Microsoft Small Basic

程式設計入門

第一章

入門

Small Basic 與程式設計

電腦程式設計被定義為使用程式設計語言創建電腦軟體的過程。就像我們可以說並且理解英語，西班牙語或者法語，電腦能理解用特定語言寫成的程式。這些特定語言被稱為程式設計語言。最初，只有很少幾種程式設計語言，並且它們都非常容易學習和理解。但是隨著電腦和軟體變得越來越精深，程式設計語言發展很快，並且隨之彙集了更複雜的概念。從而造成現代大部分程式設計語言以及相關的概念對於初學者掌握起來而言頗具挑戰性。這一事實已經開始阻礙人們學習或嘗試電腦程式設計。

Small Basic 是一門針對初學者設計的使程式設計變得非常容易，親切，有趣的程式設計語言。Small Basic 的目的 在於消除障礙，充當通往令人驚奇的電腦程式設計世界的踏腳石。

Small Basic 環境

讓我們從對 Small Basic 環境的一個快速介紹開始。當你第一次運行 SmallBasic.exe，你會看到一個如下圖所示的視窗。

圖 1 - Small Basic 環境

這就是 Small Basic 環境，我們將在這裡編寫和執行我們的 Small Basic 程式。這個環境有幾個截然不同的部分。圖中已經用數位標出。

编辑器，即標記為[1]的部分，我們將用來寫我們的 Small Basic 程式。當你打開一個示常式序或者一個先前保存過的程式，它將顯示在這個編輯器裡。這樣你就可以對其進行更改並且保存以備後用。

你也可以同時打開和工作在多個程式上。每個打開的程式將被顯示在單獨的編輯器裡。包含你當前正工作在上面的程式的編輯器被稱為*活动编辑器*。

工具栏，即標記為[2]的部分，是被用來向*活动编辑器*或環境發佈命令的。隨著我們的進展，我們將學到工具列中各種各樣的命令。

表面，即標記為[3]的部分，用來放置所有編輯器視窗。

我們的第一個程式

既然你已經熟悉了 Small Basic 環境，我們將進而在裡面開始程式設計。正如我們剛剛在上面提到的，編輯器是我們寫程式的地方。所以，讓我們先在編輯器裡輸入下麵這行。

```
TextWindow.WriteLine("世界你好")
```

這是我們的第一個 Small Basic 程式。如果你輸入正確，你應該看到與下圖相似的結果。

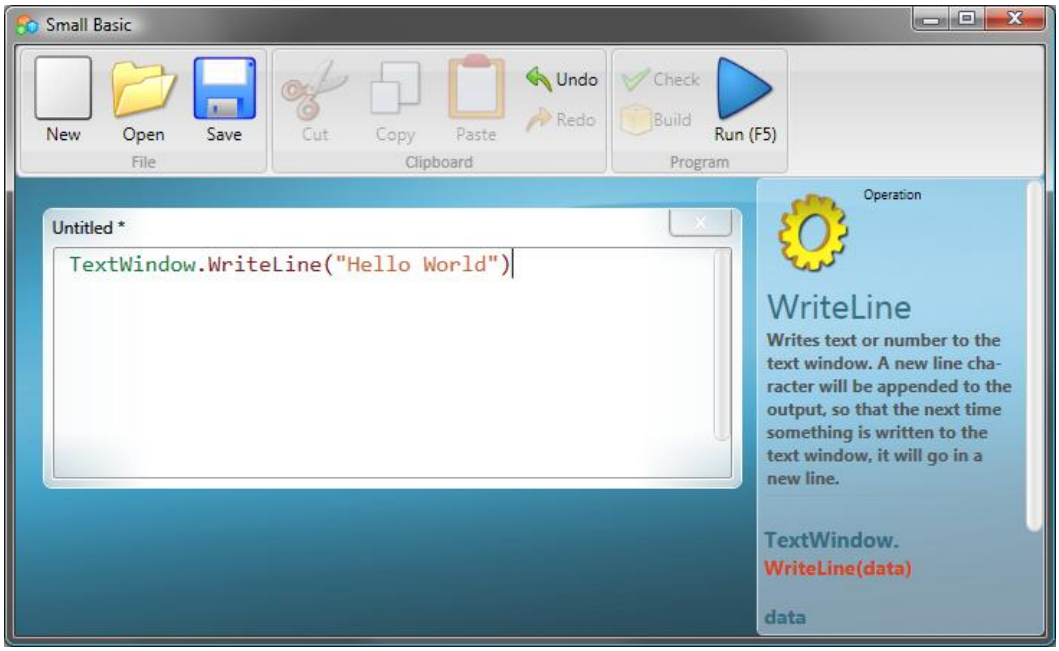

圖 2-第一個程式

既然我們已經輸入了我們的新程式，讓我們來運行它看看會發生什麼。我們可以通過點擊工具列上的*运行*按鈕或者使用鍵盤上的 F5 快速鍵來運行我們的程式。如果一切順利，我們的程式將運行並得到如下圖所示的結果。

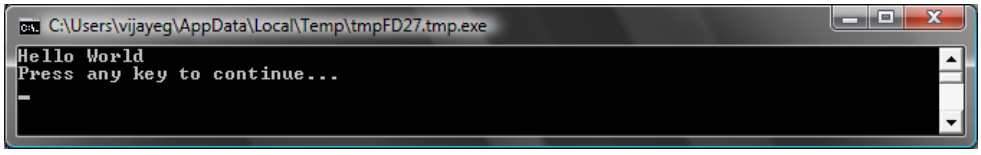

圖 3-第一個程式的輸出

恭喜！你已經編寫並運行了第一個 Small Basic 程式。一個非常簡單的小程式，然而卻是通向成為一個真正電腦程式員的一大步！現在，在繼續創建更大的程式之前，還有一個細節需要瞭解。我們必須要瞭解剛剛發生了什麼—我們到底告訴了電腦什麼並且電腦是如何知道要做什麼的？在下一章，我們將分析我們剛剛寫的程式，從而對其進行理解。

圖 4- Intellisense （智慧感知））。*這被稱作“intellisense”（智慧感知）。它說明你能更快的輸入你的程式。你可以通過按上/下方向鍵來遍歷這個清單。當你找到你想要的，你可以按回車鍵來將選中的條目插入到你的程式中。*

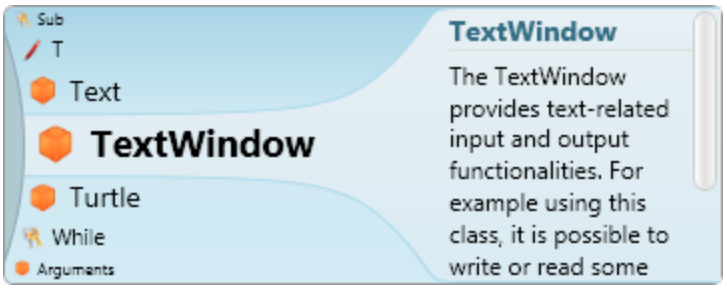

圖 4- Intellisense（智慧感知）

保存我們的程式

如果你想關閉 Small Basic 並且打算以後回來在你 剛輸入的程式上繼續工作，你可以保存程式。隨時保存程式實際上是個很好的習慣，以至於你在意外關閉或斷電時不會丟失資訊。你可以通過點擊工具列上的“保存”圖示或者使用快速鍵“Ctrl+S”（在按下 Ctrl 鍵時按 S 鍵）來保存當前程式。

第二章

解讀我們的第一個程式

什麼是真正的電腦程式?

一個程式是一組電腦指令。這些指令準確地告訴電腦做什麼，並且電腦總是遵循這些指令。就像人們一樣，電腦只能遵循用它們能明白的語言指定的指令。這些語言被稱為程式設計語言。有非常多的語言電腦能明白，**Small Basic** 是其中之一。

假像你和你的朋友之間有個會話。你和你的朋友用詞語組成句子來彼此傳遞資訊。相似地，程式設計語言包含詞語的集合用來組成句子來將資訊傳遞給電腦。程式基本上是多組語句（有時很少，有時多達數千）一起用同樣的方式 讓程式師和電腦彼此明白。

有很多電腦能明白的語言。Java, C , Python, VB, 等都是強大的現代電腦語言。它們被用來開發簡單到複雜的軟體程式。

Small Basic 程式

一個典型的 Small Basic 程式由一組*语句*組成。 程式的每一行代表一條語句。每條語句是給電腦的一條指令。當我們讓電腦執行一個 Small Basic 程式時，它取過程式並從第一個語句讀起。它明白我們說的是什麼，並且執行我們的指令。一旦執行完第一條語句，它回到程式繼續讀取並執行第二行。如此繼續下去直到程式的結尾。至此，我們的程式執行完畢。

回到我們的第一個程式

這是我們寫的第一個程式：

```
TextWindow.WriteLine("世界你好")
```

這是一個非常簡單的只包含一條*语句*的程式。這條語句告訴電腦寫一行內容為**世界你好**的文字到 Text Window。

這條語句在電腦裡可逐字翻譯成：

```
寫世界你好
```

你可能已經注意到這條語句可以依次分解成更小的片斷，就像句子可以分解成詞語一樣。在第一條語句中，我們有三個清楚的片段：

a) TextWindow
b) WriteLine
c) “世界你好”

點號，圓括弧和引號都是必須被放到語句中恰當位置的標點符號，從而使電腦能夠明白我們的意圖。

你可能還記得在我們運行我們的第一個程式時出現的黑色視窗。這個黑色視窗叫做 TextWindow，有時也被稱為主控台。那就是程式的結果輸出的地方。**TextWindow**，在我們的程式中被叫做*对象*。在我們的程式中有許多這樣的物件供我們使用。我們可以對這些物件執行一些不同的*操作*。在我們的程式裡，我們已經使用了 ***WriteLine*** 操作。你可能還注意到 WriteLine 操作後面跟著放在引號中的**世界你好**。這段文字被作為輸入傳遞給 WriteLine 操作，然後列印出來給使用者。這被稱為對該操作的 一個*輸入*。一些操作接受一個或者多個輸入，而其他的則不需要任何輸入。

我們的第二個程式

既然你已經明白了我們的第一個程式，讓我們繼續加一些顏色來讓它更新奇。

> *標點符號如引號，空格以及圓括弧在電腦程式中非常重要。基於它們的位置和數量，它們可以改變所要表達的意思。*

```
TextWindow.ForegroundColor =
"Yellow"
TextWindow.WriteLine("世界你好")
```

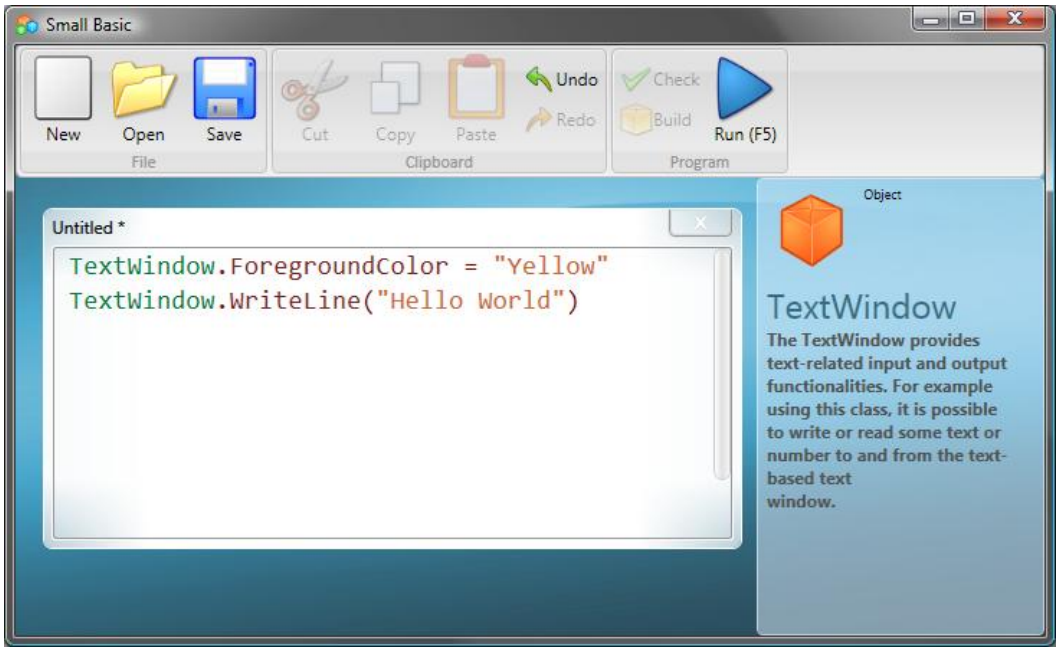

圖 5-增加顏色

當你運行上面的程式，你將注意到在 TextWindow 中輸出的還是“世界你好”，但是這次字體是黃色的而不是之前的灰色。

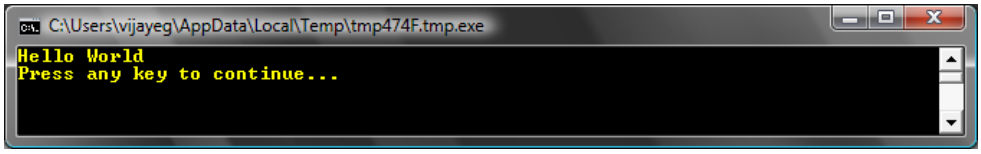

圖 6-黃色的世界你好

注意加到我們最初的程式中的這條新語句。它使用了一個新單詞， *ForegroundColor* （前景色），我們讓它等於 “*Yellow*”。這意味著我們把“Yellow”指定給 *ForegroundColor*。現在，ForegroundColor 和 WriteLine 操作的區別是 ForegroundColor 沒有接受任何輸入而且也沒有圓括弧。取而代之的是跟了一個*等子*符號和一個詞。 我們定義 ForegroundColor 為 TextWindow 的*屬性*。下麵是一些對 ForegroundColor 屬性的有效值。試著用它們中的一個替換“Yellow”來看看結果 — 不要忘記放引號，這是必要的。

```
Black
Blue
Cyan
Gray
Green
Magenta
```

```
Red
White
Yellow
DarkBlue
DarkCyan
DarkGray
DarkGreen
DarkMagenta
DarkRed
DarkYellow
```

第三章

引入變數

在我們的程式中使用變數

如果我們的程式能夠對使用者的姓名說“你好”而不只是單純的“世界你好”，那樣不是更好嗎？為了能那樣做，我們必須首先讓使用者告知他/她的姓名並將其保存到某個地方，然後輸出使用者的姓名和“你好”。讓我們一起來看看這是如何做到的：

```
TextWindow.Write("輸入你的姓名：")
name = TextWindow.Read()
TextWindow.WriteLine(name + "你好")
```

當你輸入並執行這個程式，你將看到如下輸出：

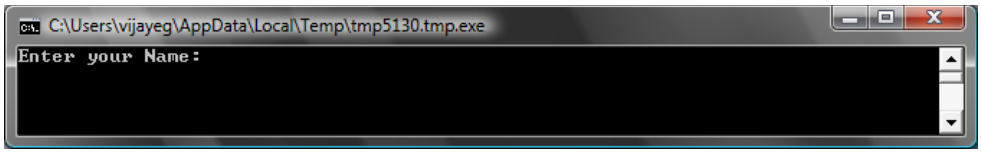

圖 7-詢問使用者姓名

當你輸入你的姓名並按 ENTER 鍵，你將看到如下輸出：

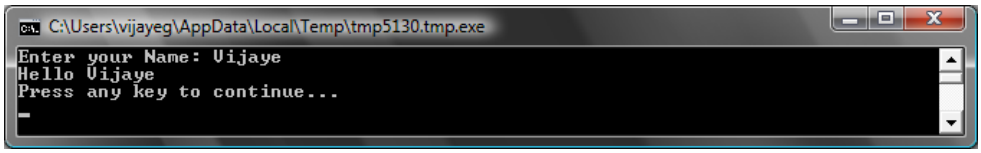

圖 8–一個溫馨的問候

現在，如果你再運行這個程式，你將被再次問同樣的問題。你可以輸入一個不同的姓名，電腦將對這個姓名說你好。

程式解析

在你剛剛運行的程式中，可能引起你注意的那行程式是：

```
name = TextWindow.Read()
```

Read() 看上去就像 *WriteLine()*，但是沒有輸入。它是一個操作並且基本上是告訴電腦等待使用者輸入些什麼並按下 ENTER 鍵。一旦使用者按下 ENTER 鍵，它將獲取使用者的輸入並將其返回給程式。有趣的一點是無論使用者輸入的什麼，現在都被存放在一個叫 **name** 的*變量*中。一個變數被定義為用來臨時存儲數值以備以後使用的地方。在上面的程式列中，**name** 被用來存儲使用者的姓名。

下麵的一行也很有趣：

```
TextWindow.WriteLine(name + "你好")
```

Write，與 WriteLine 相似，是 ConsoleWindow 上又一個操作 。Write 允許你向 ConsoleWindow 寫東西，但是後續文本將與當前文本在同一行。

這是我們使用存放在我們的變數，**name** 中的值的地方。我們取出 name 中的值並將它與“你好”一起寫到 TextWindow。

一旦一個變數被設定，你可以多次使用它。例如，你可以如下這麼做：

```
TextWindow.Write("輸入你的姓名：")
name = TextWindow.Read()
TextWindow.Write("你好，" + name + "。  ")
TextWindow.WriteLine("你最近怎麼樣，" + name + "？")
```

你將會看到如下的輸入：

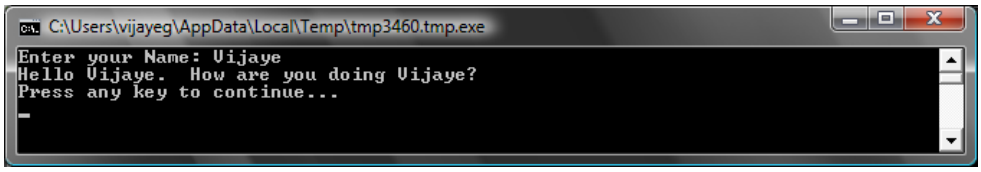

圖 9-變數重用

變數命名規則

未完成

使用數位

我們剛才看到如何用變數存儲使用者的姓名。在接下來的程式中，我們將看到如何用變數存儲和運算元字。讓我們從一個非常簡單的程式開始：

```
number1 = 10
number2 = 20
number3 = number1 + number2
TextWindow.WriteLine(number3)
```

當你運行這個程式，你將得到下麵的結果：

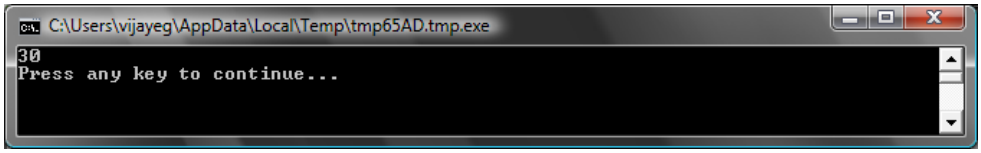

圖 10-兩數相加

在程式的第一行，你將變數 **number1** 賦值為 10。在第二行，你將變數 **number2** 賦值為 20。在第三行，你將 **number1** 和 **number2** 相加並將結果賦值給 **number3**。 因此，在這種情況下， **number3** 的值將是 30。並且這就是我們輸出到 TextWindow 的結果。

注意數位沒有放在引號中。對於數位，不需要引號。只有在使用文本時，才需要引號。

現在，讓我們對程式做輕微的修改並看看結果：

```
number1 = 10
number2 = 20
```

```
number3 = number1 * number2
TextWindow.WriteLine(number3)
```

上面的程式將 **number1** 與 **number2** 相乘並將結果存放在 **number3** 中。你可以看到如下的程式運行結果：

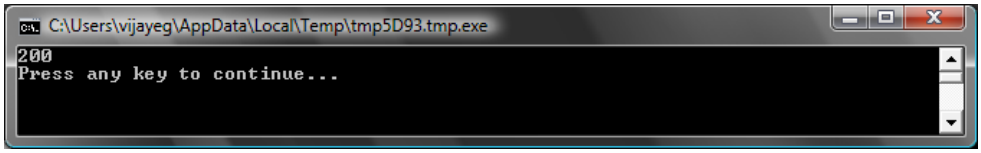

圖 11-兩數相乘

相似的，你可以對數位作減法或除法。這是減法：

```
number3 = number1 - number2
```

除法的符號是'/'。程式看上去就像這樣：

```
number3 = number1 / number2
```

這個除法的結果是：

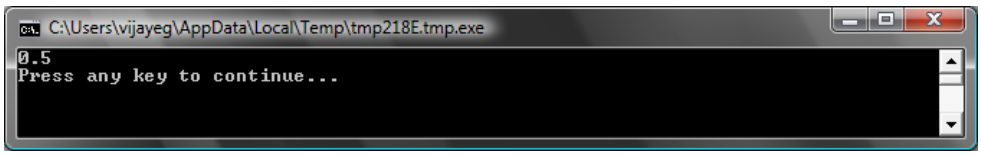

圖 12-兩數相除

一個簡單的溫度轉換器

下一個程式我們將用公式 $℃ = \frac{5(°F-32)}{9}$ 將華氏溫度轉換成攝氏溫度。

首先 ，我們從使用者那裡得到華氏溫度並存放到一個變數中。這裡有一個特殊的操作能讓我們從使用者那裡讀取數位，它就是 **TextWindow.ReadNumber**。

```
TextWindow.Write("輸入華氏溫度： ")
fahr = TextWindow.ReadNumber()
```

一旦我們有存放在變數中的華氏溫度，我們可以像這樣將它轉換為攝氏溫度：

```
celsius = 5 * (fahr - 32) / 9
```

圓括弧告訴電腦先計算 **fahr - 32** 然後再處理其它的。現在我們需要做的就是將結果輸出給使用者。將所有這些放到一起，就是我們的程式：

```
TextWindow.Write("輸入華氏溫度： ")
fahr = TextWindow.ReadNumber()
celsius = 5 * (fahr - 32) / 9
TextWindow.WriteLine("攝氏溫度是 " + celsius)
```

程式的運行結果是：

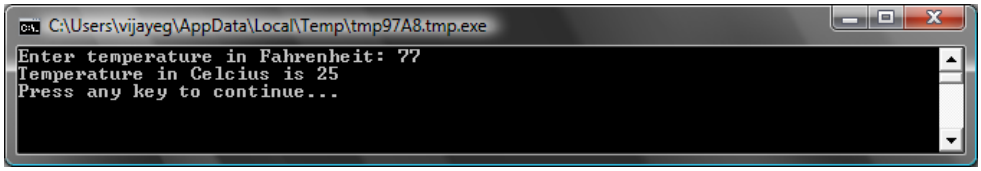

圖 13-溫度轉換

第 4 章

條件與分支

回到第一個程式。如果我們的程式不僅僅會說“*Hello World* （你好，世界）”； 而是可以根據一天中的不同時間段說“*（早上好，世界）*”或者“ *（晚上好，世界）*”，那是不是一件很酷的事情？在下一個程式中， 我們將讓電腦能夠在中午 12 點以前說“*（早上好，世界）*”，在中午 12 點以後說“*（晚上好，世界）*”。

```
If (Clock.Hour <12) Then
  TextWindow.WriteLine("Good Morning World")
EndIf
If (Clock.Hour >= 12) Then
  TextWindow.WriteLine("Good Evening World")
EndIf
```

根據你運行程式的時間，你將會看到下麵的某一個輸出。

圖 14 - Good Morning World（早上好，世界）

圖 15 - Good Evening World（晚上好，世界）

讓我們來分析一下程式的前三行。你肯定已經發現這幾行程式告訴電腦如果 Clock.Hour 小於 12 ，就輸出"Good Morning World.（早上好，世界。）"。**If**， **Then** 和 **EndIf** 是在程式運行過程中電腦能夠理解的特殊語言。**If** 後面總是會跟一個條件判斷。在這個例子中是（**Clock.Hour < 12**）。記住，為了讓電腦理解你的意圖，小括弧在這裡是必須的。條件判斷後面跟的是 **then** 和要執行的實際操作。實際操作後面跟的是 **EndIf** ， 這告訴電腦這個有條件的執行過程結束了。

在 Small Basic 中，你可以用 Clock 物件來訪問當前的日期和時間。它還提供了一組屬性，使你可以分別得到當前的日、月、年、小時、分鐘、秒。

在 **then** 和 **EndIf** 之間可能會有不只一個操作。如果條件符合，電腦將會把它們全部執行。例如，你可以寫如下程式：

```
If (Clock.Hour <12) Then
  TextWindow.Write("Good Morning.")
  TextWindow.WriteLine("How was breakfast?")
EndIf
```

Else

在這一章最開始的程式中，你可能已經注意到第二個條件有點多餘。**Clock.Hour** 的值只可能小雨或者大於 12。 我們不需要真正檢查第二個條件。對於這種情況，我們可以通過，把兩個 **if..then..endif** 語句縮短為一句。

如果我們用 **else** 重寫，你將看到的下麵的程式：

```
If (Clock.Hour <12) Then
  TextWindow.WriteLine("Good Morning World")
Else
  TextWindow.WriteLine("Good Evening World")
EndIf
```

這個程式將和最開始的程式做完全同樣的事情。這裡我們學到了電腦程式設計中非常重要的一課：

> *在程式設計過程中，我們經常可以通過很多方法做同樣的事情。很多時候，一種方法比其它的方法更有效。選擇哪一種方法是由程式師決定的。當你編寫了更多的程式獲得了更多的經驗後，你就可以瞭解不同的技術以及它們的優缺點。*

Indentation（縮進）

在所有的程式中，你都可以看到 ***If, Else*** 和 ***EndIf*** 是如何縮進的。縮進不是必須的，但它將說明你更容易的理解程式的結構。因此，在區塊間縮進語句通常是一個很好的習慣。

Even（奇數）或者 Odd（偶數）

現在我們已經學會了 **If..Then..Else..EndIf** 語句。讓我們寫一個程式。 當你給出一個數位時，它可以判斷是奇數還是偶數。

```
TextWindow.Write("Enter a number: ")
num = TextWindow.ReadNumber()
remainder = Math.Remainder(num, 2)
If (remainder = 0) Then
  TextWindow.WriteLine("The number is Even")
Else
  TextWindow.WriteLine("The number is Odd")
EndIf
```

運行這個程式，你將能看到它的輸出如下：

圖 16-奇數或偶數

在這個程式中，我們引入了一個新的很有用的操作，**Math.Remainder** 。是的，正如你已經看到的，**Math.Remainder** 將返回第一個數除以第二個數的餘數。

Branching（分支）

還記得嗎？在第二章中，你知道了電腦按從上到下的順序一次處理常式中的一個語句。然而，有一種語句可以使電腦不按照循序執行，而跳到另一條語句。我們看一下下麵的程式。

```
i = 1
start:
TextWindow.WriteLine(i)
i = i + 1
If (i <25) Then
  Goto start
EndIf
```

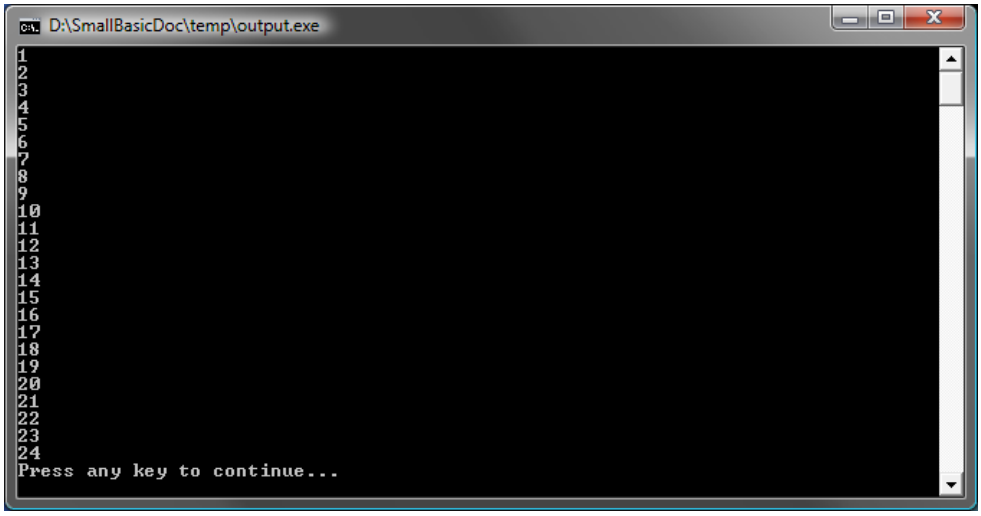

图 17-使用 Goto

在上面的程式中，我們把 1 付給變數 **i** 。接著，我們加了一個新的語句，它以冒號結束（：）

```
start:
```

這被稱作一個標記。標記就像是一個電腦可以理解的書簽。只要你能保證它們的命名是唯一的，你可以任意命名一個標記，而且可以在程式中加入任意多的標記。

另一個有趣的語句是：

```
i = i + 1
```

這句話告訴電腦把變數 **i** 加 1，然後再把結果付回給 **i** 。所以如果原來 **i** 的值是 1，在執行這句話之後，**i** 的值將變成 2。

最後，

```
If (i <25) Then
  Goto start
EndIf
```

這部分程式告訴電腦，如果 **i** 的值小於 25，就開始從標記 **start** 執行語句。

Endless execution（無限執行）

使用 **Goto** 語句，你可以讓電腦任意多次的執行一些操作。例如，你可以把奇數或偶數的程式改寫成下麵的樣子。程式將永遠執行。你可以通過點擊命令框右上角的 Close（X）來終止程式。

```
begin:
TextWindow.Write("Enter a number: ")
num = TextWindow.ReadNumber()
remainder = Math.Remainder(num, 2)
If (remainder = 0) Then
  TextWindow.WriteLine("The number is Even")
Else
  TextWindow.WriteLine("The number is Odd")
EndIf
Goto begin
```

圖 18-奇數或偶數程式無限執行

第 5 章

迴圈

For 迴圈

讓我們再以前面章節用到的代碼來舉例說明。

```
i = 1
start:
TextWindow.WriteLine(i)
i = i + 1
If (i <25) Then
  Goto start
EndIf
```

這段代碼的運行結果是：按順序列印數位 1 到 24。在提供了便捷的變數遞增方法的程式設計語言中，這樣的變數遞增過程是很常見的。以上這段程式與下麵的程式輸出結果相同：

```
For i = 1 To 24
  TextWindow.WriteLine(i)
EndFor
```

輸出結果是：

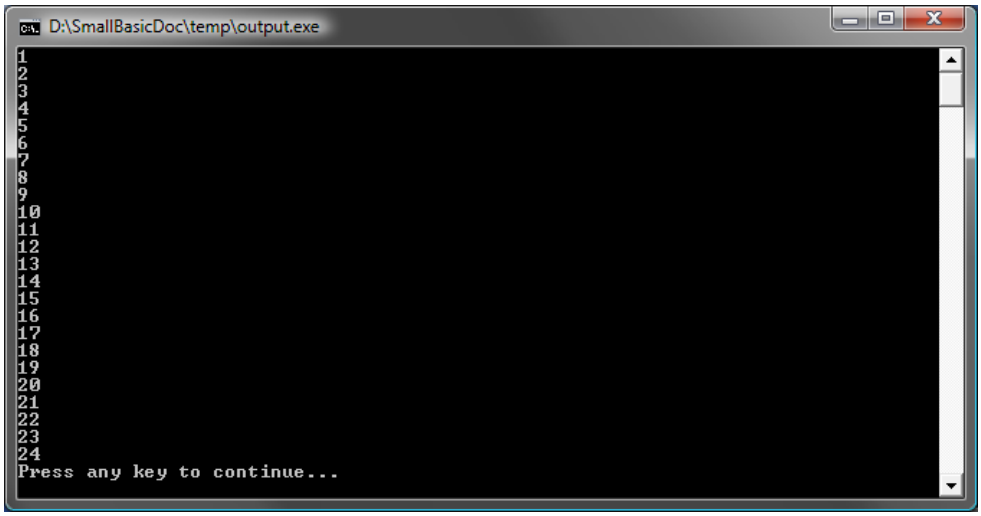

圖 19-使用 For 迴圈

請注意，我們將 8 行的程式減少到 4 行，而實際運行結果與 8 行的程式完全相同！還記得前面我們提到的"完成一個任務通常會有多種方法"嗎？這就是一個典型的例子。

For..EndFor 在程式設計術語中被稱為一個*循环*。在迴圈中可以使用一個變數，給變數賦以初值和終值，並讓電腦自動執行變數遞增。變數每遞增一次，電腦便執行一次 **For** 到 **EndFor** 之間的程式。

如果希望變數以 2（而不是以 1）為基準遞增，例如，要輸出 1 至 24 之間的所有奇數，即可以使用迴圈來實現。

```
For i = 1 To 24 Step 2
  TextWindow.WriteLine(i)
EndFor
```

圖 20-僅輸出奇數

For 語句中的 **Step 2** 告訴電腦以 2 為基數對變數 **i** 遞增（而不是以 1）。通過使用 **Step** 可以根據需要指定任意遞增基數。甚至可以指定負增量來進行倒數，示例如下：

```
For i = 10 To 1 Step -1
  TextWindow.WriteLine(i)
EndFor
```

图 21-倒数

While 迴圈

While 迴圈是另一種迴圈方法，該方法在無法預測迴圈次數的情況下非常實用。前面介紹的 For 迴圈按照預先定義的次數執行程式，而 While 迴圈則是在指定條件為真的情況下重複運行程式。以下示例中，我們將在指定變數大於 1 的情況下，對變數進行等分。

```
number = 100
While (number >1)
  TextWindow.WriteLine(number)
  number = number / 2
EndWhile
```

圖 22-等分迴圈

在以上程式中，對變數 *number* 賦以初值 100，在 *number* 大於 1 的情況下重複運行 While 迴圈。在迴圈中，輸出 *number* 的值，再除以 2，對其進行有效地等分。正如預期，程式的輸出結果為逐步被等分的序列數位。

因為我們無法事先得知程式要迴圈運行的次數，如果用 For 迴圈實現以上程式就會相當困難。通過使用 While 迴圈，可以方便地檢查條件，再確定是否要繼續執行迴圈或退出。

有趣的是，每個 While 迴圈都可以分解成一組 If..Then 語句。例如，可以將以上程式重寫如下，其輸出結果相同：

```
number = 100
startLabel:
TextWindow.WriteLine(number)
number = number / 2

If (number >1) Then
  Goto startLabel
EndIf
```

實際上，電腦內部會將每個 While 迴圈重寫成一組帶有一個或多個 Goto 語句的 If..Then 語句。

第 6 章

圖形入門

目前為止，在所有例子中，我們都用 TextWindow 解釋 Small Basic 語言的基礎知識。然而，Small Basic 還有一組強大的圖形功能。我們將在這章中學習。

GraphicsWindow 引言

就如 TextWindow 使我們可以處理 Text （文本）和 Numbers（數位）, Small Basic 提供了 **GraphicsWindow**，我們可以用它來畫圖。讓我們從顯示 GraphicsWindow 開始。

```
GraphicsWindow.Show()
```

當你運行這個程式的時候，你會發現你得到的是一個下圖所示的白色 Window（表單），而不是一個通常情況下的空的文本表單。這個表單本身沒什麼用，但這章你所有的工作都將基於它。你可以通過右上角的'X'來關閉這個視窗。

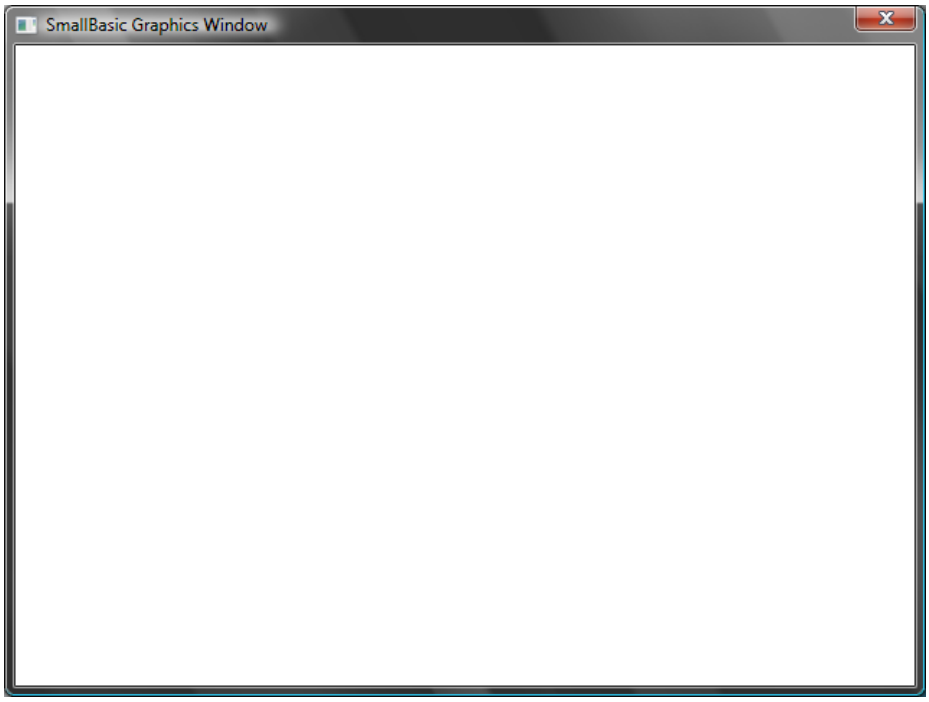

圖 23-一個空的 Graphics Window

設置 Graphics Window

這個圖形視窗允許你根據自己的需要調整它的外觀。你可以改變它的標題、背景和大小。讓我們繼續對它進行一些修改。這將說明我們熟悉表單。

```
GraphicsWindow.BackgroundColor = "SteelBlue"
GraphicsWindow.Title = "My Graphics Window"
GraphicsWindow.Width = 320
GraphicsWindow.Height = 200
GraphicsWindow.Show()
```

下麵是一個個性化的圖形視窗的樣子。你可以把它的背景顏色改變成附錄 B 中列出的任意一種顏色。試試這些屬性看看表單的外觀是如何改變的。

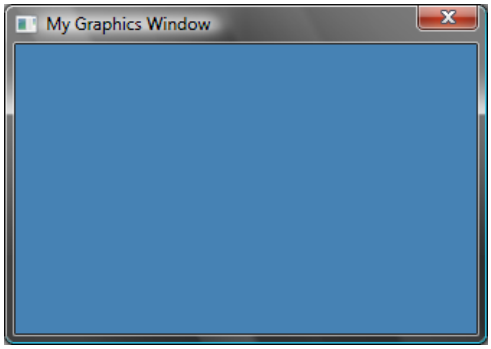

圖 24–一個 Custom Graphics Window（個性化圖形表單）

畫線

一旦你有了 GraphicsWindow，我們可以在它上面畫圖形、文本甚至是圖片。讓我們先畫一些簡單的圖形。下麵是在 Graphics Window 上畫兩條線的程式。

```
GraphicsWindow.Width = 200
GraphicsWindow.Height = 200
GraphicsWindow.DrawLine(10, 10, 100, 100)
GraphicsWindow.DrawLine(10, 100, 100, 10)
```

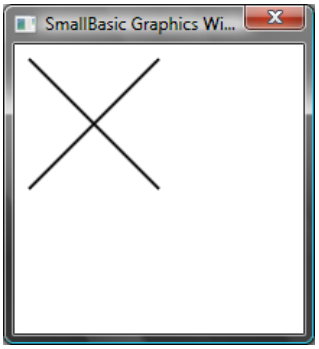

圖 25–交叉線

這兩行程式設置了表單並且畫了兩條交叉線。*DrawLine* 後面緊跟的前兩個數字指定了線開始端的 x，y 座標；另外兩個數字指定了線結束端的 x，y 座標。電腦圖形中有趣的一件事是

我們使用 web 顏色符號（(#RRGGBB）代表顏色，而不是用名字。例如，#FF0000 代表紅色，#FFFF00 代表黃色，依此類推。我們將會學更多的關於顏色的知識。[TODO Colors chapter]

（0，0）座標代表了表單的左上角。實際上，表單被認為是在坐標系的第二象限。

TODO: 插入象限圖]

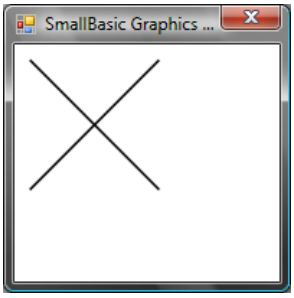

圖 26-座標圖

如果我們回到畫線的程式，有趣的是 Small Basic 允許你修改線的屬性。比如顏色和它的寬度。首先，我們用下麵的程式改變線的顏色。

```
GraphicsWindow.Width = 200
GraphicsWindow.Height = 200
GraphicsWindow.PenColor = "Green"
GraphicsWindow.DrawLine(10, 10, 100, 100)
GraphicsWindow.PenColor = "Gold"
GraphicsWindow.DrawLine(10, 100, 100, 10)
```

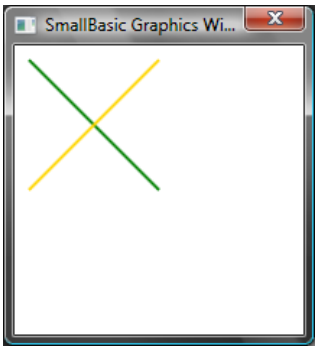

圖 27-改變線的顏色

現在，我們改變大小。在下麵的程式中，我們把線的寬度從預設的 1 改為 10。

```
GraphicsWindow.Width = 200
GraphicsWindow.Height = 200
GraphicsWindow.PenWidth = 10
```

```
GraphicsWindow.PenColor = "Green"
GraphicsWindow.DrawLine(10, 10, 100, 100)
GraphicsWindow.PenColor = "Gold"
GraphicsWindow.DrawLine(10, 100, 100, 10)
```

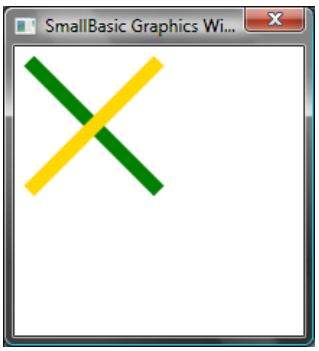

圖 28-粗的彩色線

PenWidth 和*改變畫線的筆*。它們不僅影響線，還會影響屬性改變後畫的所有圖形。

使用上一章我們學的迴圈語句，我們可以輕鬆的寫一個程式。這個程式可以畫很多條線。畫線用的筆會越來越粗。

```
GraphicsWindow.BackgroundColor = "Black"
GraphicsWindow.Width = 200
GraphicsWindow.Height = 160
GraphicsWindow.PenColor = "Blue"

For i = 1 To 10
  GraphicsWindow.PenWidth = i
  GraphicsWindow.DrawLine(20, i * 15, 180, i * 15)
endfor
```

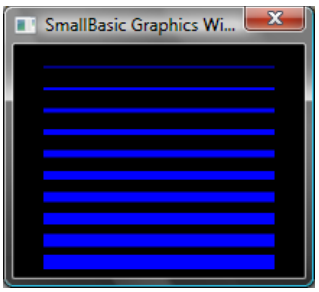

圖 29-不同的筆的寬度

這個程式有趣的部分是迴圈語句。每迴圈一次，我們加大筆的寬度，然後在原來的線下麵畫一條新線。

畫圖和填充圖形

說到畫圖形，對於每一個圖形通常都有兩類操作。它們分別是畫操作和填充操作。畫操作用筆劃出圖形的輪廓；填充操作用刷子為圖形上色。例如在下麵的程式中，有兩個矩形。一個是用紅筆劃出來的，另一個是用綠色的刷子填充的。

```
GraphicsWindow.Width = 400
GraphicsWindow.Height = 300

GraphicsWindow.PenColor = "Red"
GraphicsWindow.DrawRectangle(20, 20, 300, 60)

GraphicsWindow.BrushColor = "Green"
GraphicsWindow.FillRectangle(60, 100, 300, 60)
```

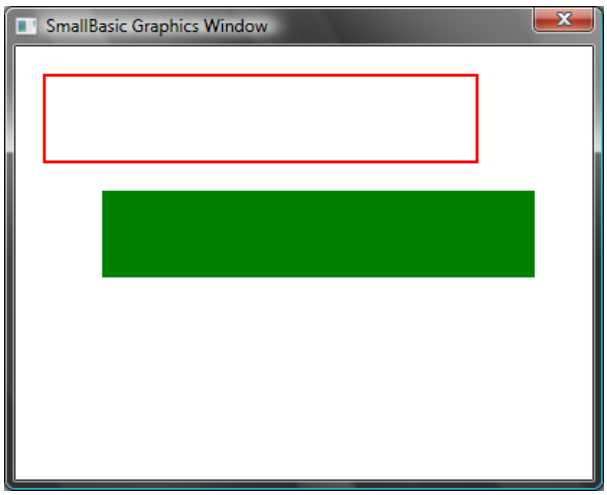

圖 30 畫圖和填充

要畫和填充一個矩形，你需要四個數字。前兩個代表矩形左上角的 X 和 Y 座標。第三個數字代表矩形的寬度，第四個代表它的高度。 實際上，畫和填充橢圓也一樣。看下麵的程式。

```
GraphicsWindow.Width = 400
GraphicsWindow.Height = 300

GraphicsWindow.PenColor = "Red"
GraphicsWindow.DrawEllipse(20, 20, 300, 60)
```

```
GraphicsWindow.BrushColor = "Green"
GraphicsWindow.FillEllipse(60, 100, 300, 60)
```

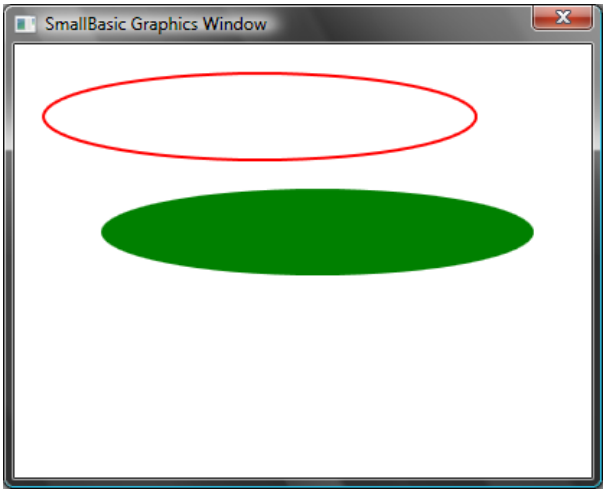

圖 31-畫和填充橢圓

橢圓形是廣義上的圓形。如果你像畫圓形，你要定義相同的寬度和高度

```
GraphicsWindow.Width = 400
GraphicsWindow.Height = 300

GraphicsWindow.PenColor = "Red"
GraphicsWindow.DrawEllipse(20, 20, 100, 100)

GraphicsWindow.BrushColor = "Green"
GraphicsWindow.FillEllipse(100, 100, 100, 100)
```

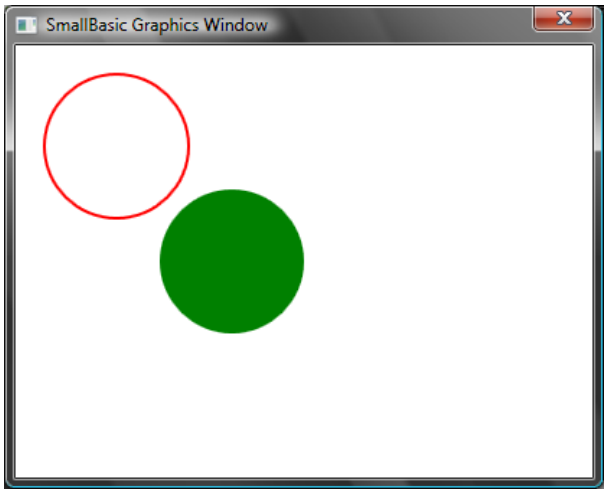

图 32-圆形

第 7 章

繪製圖形

在這一章節中，我們將運用前面學到的知識來做一些有趣的事。在本章的示例中，我們將向你展示幾個有趣的方法，讓你可以結合目前所學到的全部知識來創建看上去非常酷的應用程式。

矩形

以下的示例利用一個迴圈語句繪製一組大小逐漸遞增的矩形。

```
GraphicsWindow.BackgroundColor = "Black"
GraphicsWindow.PenColor = "LightBlue"
GraphicsWindow.Width = 200
GraphicsWindow.Height = 200

For i = 1 To 100 Step 5
  GraphicsWindow.DrawRectangle(100 - i, 100 - i, i * 2, i * 2)
EndFor
```

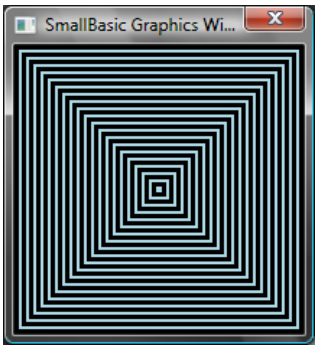

圖 33-矩形組

圓形

在前面程式的基礎上加以改變，繪製一組圓形（而不是矩形）。

```
GraphicsWindow.BackgroundColor = "Black"
GraphicsWindow.PenColor = "LightGreen"
GraphicsWindow.Width = 200
GraphicsWindow.Height = 200

For i = 1 To 100 Step 5
  GraphicsWindow.DrawEllipse(100 - i, 100 - i, i * 2, i * 2)
EndFor
```

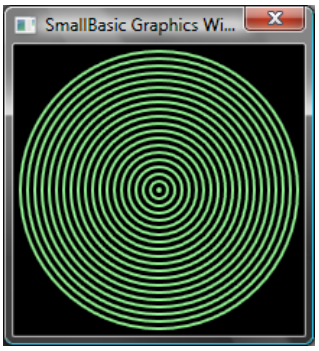

圖 34-圓形組

隨機的圖案

以下程式利用 *GetRandomColor* 為畫筆設置隨機顏色，在使用 *GetRandomNumber* 設置圓形的 x 和 y 座標。這個有趣的程式將兩種運算巧妙地結合，每次運行時都可以繪製出不同的圖案。

```
GraphicsWindow.BackgroundColor = "Black"
For i = 1 To 1000
  GraphicsWindow.BrushColor = GraphicsWindow.GetRandomColor()
  x = Math.GetRandomNumber(640)
  y = Math.GetRandomNumber(480)
  GraphicsWindow.FillEllipse(x, y, 10, 10)
EndFor
```

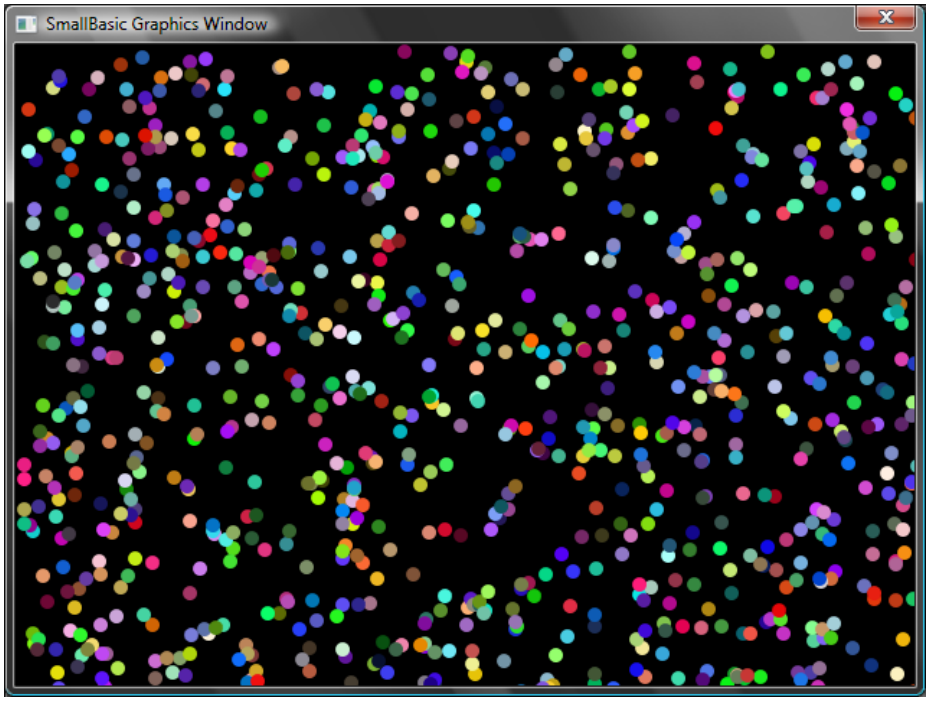

圖 35-隨機的圖案

分形

以下程式使用亂數繪製一個簡單的三角分形。分形是一種幾何圖形，它可以被分割成多個子圖形，其中每個子圖形都與該分形形狀完全相同。在這個示例中，程式將繪製一個由幾百個小三角形組成的大三角形，其中每個小三角形都與大三角形完全相同。由於這個程式需要運行幾秒鐘，我們可以在程式運行時看到三角形從小點開始，逐漸形成大三角形的過程。程式本身的邏輯很難描述，大家可以嘗試研究一下。

```
GraphicsWindow.BackgroundColor = "Black"
x = 100
y = 100
```

```
For i = 1 To 100000
  r = Math.GetRandomNumber(3)
  ux = 150
  uy = 30
  If (r = 1) then
    ux = 30
    uy = 1000
  EndIf

  If (r = 2) Then
    ux = 1000
    uy = 1000
  EndIf

  x = (x + ux) / 2
  y = (y + uy) / 2

  GraphicsWindow.SetPixel(x, y, "LightGreen")
EndFor
```

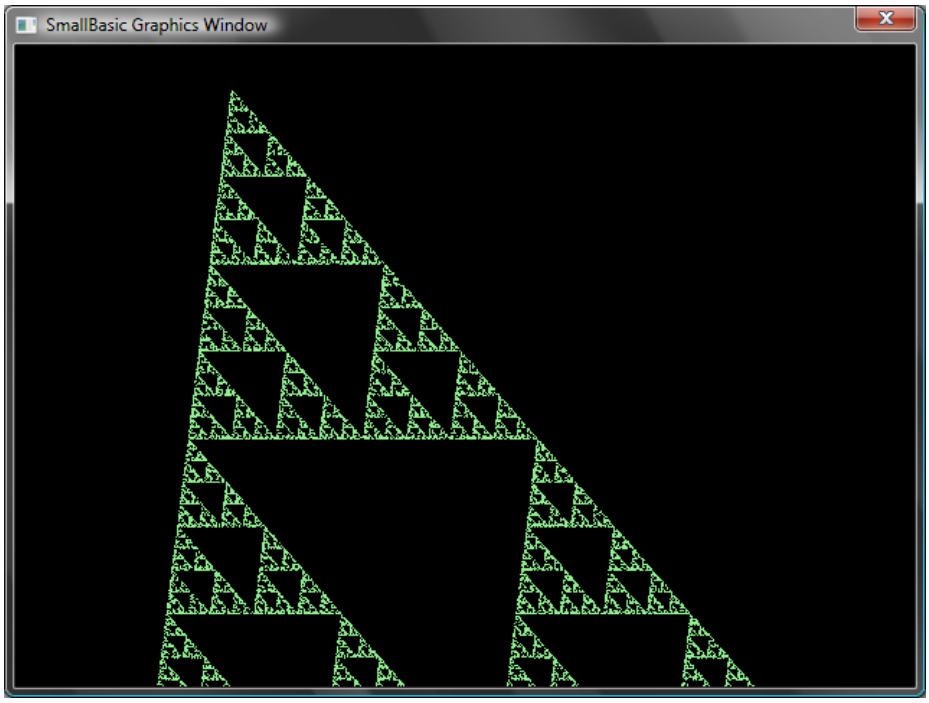

圖 36-三角分形

如果你想認真觀察小三角形逐漸形成三角分形的過程，可以在迴圈中引入 ***Delay*** 延遲。這個運算通過讀取指定的數值（以毫秒為單位），延遲程式的執行時間。以下為修改後的程式，修改的代碼以粗體顯示。

```
GraphicsWindow.BackgroundColor = "Black"
x = 100
y = 100

For i = 1 To 100000
  r = Math.GetRandomNumber(3)
  ux = 150
  uy = 30
  If (r = 1) then
    ux = 30
    uy = 1000
  EndIf

  If (r = 2) Then
    ux = 1000
    uy = 1000
  EndIf

  x = (x + ux) / 2
  y = (y + uy) / 2

  GraphicsWindow.SetPixel(x, y, "LightGreen")
  Program.Delay(2)
EndFor
```

通過增加延遲時間，可以讓程式執行得更慢。你可以將延遲時間調整到自己喜歡的數值。

我們還可以對程式進行其它修改，將以下代碼：

```
GraphicsWindow.SetPixel(x, y, "LightGreen")
```

替換為：

```
color = GraphicsWindow.GetRandomColor()
GraphicsWindow.SetPixel(x, y, color)
```

修改後的程式將以隨機顏色繪製三角形。

第 8 章

Turtle Graphics（海龜繪圖法）

Logo 語言

二十世紀 70 年代，有一種叫做 Logo 的程式設計語言，它非常簡單但功能強大，當時被一些研究人員所使用。後來開發人員在這種程式設計語言中加入了現在被稱為“Turtle Graphics（海龜繪圖法）”的技術，並在螢幕上顯示一個“海龜”圖案 ，通過諸如 *（向前移動）*，*Turn Right（向右轉）*，*Turn Left（向左转）*等命令移動龜標。通過控制龜標的移動，程式設計人員可以繪製有趣的圖形。這一新技術的引入，使得 Logo 變得更加簡單易用，並對各個年齡的程式設計愛好者產生了吸引力，因此在 80 年代廣為流行。

Small Basic 語言中也引入了 **Turtle** （龜標）物件，程式設計人員可以通過 Small Basic 自帶的很多命令調用這個物件。在本章中，我們將使用龜標繪製圖形。

Turtle

程式初始化時，我們需要在螢幕上顯示龜標。代碼如下：

```
Turtle.Show()
```

程式運行後，將顯示一個白色視窗，這個視窗與我們在前面章節中看到的視窗唯一的不同就是視窗的中央多了一個龜標。這個龜標將會按照我們的指令繪製指定的圖案。

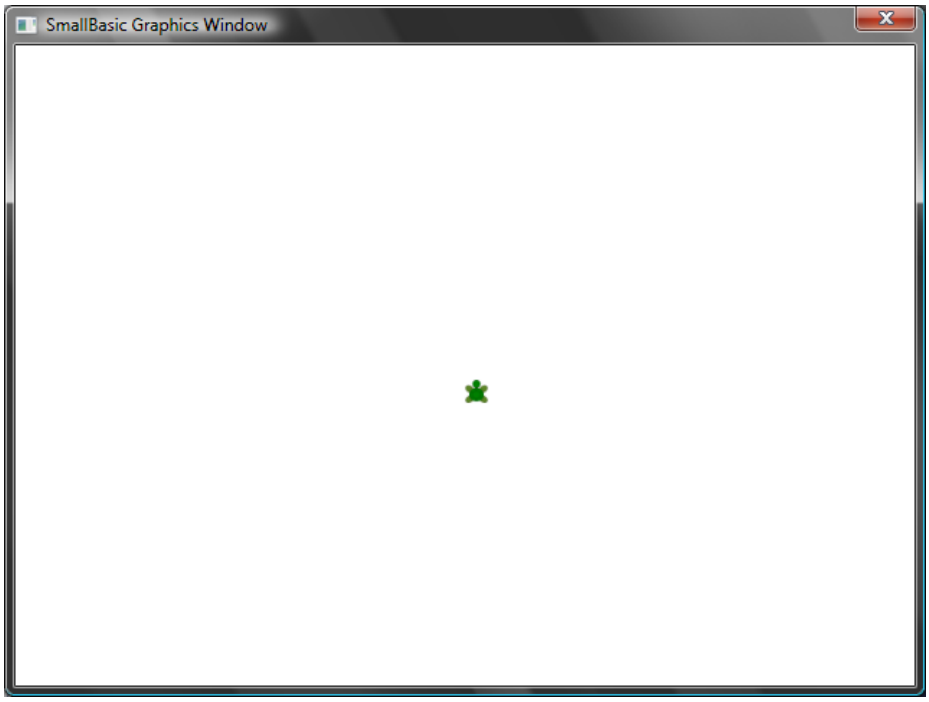

圖 37-視窗中央顯示有龜標

移動和繪製

Move（移動）是這只"海龜"可以聽懂的命令之一。這個運算會讀取使用者輸入的一個數值。該數值即是龜標要移動的距離。例如，在下麵的代碼中，我們指定龜標移動 100 象素。

```
Turtle.Move(100)
```

程式運行後，可以看到龜標緩慢地向上移動 100 象素，並在它經過的地方畫下一條線。當龜標移動到指定位置後，將畫出以下圖案。

使用龜標繪製圖形時，可以不調用 Show()。電腦在執行操作時會自動顯示龜標。

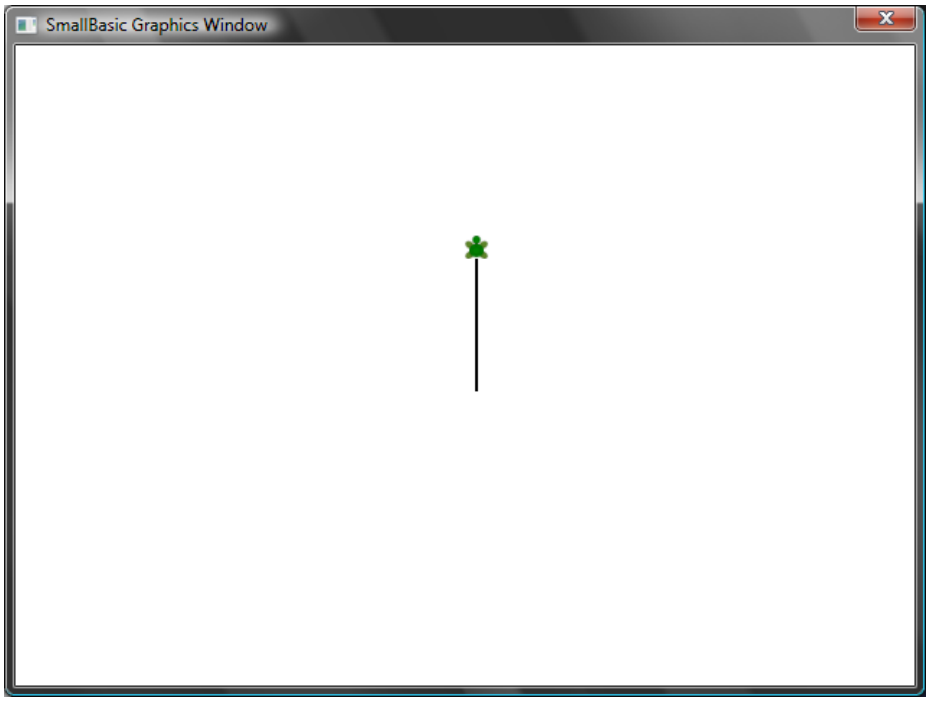

圖 38-移動 100 象素

繪製正方形

正方形由四條邊組成，兩條為水準方向，兩條為垂直方向。因此，要繪製一個正方形，我們需要用龜標畫一條線，右轉，再畫一條線，然後重複同樣的動作直至畫完四條邊。我們可以將這些動作轉換成以下代碼。

```
Turtle.Move(100)
Turtle.TurnRight()
Turtle.Move(100)
Turtle.TurnRight()
Turtle.Move(100)
Turtle.TurnRight()
Turtle.Move(100)
Turtle.TurnRight()
```

程式運行後，龜標將會逐條邊繪製出一個正方形，其結果如下圖。

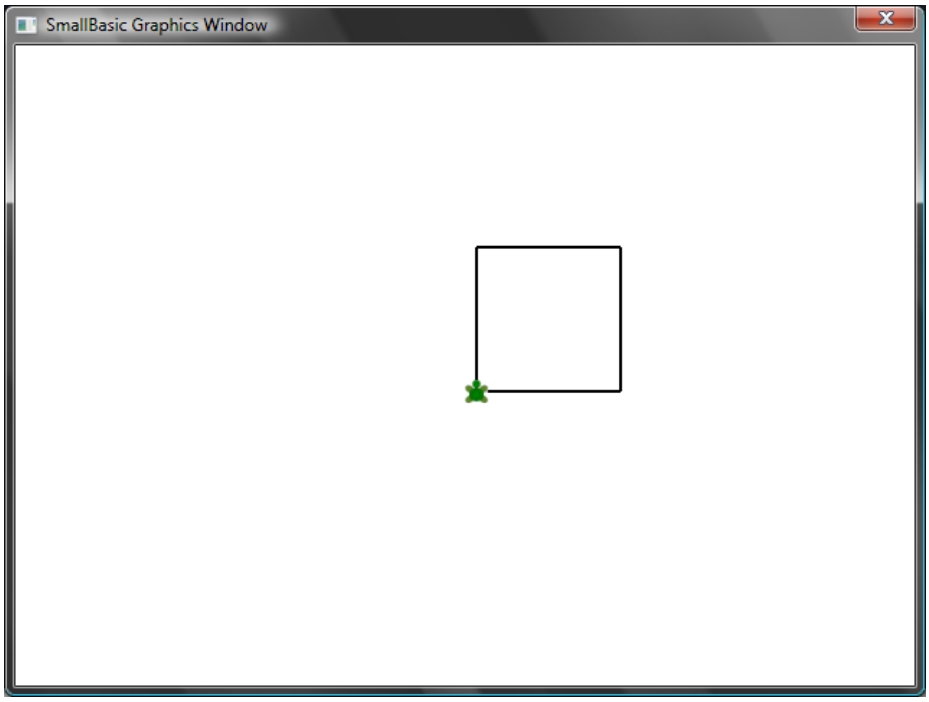

圖 39-龜標繪製出一個正方形

如果用心觀察，就會發現我們在一遍又一遍地使用同樣的指令，確切地說是重複使用了四次。我們在前面介紹過，可以使用迴圈來實現像這樣的重複命令。拿上面的這個程式來舉例，如果用 **For..EndFor** 迴圈來實現，程式就會變得簡單很多。

```
For i = 1 To 4
  Turtle.Move(100)
  Turtle.TurnRight()
EndFor
```

修改顏色

這個示例中龜標所在的 GraphicsWindow （圖畫視窗）和我們在前一章中所看到的相同。也就是說，前面所學的運算都可以在這裡使用。例如，以下這段代碼可以繪製一個每條邊顏色不同的方形。

```
For i = 1 To 4
  GraphicsWindow.PenColor = GraphicsWindow.GetRandomColor()
  Turtle.Move(100)
  Turtle.TurnRight()
EndFor
```

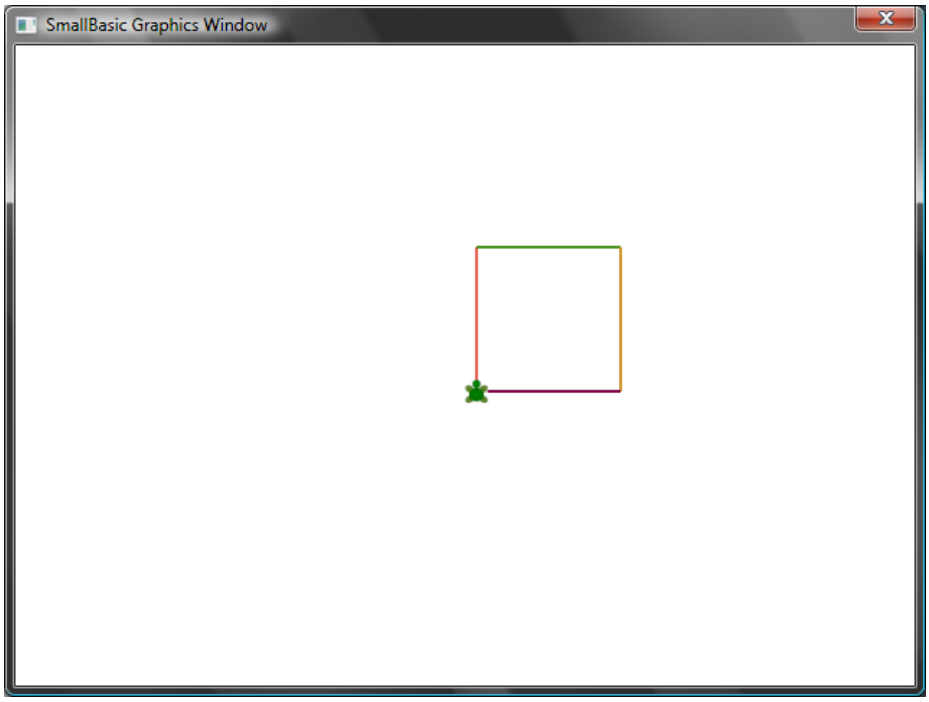

圖 40-修改顏色

繪製複雜圖形

龜標除了可以執行 **TurnRight** （向左轉）和 **TurnLeft** （向右轉）運算以外，還可以執行 **Turn**（轉向）運算。這個運算會按照使用者指定的數值將龜標旋轉相應的角度。使用這個運算，我們就可以繪製任意形狀的多邊形。以下程式可以繪製一個六邊形。

```
For i = 1 To 6
  Turtle.Move(100)
  Turtle.Turn(60)
EndFor
```

運行一下這個程式，看看它的輸出是否真的是一個六邊形。因為邊與邊的夾角是 60 度，我們就使用 **Turn(60)**。對於這樣每個邊都等長的多邊形，邊與邊之間的夾角即是 360 除以多邊形的邊數。我們可以利用這個原理並通過使用變數，實現一個可以繪製任意邊數的多邊形的通用程式。

```
sides = 12

length = 400 / sides
angle = 360 / sides
```

```
For i = 1 To sides
  Turtle.Move(length)
  Turtle.Turn(angle)
EndFor
```

使用這個程式，我們可以通過改變 **sides** （邊數）變數繪製任意多邊形。如果將變數設置為 4，將會繪製出和我們開始編寫的那個程式一樣的四邊形。如果將變數設置為一個很大的數值，比如 50，將會繪製出的一個接近圓形的圖案。

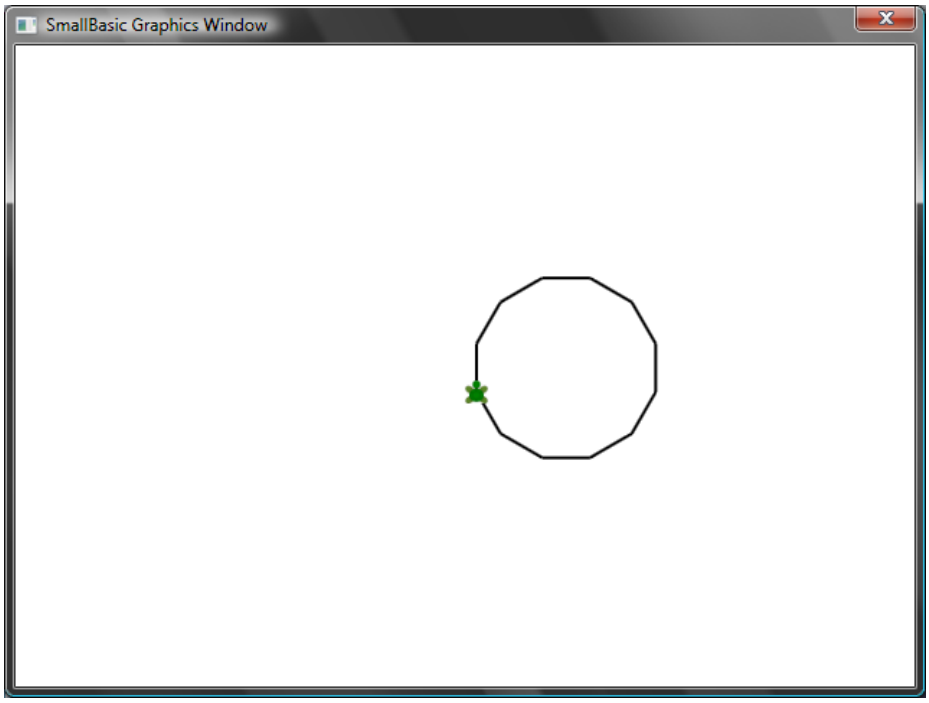

圖 41-繪製一個十二邊形

使用以上技術，利用龜標繪製多個圓形，並在繪製每個圓形之後移動一段距離，我們將會得到一個有趣的輸出結果。

```
sides = 50
length = 400 / sides
angle = 360 / sides

Turtle.Speed = 9

For j = 1 To 20
  For i = 1 To sides
    Turtle.Move(length)
```

```
    Turtle.Turn(angle)
  EndFor
  Turtle.Turn(18)
EndFor
```

以上程式中有兩個嵌套的 **For..EndFor** 迴圈，內層的迴圈 *i = 1 to sides*) 和多邊形程式相似，負責輸出圓形。外層的迴圈 *j = 1 to 20*) 負責在每次繪製圓形後移動龜標。在程式中，我們指定使用龜標繪製 20 個圓形。程式的運行結果是一個如下所示的有趣圖案。

在以上程式中，通過將 Speed （速度）設置為 9，可以讓"海龜"走得快一點兒。你也可以通過修改這個屬性設置讓"海龜"按照你喜歡的速度移動。

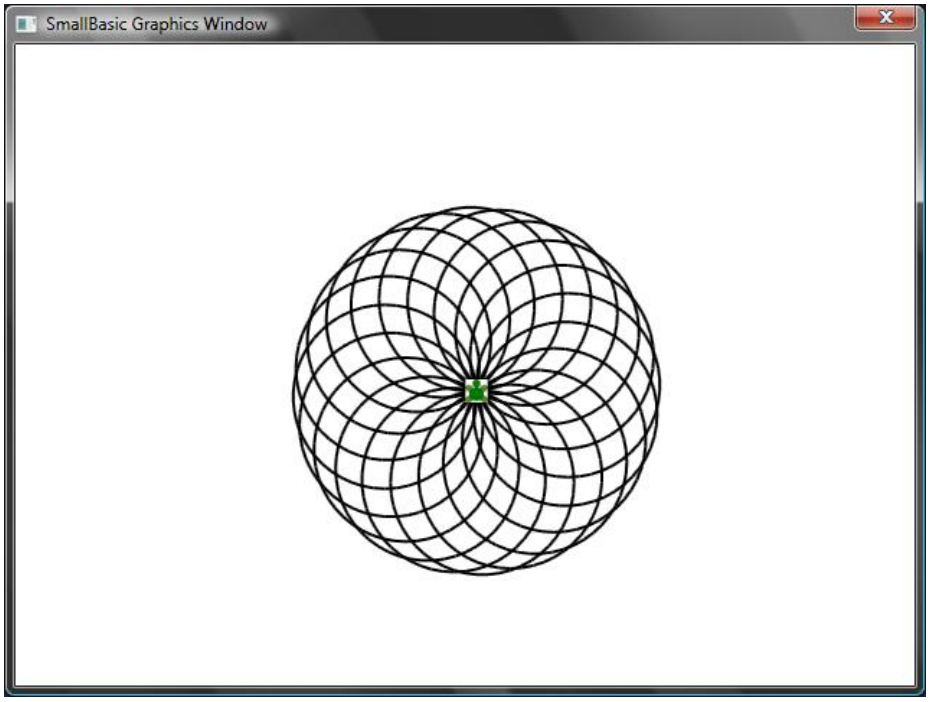

圖 42-在圓形中移動

自由移動

可以使用 **PenUp**（停筆）運算讓龜標停止移動。這樣，就可以將龜標移動到螢幕的任意位置，而不留下痕跡。如果想重新使用龜標繪製圖案，可以調用 **PenDown**（落筆）。通過使用這兩個命令，可以繪製出像虛線這樣的有趣效果。以下程式就使用這個方法繪製出一個以虛線組成的多邊形。

```
sides = 6

length = 400 / sides
angle = 360 / sides

For i = 1 To sides
  For j = 1 To 6
    Turtle.Move(length / 12)
    Turtle.PenUp()
    Turtle.Move(length / 12)
    Turtle.PenDown()
  EndFor
  Turtle.Turn(angle)
EndFor
```

和上面的程式一樣，這個程式也有兩個嵌套的迴圈，內部的迴圈繪製虛線，外部的迴圈根據指定數值繪製多邊形的邊。在這個的示例中，我們指定 **sides**（邊數）變數為 6，得到一個由虛線組成的六邊形，如下所示。

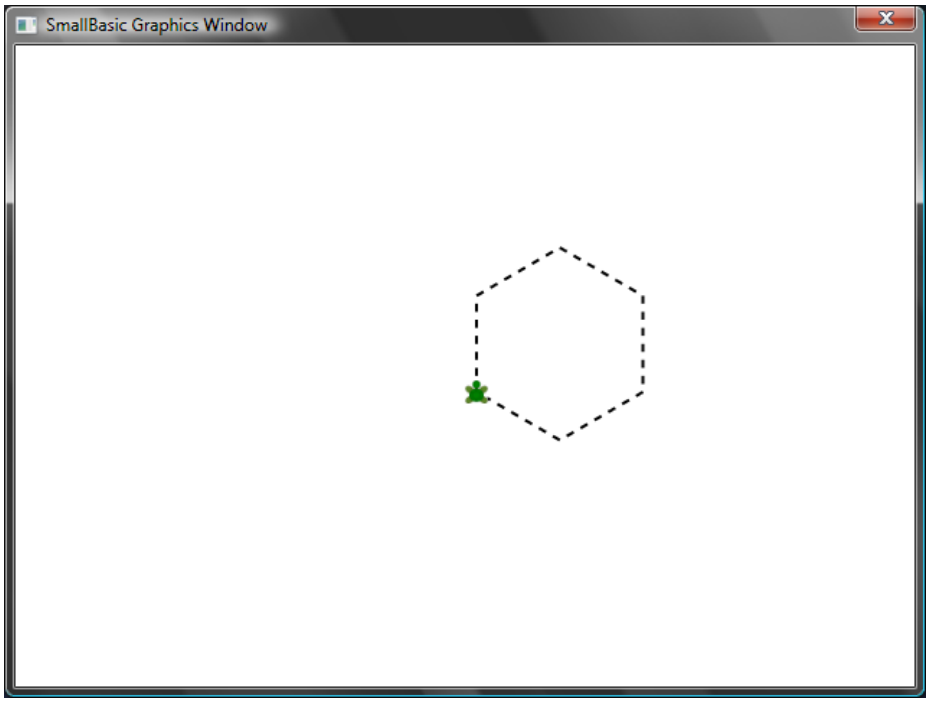

图 43-使用 PenUp 和 PenDown

第九章

子常式（Subroutines）

在程式設計時我們經常遇到需要重複執行相同一組步驟的情況。這種情況下，多次重寫相同的語句是不合理的。這時我們需要借助子常式。

一個子常式是一個較大的程式中實現特定功能的一小段代碼，子常式可以在程式的任意位置被調用。子常式由一個以 **Sub** 關鍵字開頭的名字標識並以 **EndSub** 關鍵字結束。例如，下麵的程式碼片段表示了一個叫做 *PrintTime*（列印時間）的子常式，它可以把當前時間顯示到 TextWindow（文本視窗）中。

```
Sub PrintTime
  TextWindow.WriteLine(Clock.Time)
EndSub
```

下麵的套裝程式含了該子常式並從多個地方調用它。

```
PrintTime()
TextWindow.Write("Enter your name: ")
name = TextWindow.Read()
TextWindow.Write(name + ", the time now is: ")
PrintTime()

Sub PrintTime
  TextWindow.WriteLine(Clock.Time)
```

```
EndSub
```

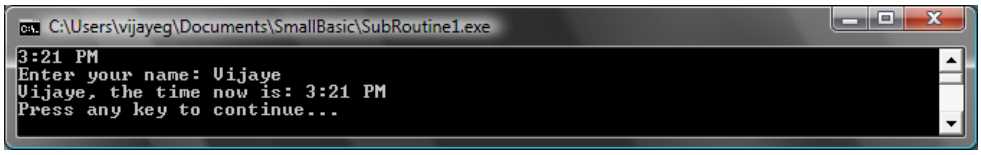

圖 44 -調用一個簡單子常式

您可以通過調用 *SubroutineName()* 來執行子常式。通常地，符號?)?對於告訴電腦您打算執行一個子常式是必需的。

使用子常式的好處

如我們在上邊看到的，子常式說明我們減少了大量程式設計時輸入的代碼。一旦子常式 *PrintTime* 寫好了，您就可以從程式的任意地方調用它來顯示當前時間。

請記住，您只能在同一個程式中調用 SmallBasic 子常式。您不能從其他程式中調用該子常式。

此外，子常式說明我們將複雜問題簡化為較簡單的片段。比方說您有一個複合公式要求解，您可以編寫幾個子常式先求解該複合公式的較小的部分。然後您可以把結果合併起來求解原始的複合公式。

子常式還能說明提高程式的可讀性。換句話說，如果您的程式中公用部分都是名稱清楚的子常式，您的程式就會變得容易讀和理解。如果您想理解其他人寫的程式或者您希望您的程式能被其他人理解，子常式尤為重要。有時，子常式對於您理解您自己的程式也會有說明，比方您一周前寫的程式。

使用變數（variables）

您可以在子常式內訪問和使用程式中的任何變數。作為例子，下麵的程式接受兩個數並列印出較大的那個數。請注意變數 x（最大）同時在子常式內部和外部被使用。

```
TextWindow.Write("Enter first number: ")
num1 = TextWindow.ReadNumber()
TextWindow.Write("Enter second number: ")
num2 = TextWindow.ReadNumber()

FindMax()
TextWindow.WriteLine("Maximum number is: "+ max)
```

```
Sub FindMax
  If (num1 >num2) Then
    max = num1
  Else
    max = num2
  EndIf
EndSub
```

程式輸出如下所示。

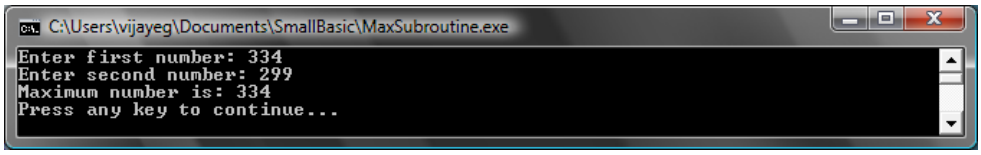

圖 45 -使用子常式得到較大數

讓我們看另一個例子以說明子常式的用法。這次我們使用一個圖形程式計算以變數 *x* 和 *y* 存儲的點。然後調用一個子常式 DrawCircleUsingCenter 來繪製以 *x* 和 *y* 為圓心的圓。

```
GraphicsWindow.BackgroundColor = "Black"
GraphicsWindow.PenColor = "LightBlue"
GraphicsWindow.Width = 480
For i = 0 To 6.4 Step 0.17
  x = Math.Sin(i) * 100 + 200
  y = Math.Cos(i) * 100 + 200

  DrawCircleUsingCenter()
EndFor

Sub DrawCircleUsingCenter
  startX = x - 40
  startY = y - 40

  GraphicsWindow.DrawEllipse(startX, startY, 120, 120)
EndSub
```

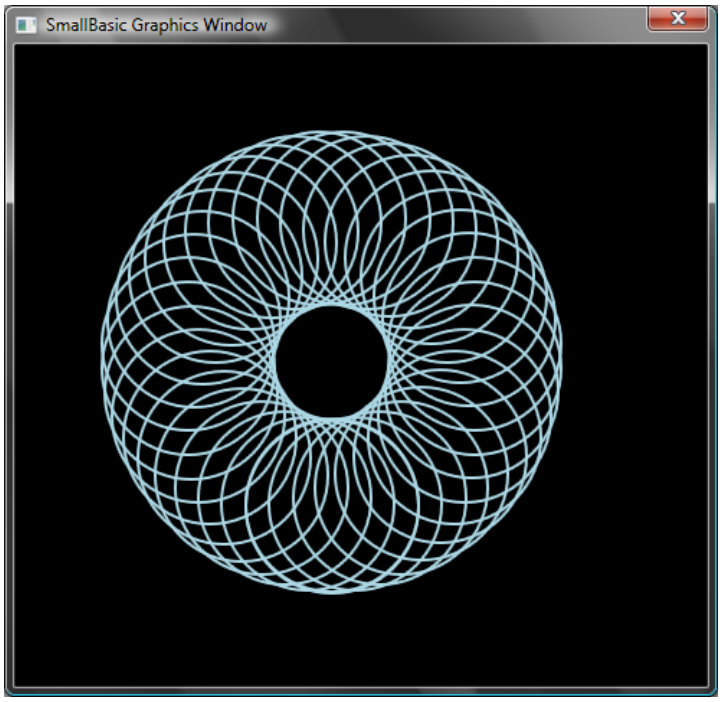

圖 46 -圖形子常式示例

在迴圈（Loops）內調用子常式

有時子常式可在迴圈內被調用，那段時間內子常式執行同一組語句但處理一個或多個變數的不同值。例如，您有個判斷一個數是否為質數的子常式 *PrimeCheck*（質數檢測）。您可以寫一個程式讓使用者輸入一個數然後您的程式通過子常式判斷這個數是否為質數。參見下麵的程式。

```
TextWindow.Write("Enter a number: ")
i = TextWindow.ReadNumber()
isPrime = "True"
PrimeCheck()
If (isPrime = "True") Then
  TextWindow.WriteLine(i + "is a prime number")
Else
  TextWindow.WriteLine(i + "is not a prime number")
EndIf

Sub PrimeCheck
  For j = 2 To Math.SquareRoot(i)
    If (Math.Remainder(i, j) = 0) Then
      isPrime = "False"
      Goto EndLoop
```

```
    EndIf
  Endfor
EndLoop:
EndSub
```

PrimeCheck 子常式得到 *i* 的值並試圖用較小的數來除它。如果有一個數能整除 *i* ，說明 *i* 不是質數。這時子常式將 *isPrime* 的值設為“False”（假）並退出。如果這個數不能被更小的數整除那麼它的 *isPrime* 值仍保持為“True”（真）。

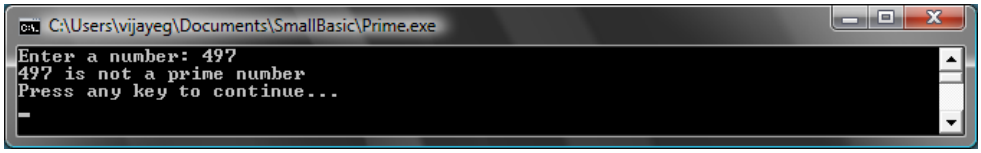

圖 47 -質數檢測

現在您有一個可以作質數檢測的子常式了，您也許想用它列出比如 100 之內的所有質數。我們可以很容易地修改程式從一個迴圈中調用 *PrimeCheck* 來實現。每次迴圈運行時都會給子常式一個不同的數。讓我們通過下麵的例子來解釋。

```
For i = 3 To 100
  isPrime = "True"
  PrimeCheck()
  If (isPrime = "True") Then
    TextWindow.WriteLine(i)
  EndIf
EndFor

Sub PrimeCheck
  For j = 2 To Math.SquareRoot(i)
    If (Math.Remainder(i, j) = 0) Then
      isPrime = "False"
      Goto EndLoop
    EndIf
  Endfor
EndLoop:
EndSub
```

上面的程式中，*i* 的值在每次迴圈時都被更新。在迴圈內部，有一個到子常式 *PrimeCheck* 的調用。子常式 *PrimeCheck* 取得 *i* 的值並計算 *i* 是否為質數。結果保存在變數 *isPrime* 中並可在子常式外部被迴圈所訪問。如果 *i* 是質數則它的值會被列印出來。由於迴圈從 3 開始直到 100，我們就得到了 3 到 100 間的所有質數的清單。下麵是程式運行結果。

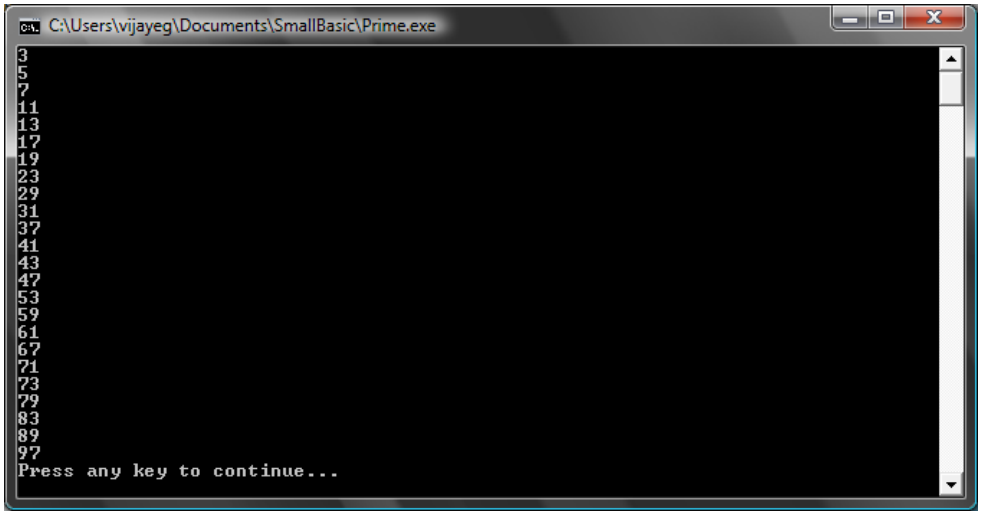

圖 48 -質數

第十章

陣列

現在您應當很清楚地瞭解如何使用變數了，並且迄今為止您仍對程式設計感到有趣，對嗎？

讓我們花一點兒時間來重溫一下我們使用變數編寫的第一個程式。

```
TextWindow.Write("Enter your Name: ")
name = TextWindow.Read()
TextWindow.WriteLine("Hello " + name)
```

在這個程式中，我們把接收並存儲使用者名的變數起名為 **name** 。然後我對這個使用者說“你好”。現在，讓我們假定有多於一位的使用者，比方說，有五位使用者。我們該如何存儲所有的名字呢？有一個辦法是這樣的：

```
TextWindow.Write("User1, enter name: ")
name1 = TextWindow.Read()
TextWindow.Write("User2, enter name: ")
name2 = TextWindow.Read()
TextWindow.Write("User3, enter name: ")
name3 = TextWindow.Read()
TextWindow.Write("User4, enter name: ")
name4 = TextWindow.Read()
TextWindow.Write("User5, enter name: ")
name5 = TextWindow.Read()

TextWindow.Write("Hello ")
```

```
TextWindow.Write(name1 + ", ")
TextWindow.Write(name2 + ", ")
TextWindow.Write(name3 + ", ")
TextWindow.Write(name4 + ", ")
TextWindow.WriteLine(name5)
```

運行上面的程式時您將看到以下結果：

圖 49 – 未使用陣列

很明顯我們肯定會有一種更好的辦法來寫這樣一個簡單的程式，對嗎？尤其是電腦非常擅長做重複性的任務，我們為什麼很麻煩地要為每一位新使用者重新寫一段相同的代碼呢？這裡的技巧就是把多個使用者的名字保存在同一個變數中。如果我們可以做到這一點，我們就可以使用前邊章節中學過的 **For** 迴圈了。這裡我們就要引入陣列了。

什麼是陣列？

一個陣列是一種特殊類型的變數，它可以同時保存多於一個的值。也就是說，如果要保存五個使用者的名字，我們不必創建 **name1**，**name2**，**name3**，**name4** 和 **name5** 的五個變數，我們可以僅僅使用 **name** 來存儲全部五個使用者名字。這種同時存儲多個值的方法叫做“索引”（index）。例如，**name[1]**，**name[2]**，**name[3]**，**name[4]** 和 **name[5]** 可以分別保存一個值。數位 1，2，3，4 和 5 被稱為陣列的的*索引*（*indices*）。

儘管 **name[1]**，**name[2]**，**name[3]**，**name[4]** 和 **name[5]** 看上去像不同的變數，但它們實際上是一個變數。您也許會問它的好處是什麼。使用陣列存儲值的最大好處就是您可以使用另一個變數來指定索引，這樣您可以很容易地在迴圈中訪問陣列。

現在，讓我們來看看如何使用我們新學的陣列知識來重寫剛才的程式。

```
For i = 1 To 5
  TextWindow.Write("User" + i + ", enter name: ")
  name[i] = TextWindow.Read()
EndFor

TextWindow.Write("Hello ")
```

```
For i = 1 To 5
  TextWindow.Write(name[i] + ", ")
EndFor
TextWindow.WriteLine("")
```

程式的易讀性大大提高了，不是嗎？請注意兩行加粗的行。第一行把值存入陣列而第二行把值由陣列讀出來。**name[1]** 中存儲的值不會被 **name[2]** 中存儲的值影響。因此很大程度上您可以把 **name[1]** 和 **name[2]** 當做具有相同身份的兩個不同變數。

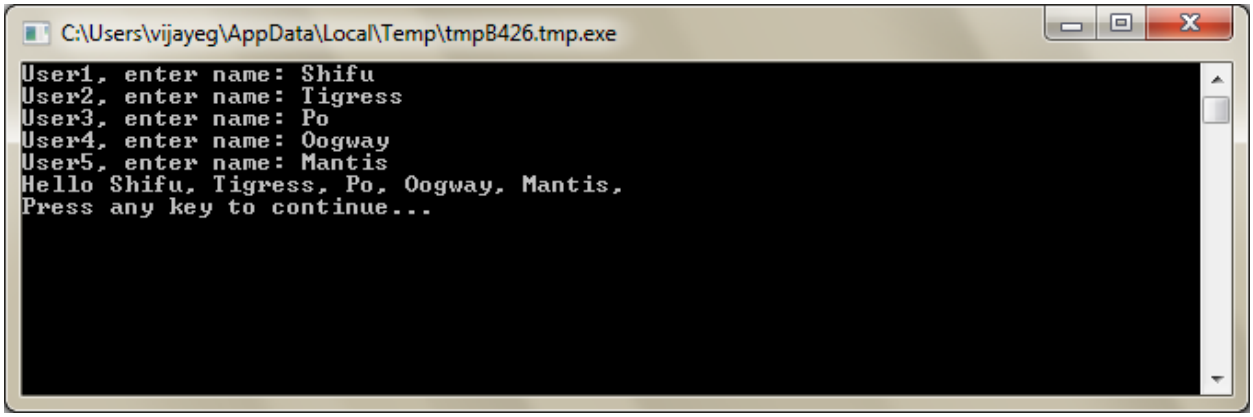

圖 50 –使用陣列

上面的程式與沒有使用陣列的程式運行結果幾乎相同，除了在 *Mantis* 後的逗號。我們可以通過重寫輸出迴圈來修正它：

```
TextWindow.Write("Hello ")
For i = 1 To 5
  TextWindow.Write(name[i])
  If i < 5 Then
    TextWindow.Write(", ")
  EndIf
EndFor
TextWindow.WriteLine("")
```

陣列的索引

在前邊的程式中您看到了我們使用數位作為索引從陣列中存取數值。實際上索引不僅僅限於數位，我們也可以使用文字索引。例如，下麵的程式中，我們詢問並保存一個使用者的資訊然後將資訊輸出。

```
TextWindow.Write("Enter name: ")
user["name"] = TextWindow.Read()
TextWindow.Write("Enter age: ")
user["age"] = TextWindow.Read()
TextWindow.Write("Enter city: ")
user["city"] = TextWindow.Read()
TextWindow.Write("Enter zip: ")
user["zip"] = TextWindow.Read()

TextWindow.Write("What info do you want? ")
index = TextWindow.Read()
TextWindow.WriteLine(index + " = " + user[index])
```

圖 51 –使用非數位索引

多於一個維度

比方說您想把您所有朋友的名字和電話號碼保存起來並能查詢他們的電話號碼，就像電話簿一樣。您應該如何編寫這樣一個程式呢？

這個例子涉及兩套索引（也被認為是陣列的維度，即二維陣列）。假設我們用朋友的“小名”來辨識每一位朋友。這就是我們的陣列的第一套索引。一旦我們得到了朋友的變數，第二套索引，**name** 和 **phone number** 就可以說明我們得到朋友的全名和電話號碼。

我們可以像這樣來存儲資料：

```
friends["Rob"]["Name"] =
"Robert"
friends["Rob"]["Phone"] = "555-
6789"

friends["VJ"]["Name"] =
"Vijaye"
friends["VJ"]["Phone"] = "555-4567"
```

陣列索引是不區分大小寫的。如同正常的變數，陣列索引的匹配會忽略大小寫。

```
friends["Ash"]["Name"] = "Ashley"
friends["Ash"]["Phone"] = "555-2345"
```

由於我們的陣列 **friends** 有兩套索引，該陣列被稱為二維陣列。

這時我們就可以使用朋友的小名作為輸入值然後輸出朋友的全名及電話號碼資訊了。下面是完整的程式碼：

```
friends["Rob"]["Name"] = "Robert"
friends["Rob"]["Phone"] = "555-6789"

friends["VJ"]["Name"] = "Vijaye"
friends["VJ"]["Phone"] = "555-4567"

friends["Ash"]["Name"] = "Ashley"
friends["Ash"]["Phone"] = "555-2345"

TextWindow.Write("Enter the nickname: ")
nickname = TextWindow.Read()

TextWindow.WriteLine("Name: " + friends[nickname]["Name"])
TextWindow.WriteLine("Phone: " + friends[nickname]["Phone"])
```

圖 52 – 一個簡單的電話簿

使用陣列表示網格

多維陣列一個通常的用法是表示網格和表格。網格有行和列，他們恰好可以放入一個二維陣列。下面是一個簡單的將箱子放入網格的示例：

```
rows = 8
columns = 8
size = 40
```

```
For r = 1 To rows
  For c = 1 To columns
    GraphicsWindow.BrushColor = GraphicsWindow.GetRandomColor()
    boxes[r][c] = Shapes.AddRectangle(size, size)
    Shapes.Move(boxes[r][c], c * size, r * size)
  EndFor
EndFor
```

這個程式把小矩形加入並調整位置以組成一個 8x8 的網格。除了放置這些小矩形，該程式也把它們存入一個陣列。這樣做使得我們可以容易地追蹤這些小矩形並在需要的時候再次使用它們。

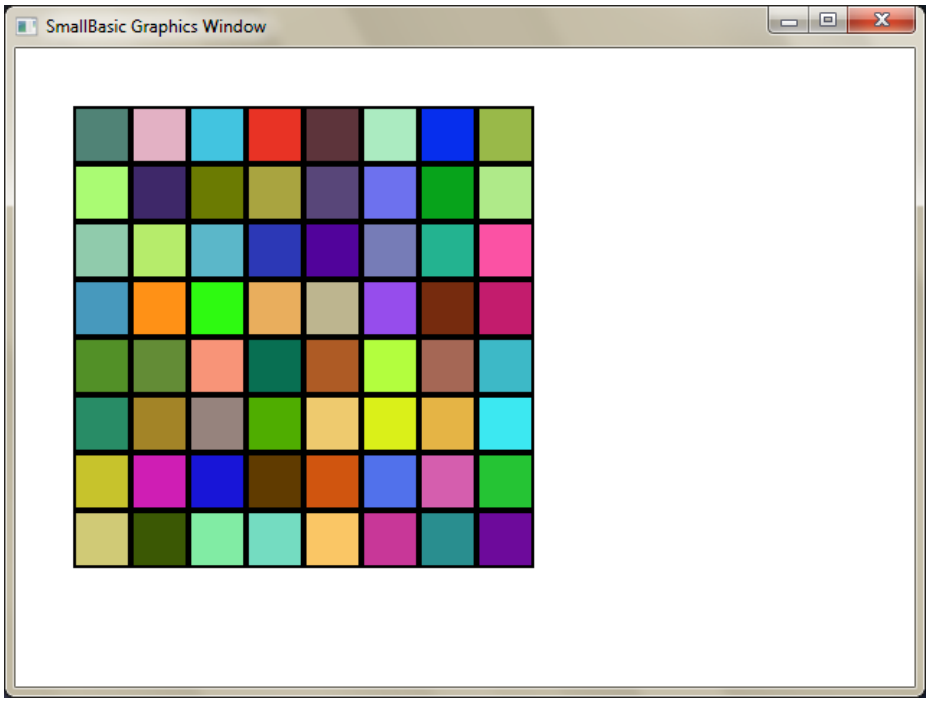

圖 53 – 在網格中放置小矩形

例如，把下麵的代碼附加到前面的程式最後就可以把小矩形以動畫的形式移動到視窗的左上角。

```
For r = 1 To rows
  For c = 1 To columns
    Shapes.Animate(boxes[r][c], 0, 0, 1000)
    Program.Delay(300)
  EndFor
EndFor
```

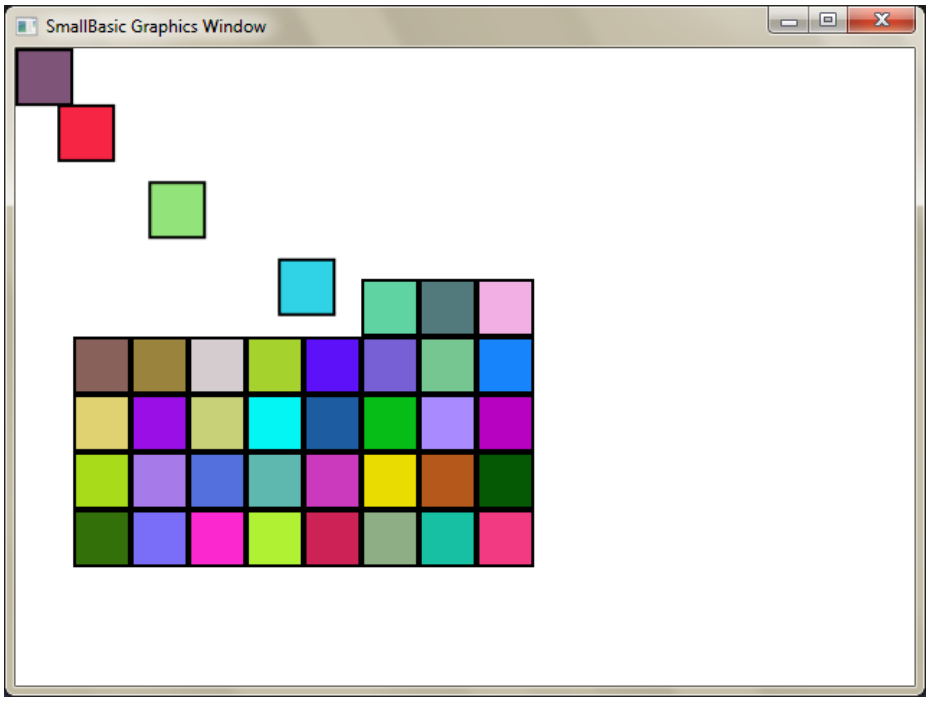

圖 54 – 在網格中追蹤小矩形

第十一章

事件（Events）和交互（Interactivity）

在最前面的兩章，我們介紹了具有屬性和操作的物件。除了屬性和操作，一些物件還帶有事件（Event）。事件就像被觸發的信號，例如，為了應對類似移動或點擊滑鼠的使用者操作。某種程度上說，事件同操作是相對的。對於操作，您作為一個程式師調用操作讓電腦做一些事情；然後對於事件，電腦通知您一些有意思的事情發生了。

事件有什麼用？

事件是介紹程式交互性的核心。如果您想讓使用者同您的程式交互，您必須使用事件。比方說，您正在編寫一個 Tic-Tac-Toe 遊戲（也有叫井字棋的，是一種類似于五子棋的連線遊戲）。您希望能讓使用者選擇他的角色，對嗎？這裡就涉及到事件-您在您的程式中使用事件來收到使用者輸入。如果這樣的解釋難以明白，不要擔心，我們來看一個非常簡單的示例來說明您理解什麼是事件以及如何使用事件。

下麵是一個非常簡單的程式，只有一行語句和一個子常式。子常式使用 *GraphicsWindow*（圖形視窗）物件的 *ShowMessage*（顯示消息）操作顯示一個對話方塊給使用者。

```
GraphicsWindow.MouseDown = OnMouseDown

Sub OnMouseDown
  GraphicsWindow.ShowMessage("You Clicked.", "Hello")
EndSub
```

上面程式中一個有意思的部分是我們將一個子常式指派到 GraphicsWindow 物件的 MouseDown（按下滑鼠）事件的那一行。您可能注意到了 MouseDown 看起來非常像一個屬性-除了指定一些值（我們將子常式 *OnMouseDown* 指派給它）。這就是事件的特殊性-當事件發生時，子常式會自動被調用。在這個例子中，每次使用者在 GraphicsWindow 上點擊滑鼠時子常式 *OnMouseDown* 都會被調用。讓我們繼續，運行並使用這個程式。任何時間當您在 GraphicsWindow 上點擊滑鼠時您都將看到如下麵所示的對話方塊。

圖 49 -對事件的反應

這種事件處理是非常強大的，它支援非常有創造性和有意思的程式。使用這種方式編寫的程式通常被成為事件驅動（event-driven）的程式。

您可以修改子常式 *OnMouseDown* 實現比彈出一個對話方塊更多的事情。例如像下麵的程式，您可以在使用者點擊滑鼠的地方繪製大藍點。

```
GraphicsWindow.BrushColor = "Blue"
GraphicsWindow.MouseDown = OnMouseDown

Sub OnMouseDown
  x = GraphicsWindow.MouseX - 10
  y = GraphicsWindow.MouseY - 10
  GraphicsWindow.FillEllipse(x, y, 20, 20)
EndSub
```

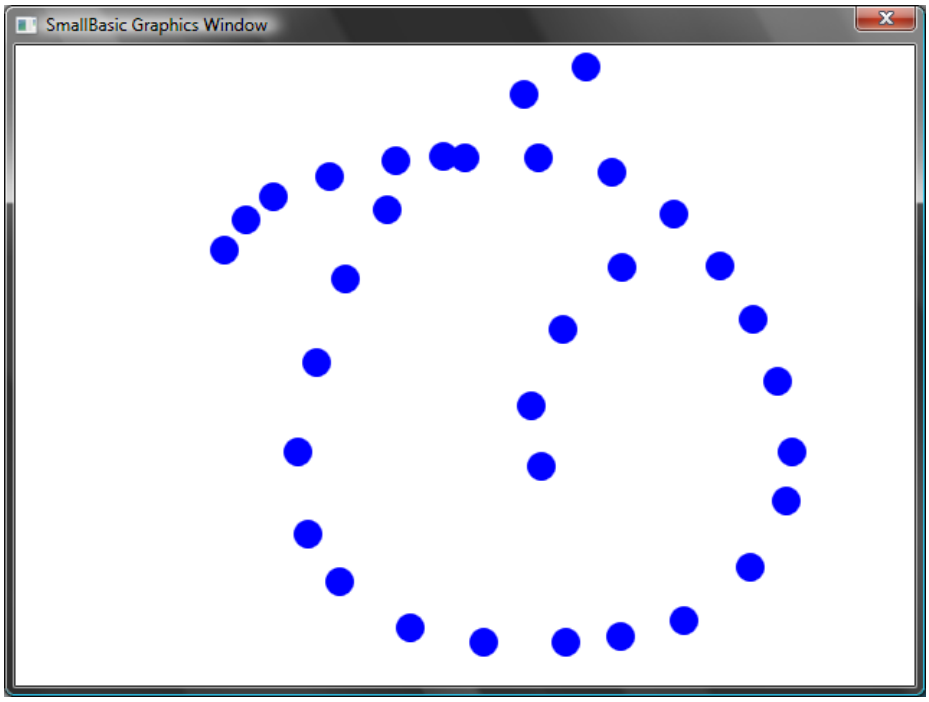

圖 50 -處理滑鼠點擊事件

請注意在上面程式中，我們使用 *MouseX* 和 *MouseY* 來取得滑鼠的座標位置。然後我們用滑鼠座標點作為圓心繪製一個圓。

處理多個事件

您想處理多少事件是沒有限制的。您甚至可以使用一個子常式來處理多個事件。但是，您每次只能處理一個事件。如果您試圖指派兩個子常式給同一個事件，只有後面的子常式會起作用。

為了說明這一點，讓我們使用前邊的示例並添加一個處理鍵盤按鍵的子常式。同時，我們讓新的子常式改變筆刷的顏色，這樣當您點擊滑鼠時您將看到不同顏色的點。

```
GraphicsWindow.BrushColor = "Blue"
GraphicsWindow.MouseDown = OnMouseDown
GraphicsWindow.KeyDown = OnKeyDown

Sub OnKeyDown
  GraphicsWindow.BrushColor = GraphicsWindow.GetRandomColor()
EndSub

Sub OnMouseDown
  x = GraphicsWindow.MouseX - 10
  y = GraphicsWindow.MouseY - 10
```

```
  GraphicsWindow.FillEllipse(x, y, 20, 20)
EndSub
```

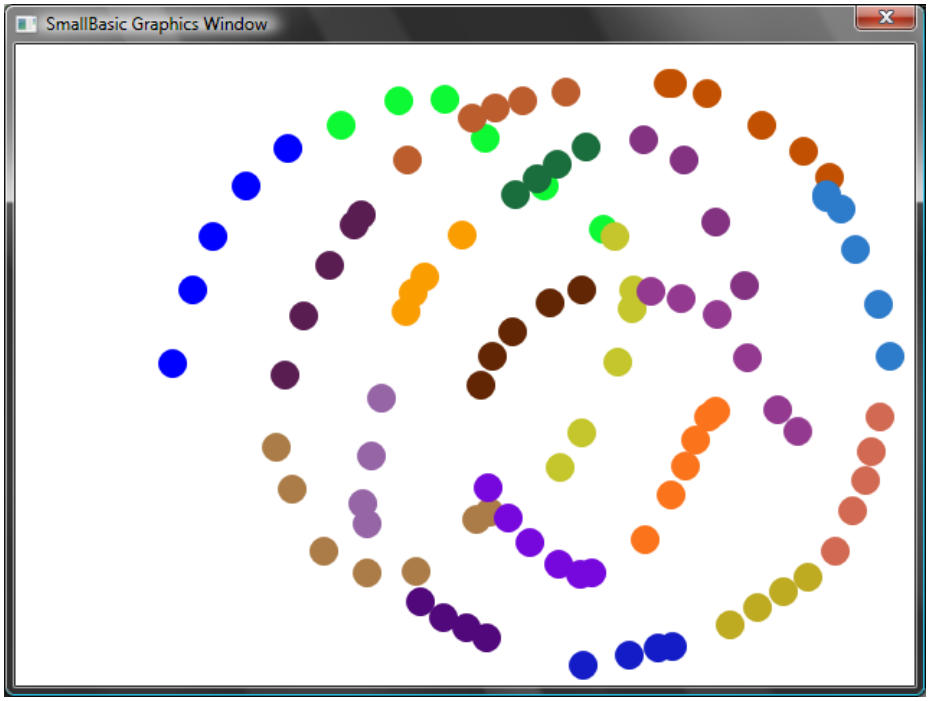

圖 51 -處理多個事件

原來當您運行程式並在視窗中點擊時，您將看到藍色的點。現在，如果您按鍵盤上的任意鍵然後再點擊滑鼠，您將看到不同顏色的點。當您按下鍵盤上的鍵時子常式 *OnKeyDown*（按下鍵盤按鍵）被執行，它將筆刷的顏色改為隨機顏色。然後您再點擊滑鼠時，您就會看到一個使用新設置的顏色（隨機產生的顏色）的圓點。

一個繪圖程式

由於事件和子常式的說明，我們現在可以編寫讓使用者在視窗中繪圖的程式了。只要我們把問題分解為較小的模組，編寫這樣一個程式就變得簡單得令人吃驚了。第一步，我們寫一個允許使用者在圖形視窗任意移動滑鼠的程式，滑鼠走過的地方會留下一道軌跡。

```
GraphicsWindow.MouseMove = OnMouseMove

Sub OnMouseMove
  x = GraphicsWindow.MouseX
  y = GraphicsWindow.MouseY
  GraphicsWindow.DrawLine(prevX, prevY, x, y)
  prevX = x
```

```
  prevY = y
EndSub
```

不過，當您運行這個程式時，第一道軌跡總是從視窗的左上角即(0, 0) 開始。我們可以通過處理 *MouseDown* 事件並捕捉當事件發生時的 *prevX* 和 *prevY* 值來修復這個問題。

同時，我們只需要把使用者按下滑鼠時的軌跡記錄下來。在其他時間我們不必繪製軌跡線。為了實現這樣的行為，我們將使用 **Mouse**（滑鼠）物件的 *IsLeftButtonDown*（滑鼠左鍵按下）屬性。該屬性告訴我們滑鼠左鍵是否被按下。如果這個值為 true（真），我們就畫軌跡線，否則就不畫。

```
GraphicsWindow.MouseMove = OnMouseMove
GraphicsWindow.MouseDown = OnMouseDown

Sub OnMouseDown
  prevX = GraphicsWindow.MouseX
  prevY = GraphicsWindow.MouseY
EndSub

Sub OnMouseMove
  x = GraphicsWindow.MouseX
  y = GraphicsWindow.MouseY
  If (Mouse.IsLeftButtonDown) Then
    GraphicsWindow.DrawLine(prevX, prevY, x, y)
  EndIf
  prevX = x
  prevY = y
EndSub
```

（待續）

附錄 A

趣味實例

龜標分形圖

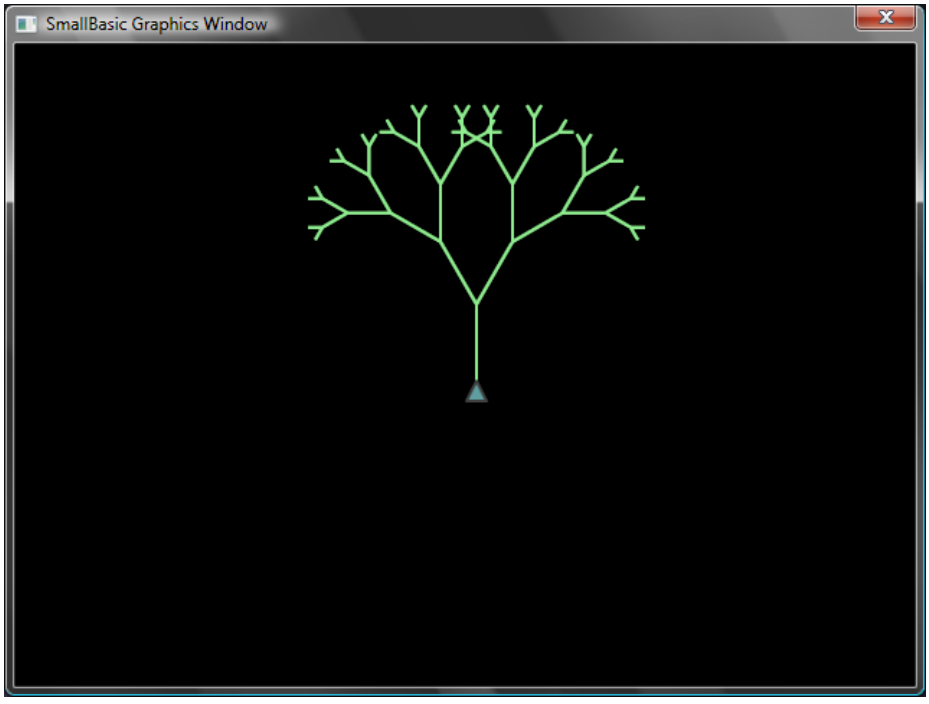

圖 44-龜標樹狀圖

```
angle = 30
delta = 10
distance = 60
```

```
Turtle.Speed = 9
GraphicsWindow.BackgroundColor = "Black"
GraphicsWindow.PenColor = "LightGreen"
DrawTree()

Sub DrawTree
  If (distance >0) Then
    Turtle.Move(distance)
    Turtle.Turn(angle)

    Stack.PushValue("distance", distance)
    distance = distance - delta
    DrawTree()
    Turtle.Turn(-angle * 2)
    DrawTree()
    Turtle.Turn(angle)
    distance = Stack.PopValue("distance")

    Turtle.Move(-distance)
  EndIf
EndSub
```

來自于 Flickr 的照片

圖 45-抓取 Flickr 的照片

```
GraphicsWindow.BackgroundColor = "Black"
GraphicsWindow.MouseDown = OnMouseDown

Sub OnMouseDown
    pic = Flickr.GetRandomPicture("mountains, river")
    GraphicsWindow.DrawResizedImage(pic, 0, 0, 640, 480)
EndSub
```

動態桌面牆紙

```
For i = 1 To 10
  pic = Flickr.GetRandomPicture("mountains")
  Desktop.SetWallPaper(pic)
  Program.Delay(10000)
EndFor
```

船槳彈球遊戲

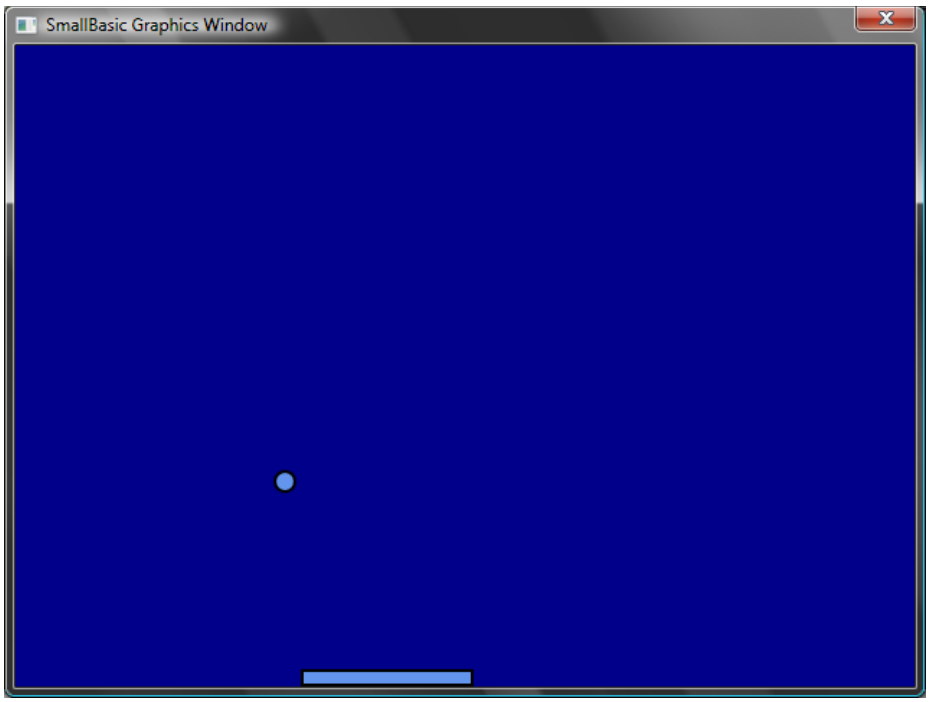

圖 46-船槳彈球遊戲

```
GraphicsWindow.BackgroundColor = "DarkBlue"
paddle = GraphicsWindow.AddRectangle(120, 12)
ball = GraphicsWindow.AddEllipse(16, 16)
GraphicsWindow.MouseMove = OnMouseMove

x = 0
y = 0
deltaX = 1
deltaY = 1

RunLoop:
  x = x + deltaX
  y = y + deltaY

  gw = GraphicsWindow.Width
  gh = GraphicsWindow.Height
  If (x >= gw - 16 or x <= 0) Then
    deltaX = -deltaX
  EndIf
```

```
  If (y <= 0) Then
    deltaY = -deltaY
  EndIf

  padX = GraphicsWindow.GetLeftOfShape(paddle)
  If (y = gh - 28 and x >= padX and x <= padX + 120) Then
    deltaY = -deltaY
  EndIf

  GraphicsWindow.MoveShape(ball, x, y)
  Program.Delay(5)

  If (y <gh) Then
    Goto RunLoop
  EndIf

GraphicsWindow.ShowMessage("You Lose", "Paddle")

Sub OnMouseMove
  paddleX = GraphicsWindow.MouseX
  GraphicsWindow.MoveShape(paddle, paddleX - 60, GraphicsWindow.Height - 12)
EndSub
```

附錄 B

顏色

TODO: 描述顏色 + 十六進位數

這裡是一個 Small Basic 支援的顏色清單, 按它們的基礎色調分類的.

Red Colors （紅色）

IndianRed	#CD5C5C
LightCoral	#F08080
Salmon	#FA8072
DarkSalmon	#E9967A
LightSalmon	#FFA07A
Crimson	#DC143C
Red	#FF0000
FireBrick	#B22222
DarkRed	#8B0000

Pink Colors （粉色）

Pink	#FFC0CB
LightPink	#FFB6C1
HotPink	#FF69B4
DeepPink	#FF1493
MediumVioletRed	#C71585
PaleVioletRed	#DB7093

Orange Colors （橘紅色）

LightSalmon	#FFA07A
Coral	#FF7F50
Tomato	#FF6347
OrangeRed	#FF4500
DarkOrange	#FF8C00
Orange	#FFA500

Yellow Colors （黃色）

Gold	#FFD700
Yellow	#FFFF00
LightYellow	#FFFFE0
LemonChiffon	#FFFACD
LightGoldenrodYel	#FAFAD2

low	
PapayaWhip	#FFEFD5
Moccasin	#FFE4B5
PeachPuff	#FFDAB9
PaleGoldenrod	#EEE8AA
Khaki	#F0E68C
DarkKhaki	#BDB76B

Purple Colors （紫色）

Lavender	#E6E6FA
Thistle	#D8BFD8
Plum	#DDA0DD
Violet	#EE82EE
Orchid	#DA70D6
Fuchsia	#FF00FF
Magenta	#FF00FF
MediumOrchid	#BA55D3
MediumPurple	#9370DB
BlueViolet	#8A2BE2
DarkViolet	#9400D3
DarkOrchid	#9932CC
DarkMagenta	#8B008B
Purple	#800080
Indigo	#4B0082
SlateBlue	#6A5ACD
DarkSlateBlue	#483D8B
MediumSlateBlue	#7B68EE

Green Colors （綠色）

GreenYellow	#ADFF2F
Chartreuse	#7FFF00
LawnGreen	#7CFC00

Lime	#00FF00
LimeGreen	#32CD32
PaleGreen	#98FB98
LightGreen	#90EE90
MediumSpringGreen	#00FA9A
SpringGreen	#00FF7F
MediumSeaGreen	#3CB371
SeaGreen	#2E8B57
ForestGreen	#228B22
Green	#008000
DarkGreen	#006400
YellowGreen	#9ACD32
OliveDrab	#6B8E23
Olive	#808000
DarkOliveGreen	#556B2F
MediumAquamarine	#66CDAA
DarkSeaGreen	#8FBC8F
LightSeaGreen	#20B2AA
DarkCyan	#008B8B
Teal	#008080

Blue Colors（藍色）

Aqua	#00FFFF
Cyan	#00FFFF
LightCyan	#E0FFFF
PaleTurquoise	#AFEEEE
Aquamarine	#7FFFD4
Turquoise	#40E0D0
MediumTurquoise	#48D1CC
DarkTurquoise	#00CED1
CadetBlue	#5F9EA0

SteelBlue	#4682B4
LightSteelBlue	#B0C4DE
PowderBlue	#B0E0E6
LightBlue	#ADD8E6
SkyBlue	#87CEEB
LightSkyBlue	#87CEFA
DeepSkyBlue	#00BFFF
DodgerBlue	#1E90FF
CornflowerBlue	#6495ED
MediumSlateBlue	#7B68EE
RoyalBlue	#4169E1
Blue	#0000FF
MediumBlue	#0000CD
DarkBlue	#00008B
Navy	#000080
MidnightBlue	#191970

Brown Colors（棕色）

Cornsilk	#FFF8DC
BlanchedAlmond	#FFEBCD
Bisque	#FFE4C4
NavajoWhite	#FFDEAD
Wheat	#F5DEB3
BurlyWood	#DEB887
Tan	#D2B48C
RosyBrown	#BC8F8F
SandyBrown	#F4A460
Goldenrod	#DAA520
DarkGoldenrod	#B8860B
Peru	#CD853F
Chocolate	#D2691E

SaddleBrown	#8B4513
Sienna	#A0522D
Brown	#A52A2A
Maroon	#800000

White Colors（白色）

White	#FFFFFF
Snow	#FFFAFA
Honeydew	#F0FFF0
MintCream	#F5FFFA
Azure	#F0FFFF
AliceBlue	#F0F8FF
GhostWhite	#F8F8FF
WhiteSmoke	#F5F5F5
Seashell	#FFF5EE
Beige	#F5F5DC
OldLace	#FDF5E6
FloralWhite	#FFFAF0
Ivory	#FFFFF0
AntiqueWhite	#FAEBD7
Linen	#FAF0E6
LavenderBlush	#FFF0F5
MistyRose	#FFE4E1

Gray Colors（灰色）

Gainsboro	#DCDCDC
LightGray	#D3D3D3
Silver	#C0C0C0
DarkGray	#A9A9A9
Gray	#808080
DimGray	#696969
LightSlateGray	#778899

SlateGray	#708090
DarkSlateGray	#2F4F4F
Black	#000000