

第 1 章

一個程式就是一連串的命令

如果您要學習怎麼開車，您必須學著處理許多事情。您必須了解離合器、油門與煞車。程式設計也需要您把許多事情牢記在心，否則您的程式將會當掉。如同您在還沒有開始學習開車前就要先熟悉車子內部一樣，馬上使用 Xcode 來學習程式設計並不會獲得任何好處。為了不要讓您被擊潰，我們把實際的程式設計環境留到稍後的章節。首先，我們要利用一些您非常熟悉的基本數學來讓您習慣 Objective-C 程式碼。

小學時您必須做些計算，把答案填入框框中：

$$2 + 6 = \square$$

$$\square = 3 * 4 \text{ (星號 * 在電腦鍵盤上是用來表示乘號)}$$

中學時，框框不流行了，取而代之的是被稱為 x 與 y 的變數（以及一個很炫的名稱，代數）。回首過往，您可能會奇怪為什麼人們會對這麼微小的改變感到恐懼。

$$2 + 6 = x$$

$$y = 3 * 4$$

Objective-C 也使用變數。變數只是一個用來方便代表某個特定資料的名字罷了，例如一個數字。以下是一個 Objective-C 述句（statement），亦即一程式碼，用來給一個變數指定一個特定的值。

```
[1]
```

```
x = 4;
```

變數 x 被指定了 4 這個值。您應該注意到了在述句結尾的地方有個分號，而那是因為每句述句都需要以分號作結。為什麼？呃...範例 [1] 的程式片段在您眼中可能很普通，但是電腦對它卻是完全不知從何下手。因此我們需要一個特別的程式，稱為編譯器（compiler），來把您輸入的文字轉換成一連串您的 Mac 所能理解的 0 與 1。對於編譯器而言，要閱讀與了解人類輸入的文字是非常困難的，因此您需要給它一些線索，例如一個述句是怎麼作結尾的。而那就是您使用分號這個標記的目的。

如果您在程式碼中忘了分號，程式碼就無法編譯，那代表它無法轉換成您的 Mac 可以執行的程式。不過別太擔心，如果編譯器沒辦法編譯您的程式碼，它會向您抱怨，而就如同我們在未來的章節會見到的，它會試著協助您找出錯誤的地方。

雖然變數名本身對於編譯器而言並沒有特殊的意義，但是具有意義的變數名可以讓程式更容易閱讀，也更容易理解。這在您需要在程式碼中追蹤一個臭蟲的時候是一大利多。

程式裡面的錯誤傳統上稱為臭蟲 (bug)。找出並修正他們被稱為除蟲 (debugging)。

因此，在真正的程式碼中我們會避免使用像 `x` 這樣沒有意義的變數名稱。例如，用來表示圖片寬度的變數名稱可以稱為 `pictureWidth` [2]。

[2]

```
pictureWidth = 8;
```

忘了分號就會讓編譯器引起軒然大波，您可以由此體會到程式設計是非常注重細節的。其中一個需要注意的細節就是程式碼的大小寫是不同的。亦即，您有沒有使用大寫是有差異的。變數名稱 `pictureWidth` 與 `pictureWIDTH`、或者 `PictureWidth` 都不相同。根據一般慣例，我結合幾個字來作為變數名，其中第一個字全部小寫，其他每個字都以大寫字母開頭，就如同您在範例 [2] 中所見。依循這個規則，我大幅地降低了設計程式時因為大小寫而出錯的可能。

請記得一個變數名總是包含一個單字（或者至少一個字母）。

雖然您有充分的自由來命名變數，仍然有些規則需要遵守。儘管我可以把它們全部列出，但是那將會非常無趣。一條必須遵守的主要規則是，您的變數名稱不能是 Objective-C 的保留字（亦即在 Objective-C 中有特殊意義的字）。藉由組合幾個單字來作為變數名，例如 `pictureWidth`，您總是安全的。為了保持名字的可讀性，建議在變數名中使用大寫字母。如果遵守這些原則，您程式中的臭蟲將可望減少。

如果您堅持要再學習一些規則，請讀完這個段落。除了字母，使用數字是被允許的，但是變數名稱不允許使用數字開頭。另一個也允許使用的是字元：“`_`”。

以下是一些變數名的例子。

正確的變數名：`door8k`，`do8or`，或 `do_or`

不允許的變數名：`door 8`（包含了一個空白），`8door`（開頭是數字）

不建議的變數名：`Door8`（開頭是大寫字母）

現在我們知道要如何給變數一個值，我們可以進行計算。讓我們看看用來計算圖片面積的程式碼。以下是用來作這件事的程式碼 [3]。

[3]

```
pictureWidth=8;
```

```
pictureHeight=6;
```

```
pictureSurfaceArea=pictureWidth*pictureHeight;
```

令人意外的，編譯器對空白並不在意（除了位於變數名稱中、關鍵字等等！）要讓程式碼更順眼，我們可以使用空白。

[4]

```
pictureWidth = 8;
```

```
pictureHeight = 6;
pictureSurfaceArea = pictureWidth * pictureHeight;
```

現在，看看範例 [5]，尤其是開頭兩個述句。

```
[5]
pictureWidth = 8;
pictureHeight = 4.5;
pictureSurfaceArea = pictureWidth * pictureHeight;
```

一般而言數字可以分成兩種類別：整數（*integers*）與小數。您可以依序在述句 [5.1] 與 [5.2] 中看到例子。整數是用來計數，例如我們想要重複執行一連串的命令某個特定的次數時會使用它（參見第七章）。您可以藉由例如籃球命中率來了解小數或浮點數（*floating-point*）。

範例 [5] 的程式碼並不能運作。問題在於編譯器要您進一步告訴它您要在程式中使用哪個變數名稱，以及它們代表何種資料形別，亦即整數或者浮點數。在電腦怪胎的術語中，這稱為“宣告一個變數”。

```
[6]
int pictureWidth;
float pictureHeight, pictureSurfaceArea;
pictureWidth = 8;
pictureHeight = 4.5;
pictureSurfaceArea = pictureWidth * pictureHeight;
```

在 [6.1] 行中，*int* 代表了變數 *pictureWidth* 是一個整數（*integer*）。下一行中，我們藉著用逗號分隔兩個變數名稱一次宣告了兩個變數。更精確地，述句 [6.2] 說這兩個變數都是 *float*（浮點數）形別，亦即包含了小數部份的數。在這個例子中有點愚蠢的是，*pictureWidth* 與另外兩個變數是不同的形別。但您可以看到的是，如果您將整數（*int*）與浮點數（*float*）相乘，運算結果將是一個浮點數，那就是為什麼您應該將變數 *pictureSurfaceArea* 宣告為浮點數 [6.2]。

為什麼編譯器要知道一個變數代表的是整數或一個有小數部份的數呢？嗯，電腦程式需要使用一部份的電腦記憶體，而編譯器為每個它遇到的變數保留了記憶體（位元組）。因為不同的資料型別，在本例中是 *int* 與 *float*，需要不同數量的記憶體以及不同的表示方式，編譯器需要保留正確數量的記憶體並採用正確的表示法。

那如果我們使用非常大或需要非常精確的數字呢？編譯器幫它們保留的小小位元組容不下它們，不是嗎？沒錯。對此，我們有兩個解答：第一個，*int* 與 *float* 都有相似的型別可以用來儲存更大（或更精確）的數字。在大多數的系統上，它們分別是 *long long* 與 *double*。但即使是它們仍然可能被填滿，這引導我們到第二個解答：作為一個程式設計師，提防發生問題是您的工作。無論如何，這不在一本導論書籍的探討範圍之內。

此外，就像您知道的銀行戶頭一樣，整數與小數都可以是負的。但是如果您知道一個變數的值絕對不會是負的，您就可以將變數值的範圍擴展以充分利用所有可用的位元組。

[7]

```
unsigned int chocolateBarsInStock;
```

巧克力棒的數目不可能為負，因此 `unsigned int` 可以用在這裡。`unsigned int` 型別表示數字大於或等於零。

您可以一次宣告一個變數並把值指定給它 [8]。

[8]

```
int x = 10;  
float y = 3.5, z = 42;
```

這可以幫您節省一些輸入的麻煩。

在前面的範例中，我們進行了一個乘法運算。我們可以使用下列的符號，正式的稱呼是運算子（`operators`），來進行基本的數學計算。

+ 來進行加法

- 來進行減法

/ 來進行除法

* 來進行乘法

使用這些運算子，我們可以進行許多種類的計算。不過如果您看看專業的 Objective-C 程式設計師的程式碼，您將會發現一些特性，這可能是因為他們懶得打字所造成的。

程式設計師通常會將 `x = x + 1;` 重新排列為如範例 [9] 或 [10] 裡面的寫法

[9]

```
x++;
```

[10]

```
++x;
```

這兩個例子都代表：把 `x` 加一。在某些情況下，`++` 位於變數名稱前或後是很重要的。請看一下接下來的範例 [11] 以及 [12]。

[11]

```
x = 10;  
y = 2 * (x++);
```

[12]

```
x = 10;  
y = 2 * (++x);
```

在範例 [11] 中，當所有指令都執行之後，y 等於 20 而 x 等於 11。相對的，在述句 [12.2] 中，x 在乘以 2 執行之前就先加了 1，因此結束時 x 等於 11 而 y 等於 22。範例 [12] 中的程式碼與範例 [13] 的效果相同。

[13]

```
x = 10;  
x++;  
y = 2 * x;
```

因此，程式設計師事實上是把兩個述句合而為一。在我個人的觀點中，我認為這讓程式比較難閱讀。如果您採用這個捷徑，沒有關係，但是小心臭蟲可能就潛伏在此。

如果您已經從高中畢業，相信對您已經是陳腔濫調，但是我們仍然要告訴您可以用括號來決定運算的先後順序。一般來說，* 與 / 比 + 與 - 要早運算。因此 $2 * 3 + 4$ 等於 10。藉著使用括號，您可以強制先計算較低優先權的加法： $2 * (3 + 4)$ 等於 14。

除法需要您的一些特別留意，因為它在用於整數與浮點數時有些不同。請看看下面的範例 [14, 15]。

[14]

```
int x = 5, y = 12, ratio;  
ratio = y / x;
```

[15]

```
float x = 5, y = 12, ratio;  
ratio = y / x;
```

在範例 [14] 的情況中，結果是 2。只有在 範例 [15] 的第二個情況中，結果才是您預計的：2.4。

% 是一個您可能不熟悉的運算子。運算子 % 的結果是第一個運算元 (operand) 與第二個運算元進行整數除法的餘數 (如果第二個運算元是零，則 % 的行為是沒有定義的)。

[16]

```
int x = 13, y = 5, remainder;  
remainder = x % y;
```

現在，在 *remainder* 中的結果等於 3，因為 x 等於 $2*y + 3$ 。

以下是一些其他的例子：

21 % 7 等於 0

30 % 2 等於 0

50 % 9 等於 5

22 % 7 等於 1

31 % 2 等於 1

60 % 29 等於 2

23 % 7 等於 2

32 % 2 等於 0

24 % 7 等於 3

33 % 2 等於 1

27 % 7 等於 6

34 % 2 等於 0

有時候它會很有用，但是請記得只能用在整數上。