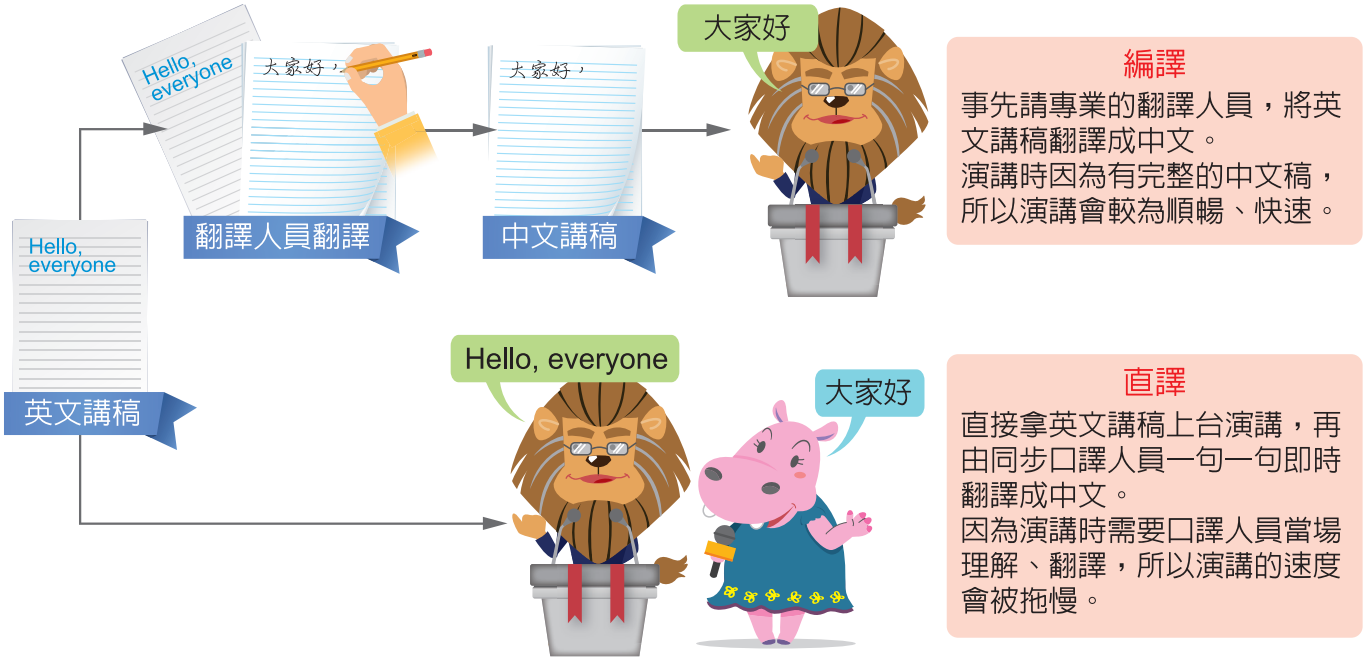


圖3-4　編譯與直譯

程式的語法與語意

撰寫程式時，可能出現的錯誤可分爲兩種類型：**語法錯誤**與**語意錯誤**。

語法錯誤

就是電腦看不懂我們的程式，例如：規定要寫$a=b+c$，我們卻寫成$a=b$加c，而電腦看不懂這裡的「加」。

語意錯誤

就是我們希望把b和c「加」起來，但卻不小心打成$a=b\times c$，這和我們眞正想要的結果不同；然而，電腦不知道我們實際在想什麼，所以它不認爲這是錯的，但結果實際上不合我們的預期。

3-2 程式中常見的結構

3-2-1 基本結構

設計程式時，有許多常見的結構可以使用；利用這些結構來設計程式，可以使程式容易理解與維護，這種方式稱爲**結構化程式設計**（Structured Programming）。

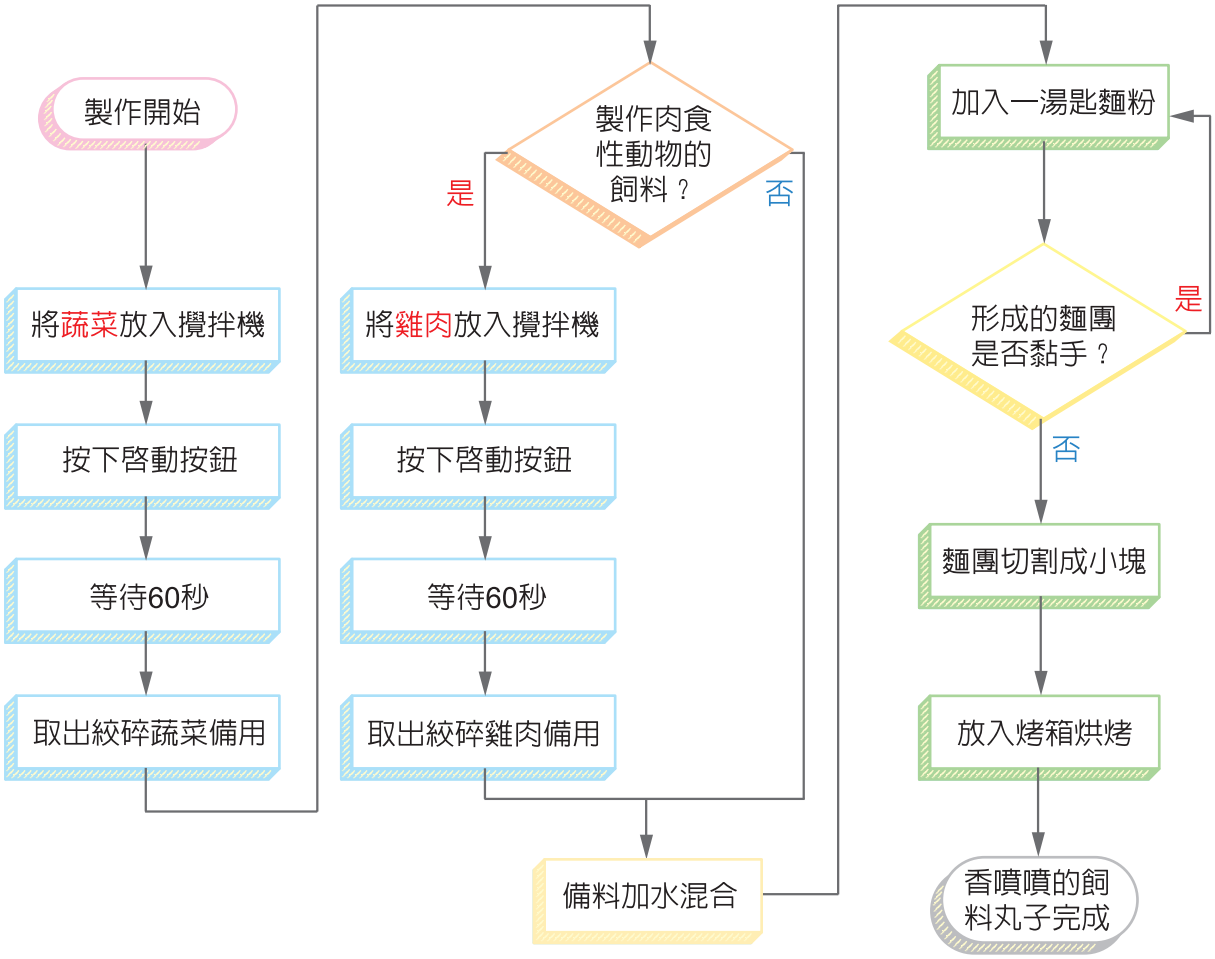


圖3-5 動物飼料丸子的細部製作流程圖

常見的程式結構包含**循序結構**、**選擇結構**和**重複結構**，接下來我們將配合「動物飼料丸子的細部製作流程圖」（圖3-5），一一介紹它們。

循序結構

循序（Sequence）結構是指電腦依循程式碼的**順序**，一行一行往下執行，而這也是電腦最基本的運作方式（圖3-6左）。

在動物飼料丸子的細部製作流程圖（圖3-5）中，前幾個步驟便是循序結構的例子（圖3-6右），製作飼料的人會依循流程圖的箭頭指示，先執行第一個步驟，接著再執行第二個步驟。

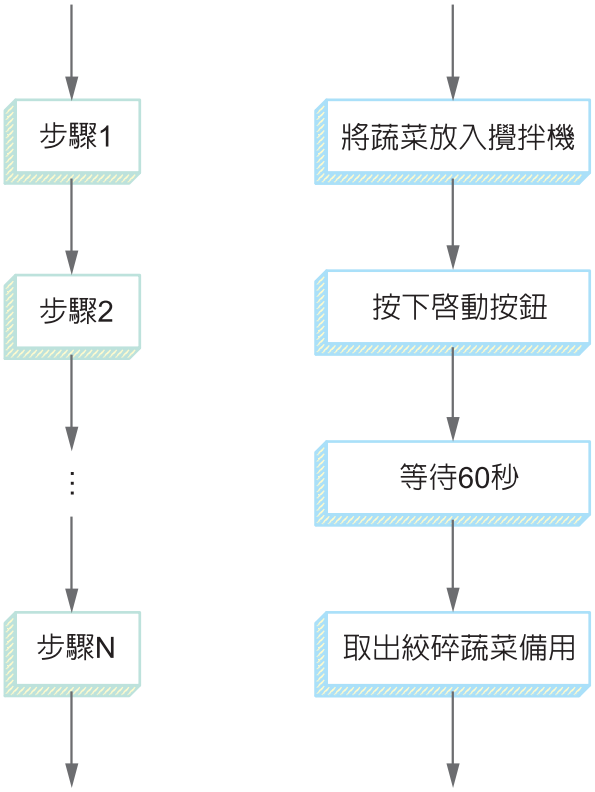


圖3-6　循序結構（左）及例子（右）

選擇結構

選擇（Selection）結構是指電腦根據「**條件**」，決定接下來要執行哪一種程式碼（圖3-7）。

動物飼料丸子製作流程圖中，製作者會根據飼料是不是給肉食性動物吃的，來決定是否要加入雞肉，這個條件判斷便是透過選擇結構來達成。

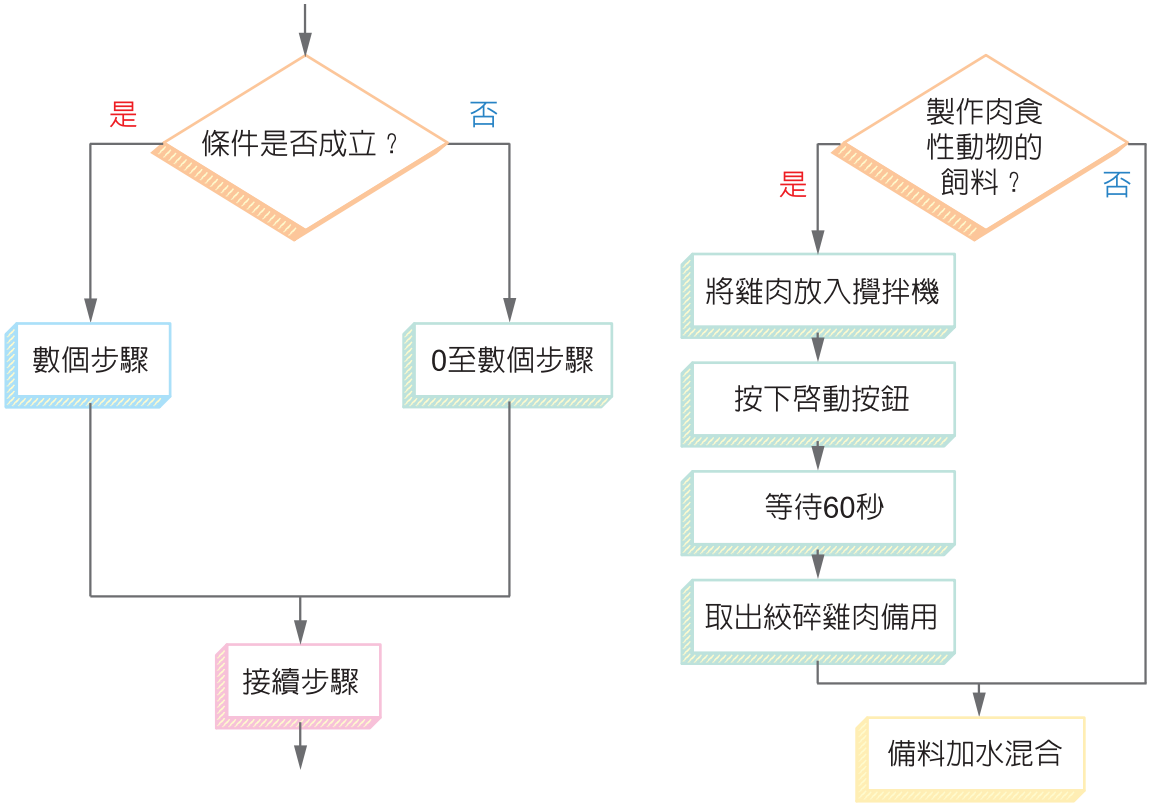


圖3-7　選擇結構（左）及例子（右）

重複結構

重複（Iteration）結構是指電腦重複執行同一段程式碼，直到停止的條件達成爲止（圖3-8）。

在飼料製作流程中，我們必須不斷地加入麵粉（動作），直到形成的麵糰不黏手（條件）爲止。這種不斷重複相同動作，直到條件滿足爲止的操作，就可以利用重複結構來達成。

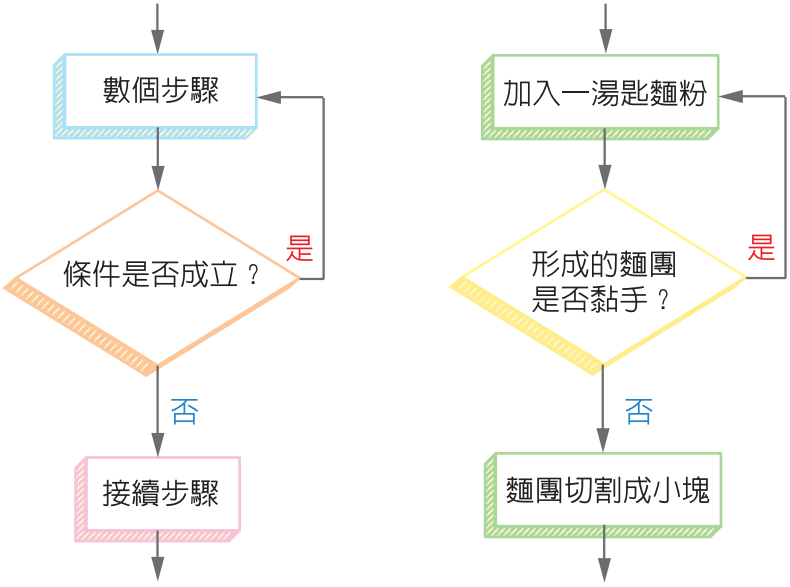


圖3-8　重複結構（左）及例子（右）

整合應用：結構化的氣泡排序演算法

撰寫程式時，循序、選擇和重複結構往往會搭配著使用，例如：氣泡排序法（圖3-9）同時使用這三種結構。

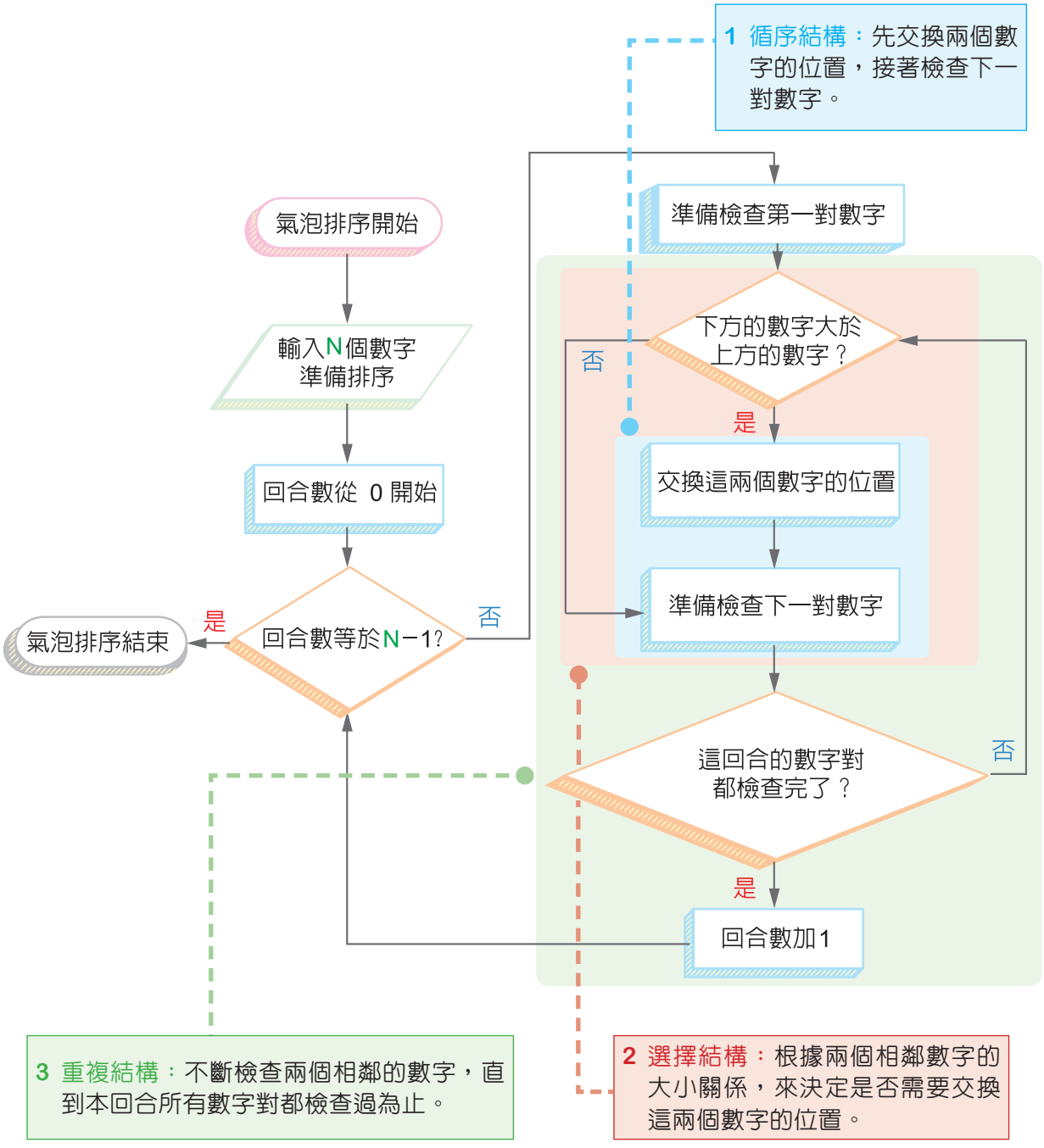


圖3-9　氣泡排序法中有循序、選擇及重複結構

3-2-2 模組化與函數的概念

模組化是指以功能爲單位，將一個大程式分割成許多功能不同的小模組（Module），各個模組可以獨立開發與管理。這種作法可以使得程式容易重複使用、容易測試、容易閱讀與方便管理，在多人共同開發程式時，也能使得分工更加容易。

在撰寫程式時，將重複出現許多次的程式碼獨立出來，寫成**函數**（Function）；或依照功能切割程式碼，使其方便重複使用或維護，這些都是模組化的概念。

有了函數，後續撰寫程式時，就可以透過呼叫函數的方式來使用函數內的功能，而不用將該功能重寫，免去重複且冗餘的程式碼，也增加程式的可重用性。爲了方便說明，我們一樣使用飼料丸子製作流程的例子來解說。

仔細觀察製作流程圖，可以發現許多重複的步驟，像是「絞碎蔬菜」與「絞碎雞肉」，它們都是將食材放進攪拌機絞碎，再取出備用（圖3-10）。

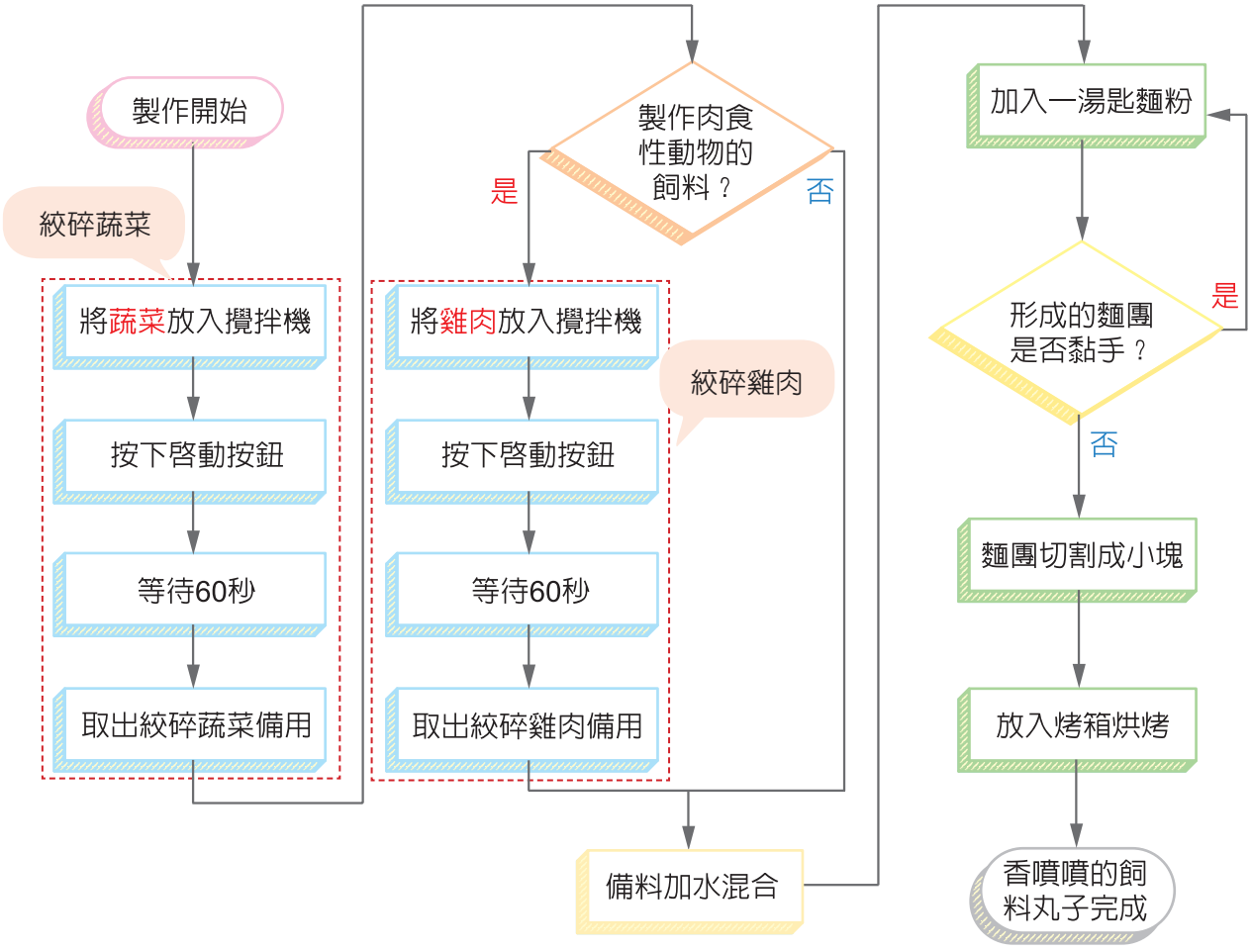


圖3-10　可以抽取成函數的步驟

因此，我們可以撰寫一個函數「將（目標食材）絞碎成備料」，負責將輸入函數的物體（也就是目標食材）絞碎成備料（如表3-1）。

表3-1　絞碎食材成為備料的函數

函數：將（目標食材）絞碎成備料
1. 將目標食材放入攪拌機
2. 按下啓動按鈕
3. 等待60秒
4. 取出絞碎的目標食材備用

數學上，我們會透過$y=f(x)=ax+b$這種表示法，用$f(x)$代表將x代入到$ax+b$的過程，而y則是輸出的結果。類似的，將物體輸入到函數內部，我們也使用相同的方法來表示。我們可以發現，這個函數需要一個輸入（變數 x），也就是需要絞碎的目標食材，並且經過一連串的步驟（函數 $f(x)$），產生一個輸出（變數 y），也就是已經絞碎的目標食材。

因此，函數就是一種模組化的方式，它會接受輸入，並在函數內部進行一系列的運算，最後產生輸出。我們將重複的操作包裝成一個獨立的函數，使得飼料製作流程圖更加單純、清楚，也容易閱讀（圖3-11）。

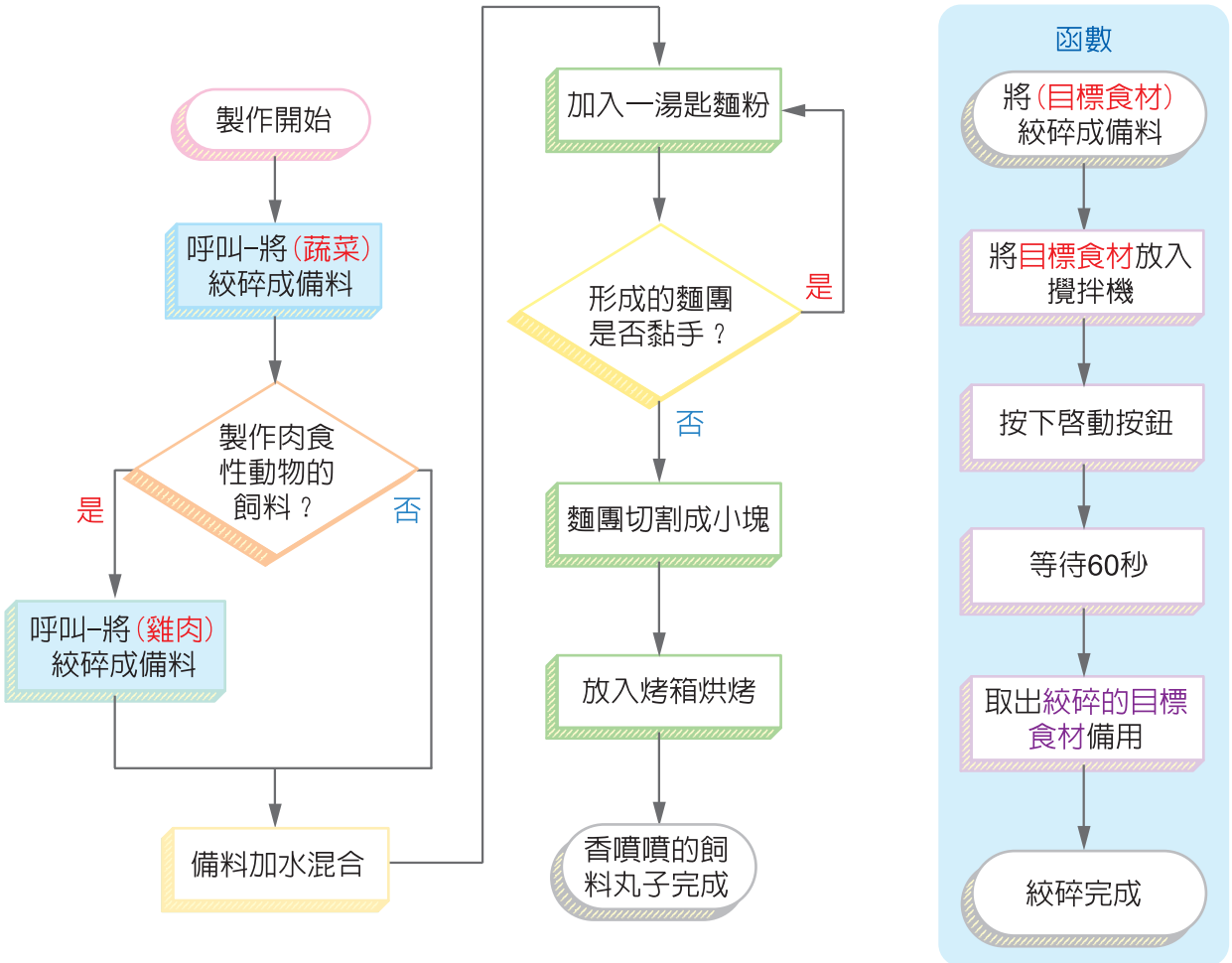


圖3-11　抽取出函數後的製作流程圖

3-3 程式設計工具

俗話說：「工欲善其事，必先利其器」，常見的程式語言工具有C、C++、Java、Visual Basic及Python等。Python是一個容易入門的程式設計語言，程式碼容易閱讀，非常適合初學者學習程式設計的概念。Python的官方網站（http://www.python.org）可以下載[1]最新版的Python開發環境。

TIP Python屬於直譯的程式語言，預設副檔名為「.py」。

近年流行不必安裝，直接使用雲端上的「互動式線上程式開發平台」，如用Google公司的Google Colaboratory（簡稱爲Colab）[1]編寫程式。我們可在 Google 雲端硬碟上外掛 Colab(圖 3-12)。

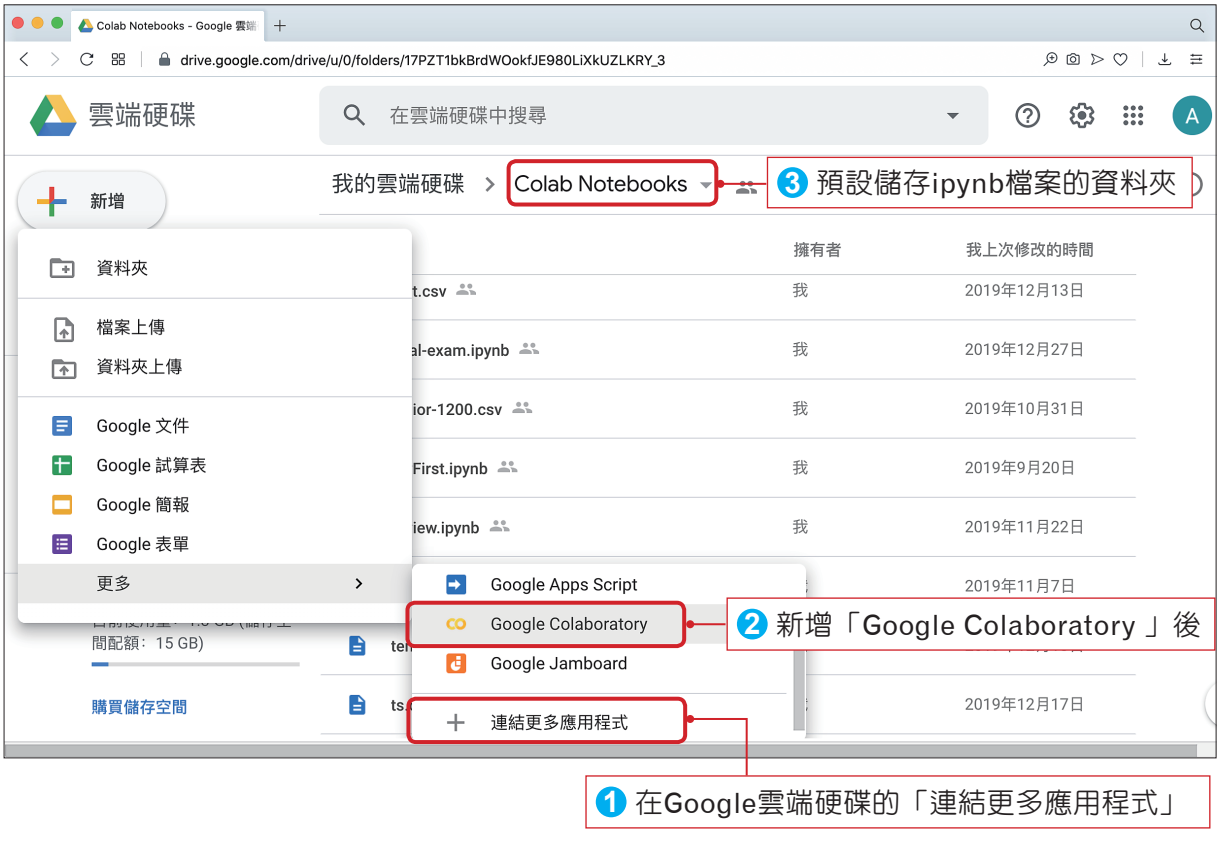


圖 3-12 Google 雲端硬碟中外掛 Colab

1 https://colab.research.google.com。

Colab是一個免費的 Jupyter Notebook(筆記本)環境(圖 3-13)，我們撰寫的程式碼可以存成「.py」的程式檔，也可存成「.ipynb」的筆記本，互動式編寫及執行程式都在一個 Web 網頁上完成，筆記本則自動儲存在 Google 雲端硬碟上。

圖3-13　Colab環境介紹

首先，讓我們來打個招呼吧！**print( )**可以把括號內的資料印出來（顯示在螢幕上），這個函數在接下來的過程中將會不斷使用到。

程式碼（檔案：Ch3-1.py）

```
1 print('哈囉  資訊科技!')
```

執行結果

```
哈囉  資訊科技!
```

IT加油站

以**print( )**函數輸出字串時，我們可以輸入

```
print('哈囉  資訊科技!')或
print("哈囉  資訊科技!")
```

也就是以雙引號（" "）或單引號（' '）把字串包圍起來。

變數的概念

變數就與數學課中x,y,z等代數很相似，對y=2x這個方程式來說，如果我們讓x=10，就會得到y=20；讓x=20，就會得到y=40。因爲x的數值可以不停的變化，我們就稱之爲**變數**。我們可以把變數想像成一個容器（如圖3-14中的箱子），裡面只能裝一個數值，但是我們可以替換這個容器內部的數值。

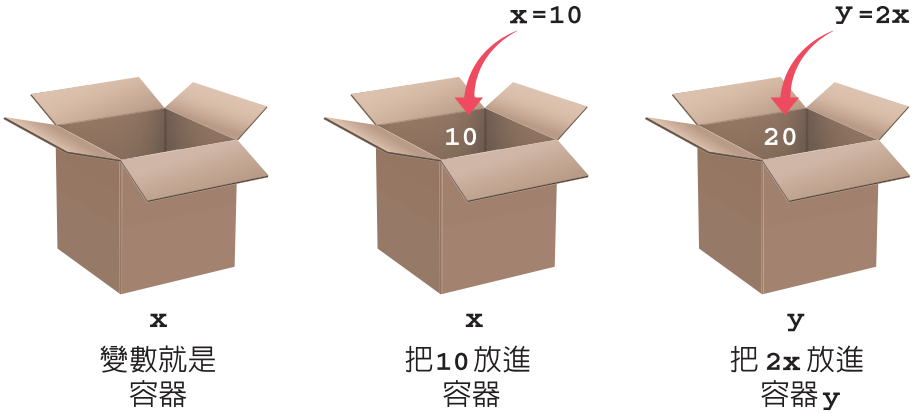


圖3-14　變數就像是個容器

我們可以透過x=10這種方式來建立一個變數（箱子），並把10存入變數x（箱子）中。

程式碼（檔案：Ch3-2.py）

```
1  x = 10
2  y = 20
3  x = 30
4  print(x, y)
```

執行結果

```
30 20
```

IT加油站

以print( )函數輸出變數時，變數不必再加引號，如print(x)或print(x,y)。print('x')會輸出字母x，而不是變數x的值。

變數的型態[2]

變數除了可以儲存數值以外，也可以儲存字串、小數等型態的資料，如表3-2所列。

表3-2　變數的型態

型態	描述	範例
整數	數學意義上的整數	`x = 10`
小數	數學意義上的小數	`pi = 3.1415`
字串	由雙引號（" "）或單引號（' '）包圍起來的一串字稱為字串	`s = "Hello, Computer."`
布林	用來表示「對」或「錯」的概念，例如： `10 > 5`是`True`；`10 < 5`是`False`	`apple_is_red = True` `apple_is_blue = False`

① 數學中的整數與小數和程式變數中的整數與小數有些許差異：數學的整數與小數範圍可以無限延伸，也就是數值可以無限大或無限小；但是程式的變數能夠儲存的數值範圍是有限的（受限於變數的儲存空間）。

② 字串變數：以`input( )`取得鍵盤輸入的字串，並把字串存在變數 `n` 中，再用 `print( )` 輸出在螢幕上。


```
name = input('請輸入你的名字:')
print('Hello,',n)
```


```
請輸入你的名字:Sandy
Hello, Sandy
```

變數的四則運算

現實中，我們可以將數字透過＋、－、×、÷等運算符號進行算術運算（表3-3）；在程式語言中，我們也可以做到相同的事情，但是符號有些不同。

表3-3　算術運算

優先順序	運算符號	功能	範例	結果
高	`**`	次方	`5**3`	`125`
↕	`-`	負數		

↕	`*`
	`/`
	`%`
		乘
		除
		除法求餘數
			`10*3`
			`10/3`
			`10%3`
				`30`
				`3.33…`
				`1`

低	`+ -`	加減	`3+5`	`8`

2　許多程式語言中，變數在執行前需要宣告資料型態，且執行中不能改變變數的資料型態。Python語言則沒有這樣的限制。

程式碼（檔案：Ch3-3.py）

```
1  x = 10 + 5   # 加：x變成15
2  y = 20 - x   # 減：y變成5
3  z = x * y    # 乘：z變成75
4  q = x / y    # 除：q變成3.0
5  print(x, y, z, q)
```

執行結果

```
15 5 75 3.0
```

IT加油站

註解

在Python的程式語法規範裡面，一行程式碼中，「#」之後的文句便是註解。程式碼的註解內可以寫任意的文句，而不會被電腦執行，通常會用口語化的敘述，來標示程式碼的用途，讓其他人可以快速理解程式。

變數的比較

變數除了四則運算外，還可以互相比較大小。常用的比較運算符號如表3-4所列。

表3-4　比較運算符號

運算	描述	範例	比較結果（假設x = 5, y = 3）
>	大於	x > y	True
>=	大於或等於	x >= y	True
<	小於	x < y	False
<=	小於或等於	x <= y	False
==	等於	x == y	False
!=	不等於	x != y	True

程式碼（檔案：Ch3-4.py）

```
1  x = 10
2  y = 20
3  print(x > y)
4  print(x < y)
5  print(x == y)
```

執行結果

```
False
True
False
```

IT加油站

在Python等程式語言中，`x = y`與`x == y`是不一樣的。

`x = y`指的是把變數`y`的值「給」`x`（右邊「賦值」給左邊），

`x == y`指判斷`x`與`y`是否「相等」。

整合應用：簡單的程式——交換兩個變數

還記得在第2章氣泡排序的演算法中，有一個關鍵的步驟是交換兩個數字嗎？我們將會在這一小節實作這個簡單的小功能。

首先，讓我們先建立兩個儲存不同數值的變數`x`和`y`。憑直覺，我們最先想到的方法可能是互相把兩個變數指定給對方。

程式碼（檔案：Ch3-5.py）

```
1  x = 10          # 交換前的x為10
2  y = 20          # 交換前的y為20
3  x = y           # 把y值給x
4  y = x           # 把x值給y
5  print(x, y)
```

執行結果

```
20 20
```

結果，`x`和`y`都變成`20`了！爲什麼會這樣呢？如同前面所說的，變數就像只能裝「一個」數值的箱子，如果要把`y`的數值放進`x`中，我們必須丟棄`x`原本儲存的數值，才能把`y`的數值放進去。所以在第3行中，我們將`y`的數值`20`放入`x`後，第4行再將`x`此時的數值`20`放回`y`，這樣並沒有辦法達到交換兩個變數的效果（圖3-15）。

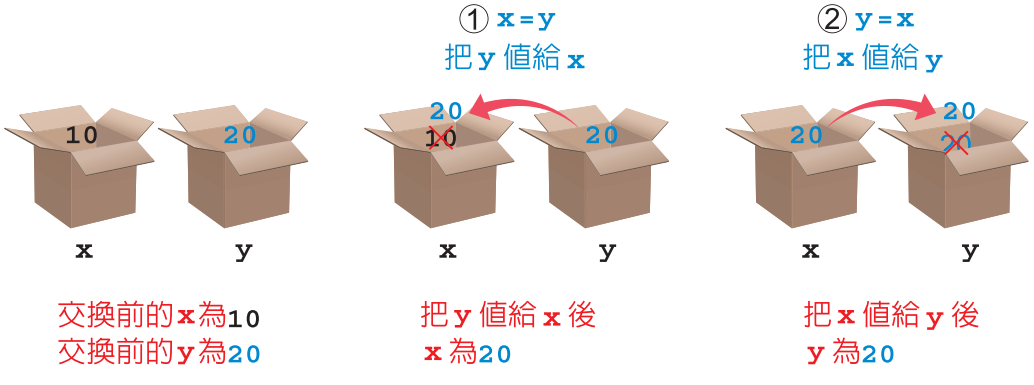


圖3-15　錯誤的變數交換

爲了記住x原本的數值，必須新增一個變數temp，用來暫時儲存x原本的值（圖3-16）。因此，我們先將x的數值放進temp，接著才將y的數值放進x，最後，把原本x的值（現在存在temp中）放進y裡，就可達成交換兩個變數的目的。

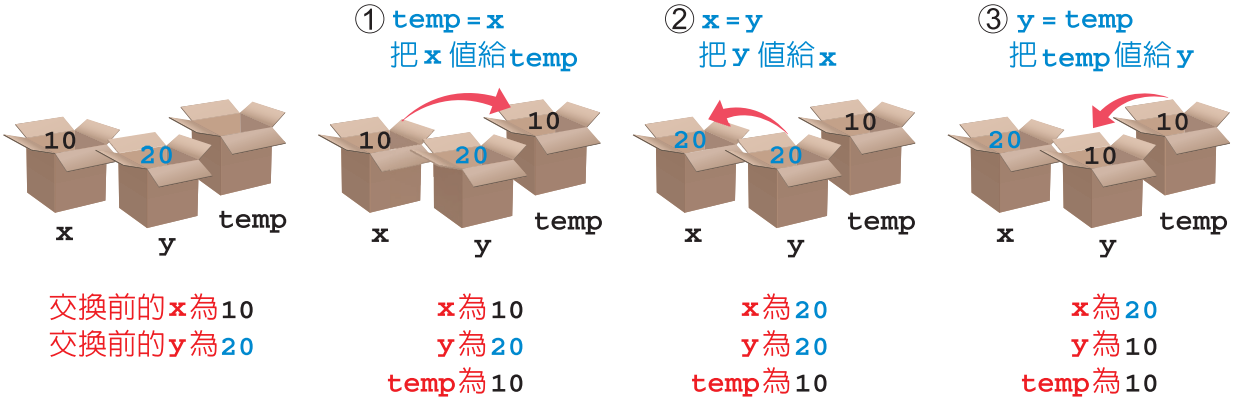


圖3-16　正確的變數交換

程式碼（檔案：Ch3-6.py）

```
1  x = 10          # 交換前的x為10
2  y = 20          # 交換前的y為20
3  temp = x        # 把x值給temp
4  x = y           # 把y值給x
5  y = temp        # 把temp值給y
6  print(x, y)
```

執行結果

```
20 10
```

實力挑戰

選擇題

( ) 1. 請問程式的基本結構不包含下列哪一個？

(A)循序結構 (B)選擇結構 (C)量子結構 (D)重複結構。

( ) 2. 請問下列何者是選擇結構的用途？

(A)重複執行數個步驟 (B)依照順序一個步驟一個步驟執行
(C)隨機挑選步驟來執行 (D)依照條件成立與否，決定要執行的步驟。

( ) 3. 下列何者是程式中用來存放變動數值的概念？

(A)變數 (B)常數 (C)定義數 (D)無理數。

( ) 4. 在設計電腦程式時，使用函數的優點並不包含？

(A)增加可讀性 (B)避免重複的程式碼
(C)使程式碼容易維護 (D)加快執行速度。

( ) 5. 相對於低階語言，下列何者不是高階語言的特性？

(A)可讀性較高 (B)使用者較易學習
(C)較容易除錯 (D)程式執行速度較快又較有效率。

( ) 6. 下列敘述何者最正確？

(A)直譯語言不需要編譯便可執行
(B)組合語言可以直接輸入電腦中執行，不需要額外的轉換
(C)機器語言的指令由英文字母組成
(D)機器語言比高階語言容易閱讀與理解，所以有利於機器執行。

多元練習

1. 寫一支程式，先指定a,b,c,d的值，再計算$\frac{a-b}{c+d}$的值。請先畫出流程圖，再撰寫程式，最後執行、取得結果。

   例如：$a=50, b=30, c=6, d=2$，則應輸出2.5。

2. 寫一支程式，先指定a,b的值，再計算$\sqrt{a^2+b^2}$的值。請先畫出流程圖，再撰寫程式，最後執行、取得結果。

   例如：$a=4, b=3$，則應輸出 c 的值為 5。

   提示：「次方」運算符號為「**」

   例如：
   $a=4$
   $b=a$ ** 2
   $c=a$ ** 0.5　　則b 的值為16，c 的值為 2。