簡介

當我還是小孩時，第一次玩電動就著迷了，我不僅止於玩而已，我還想要自製電玩。我找到一本像這樣的書，教我怎麼寫程式和設計遊戲，真是簡單又有趣。我編寫的遊戲程式也像在本書中的程式範例，雖然它不像我爸媽買的任天堂遊戲那麼厲害，但這些都是我自己設計製作的遊戲程式。

現下的我已長大成人，我還一直樂在程式設計的工作並以此謀生。就算你不想當電腦程式設計師，但程式設計還是一項有用又有趣而值得培養的技能。程式設計能訓練和培養我們的邏輯思考與制定計劃的能力，在我們從編寫的程式碼中發現錯誤時，也可重新思考我們的設計想法。

大多數針對初學者的程式設計書籍可分為兩類。第一類的書籍講述過多的「遊戲製作軟體」理論，而程式設計實作卻教太少，或是把程式語法講述太過簡化，以致於不像在教程式設計。第二類的書籍把程式設計寫得像數學教科書，講述一大堆的理論原則但缺少實際的應用。本書則以不同的方式來教你如何編寫程式和製作電玩遊戲。我會把遊戲的程式碼直接完整地展示出來，並以實例來解釋說明程式設計的原理。這種方式正是我在學習程式設計時的關鍵方法。越能從別人的程式實例中學習，就越能在自己編寫程式時有更多想法。

只要有台電腦，免費的 Python 軟體以及本書，這樣就能搞定一切。當您能從本書中學會如何製作遊戲程式，那您就有能力可開發出屬於自己的遊戲程式。

電腦是台很神奇的機器，而學習程式設計並不像大多數人所想的那麼困難。所謂的「電腦程式」是指一堆電腦能理解的指令，就像故事書中一大堆的句子會讓讀者能看懂一樣。若想要指揮電腦運作，則要使用電腦看得懂的語言。本書所教的程式語言為 Python。在現今還有很多可學習的程式語言，如 BASIC、Java、JavaScript、PHP 和 C++等。

當我還是小孩時，所學的是 BASIC，現今新型的程式語言如 Python 則更簡單好學，很多專業的程式設計師在職場或平常玩樂應用中也都使用 Python 來設計程式。Python 是一套完全免費的工具，只要從網路上下載安裝就能使用。

由於電玩遊戲程式也是一種電腦程式，所以也是由一堆指令所組成。本書中所講述的遊戲程式比起 Xbox、PlayStation 或 Nintendo 來看是較簡單，這些範例沒有很炫的影像圖樣，原因是想要專注在講解和指導程式設計的基本觀念，故意這麼簡化的目的是讓讀者能專心在學習程式設計。電玩遊戲程式不一定要繁雜酷炫才是好玩的！

本書的適用對象

程式設計並不困難，但卻很難找到能教您利用程式設計來做有趣事情的教材。有些電腦書籍講述過多新手程式設計師不太需要的內容了。本書會教您如何寫出自己的遊戲程式，您會學到有用的技能，並作出能秀給別人看的電玩遊戲程式。這本書所適用的讀者為：

- 完全沒有經驗，但想要學習程式設計的初學者。
- 想製作電玩遊戲，也想學會程式設計的青少年和小朋友。
- 想教別人程式設計的成年人和老師。
- 想藉由學習專業的程式語言來學會如何編寫程式的所有人，包括大人小孩皆適用。

本書簡介

書中大多數的章節都以一個嶄新的遊戲主題來說明講解，部分章節則提供一些像如何除錯的有用專題。在講解遊戲製作時新的程式設計觀念也會一併說明，而書中章節安排順序是有其用意的，以下為各章節簡介：

- **Chapter 1：互動式 Shell**，講解如何使用 Python 的互動 Shell 模式來逐行實作程式指令。
- **Chapter 2：編寫程式**，講解如何在 Python 的 File editor 中編寫完整的程式碼。
- **Chapter 3：猜數字**，本書講解的第一個遊戲程式，此遊戲是要讓玩的人猜出一組數字，在猜的過程中程式會提供猜的數字太高或太低的提示。
- **Chapter 4：腦筋急轉彎**，會編寫設計一個簡單的程式對使用者說幾個腦筋急轉彎笑話。
- **Chapter 5：龍域（Dragon Realm）**，會編寫設計一個猜測遊戲，讓玩家必須在兩個洞穴中作出選擇，結果會出現善龍或是惡龍。
- **Chapter 6：使用 Debugger**，講解如何使用除錯器來修正程式中的錯誤。
- **Chapter 7：使用流程圖設計 Hangman 遊戲**，講解怎麼用流程圖來設計規劃較長的程式，例如猜單字遊戲（Hangman）。
- **Chapter 8：編寫 Hangman 猜單字遊戲的程式碼**，依據 Chapter 7 設計的流程圖來編寫出猜單字遊戲的程式。
- **Chapter 9：擴充 Hangman 猜單字遊戲**，利用 Python 的字典資料型別來擴充猜單字遊戲新功能。
- **Chapter 10：井字棋遊戲程式**，講解如何編寫使用到人工智慧的人機對戰的 OX 井字棋遊戲（tic-tac-toe）。

- **Chapter 11：Bagels 推理遊戲**，講解如何編寫 Bagels 推理遊戲，讓玩家依照線索來推理猜測出正確的神秘數字。
- **Chapter 12：笛卡兒座標系統**，本章介紹說明後面章節在設計遊戲時會用到的笛卡兒座標系統。
- **Chapter 13：聲納尋寶遊戲**，介紹如何編寫尋寶遊戲，讓玩家在茫茫大海中尋找失落的寶箱。
- **Chapter 14：凱撒密碼（Caesar Cipher）**，製作一個簡單的編碼程式，讓我們可以編寫和解碼秘密訊息。
- **Chapter 15：黑白棋**，會製作一個高階的人機對戰黑白棋遊戲，程式會使用到難以打敗的人工智慧元件。
- **Chapter 16：黑白棋人工智慧模擬**，延展 Chapter15 中黑白棋遊戲的應用，製作多個人工智慧來讓電腦對電腦進行比賽，比較不同的 AI 演算法。
- **Chapter 17：繪製圖案**，介紹 Python 的 pygame 模組，並示範怎麼用它來繪製平面圖案。
- **Chapter 18：動畫**，介紹如何使用 pygame 來製作動畫。
- **Chapter 19：碰撞偵測**，講解如何在 2D 平面遊戲中偵測物件的碰撞。
- **Chapter 20：使用聲音和影像**，介紹怎麼在遊戲中加入聲音和影像。
- **Chapter 21：使用了聲音和影像的 Dodger 遊戲**，結合第 17 章到第 20 章的知識與觀念來製作出 Dodger 動畫遊戲。

第 1 章

互動式 Shell

在您製作電玩遊戲之前，先要學會一些基本的程式設計概念。從本章開始，您會學到如何使用 Python 的互動式 Shell，以及執行基本的算數運算。

本章包含的主題：

- ▶運算子
- ▶整數與浮點數
- ▶值
- ▶表示式
- ▶語法錯誤
- ▶把值儲存到變數中

一些簡單的數學

請參考「簡介」中「啟動 IDLE」小節的步驟來開啟 IDLE 的 Python Shell 視窗。首先我們要用 Python 來求解一些簡單的數學問題。互動式 Shell 可以像個計算機一樣來算數學問題。請在互動式 Shell 中 >>> 提示符號後輸入 2 + 2，再按下 Enter 鍵（有些鍵盤是 Return 鍵）。如圖 1-1 所示，互動式 Shell 中求解這個簡單數學問題的過程，請留意其回應的結果是 4。

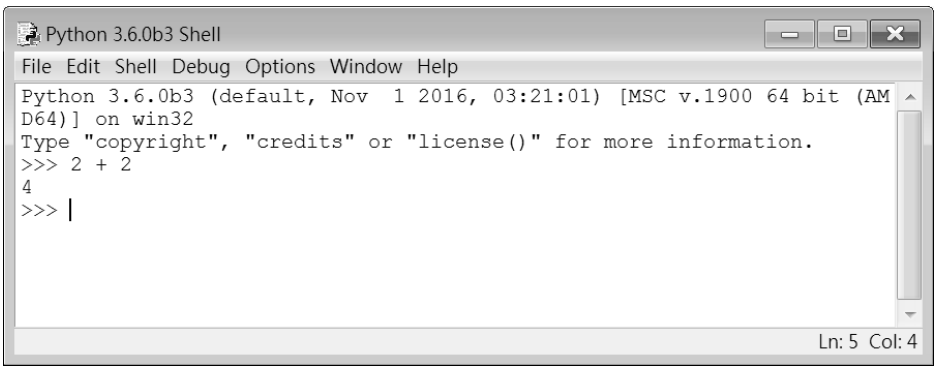


圖 1-1：在互動式 Shell 中輸入 2 + 2

這個數學問題是個簡單的程式指令，其中的加號（+）是告訴電腦要把兩個數字 2 加起來的意思。電腦處理後在下一行回應數字 4 的結果。表 1-1 列出了在 Python 中可使用的數學符號。

表 1-1：數學運算子

運算子	運算
+	加法
-	減法
*	乘法
/	除法

減號（-）是處理數字的相減，星號（*）則是數字的相乘，而斜線符號（/）則是數字相除。當以這種方式來使用時，它們都稱為運算子。運算子是告知 Python 對數字要進行什麼樣的運算。

整數和浮點數

整數（Integers 或縮寫成 Ints）是指所有像 4、99 和 0 這樣的數值。而浮點數（Floating-point numbers 或縮寫成 floats）是指分數或有小數點的數值，例如 3.5、42.1 和 5.0 等。在 Python 中，5 是整數，但 5.0 是浮點數。這些數字稱為「值（values）」。後面我們還會學到其他不是數字類的「值」。在您輸入到 Shell 中的數學問題，兩個 2 都是整數值。

表示式

在這個範例中的數學問題「2 + 2」就是個「表示式（expressions）」。如圖 1-2 所示，表示是由值（數字）和運算子（數學符號）連接組成，該表示式會生成程式碼可使用的新值。電腦能幾秒內處理求解數百萬個表示式。

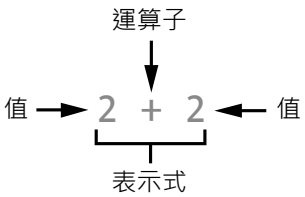


圖 1-2：表示式是由值和運算子組成

請如下在互動式 Shell 中輸入一些表示式，並分別在每個表示式後按下 Enter：

```
>>> 2+2+2+2+2
10
>>> 8*6
48
>>> 10-5+6
11
>>> 2 +        2
4
```

這些表示式看起來都像是正規的數學方程式，但請留意範例「2 + 2」中多了很多的空格。在 Python 中，值和運算子之間可加入任意的空格，都能正確運算求解。但有一點要注意，在互動式 Shell 的 >>> 提示符號後輸入指令時，每一行的開頭是不能有空格的哦。

表示式運算求解

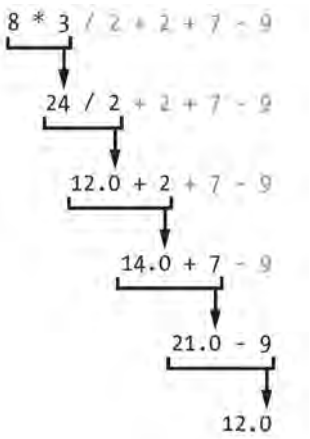
當電腦求解 10 + 5 表示式並傳回 15，就是在「運算」這個表示式。表示式會經過運算求解成單一個「值」，過程就像是對數學問題求解成一個數字答案。舉例來說，表示式 10 + 5 和 10 + 3 + 2 其運算結果都是 15。

當 Python 運算表示式時，運算子的運算順序和處理數學問題是一樣的，會遵守下列幾項規則：

- 括號內的表示式先運算求解。
- 先乘除後加減。
- 運算順序是由左而右。

表示式 1 + 2 * 3 + 4 的運算結果為 11，而不是 13，因為 2 * 3 會先運算。如果表示式變成 (1 + 2) * (3 + 4)，則運算結果為 21，因為 (1 + 2) 和 (3 + 4) 在括號內會先運算，然後才會相乘。

表示式長短不一，但都運算求解成單一個值，就算這個單個值為表示式也可以。舉例來說，表示式 15 其運算結果也是 15。表示式 8 * 3 / 2 + 2 + 7 - 9 的運算結果為 12.0，其運算過程如下：



電腦會照上述步驟過程來運算，但您不會在互動式 Shell 中讓您看到這個過程，Shell 會直接告訴您結果：

```
>>> 8 * 3 / 2 + 2 + 7 - 9
12.0
```

請留意表示式中「/」除法運算子會把運算結果變成浮點數，例如，24 / 2 運算求解結果為 12.0。數學運算子在運算時若運算對象中有一個是浮點數，則求解結果也會是浮點數，因此 12.0 + 2 的結果是 14.0。

語法錯誤

如果您在互動式 Shell 中輸入 5 +，再按下 Enter 鍵，則會顯現錯誤訊息：

```
>>> 5 +
SyntaxError: invalid syntax
```

這個錯誤訊息是告訴您 5 + 並不是合法的表示式。合法的表示式中運算子要連接兩個值，也就是 + 運算子的**前與後**都會有值來連接。當有個值不見了，錯誤訊息就會顯示。

SyntaxError 是指 Python 看不懂這個指令，因為您輸入錯誤。電腦程式設計並不僅只是丟指令給它執行而已，還有知道怎麼正確地下指令才行。

別擔心會出錯，這些錯誤並不會弄壞您的電腦，只要在互動式 Shell 的 >>> 提示符號後重新輸入正確的指令就行了。

把值存放到變數中

當表示式運算求解出一個值，那您可以把這個值存放到「**變數**」中，以後就可取用。您可以把變數想像成一個可以裝東西的盒子。

指定陳述句（assignment statement）會把值存放到變數中。輸入變數的名稱，接著輸入等號（=），這個等號就是指定運算子，然後是要放入變數的值。舉例來說，在互動式 Shell 中輸入如下內容：

```
>>> spam = 15
>>>
```

這個 spam 變數盒子就會裝入 15 的值，如圖 1-3 所示。

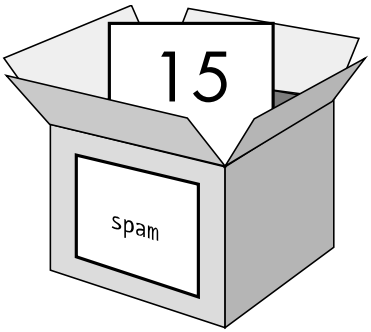


圖 1-3：變數就是個可以裝東西的盒子

當您按下 Enter 鍵，您看不到什麼反應，但在 Python 中，如果下的指令正確成功，則不會有錯誤訊息顯示，>>> 提示符號會再次顯示等待您的輸入。

與表示式（expressions）不一樣，**陳述句**（statements）是不會運算求解出任何值的指令。這就是為什麼在互動式 Shell 中輸入 spam = 15 並按下 Enter 後不會有什麼東西顯示。如果您會搞混表示式和陳述句，請記住，表示式是會運算求解出一個值，而其他的指令則都是陳述句。

變數能存放值，但不是表示式。舉例來說，請看這兩個陳述句的例子：spam = 10 + 5 和 spam = 10 + 7 - 2。這兩者都會運算求出 15，所以前述兩個陳述句結果都相同，都是把 15 的值指定到變數 spam 中。

好的變數所取的名字本身會把該變數存放的內容描述出來。假設您搬家時，紙箱貼的標籤都只寫著「**東西**（stuff）」，那拆箱時要找東西就很困難了。本書後續的範例中會用到像 spam、eggs 和 bacon 的變數名稱。

若第一次在指定陳述句中有用到某變數時，Python 就會建立該變數。若要檢查取用某個變數存了什麼內容，則請在互動式 Shell 中輸入變數名稱並按下 Enter：

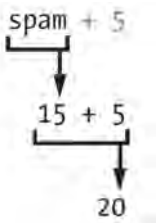
```
>>> spam = 15
>>> spam
15
```

spam 這個表示式會運算求取存放在 spam 變數中的值 15。

我們也可以在表示式中使用變數，請在互動式 Shell 中試著輸入如下內容：

```
>>> spam = 15
>>> spam + 5
20
```

這例子是把 15 存放到 spam 變數中，所以輸入 spam + 5 就等於是輸入 15 + 5。以下是 spam + 5 的運算求解過程：



您不能在變數用指定陳述句建立它之前就使用，如果您這樣做，Python 會丟出一個 NameError 的錯誤訊息，告訴您目前並沒有這樣的變數。錯用變數名稱會產生錯誤的：

```
>>> spam = 15
>>> spma
Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    spma
NameError: name 'spma' is not defined
```

上述例子顯示錯誤是因為建立的是 spam 變數，而不是 spma。

我們可以藉由輸入另一個指定陳述句來變更存放在變數中的值。例如，在互動式 Shell 中試著輸入如下內容：

```
>>> spam = 15
>>> spam + 5
20
>>> spam = 3
>>> spam + 5
8
```

當您第一次輸入 spam + 5 時，表示式運算結果為 20，因為一開始的 spam 變數是存放了 15。隨後輸入了 spam = 3，則原本的 15 就會被新的 3 取代（或覆寫），因為變數一次只能放一個值。由於 spam 存放的是 3，後面的 spam + 5 運算結果就是 8。變數覆寫的過程就像是把東西從盒子中取出，再放入新的東西，如圖 1-4 所示。

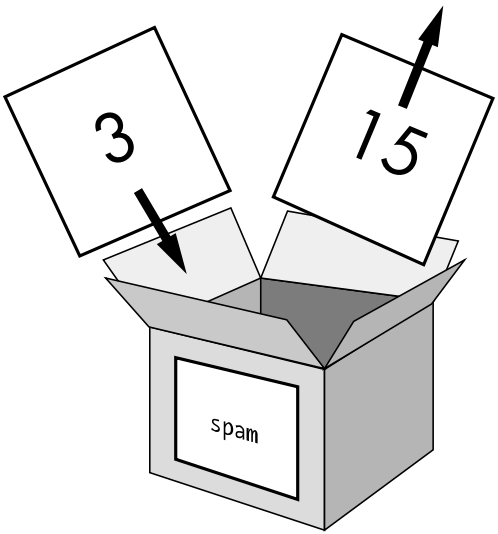


圖 1-4：原本存在 spam 的 15 會被 3 覆寫蓋過去

甚至還可用 spam 變數來當成指定陳述句中的值來運算，再指定回 spam 變數中：

```
>>> spam = 15
>>> spam = spam + 5
20
```

spam = spam + 5 這行指定陳述句是說：「原本存放在 spam 的值加 5 再指定回 spam 中當成新的值」。繼續對 spam 加 5 幾次，在互動式 Shell 中輸入如下內容：

```
>>> spam = 15
>>> spam = spam + 5
>>> spam = spam + 5
>>> spam = spam + 5
>>> spam
30
```

在這個範例中，第一行 spam 指定了 15 的值，在下一行中 spam + 5 的結果又指定回 spam 中，值是 20。當加了 3 次後，spam 就變成 30 了。

到目前為止都只介紹了一個變數，但其實我們可以在程式中建立多個變數，舉例來說，指定不同的值到 eggs 和 bacon 兩個變數中，如下所示：

```
>>> bacon = 10
>>> eggs = 15
```

現在 bacon 變數中放入 10，而 eggs 變數則放了 15。每個變數都有屬於自己的盒子來存放東西，如圖 1-5 所示。

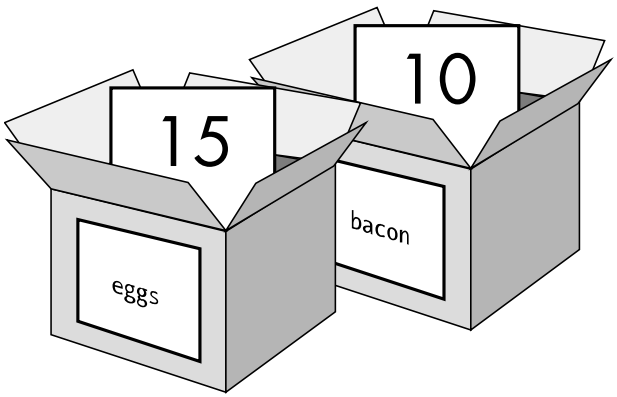


圖 1-5：bacon 和 eggs 變數各自存放不同的值

輸入 spam = bacon + eggs 到互動式 Shell 中，再查看 spam 的新結果為何：

```
>>> bacon = 10
>>> eggs = 15
>>> spam = bacon + eggs
>>> spam
25
```

上述例子結果，spam 為 25。當我們把 bacon 和 eggs 加起來時，其實是把存在變數中的值拿來相加，也就是 10 + 15。各自的變數含有各自的值，而不是表示式，所以 spam 變數最後會以 25 指定進去，而不是 bacon + eggs 這個表示式。經過 spam = bacon + eggs 指定陳述句的指定後，是把 25 的值指定到 spam 中，之後就算變更 bacon 和 eggs 變數的內容也不會影響 spam。

總結

在本章中，我們學習了編寫 Python 程式指令的基本觀念。由於電腦只看得懂特定的指令而不是一般的語句文法，因此 Python 需要用正確的語法來指揮它幫您做事。

表示式由多個值（如 2 或 5）和運算子（如 + 或 - ）連接在一起的。Python 會運算求解表示式，也就是把表示式運算求解成為單一個值。我們可以把值存放到變數中，這樣我們就能在程式中隨時取用。

在 Python 中有各式各樣的運算子和值，在下一章的內容中，我們要學習更多基本觀念和寫出第一支程式。也會學習在表示式中處理文字，Python 並不只限於運算數字而已，它還有更多的功用呢！

第 2 章
編寫程式

現在我們來學習 Python 對文字能作什麼樣的處理，幾乎所有的程式都會顯示文字與使用者溝通，而使用者也會由鍵盤輸入文字到程式中。在本章內容中，將製作屬於您的第一支程式，這程式就包含上述的兩項功用。我們將學到怎麼把文字存放到變數中，怎麼合併文字，以及將文字顯示到畫面中。這支程式會顯示問候文字「Hello World!」，並詢問使用者的名字。

本章包含的主題：

- ▶字串
- ▶字串的連接
- ▶資料型別（如字串或整數型別）
- ▶使用 File Editor 來編寫程式

▶在 IDLE 中儲存和執行程式

▶執行的流程

▶注釋

▶print()函式

▶input()函式

▶區分大小寫

字串

在 Python 中，文字值都稱為**字串**（strings）。字串值能像整數或浮點數一樣取用，也能存放到變數內。在程式碼中，字串值的起始和結尾都是用半型單引號「'」來包住。請在互動式 Shell 中輸入如下內容：

```
>>> spam = 'hello'
```

單引號告知 Python 字串的開頭和結尾，但它們並不屬於字串值本身。現在假設您在互動式 Shell 中輸入 spam，就會看到 spam 變數中所存放的內容了。請記住，Python 會以存放在變數中的值來運算，在這個例子中，變數中的值就是字串'hello'。

```
>>> spam = 'hello'
>>> spam
'hello'
```

字串可以是鍵盤任何的字元，其長短隨意。以下是字串的一些例子：

```
'hello'
'Hi there!'
'KITTENS'
'7 apples, 14 oranges, 3 lemons'
'Anything not pertaining to elephants is irrelephant.'
'A long time ago, in a galaxy far, far away...'
'O*&#wY%*&OCfsdYO*&gfC%YO*&%3yc8r2'
```

字串連接

利用運算子製作表示式來連接字串，其作法和處理整數與浮點數是一樣的。當您想要把兩個字串連接在一起，可用+運算子來處理，這就是**字串連接**（string concatenation）。請在互動式 Shell 中輸入'Hello' + 'World!'試試：

```
>>> 'Hello' + 'World!'
'HelloWorld!'
```

這行表示式會運算成單一個字串值 'HelloWorld!'。字串中沒有空格，因為在例子中要連接的兩個字串本來就沒有放空格，不像下面的例子：

```
>>> 'Hello ' + 'World!'
'Hello World!'
```

+ 運算子在處理字串和整數值的結果是完全不同的，原因在於處理的**資料型別**（data type）不同。所有的值都有其資料型別，像 'Hello' 值的資料型別為字串，而 5 這個值的資料型別為整數。資料型別告知 Python 在運算表示式時，運算子要以什麼方式來運算求解。+運算子會對字串進行連接，而會對整數和浮點數進行數學的加法運算。

在 IDLE 的 File Editor 中編寫程式

直到現在為止，我們都是在 IDLE 的互動式 Shell 中一次輸入一行指令程式，當我們在編寫程式時需要很多行指令，並希望它們一起執行時，就要用到下面我們講述的作法了，現在我們就來編寫第一支程式吧！

除了直譯器之外，IDLE 還有一個 File Editor 功能，請點選互動式 Shell 上方的 **File** 功能表，然後選取 **New File** 指令項，這樣就會開啟一個空白的視窗可以讓我們輸入程式碼，如圖 2-1 所示。

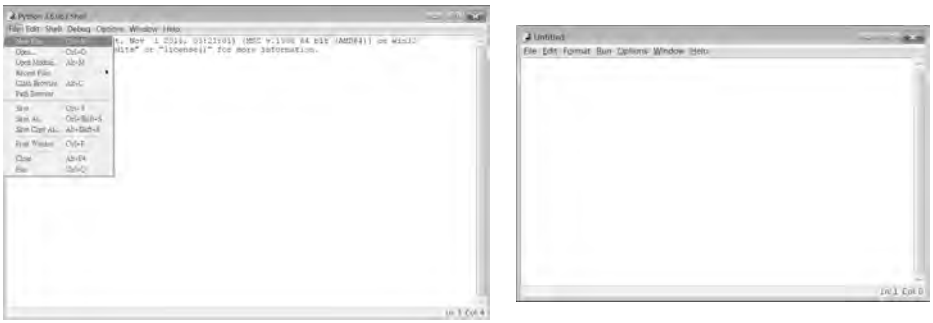


圖 2-1：左側為互動式 Shell 視窗，右側為開啟的 File Editor

兩個視窗有點像，但請記住：互動式 Shell 中有 >>> 提示符號，而 File Editor 中則沒有。

製作 Hell World 程式

依照傳統來說，程式設計新手製作的第一支程式通常是在螢幕上顯示「Hello World!」字樣。我們現在就來開始製作吧。

當您要輸入程式時，請記住不要在每行開頭加入行號，書中的例子是為了閱讀上的方便和引用時才印出行號的。在 File Editor 視窗右下角會顯示目前閃動游標所在的行號，如圖 2-2 所示目前是位在第 1 行（Ln: 1）和第 0 欄（Col: 0）。

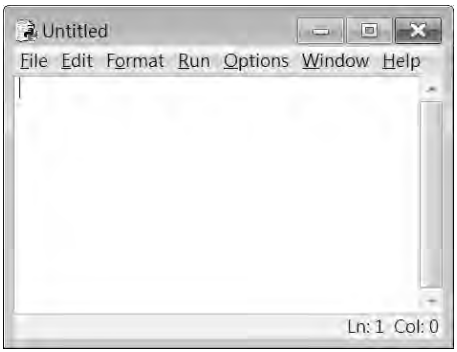


圖 2-2：File Editor 視窗右下角會顯示目前閃動游標的所在行和欄

請照下列內容輸入到新的 File Editor 空白視窗中，這就是程式的**原始程式碼**（source code）。內容含有讓 Python 逐一執行處理的指令。

⬇hello.py

```
1. # This program says hello and asks for my name.
2. print('Hello world!')
3. print('What is your name?')
4. myName = input()
5. print('It is good to meet you, ' + myName)
```

IDLE 的 File Editor 會對不同的指令碼以不同的彩色呈現。

在您輸入完上述的程式碼後，視窗看起來會如圖 2-3 所示。

圖 2-3：在 File Editor 中輸入程式碼後的樣子

請檢查一下您的 IDLE 視窗是不是看起來也像這樣。

儲存程式

當您輸入完程式碼後，請點選 **File→Save As** 指令，或按下 Ctrl-S 鍵來儲存。隨即會顯示另存新檔對話方塊，如圖 2-4 所示，在「檔案名稱」方塊中輸入 hello.py，再按下「**存檔**」鈕。

在編寫程式時最好隨時儲存，才不會在電腦當機或不小心結束 IDLE 時，讓辛苦輸入的程式遺失。

若想要開啟載入之前儲存的程式檔時，選取 **File→Open** 指令，選取對話方塊中存檔路徑內的 hello.py，再按下「**開啟舊檔**」鈕即可，剛才儲存的 hello.py 程式檔就會在 File Editor 中開啟。

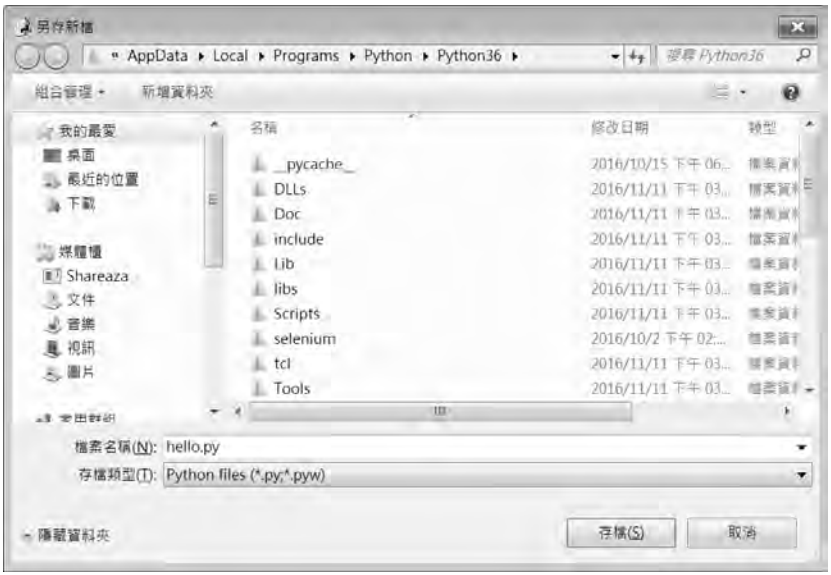


圖 2-4：儲存程式

執行程式

現在正是執行程式的時候了，請在 File Editor 視窗中點選 **Run→Run Module** 指令，或按下 F5 鍵。這樣程式就會在互動式 Shell 中執行。

這個範例程式會印出 Hello world! 並詢問您的名字，如圖 2-5 所示。

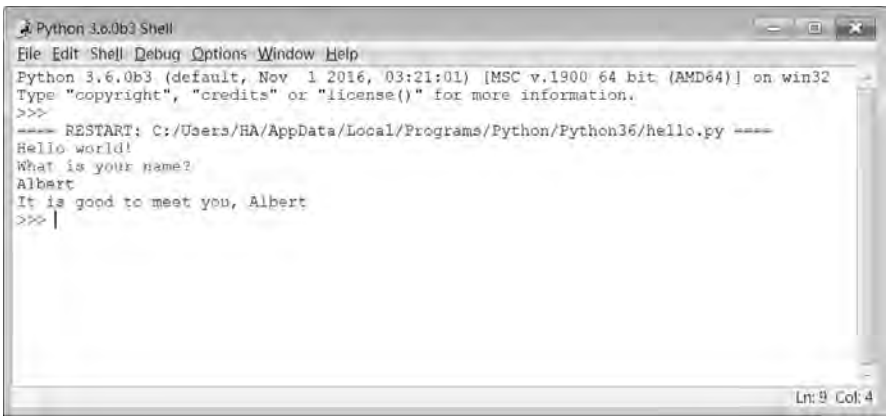


圖 2-5：執行 hello.py 程式後的互動式 Shell 畫面

當您輸入名字並按下 Enter 鍵，程式就會以您輸入的名字，秀出問候的文句。恭禧呀！您已編寫出第一支程式了，您現在已經是位電腦程式設計師了哦。按下 F5 再次執行，試著輸入別的名字看看結果如何。

如果您在上述程式中出現錯誤，請利用 https://inventwithpython.com/diff/#diff 網站的比對功能，可以幫您把輸入的程式碼和書中的範例程式碼進行比對，請複製和貼上您所輸入的程式碼，並按下「**Compare**」鈕。這樣就能找出差異所在，也會用反白的方式呈現在網站中，如圖 2-6 所示。

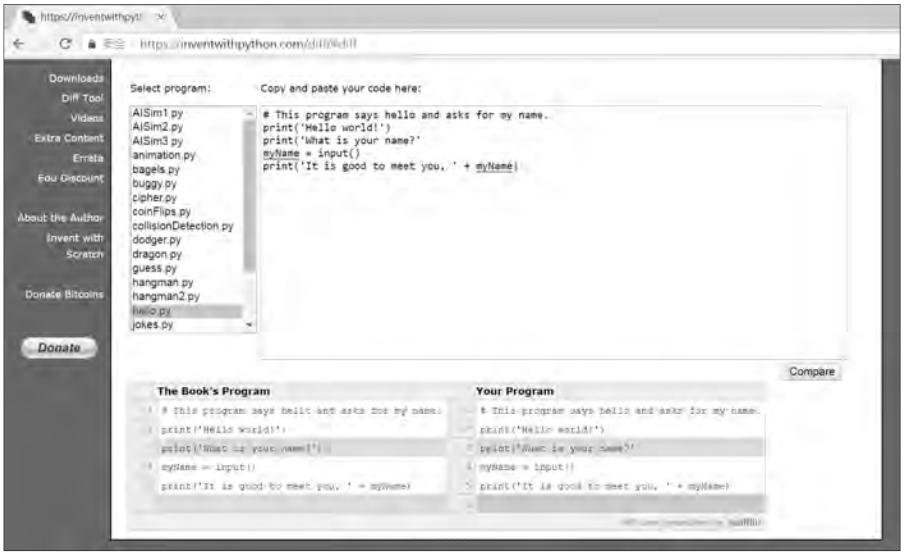


圖 2-6：使用 https://inventwithpython.com/diff/#diff 網站的比對功能

當我們在編寫執行程式時，如果出現如下所示的 NameError 錯誤訊息，則有可能是用了 Python 2 的舊版本而不是 Python 3。

```
Hello world!
What is your name?
Albert
Traceback (most recent call last):
  File "C:/Python26/test1.py", line 4, in <module>
    myName = input()
  File "<string>", line 1, in <module>
NameError: name 'Albert' is not defined
```

要修正這樣的錯誤，請安裝 Python 3.4 以上的版本來重新執行這支程式（安裝的相關訊息在本書「簡介」中有說明）。

Hello World 這支程式的運作原理

程式碼中每一行指令都會被 Python 直譯，這些指令組成了一支程式。電腦程式的指令就像是作料理時的步驟，Python 會依照順序完成每條指令，從程式最上端逐一往下處理到最後一行。

程式在 Python 中逐一處理的動作就稱之為「**執行**（execution）」。當程式啟動時，會由第一行指令開始執行，之後 Python 往下移到第二個指令，以此類推。

接下來我們一起看一看每行程式碼都是做什麼用的。首先從程式的第一行開始講解。

注釋

在 Hello World 程式中的第一行是個「注釋（comment）」。

```
1. # This program says hello and asks for my name.
```

任何在 # 字號後面的文字都是注釋，是程式設計師的筆記，寫下關於這程式是在做什麼的說明。Python 本身不會幫您寫，而是作為程式設計師的您自己為程式寫下的說明。程式設計師通常會在程式的最開頭放入注釋，用來寫下這支程式的標題及其相關說明。在 Hello World 程式中的注釋說明了這支程式會打招呼和詢問您名字。

函式：在程式中的小程式

函式（function）就像是程式中的小程式，內有一些指令可讓 Python 來執行。函式最大的好處是您只需要知道它們有什麼用途，而不用了解其內部是如何運作的。Python 本身提供了不少內建的函式可取用。我們在 Hello World 程式中所使用的 print() 和 input() 就是內建的函式。

函式呼叫（function call）是下一道指令告知 Python 要執行某函式中的程式碼。舉例來說，您的程式呼叫 print() 函式在螢幕上顯示字串，print() 函式就會把括號內您所輸入的字串如實顯示在螢幕上。

print() 函式

Hello World 程式中的第 2 行和第 3 行是呼叫 print() 函式：

```
2. print('Hello world!')
3. print('What is your name?')
```

在函式呼叫中以括號括住的值稱為**引數**（argument）。在第 2 行 print() 函式呼叫中的引數就是 'Hello world!'，而在第 3 行 print() 函式呼叫中的引數則是 'What is your name?'，這就是**傳**（passing）引數到函式中的意思。

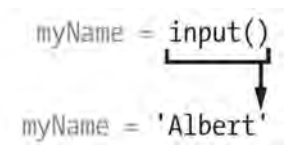
Input() 函式

第 4 行中的指定陳述句中是以函式呼叫 input() 指定到變數是 myName 中：

```
4. myName = input()
```

當 input() 被呼叫時，程式會等待使用者輸入文字，使用者輸入的文字字串就是函式呼叫運算的結果。在表示式中值可以用的地方，函式呼叫也一樣可以用。

這個函式呼叫的值稱之為**返回值**（return value。事實上「函式返回的值」和「函式呼叫運算的結果」意思是相同的）。在這個例子中，input() 所返回的值正是使用者所輸入的名字。如果使用者輸入 Albert，則 input() 函式呼叫運算結果就是字串 'Albert'，其過程如下：



這就是'Albert'字串值怎麼存放到 myName 變數中的過程。

在函式呼叫中的表示式

Hello World 程式中的最後一行有另一個 print() 函式呼叫：

```
5. print('It is good to meet you, ' + myName)
```

在 print()函式的括號中表示式為'It is good to meet you, ' + myName，由於引數通常都會是單一個值，Python 會先運算表示式成一個結果值再當成引數來傳送。

如果 'Albert' 存放在 myName 變數中，則這一整個表示式的運算過程如下：

```
print('It is good to meet you, ' + myName)
print('It is good to meet you, ' + 'Albert')
print('It is good to meet you, Albert')
```

這就是程式為何會以使用者輸入的名字來顯示問候文句的原理。

程式的結束

當程式執行到最後一行時就會終止或離開，意思是程式會停止執行。Python 會清掉所有放在變數中的值，包括存放在 myName 變數的字串也會被丟掉。如果您再次執行程式，並輸入另一個不同的名字，程式又會以新的名字來顯示：

```
Hello world!
What is your name?
Carolyn
It is good to meet you, Carolyn
```

請記住，電腦有點笨，它只會準確地按照您的程式指令去執行，也不會管您輸入的是誰的名字或什麼超蠢的文字，隨便您輸入什麼，電腦都會照樣處理：

```
Hello world!
What is your name?
poop
It is good to meet you, poop
```

變數的取名

為變數取一個能描述其存放內容的名字，會讓程式更易讀好懂。以前述的例子來說，您可以把 myName 變數取名為 abrahamLincoln 或 nAmE，對 Python 來說都只是個變數名字而已，它都能照樣執行，但這樣的名字會讓人搞不清楚這個變數是用來存放什麼內容的。如同第 1 章已提過的例子，如果您正為搬家打包東西，如果每個打包的箱子都貼上「**東西**（Stuff）」字樣，那跟沒貼是一樣的，對箱子裡到底裝了什麼一點幫忙都沒有！本書在互動式 Shell 所舉的例子中用了 spam、eggs 和 bacon 等變數名稱，那是因為舉的例子都很小，也僅是說明示範之意，所以沒什麼關係。不過，在本書後面較大型的範例中，則都會取有意義的變數名稱，因此將來您自己設計程式時也一樣要用有意義的變數名稱。

變數名稱的英文是有區分大小寫的，意思是相同的變數名稱若用了不同大小寫的英文，也會被看成是不同的變數。例如，spam、SPAM、Spam 和 sPAM 等對 Python 來說是四個不同的變數名稱，它們能各自存放各自的內容。在程式中最好不要取上述這種只有大小寫不同的變數名稱，能用更具描述性的單字來代表是比較好的取名方式。

變數名稱一般來說都是小寫的，但若變數是用二個以上的英文單字組合而成，則第一個單字之後的字母以大寫開頭來分隔是不錯的用法。舉例來說，變數名稱 whatIHadForBreakfastThisMorning 會比 whatihadforbreakfastthismorning 易讀。這種大小寫的取名法稱為「**駝峰式（camel case）**」（因為像駱駝背上的駝峰高低參差），能讓我們的程式更好閱讀。程式設計師在為變數取名字的時候也會儘量使用簡短易懂的組合：例如，使用 breakfast 或是 foodThisMorning 會比 whatIHadForBreakfastThisMorning 更好。這只算是慣例而已，是選擇性可不做的，但在 Python 程式設計中這是標準的做法。

總結

一旦您學會了如何使用字串和函式，就能製作出與使用者互動溝通的程式。這很重要，因為文字是電腦與使用者彼此互動交流的主要管道。使用者可藉由鍵盤輸入文字到 input() 函式內，而電腦則將文字透過 print() 函式顯示在螢幕上。

字串只是種新的資料型別，所有的值都有其資料型別，其資料型別會影響 + 運算子在運算處理時的作法。

在程式中，函式通常是用來帶出較為複雜的指令動作，Python 有很多內建的函式可以使用，將來我們在本書中會陸續學到。在表示式中，函式呼叫的用法和使用值的時候是一樣的。

Python 對程式的指令和步驟進行運算處理，就稱之為執行。在第 3 章中，我們將要學習更多關於執行處理的不同方式，除了由上而下直線式的流程外，還有其他的流程控制，一旦學會流程控制，就更具備了製作遊戲程式的能力。