

# 第 3 章

## 函式

到目前為止，我們看到的最長的程式碼也只有五行述句。要到包含數千行的程式似乎還有一段路要走，但是因為 Objective-C 的天性使然，我們必須初步的探討程式是如何組織化的。

如果一個程式包含了一連串又臭又長的述句，它將會非常難以尋找並修正臭蟲。此外，一段相同的程式碼可能會在您的程式中重複出現好幾次。如果裡面有隻臭蟲，您就必須在好幾個地方修正一模一樣的臭蟲。這是一個夢魘，因為非常容易就會忘記一（或兩）個地方！因此，人們想出了一個方法來組織化程式碼，讓它更容易修正臭蟲。

這個問題的解決之道是把述句根據它們的功能群組起來。例如，您可能有一組述句可以用來計算圓面積。一旦您檢查過這些述句的結果是可信賴的，您將再也不需要查看這些程式碼來看看臭蟲是不是在其中。這組稱為函式（function）的述句有自己的名字，您可以使用這個名字來呼叫這組函式並讓這些程式碼被執行。使用函式這個概念非常基本，因此每個程式都至少有一個函式在其中：`main()` 函式。編譯器會去找尋 `main()` 這個函式，這樣它才知道在執行期間要從哪裡開始執程式碼。

讓我們仔細一點看看 `main()` 函式。[1]

```
[1]
main()
{
    // main() 函式主體。請將您的程式碼置於此處。
}
```

述句 [1.1] 顯示了函式名稱，亦即 "main"，接下來是左大括號跟右大括號。不同於 "main" 是個保留字而且 `main()` 函式是一定要出現的，當您定義自己的函式時，您可以以幾乎任何方式來為它們命名。小括號的存在是有理由的，但是我們將在本章稍後才討論它。在接下來的 [1.2,1.5] 行有一對大括號，我們必須把我們的程式碼放在這對大括號之間。任何位於這對大括號之間的東西稱為這個函式的主體。我從第一章拿了些程式碼過來並把它們放置到它們該去的位置 [2]。

```
[2]
main()
{
    // 變數們在此宣告
    float pictureWidth, pictureHeight, pictureSurfaceArea;

    // 我們初始化這些變數（我們給這些變數一個值）
```

```
pictureWidth = 8.0;
pictureHeight = 4.5;

// 底下是真正的計算進行的地方
pictureSurfaceArea = pictureWidth * pictureHeight;
}
```

如果我們繼續增加程式碼到 `main()` 主體中，我們就會遭遇除蟲的困難，沒有結構的程式碼是我們說過要避免的。讓我們來寫另一個程式，現在它包含了一些結構在內。除了必要的 `main()` 函式之外，我們將建立一個 `circleArea()` 函式 [3]。

```
[3]
main()
{
    float pictureWidth, pictureHeight, pictureSurfaceArea;
    pictureWidth = 8.0;
    pictureHeight = 4.5;
    pictureSurfaceArea = pictureWidth * pictureHeight;
}

circleArea() // [3.9]
{

}
```

這很簡單，但是我們在述句 [3.9] 開始的自定函式還沒有作任何事情！請注意函式的規格是位於 `main()` 函式主體之外的。換句話說，函式不是巢狀的。

我們的新函式 `circleArea()` 必須由 `main()` 函式中呼叫。讓我們看看要如何做到它 [4]。

```
[4]
main()
{
    float pictureWidth, pictureHeight, pictureSurfaceArea,
          circleRadius, circleSurfaceArea; // [4.4]
    pictureWidth = 8.0;
    pictureHeight = 4.5;
    circleRadius = 5.0; // [4.7]
    pictureSurfaceArea = pictureWidth * pictureHeight;
```

```
// 我們在此呼叫我們的函式！
circleSurfaceArea = circleArea(circleRadius); // [4.11]
}
```

請注意：這個程式其餘的部份並沒有列出（請見範例 [3]）。

我們新增加了一對 `float` 變數 [4.4]，並且初始化（initialize）變數 `circleRadius`，亦即給它一個值 [4.7]。最引人注意的應該是 [4.11] 行，就是 `circleArea()` 函式被呼叫的地方。如您所見，變數名 `circleRadius` 被放置在小括號之間。它是一個 `circleArea()` 函式的參數（argument）。變數 `circleRadius` 的值將被傳遞給函式 `circleArea()`。當函式 `circleArea()` 把它真正進行運算的工作完成之後，它必須傳回（return）結果。讓我們修改在範例 [3] 中的 `circleArea()` 函式來反映這件事 [5]。

請注意：底下只列出 `circleArea()` 函式。

```
[5]
circleArea(float theRadius) // [5.1]
{
    float theArea;
    // 圓周率 pi 乘以半徑 r 平方
    theArea = 3.1416 * theRadius * theRadius; // [5.5]
    return theArea;
}
```

在 [5.1] 我們定義了 `circleArea()` 函式需要一個浮點數值作為輸入。當接收到之後，這個數值被儲存在命名為 `theRadius` 的變數中。我們使用第二個變數，也就是 `theArea` 來儲存在 [5.5] 中計算的結果，因此我們必須在 [5.3] 中使用與在 `main` 函式 [4.4] 中一樣的方法來宣告它。您會注意到變數 `theRadius` 的宣告是在 [5.1] 行的小括弧間完成的。第 [5.6] 行傳回結果給呼叫這個函式的程式片段。最後，在第 [4.11] 行，變數 `circleSurfaceArea` 被設定為這個值。

在範例 [5] 中的函式已經完成，除了一個地方之外。我們還沒指定這個函式回傳的資料型別。編譯器需要我們作這個動作，而且沒有轉圜的餘地，因此在這邊我們標明它是屬於 `float` 型別 [6.1]。

```
[6]
float circleArea(float theRadius)
{
    float theArea;
    theArea = 3.1416 * theRadius * theRadius;
    return theArea;
}
```

如同第 [6.1] 行的第一個字表示的，這個函式傳回來的資料（亦即，變數 *theArea* 的值）是屬於 `float` 型別。作為一個程式設計師，您必須確定位於 `main()` 函式中的變數 *circleSurfaceArea* [4.11] 也有著相同的資料形態，如此一來編譯器就沒有理由在這件事上繼續煩我們。

並非所有的函式都需要參數。如果沒有參數，小括號仍然是需要的，即使它們是空白的

```
[7]
int throwDice()
{
    int noOfEyes;

    // 用來生成介於 1 到 6 之間亂數的程式碼

    return noOfEyes;
}
```

也並非所有的函式都會回傳一個值。如果一個函式沒有回傳值，它屬於“void”型別。而“return”述句則是可有可無的。如果您使用它，則 `return` 關鍵字之後就不可以跟著數值或變數名。

```
[8]
void beepXTimes(int x);
{
    // 發出 x 次逼聲的程式碼。
    return;
}
```

如果一個函式有多於一個參數，就像底下的 `pictureSurfaceArea()` 函式一樣，各個參數間要用逗號分開。

```
[9]
float pictureSurfaceArea(float theWidth, float theHeight)
{
    // 程式碼在此
}
```

`main()` 函式依照慣例應該要回傳一個整數，因此，沒錯，它的確也有一個 `return` 述句。它應該要回傳 0（零，[10.10]）來表示這個函式順利執行沒有任何問題。既然 `main()` 函式回傳一個整數，我們必須在 `main()` 之前寫 `"int"` [10.1]。讓我們把現在所有的程式碼列在一起。

```
[10]
int main()
```

```
{
    float pictureWidth, pictureHeight, pictureSurfaceArea,
           circleRadius, circleSurfaceArea;
    pictureWidth = 8;
    pictureHeight = 4.5;
    circleRadius = 5.0;
    pictureSurfaceArea = pictureWidth * pictureHeight;
    circleSurfaceArea = circleArea(circleRadius);
    return 0; // [10.10]
}
```

```
float circleArea(float theRadius) // [10.13]
{
    float theArea;
    theArea = 3.1416 * theRadius * theRadius;
    return theArea;
}
```

就如您在範例 [10] 中所見，我們在第 [10.1] 行有個 `main()` 函式，在第 [10.13] 行有另一個自己定義的函式。如果我們編譯這份程式碼，編譯器仍然會停頓下來。在第 [10.9] 行，它會宣稱不知道有任何一個名為 `circleArea()` 的函式。為什麼？看起來編譯器似乎開始讀 `main()` 函式，然後就突然遇到某個它不認識的東西。它不再繼續讀下去而且給了您一個警告（warning）。要滿足編譯器，只要在包含 `int main()` 的述句之前加入函式宣告（**function declaration**）即可 [11.1]。這沒有什麼困難，它跟第 [10.13] 行一模一樣，除了以分號結尾之外。

```
[11]
float circleArea(float theRadius); // 函式宣告
```

```
int main()
{
```

請注意：程式的其他部份並未列出（請參閱範例 [10]）。

我們很快就要來實際的編譯這個程式了，但是要先把一些剩下來的瑣碎事情處理完。

當我們撰寫程式時，建議您把保持程式碼未來的可再用性牢記在心。我們的程式可能有個 `rectangleArea()` 函式，就像下面範例 [12] 中的一樣，而這個函式可以在我們的 `main()` 函式中呼叫。即使我們放在一個函式中的程式碼只被用過一次，這仍然是有幫助的。因為 `main()` 函式會變得比較容易閱讀。如果您必須對您的程式碼除蟲，這會讓找出臭蟲可能位於您程式

中何處的工作變得容易些。您可能會發現它位於某個函式中。因此您不用大費周章的查看一長串的述句，只需要檢查函式中的述句即可，讓尋找臭蟲的工作變得容易許多。這一切都要感謝左右括號。

[12]

```
float rectangleArea(float length, float width)
{
    return (length * width);
}
```

如您所見，在像這樣的簡單例子中，可能會有一句單一述句 [12.3] 同時計算並傳回結果。我之前在 [10.15] 中使用一個不必要的變數 *theArea* 只是為了向您示範如何在函式中宣告變數。

雖然我們在這章裡面自己定義的函式很直觀，但是它對於了解一件事是非常有幫助的，那就是您可以修改一個函式卻不影響呼叫它的程式碼，只要您沒有改變這個函式的宣告（亦即，它的第一行）的話。

舉例來說，您可以改變函式內的變數名稱，但是它卻仍然正常運作（而且這將完全不影響程式的其他部份）。其他人可以撰寫這個函式，而您可以在完全不了解它內部運作的情況下使用它。您所需要知道的只有如何使用這個函式，也就是了解：

- 這個函式的名字
- 這個函式的參數個數、順序以及型別
- 這個函式傳回什麼（矩形面積的值）以及結果的型別

在範例 [12] 中，這些答案分別是：

- `rectangleArea`
- 兩個參數，都是浮點數 `float`，第一個代表長，第二個代表寬。
- 這個函式傳回了某個東西，而結果是浮點數 `float` 型別（可以由述句 [12.1] 的第一個字得知）。

在函式中的程式碼被從主程式中隔離開來，也跟其他函式隔離。這是 Objective-C 一個最不可或缺的功能。在第五章中，我們將再次探討這個行為。但是現在，我們要開始啟動 Xcode 並且執行上面的程式 [11]。