Chapter 3  
迴圈與集合

上一章我們開始學習使用了簡單的變數，以及基礎的邏輯判斷。然而，每個動作都要下一句指令，實在讓人感覺不出學習程式語言到底有什麼用，反而比人工更麻煩。

事實上電腦程式就是這樣，一個口令一個動作，然而在這一章，我們將會學習如何使用單純的指令，讓程式執行重複的工作，而不是每碰到重複的動作時就需要打一行程式碼。此章的重點在於使用迴圈執行重複的指令，並且搭配迴圈，操作集合。

##### 完成本章後，您可以學會:

* 使用迴圈讓程式執行重複的動作。
* 使用不同的迴圈以滿足不同的重複條件。
* 了解陣列與列表。
* 使用迴圈操作集合。

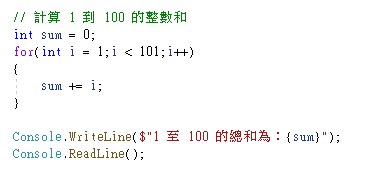
## 3.1 迴圈(Loop)

上一章我們學習了使用變數及邏輯判斷，然而每一個動作就必須下一道指令，這不會讓我們的工作更加輕鬆。請記住，程式的目的是以電腦協助人類工作，以降低成本並提高效率，如果沒辦法簡化人類的工作，那就不會有人想用。

回到主題，我們要如何讓程式自動執行重複的工作？在這個問題上，我們要先思考幾個問題：

1. 需要重複執行的工作內容是什麼？
2. 程式在什麼情況下需要重複執行？
3. 程式要執行幾次，或者說在什麼情況下要結束重複執行？

第一和第二個問題也許不難，但第三個問題，有的時候我們並不能確切的說出，程式應該執行幾次才會結束，而是在執行結果滿足了某個條件的情況下，我們才會讓程式結束。而依照結束條件的不同，我們可以分為特定次數的For迴圈，以及不定次數的While迴圈。以下我們先來介紹For迴圈的寫法：



上述的程式為計算1的100的整數之和。可以看見，我們並不需要

sum = sum + 1;

sum = sum + 2;

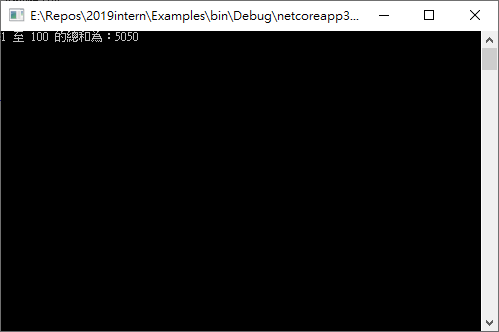
………………………

sum = sum + 100;

這樣真的寫100行從1加到100，我們使用了for迴圈來協助我們完成任務。來看看for迴圈的結構，首先要以for開頭，後接小括號，括號中的內容則是迴圈的執行條件，而迴圈要執行的工作內容，則是在小括號後方以大括號包覆。

我們的第一個條件，是在此迴圈內宣告了一個整數類型的索引值i，而它的初始值是1。i++表示每當此迴圈執行完大括號的內容後，便將i索引值+1；當i < 101 的時，迴圈會繼續執行；當i >= 101時，迴圈則會結束。

我們的目的是要計算 1 ~ 100的整數和，所以我們的條件是從1開始，到100結束(小於101的整數中，最大就只到100，當然也可以寫作 <= 100)，而每執行完一次加總，下個要加的數會比目前的i多1，所以每次執行完迴圈就將i + 1。而相加之和則記錄於 sum 變數中，最後輸出sum之值即為結果。這樣寫是不是簡單多了？



|  |  |
| --- | --- |
|  | sum += i的寫法等同於 sum = sum + i，其意為將sum目前之值與i之值相加後，存回sum中。同樣的寫法還有 -=、\*=、/=。 |

|  |  |
| --- | --- |
|  | For迴圈的索引值一般習慣用i，如果i變數已在同區域中使用，則會使用j、k…依次類推，不過並沒有強制規定索引值的名稱，愛叫什麼都可以。 |

|  |  |
| --- | --- |
|  | 迴圈可以重複執行工作，但不一定是最快速的解法。以1~100的整數和為例，最快的解法是(1 + 100) \* 100 / 2 = 5050。 |

除了For迴圈以外，別忘了我們還有不定次數的While迴圈，讓我們看看以下的範例：



假設我們要找出某個數是不是質數，我沒什麼比較好的方法，就是試除看看。但要試到什麼時候為止，我們不需要從2到該數全試一遍，只要試到該數的平方根即可。所以條件是div \* div <= target。當迴圈每執行過一次，程式便會判斷div \* div <= target 的條件是否成立，如果條件成立則會繼續執行迴圈內容，否則停止。

而當我們找到有一個數可以整除97時，也不用再試了，因為它已經不是質數，可以使用break指令跳出迴圈。

|  |  |
| --- | --- |
|  | 由於while迴圈不像for迴圈在條件中定義了每次執行完索引值的變化，所以要切記每次執行完迴圈，最後一定要加上推進條件(div++)，否則div的值永遠不會變化，程式也就永遠不會停止，稱之為無窮迴圈(infinite loop)，這是程式設計上常見的bug之一。 |

|  |  |
| --- | --- |
|  | 除了break以外，迴圈中能改變控制流程的指令還有continue。當迴圈碰到break時，會直接中止迴圈的執行；碰到continue時，則是本輪不做動作直接跳下一輪執行。 |

|  |  |
| --- | --- |
|  | 這個範例是找出97是否為質數，是個定值。讀者可搭配上一章讀取使用者輸入及轉型的實作方式，將此範例修改為「找出使用者輸入的數是否為質數」，而不是定值97。試著修改看看，如果不知道如何修改建議要再回頭看上一章的內容。 |

## 3.2 集合的使用

關於迴圈的應用，相信大家已經有了基礎的認識。但大家畢竟不是數學家，更多的使用情境可能是在日常生活的特定範圍中使用，而不是自然數的領域。這個時候我們就必須導入集合了。.Net框架提供了不少集合種類，諸如HashSet、HashTable、陣列(Array)、列表(List)等等，這裡我們只介紹兩個初學者最常用也最有機會接觸到的集合：陣列與列表。

集合是具有某種特性的事物的整體，或是一些確認物件的匯集。構成集合的事物或物件稱作**元素**或是**成員**。集合的元素可以是任何事物。

### 3.2.1 陣列(Array)

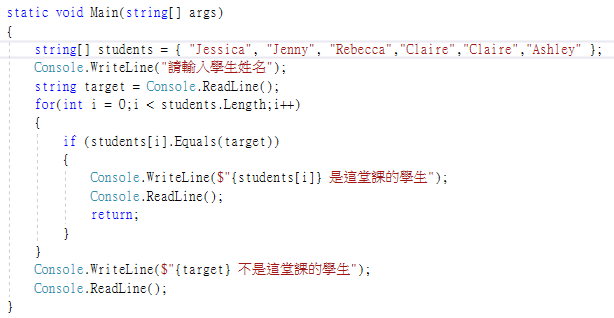
我們假設，一門課有6個學生，分別是 Jessica、Jenny、Rebecca、Claire、Ashley。我們可以用一個陣列將學生儲存在一起。陣列的寫法如下：



開頭的string，表示這個陣列儲存的是string(字串)型別的資料；後方必須接中括號[]；students為這個陣列的名稱；而大括號{}中的則是此陣列的內容，各資料間以逗號做區隔，因為資料型別是string，所以需要用雙引號包覆。

現在，如果我們要找出某個學生是不是這門課的學生，我們可以對整個陣列跑一個迴圈，一一比對學生的姓名是否吻合。如果沒發現吻合的，那我們就知道該學生不是此門課程的學生。

這樣的程式，寫法如下：



當我們迴圈的範圍是特定陣列時，我們需要使用「索引值」來對陣列的特定元素進行操作。陣列的第一個元素索引值為0，也就是students[0] 代表的是Jessica這位學生，接著依序增加1，所以students[5]為Ashley，而陣列提供了一個屬性students.Length，代表陣列中的元素個數，students中有6個元素，所以students.Length = 6；而迴圈的索引值，從0到5(小於6的整數最大就是5)，剛好可以跑完整個陣列。

|  |  |
| --- | --- |
|  | 上面的範例中，我們要是找到符合的名字，就可以不用再找下去跳出迴圈了。不過，筆者在此使用的是return關鍵字，而非上一小節的break。請注意這兩個關鍵字用法的不同：  使用return是直接中止這個函式的執行，並回傳值(如果有的話)；而break則是中止迴圈的執行。換句話說如果使用break，程式會繼續執行迴圈外的印出「XXX不是這堂課的學生」，這樣的執行結果就不對了。 |

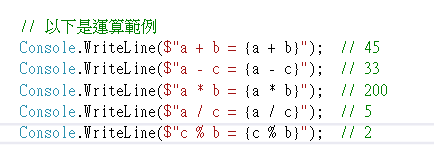
|  |  |
| --- | --- |
|  | 有沒有發現，陣列裡有兩個名叫Claire的學生？陣列是可以加入重複元素的，就像班上也可以有兩個同名的學生。在這個範例中，我們只判斷名字，所以只要找到有叫做Claire的學生，程式就結束了。當然也有不允許加入重複元素的集合，如HashSet，不過在這章節中我們不提。 |

|  |  |
| --- | --- |
|  | 陣列在宣告時，必須確定陣列的大小。此範例在宣告時除了確定元素個數外，也確定了元素內容。也有在宣告時只宣告個數，但內容在之後才加入陣列的寫法，如下：    由此亦可知，陣列中的元素亦可重新指派新值，而不是加入後就不能修改的。 |

### 3.2.2 列表(List)

列表是另一種常使用的資料集合，與陣列不同的是，列表的大小是不固定的，可以隨著元素的加入會移除而動態變化。

我們先宣告一些變數，有數值類(整數、二進位浮點數、十進位浮點數)，也有字串。接著，我們來試著對數值做點運算：

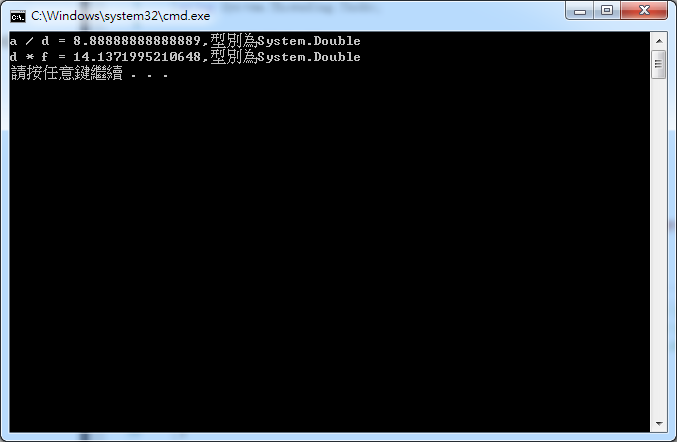
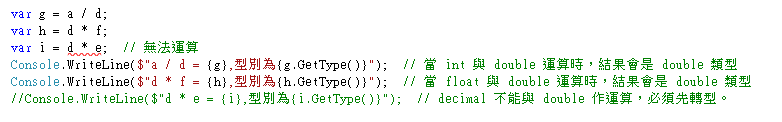


以上只是簡單的加減乘除，以及取餘數。這裡可以看到，由於 a 與 c 都是整數，所以相除後結果也會是整數，小數的部分就無條件捨去了。

|  |  |
| --- | --- |
|  | C#是編譯式語言，在執行前會先將程式碼透過編譯器編譯成可執行檔案(.exe)，如果程式碼在編寫時有誤，編譯器便可以在執行前將之揪出。然而，也有的情況是只有在執行期才會發生的錯誤(Runtime Error)，即使通過了編譯，也不代表程式必定可以順利執行。  例如，以上述的例子而言，當我們把程式碼改為a / 0(除法不能除以0)，雖然在編譯期不會出現問題，程式執行時就會跳出DivideByZeroException例外而無法正常執行。 |

|  |  |
| --- | --- |
|  | 加減乘除相信大家都知道，%的運算是取餘數，讀作modulus，例如 10除以7，商數是1，餘數為3，則 10 % 7 = 3。  另外，除法是不能除以0的，所以 % 一樣不能除以0。 |

接著我們來看一下，當不同類型的數值一同運算時，會發生什麼樣的事情。



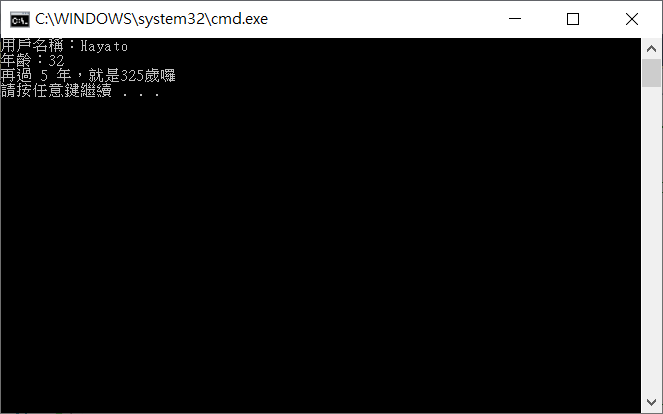
如上圖，當 int 與 float 碰到 double 時，計算出來的結果都會被改為 double 型別；而 decimal 型別是不能直接與 double 作計算的。

|  |  |
| --- | --- |
|  | Double和decimal明明都是浮點數，為什麼不能直接和decimal作運算？簡單的說，因為decimal(十進位浮點數)的精確度比double(二進位浮點數)來得高，所以double和decimal間不能直接運算。而float是double的子集合，所以可以直接隱含轉換。  那，什麼時候使用decimal呢？由於decimal的特性，非常適合在財務與金融計算上使用。畢竟，算錢的時候，要使用精確度較高的資料型別，才能坦蕩蕩避免起爭議嘛！ |

除了數值型別，我們也可以來看看字串型別的運算。



當我們試圖印出用戶五年後的年齡時，age + 5印出來的並不是 32 + 5 = 37，而是32和5連在一起的字串325。如下圖：

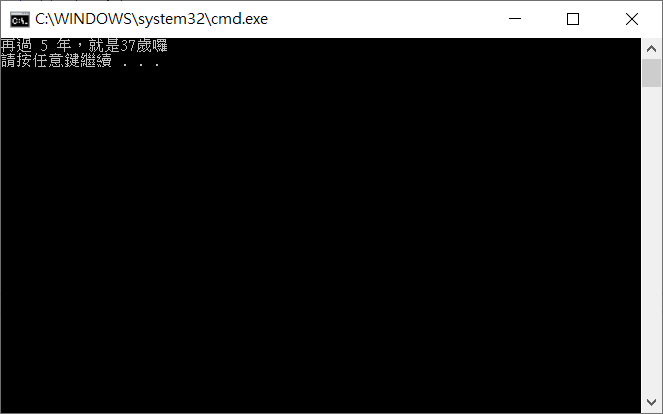


這顯然是有問題的，會造成這樣的原因，是因為age我們在宣告時，宣告為字串(string)型別，當放在Console.WriteLine()中時由於其特殊的機制，程式自動將age + 5 的5轉換為字串類型，而在字串間使用「+」的意思是將字串作拼接，所以輸出結果是325。

如果我們想讓字串型別的資料，像數值一樣作運算，就必須先將其轉型。如下圖：



使用Convert.ToInt32()函式將字串型別轉換成整數型別，這樣就可以運算了。



|  |  |
| --- | --- |
|  | Console.WriteLine()這個方法會使用複合格式，將物件的值轉換成文字表示，所以即使age是字串，直接與整數運算在Console.WriteLine()中是不會不能執行的，否則正常情況下編譯器會直接顯示錯誤。 |

|  |  |
| --- | --- |
|  | 轉型的方法很多，在C#語言中可分為隱含轉換、明確轉換、使用者定義轉換及使用協助程式類別轉換。上面的例子為使用協助程式類別轉換的一種。每種轉型的情況雖然不盡相同，但都有可能發生例外或資訊遺失，也就是程式有可能因為錯誤而中斷或轉換後的數值與原數值有誤差。例如我們如果想把一個字串”abc”轉型成整數類型，而使用Convert.ToInt32()，就會發生例外情況，因為原始字串並不能看成數值類型。 |

## 2.3 邏輯分支判斷

本章的最後一節，會提到邏輯分支判斷。

假設，我們設計一個諮詢程式，讓使用者輸入他的年收入(萬元)，並依照年收入的多寡來回應對該使用者增加年收入的建議。我們定義的條件如下：

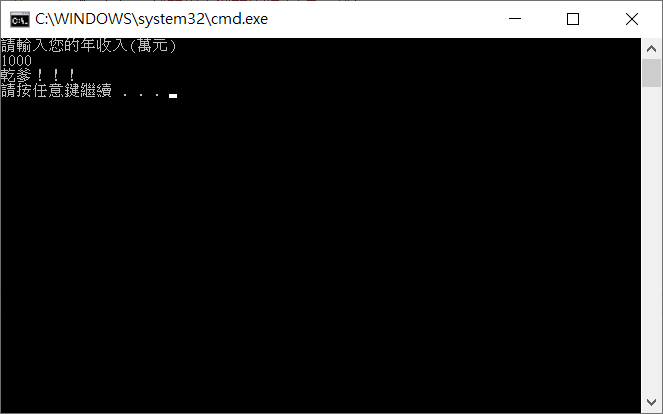
1. 年收入未達50萬：輸出「…我敢肯定這本書對你會有幫助的。」
2. 年收入50萬以上未滿100萬：輸出「…好好把這本書讀完，你就有機會再進一步。」
3. 年收入100萬以上未滿500萬：輸出「恭喜你，人生勝利組。」
4. 年收入500萬以上：輸出「乾爹！！！」

依照上述的規則，我們需要讀取使用者輸入的資料，並作出判斷來回應。本章開頭已經示範過如何讀取使用者輸入，還不熟悉的讀者請回頭看前兩章；而在邏輯分支判斷上，C#常用的語法有兩種，一種是if-else if-else，一種是switch。這邊的情境，我們較適合使用if-else if-else作處理，範例如下：



使用if時，需要使用小括號將敘述式包覆起來。而如果同時有多個條件，則可以用 ==(Equal)、&&(And)、||(Or)、!(NOT)及小括號作連接。當然有的型別也提供判斷是否相等或包含的函式，只要結果可得出一個布林值(真或假)的敘述式都可以。而條件成立要執行的程式碼區塊，則需使用大括號包覆起來。

最後，看看我們的執行結果吧。



|  |  |
| --- | --- |
|  | if條件之後不一定要直接接else條件，可以同時多個if或else if 並列，也可以沒有else條件。多個if條件並列，與if-else if條件並列的區別在於，如果是多個if條件並列，則每一個if條件都會判斷，如果同時符合多個if條件，則這些區塊都會被執行；如果是if – else if，程式會由上而下依序判斷，只要其中有一個條件成立，程式在執行過後就不會再執行這組if-else if中的其他條件區塊。  另外，else區塊作為與if相對應的條件區塊，並不需要明確的條件敘述，只要if不成立就會直接走進else區塊。在if-else if-else中，則是上述所有if及else if條件都不成立的話，就會執行else區塊。在設計上應想好是不是有一個預設的機制在所有條件都不成立時要執行，如果有的話，就應該放在else區塊中，反之則應檢查else中是不是有不應放入的程式碼，造成程式邏輯錯誤。 |

|  |  |
| --- | --- |
|  | 您沒有看錯，我也沒有打錯，在邏輯分支中，等於就是以兩個等號「==」來作判斷是否相等，這不是網路聊天用的表情符號。其他像「且」則是以兩個「&&」判斷兩個敘述式是否同時成立，「或」則是以兩個「||」(這個鍵通常是跟「\」在同個按鈕上，請按住shift鍵打出)判斷，只要有其中一個條件成立即可。  And與Or存在捷徑運算，對And來說，只要第一個敘述式為假，就不會再判斷第二個；Or則是第一個敘述式為真，就不會判斷第二個。 |