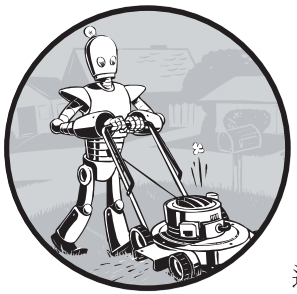
簡介



「您在二小時內完成的事，我們三個人花了兩天才搞定呀。」在 2000 年左右，我的大學室友在一家電器零售店工作，有時店裡會從競爭店家那收到一份有幾千件商品價格目錄的試算表檔案，由三位員工所組成的小組要印出這份試算表，一一拿來比對他們店裡的商品價格，並記錄下價格比競爭店家賣得貴的所有商品，通常這項工作得花上好幾天的時間。

「各位，您們知道嗎？如果有印出試算表的原始檔案，我可以寫一支程式來幫您們完成這項工作哦。」我的室友告訴他們，當時他看到這幾個人蹲在地上忙著查價格，周圍還散落一堆印出的表單紙張。

幾個小時後，他編寫了一支簡短的程式，從檔案中讀取競爭店家的價格資料，在自己店裡的資料庫中搜尋相同的商品來比對，並記錄下是否比競爭對手的價格高。他當時還是個程式設計新手，花了一些時間在一本程式設計書中查看說明文件，邊學邊寫出這支程式。最後程式實際上只花了幾秒鐘就跑完了，我的室友和他的同事們在那天享受了超長的午休時間。

這就是電腦程式設計的威力，電腦就像把瑞士刀，能幫您完成無數的工作。許多人花了數小時來點按滑鼠和鍵盤輸入來執行著重複的工作，但卻沒有體認到，只要對機器下達正確的指令，就能在幾秒鐘完成這些工作。

本書的適用對象

軟體是我們現在使用的許多工具中的核心：幾乎所有人都在使用社群網路來互動交流，很多人的手機都像是台連上網路的小電腦，現今大多數的辦公室工作都需要操作電腦來完成，因此，程式設計人材的需求很大。一大堆的書籍、互動式網路教室和系統開發新手補習班，都很想要把初學者變成軟體工程師，讓他們都有百萬年薪。

本書並不針對這類人而寫的，而是針對所有其他的人而設計的。

這本書不會讓您變成職業級的軟體開發人員，就像幾堂吉他課程不會讓您成為搖滾巨星。但如果您是辦公室的職員、經理人，在學研究者，或是會使用電腦來工作或娛樂的任何人，您都能在這本書中學到程式設計的基本知識，並能完成下列的這些簡單的自動化工作：

- 搬移並對數千個檔案重新命名，將其分類放入不同的資料夾。
- 填寫線上表單自動化，不用自己打字鍵入。
- 在網站有更新時，從網站下載檔案或複製想要的文字。
- 使用電腦對客戶傳送簡訊通知。
- 更新與美化 Excel 試算表。
- 檢查 Email 並傳送預先寫好的回覆。

這些工作對人來說很簡單，但卻要花很多時間。這些工作大都很零散瑣碎，都有其特殊性而沒有現成的軟體可幫忙完成，但只要有一點程式設計的知識，就能利用電腦來幫您完成這些工作。

本書的程式風格

本書並不是定位成一本參考手冊，而是定位成初學者的指南，書中程式風格有時並不是最佳的實作示範（例如，有些程式會用全域變數），但會折中取捨，好讓程式碼變更簡單，方便學習。本書的目標是讓大家設計寫出能用但也可以馬上丟棄的程式碼，所以不會花太多時間來關心風格和優雅的議題。複雜的程式設計觀念（例如，物件導向程式設計、串列的原理和產生器之類）在本書中不會介紹，因為會增加太多複雜性。程式設計老手可能會覺得本書中有些程式碼可修改得更有效率，但這不是首要考量，本書主要的重心是放在用最少的時間和工作量就能讓程式運作。

什麼是程式設計

在電視劇和電影中，有時會看到程式設計人員在閃動的螢幕中快速輸入一大堆神秘 0 與 1 之類的編碼，但現代的程式設計並沒有這麼神秘。程式設計只是簡單地輸入指令來指示電腦執行的過程，這些指令可能會運算一些數字、會修改文字、會在檔案中搜尋資訊，或利用網路與其他電腦通訊。

所有程式都把基本指令當作積木組件，下列是一些常用的指令，並以自然口語的形式來表示：

「做這個，然後這那個。」

「如果這個條件為真，執行這個動作，不然執行那個動作。」

「依照執行這個動作 27 次。」

「一直做這個，直到條件為真才停止。」

也可把這些積木組件組合起來，作出更複雜的決定，舉例來說，這裡有些程式指令稱為原始程式碼，是用 Python 語言所設計編寫的一個簡單程式。Python 軟體會從這個例子的開端執行每一行程式碼（有些程式只有在 if 特定的條件為真時才執行，或者執行 else 部分的程式），一直到最底部。

```
❶ passwordFile = open('SecretPasswordFile.txt')
❷ secretPassword = passwordFile.read()
❸ print('Enter your password.')
  typedPassword = input()
❹ if typedPassword == secretPassword:
❺    print('Access granted')
❻    if typedPassword == '12345':
❼        print('That password is one that an idiot puts on their luggage.')
  else:
❽    print('Access denied')
```

您或許對程式設計沒有概念，但讀了上述的程式碼範例，也能合理猜出它在做什麼事情。首先，開啟了 SecretPasswordFile.txt 檔❶，讀取了其中的密碼資料❷，隨後提醒使用者（使用鍵盤）輸入一個密碼❸，接著比對這兩個密碼❹，如果相同則程式在螢幕上印出 Access granted ❺，隨即檢查密碼是否為 12345 ❻，如果是則提示說明這不是個好的密碼❼。如果比對兩個密碼後不相同，程式在螢幕上印出 Access denied ❽。

什麼是 Python

Python 指的是一種程式語言（包含語法規則，用來設計寫出有效合法的 Python 程式碼），和 Python 直譯器軟體，它讀取程式碼（以 Python 語法寫成），並執行其中的指令。Python 直譯器可從 http://python.org/ 免費下載，有 Linux、macOS 和 Windows 等版本。

Python 的名字的靈感取自於英國超現實喜劇劇團 Monty Python，而不是指「蟒蛇」。Python 程式設計人員被暱稱為 Pythonistas。Monty Python 和蛇的一些參考常出現在 Python 的教學指南和說明文件中。

程式設計人員其實不用懂太多數學

我聽過關於學習程式設計最常見的顧慮，就是大家都認為要懂很多數學知識，事實上大多數程式設計所需要的數學知識大都是基本的算術運算而已。實際上，很會設計程式和很會解數獨遊戲沒什麼太大差別。

要解出數獨問題，數字 1 到 9 必須要填入 9×9 的棋格上的每一欄每一列，以及每個 3×3 的內部方塊中。利用推導和起始數字的邏輯，您就能找到答案。舉例來說，在圖 1 中的數獨問題內，5 出現在第一和第二列，它就不能再出現在這二列中，所以右上角的 3×3 方格中 5 就要放在第三列。由於最左一欄已有 5，

所以 5 不能放在 6 的右側，只能放 6 的左側。每次解決一列、一欄或一個方塊，為剩下的部分找出更多數字的線索，當完成一組 1-9 的數字後再繼續完成其他的，這樣很快就能填完整張數獨表格。

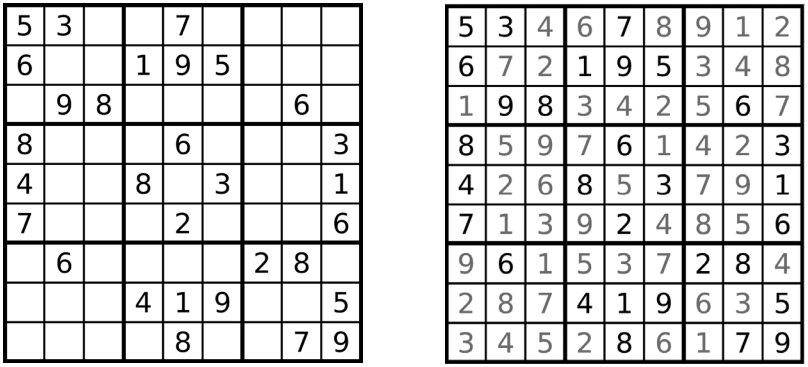


圖 1　一個新的數獨問題（左圖）和解答（右圖）。雖然用了數字，並不表示要用很多數學知識才能解題（Images © Wikimedia Commons）

只因為數獨問題用了數字，並不表示要精通數學才能解出答案，程式設計也是這種情況。就像解開數獨問題一樣，設計程式需要把問題分解出單個和細部的步驟。相同地，在程式除錯時（是指找出錯誤和修改錯誤），您會耐心觀察程式在做什麼，找出有問題的原因。就和所有其他技能一樣，設計和寫出愈多程式，您就愈能掌握其技巧。

學程式設計並不受年齡限制

第二個關於學習程式設計最常見的顧慮是，覺得自己太老了而無法學習程式設計。我從很多網路評論和留言中看到，不少人認為自己當下想要學習已晚了，因為已經（天啊！）23 歲了。顯然這不是「太老」而無法學習程式設計，因為有很多人在更老的年級都還在學習。

您不需要從小就開始成為很有實力的程式設計師，但是會程式設計的小孩一直給人有神童般的形象。不幸的是我也為這樣的神話做出了貢獻，因為我告訴別人我在讀小學時就開始寫程式了。

但是，現在的程式設計比 1990 年代更容易學習。現今有更多的書籍、更好的網路搜尋引擎，以及更多的社群網站可參考。最重要的是，程式語言本身更容

易使用。由於這些原因，**現在大約只要花幾個週末好好學習，就可以學到我從小學到高中畢業所累積的程式設計知識**。說實在的，我並沒有真的因為很早學習而贏在起跑點。

更重要的是對程式設計要有「累積成長的心態」，換句話來說，就是要了解到大家是透過實務應用來累積程式設計的技能。大家都不是一生下來就是程式設計師，現在還不具備程式設計的技能，並不代表以後不能成為專家。

程式設計是創造性的活動

程式設計是創造性的工作，有點像繪畫、寫作、編織，或用樂高積木建構一座城堡。就像在空白畫布上繪畫創作，製作軟體有很多限制，但也有無限可能。

程式設計與其他創造性活動的不同之處是在編寫程式時，您所需要的所有原料都在電腦中，您不用再額外採買畫布、顏料、底片、紗線、樂高積木或電子零件了。就算是十年前的舊電腦也足夠強大來讓我們編寫程式。在程式寫好之後，就能無限制地複製取用，編織好的毛衣只能讓一個人溫暖，但好用的程式卻能很容易地分享到全世界。

本書簡介

本書的 Part 1 為介紹 Python 程式設計的基本概念，Part 2 則是以一些不同的應用實例專題導入，教您用電腦來完成自動化的作業。Part 2 的每一章都有一些範例程式專題讓您實作學習。下面簡單介紹每一章的內容：

Part 1：Python 程式設計基礎

第 1 章：Python 基礎，內容包括表示式、大部分的 Python 指令的基本型別，以及怎麼使用 Python 互動環境來執行程式碼。

第 2 章：流程控制，解釋了如何讓程式決定執行哪些指令，好讓程式碼能聰明地回應不同的情況。

第 3 章：函式，介紹如何定義自己的函式，這樣能把程式碼組織成好管理的區塊。

第 4 章：串列，介紹了串列資料型別，並討論介紹了如何組織管理資料的方法。

第 5 章：字典與結構化資料，介紹了字典資料型別，並示範了更強大的資料組織管理方法。

第 6 章：字串的操作，內容含括處理文字資料（在 Python 中稱為字串）的各種操作。

Part 2：自動化專題實作

第 7 章：使用正規表示式進行模式比對，內容包含 Python 如何使用正規表示式處理字串和搜尋的文字模式。

第 8 章：輸入驗證，本章說明程式如何驗證使用者所提供的資訊，以確保使用者的資料用了正確的格式，不會在程式的其餘部分引起錯誤。

第 9 章：讀寫檔案，本章說明了程式如何讀取文字檔的內容，以及怎麼將資訊儲存到硬碟的檔案中。

第 10 章：檔案的組織管理，本章示範 Python 如何使用比手動操作更快速的複製、搬移、重新命名和刪除大量的檔案，也介紹了壓縮和解壓縮檔案的內容。

第 11 章：除錯，本章示範如何使用 Python 的各種 Bug 搜尋和修復工具。

第 12 章：從 Web 擷取資訊，示範了如何以程式來自動下載網頁，並解析取得資訊，這就是 Web 擷取資訊。

第 13 章：處理 Excel 試算表，本章介紹如何編寫程式來自動化處理 Excel 試算表，如果您需要分析和處理數千個試算表檔案時，這會很有幫忙。

第 14 章：處理 Google 試算表，本章介紹如何使用 Python 來讀取和更新 Google 試算表，這套著名的網路試算表軟體。

第 15 章：處理 PDF 與 Word 文件，本章介紹如何編寫程式來讀取和處理 Word 和 PDF 檔。

第 16 章：處理 CSV 檔和 JSON 資料，繼續說明怎麼設計和編寫程式來處理文件，本章討論的是 CSV 和 JSON 檔案。

第 17 章：保持時間、工作排程和程式啟動，本章說明了 Python 程式怎麼處理時間和日期，如何排程電腦在特定時間執行工作。同時也示範了 Python 程式如何啟動電腦中的其他應用程式。

第 18 章：發送 Email 和文字簡訊，本章介紹了如何設計和編寫程式來傳送 Email 和簡訊。

第 19 章：處理影像圖片，本章解釋了如何設計和編寫程式來處理 JPG 或 PNG 這類的影像圖片。

第 20 章：以 GUI 自動化來控制鍵盤和滑鼠，本章說明如何設計和編寫程式來控制鍵盤和滑鼠，自動化來點按滑鼠和鍵盤的鍵入。

附錄 A：安裝第三方模組，介紹怎麼利用其他模組來擴充 Python 功能。

附錄 B：執行程式，說明如何在 Windows、macOS 和 Linux 中從程式碼編輯器外部執行 Python 程式。

附錄 C：習題解答，提供每一章後習題的解答以及一些額外參考內容。

下載和安裝 Python

可依自己系統的需要從 http://python.org/downloads/ 免費下載 Windows、macOS 或 Ubuntu 的 Python 版本，就算您從該網站的下載頁面下載了最新版本，本書中的相關程式範例應該都能運作。

NOTE

請確定您下載的 Python 是 3 以上的版本（例如 3.8.0）。本書中的程式是以 Python 3 以上版本來測試執行的，有部分程式在 Python 2 版中也許不能正常執行。

您必須先了解自己的電腦作業系統是什麼，然後在下載頁面中找到是針對 64 位元或 32 位元系統或特定作業系統的 Python 安裝程式，如果您的電腦是 2007 年之後才購買的，很可能作業系統是 64 位元的，不然就是 32 位元的系統。您可以從下列方法來確定：

- 在 Windows 系統中，點選「**開始→控制台→系統**」，然後檢查系統類型是 64 位元或是 32 位元的。

- 在 macOS 系統中，進入 Apple 功能表，點選「**About This Mac→More Info →System Report→Hardware**」，然後檢查 Processor Name 欄位。如果是 Intel Core Solo 或 Intel Core Duo，則機器是 32 位元的，如果是其他（Intel Core 2 Duo），則機器是 64 位元的。
- 在 Ubuntu Linux 系統中，開啟終端機，輸入執行 **uname -m**，結果顯示是 i686 的話是 32 位元的，若是 x86_64 則表示是 64 位元的。

在 Windows 系統中，下載 Python 安裝程式（副檔名為 .msi），連按二下執行，然後依照安裝程式的指引來完成安裝。其步驟大致如下：

1. 選擇 **Install for All User**，然後按下 **Next**。
2. 接受預設的選項，按下 Next 鈕經過多個步驟，完成安裝。

在 macOS 系統中，下載適合您系統版本的 .dmg 檔，連按二下執行，然後依照安裝程式的指引來完成安裝。其步驟大致如下：

1. 當 DMG 套件在新視窗中開啟時，連按二下 Python.mpkg 檔。您可能要輸入系統管理員的密碼。
2. 點按 **Continue**，跳過歡迎部分，按下 **Agree**，同意授權。
3. 在最後的視窗按下 **Install**。

如果是在 Ubuntu 系統中，可從終端機視窗中安裝 Python，步驟大致如下：

1. 開啟終端機。
2. 輸入 **sudo apt-get install python3**
3. 輸入 **sudo apt-get install idle3**
4. 輸入 **sudo apt-get install python3-pip**

下載和安裝 Mu

Python 直譯器是執行 Python 程式所需的軟體，而 Mu 編輯器軟體則是讓我們輸入程式碼的地方，這裡的工作方式很像在文書處理器中輸入內容。讀者可以從 https://codewith.mu/ 下載 Mu。

在 Windows 和 macOS 中，下載適用於自己作業系統的安裝程式，然後連按二下安裝程式檔即可執行安裝。如果您使用的是 macOS，則執行安裝程式時會開啟一個視窗，您必須把其中的 Mu 圖示拖曳到 Applications 檔案夾圖示才能繼續安裝。如果您使用的是 Ubuntu 系統，則需要把 Mu 當作 Python 軟體套件來安裝。在這種情況下，請點按 Download 頁面「Python Package」部分中的「Instructions」按鈕。

啟動 Mu

安裝之後，就可啟動 Mu：

- 在 Windows 7 或更新的版本中，點按 Windows 畫面左下角的開始圖示，在搜尋方塊中輸入 **Mu**，然後選取它。
- 在 macOS 中，開啟 Finder 視窗，點按 **Application**，然後點按 **mu-editor**。
- 在 Ubuntu 系統中，選取 **Applications→Accessories→Terminal**，然後輸入 **python3 -m mu**。

Mu 第一次執行時，會顯示一個「Select Mode」視窗，其中包含選項 Adafruit CircuitPython、BBC micro:bit，Pygame Zero 和 Python 3。請選取 **Python 3**。之後可透過編輯器視窗最上方的「Mode」按鈕來更改模式。

NOTE

請下載 Mu 版本 1.10.0 或更高版本，這樣才能安裝本書所介紹的第三方模組。在撰寫本書時釋出的是 1.10.0 Alpha2 版本（2020 年 4 月），在 Download 頁面中與主要下載連結分開，這項連結單獨列在最上方。

啟動 IDLE

Python 直譯器是執行 Python 程式的軟體，而互動開發環境（IDLE）是輸入程式指令的視窗，像是個文書處理軟體。現在讓我們啟動 IDLE 吧！

- 在 Windows 7 或更新的版本中，按下螢幕左下角的開啟圖示鈕，在搜尋文字方塊中輸入 **IDLE**，再點選顯示在功能表上的 **IDLE (Python GUI)**。
- 在 macOS 中，開啟 Finder 視窗，點按 **Applications**，再點按 **Python 3.8**，隨後點按 IDLE 圖示啟動。
- 在 Ubuntu 中，選取 **Application→Accessories→Terminal**，然後輸入 **idle3**（也許您也可以點按螢幕上的 **Applications**，選取 **Programming**，再按下 **IDLE 3**）。

Shell 互動模式

執行 Mu 時，出現的視窗稱為檔案編輯器（file editor）視窗。可透過點按 REPL 按鈕開啟互動式 Shell 模式。Shell 是個程式，能讓我們在電腦中輸入指令，就像在 macOS 和 Windows 的「終端機」或「命令提示字元」模式一樣。使用 Python 的互動式 Shell 模式，可以輸入讓 Python 直譯器軟體執行的指令。電腦會讀取您輸入的指令並馬上執行。

在 Mu 中的互動式 Shell 視窗會顯示類似下列這般的版本文字訊息，而視窗後半部分都是空的：

```
Jupyter QtConsole 4.3.1
Python 3.6.3 (v3.6.3:2c5fed8, Oct 3 2017, 18:11:49) [MSC v.1900 64 bit
(AMD64)]
Type 'copyright', 'credits' or 'license' for more information
IPython 6.2.1 -- An enhanced Interactive Python. Type '?' for help.

In [1]:
```

如果您執行的是 IDLE，第一次啟動所顯示的是互動式 Shell 視窗，此視窗大部分都是空白，只顯示像下面這樣的文字：

```
Python 3.8.0b1 (tags/v3.8.0b1:3b5deb0116, Jun 4 2019, 19:52:55) [MSC v.1916
64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

In [1]: 和 >>> 都是提示符號，書中的範例會使用 >>> 來作為互動式 Shell 的提示符號，因為它最常見，如果您從終端機或命令提示字元模式執行 Python，也一樣會使用 >>> 這個提示符號。In [1]: 這個符號最先由 Jupyter Notebook 使用，這也是個很著名的 Python 編輯器。

舉例來說，在 Python Shell 互動環境中的 >>> 提示符號後面輸入如下指令：

```
>>> print('Hello world!')
```

輸入完這行指令後按下 Enter 鍵，則 Shell 互動模式會顯示如下內容來回應：

```
>>> print('Hello world!')
Hello world!
```

剛剛給電腦下了一條指令，而它也完成了要執行的操作！

安裝第三方模組

有些 Python 程式碼需要在程式中匯入模組。其中某些模組是 Python 內建隨附的，而有些模組則是由 Python 核心開發團隊之外的開發人員所建置的第三方模組。附錄 A 詳細介紹了怎麼使用 pip 程式（在 Windows 中）或 pip3 程式（在 macOS 和 Linux 中）安裝第三方模組。當本書指示您要安裝某個特定的第三方模組時，請查閱參考附錄 A。

如何尋找說明文件

程式設計師很喜歡透過在網路搜尋問題的答案來學習。這與許多人所習慣的學習方式不同，一般人大都是透過親自上課和可回答問題的老師來學習的。把網路當作教室的最大好處是，整個社群的高手們都可以回答您的問題。

實際上，您的問題很可能別人已提問過，這些答案正在線上等著您找出來。如果您遇到某個錯誤訊息或某段程式碼無法正常運作，其實您可能不是第一個遇到這些問題的人，找答案比您所想像的要容易多了。

舉例來說，我們故意製造一些錯誤：在 Shell 互動環境中輸入「**'42' + 3**」，現在您不需要知道這指令是什麼意思，但執行結果會像下列這般：

```
>>> '42' + 3
❶ Traceback (most recent call last):
  File "<pyshell#0>", line 1, in <module>
    '42' + 3
❷ TypeError: Can't convert 'int' object to str implicitly
>>>
```

這裡顯示了錯誤訊息❷，因為 Python 不能理解您輸入的指令，錯誤訊息中的 Traceback 部分❶顯示了 Python 所遇到問題的特定指令和行號，如果您不知怎麼處理這個錯誤訊息，可連上網路來搜尋這條錯誤訊息，在您慣用的搜尋引擎網頁中輸入 **"TypeError: Can't convert 'int' object to str implicitly"**（包含引號），您就會找到一堆連結，其中有許多解釋這條錯誤訊息的資訊，以及什麼原因造成這項錯誤的發生，如圖 2 所示。

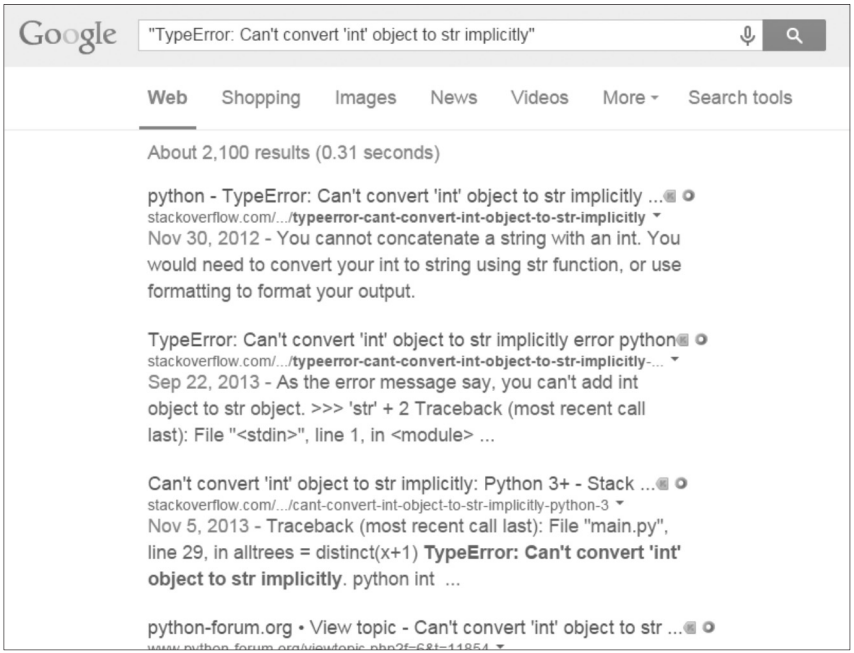


圖 2　使用 Google 搜尋錯誤訊息的結果

您會發現，網路上有一堆人也遇到了同樣的問題，而且已經有好心人回答解決了這個問題。沒人能全面掌握所有程式設計的各種技能，所以所有的軟體開發人員每天都有項重要的工作，那就是上網尋找技術問題的解答。

正確地發問

如果在網路的搜尋中找不到答案，請試著在連到 Stack Overflow 網站的討論區（http://stackoverflow.com/）、或是在 "learnprogramming" subreddit 網站留言板（https://reddit.com/r/learnprogramming/），或一些程式設計討論群組、論壇中提問。但請記住，要問對問題並正確地發問，這有助於讓別人協助您。先閱讀討論區或論壇中的 FAQ（常見問題），了解正確發問的方式。

在提出問題時，請記住下列幾點：

- 說明您想要做的是什麼，而不是只說您做了什麼。這樣能讓幫助您的人知道您是否走錯路了。
- 明確指出發生錯誤的地方，它是在程式每次執行時發生，還是您在做了什麼動作後才發生的。
- 將完整的錯誤訊息和您的程式碼複製貼上 http://pastebin.com/ 或 http://gist.github.com/。

  這些網站能讓您很容易在網路上與別人共享大量的程式碼，程式放在這些網站上其編排格式不會遺失。隨後您可以把貼上程式碼網站的 URL 放入 Email 或論壇的貼文中。例如，下列是我貼出的一些程式碼內容：http://pastebin.com/SzP2DbFx/ 和 https://gist.github.com/asweigart/6912168/。
- 解釋您為解決這個錯誤和問題已經試過了那些方法，這也會讓別人知道您已做了一些功課，而不是無腦的提問。
- 列出您使用的 Python 版本（Python 2 的直譯器和 Python 3 的直譯器有些不同）。最好也說明您所使用的作業系統和版本。
- 如果錯誤在您改了程式之後才出現，請準確指出您到底改了什麼？
- 說明這個錯誤是否在您每次執行這程式時重現，或是它在某些特定操作下才會發生，如果是這樣，要說明一下您做了什麼操作。

請遵守網路的禮節，例如，不要都寫大寫的英文提問，或對試圖幫您的人提出無理的要求。

您可以在 https://autbor.com/help/ 的部落格文章中找到有關如何尋求程式設計協助的更多資訊。也可以在 https://www.reddit.com/r/learnprogramming/wiki/faq/ 上找到關於程式設計常見問題的清單，並可在 https://www.reddit.com/r/cscareerquestions/wiki/index/ 上找到有關從事軟體開發工作的類似清單。

我很喜歡幫助大家發掘 Python 的魅力。我在 https://inventwithpython.com/blog/ 網站中的部落格內寫了不少關於程式設計的教材資料，請隨時連上去查閱參考。另外如果有什麼疑問，您可以利用 al@inventwithpython.com 電子郵件與我連絡。此外，您還可以把問題發佈到 https://reddit.com/r/inventwithpython/，我會盡可能回應。

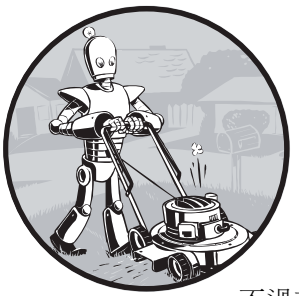
總結

對大部分的人來說，電腦只是某種裝置設備而不是工具，但如果學會如何設計編寫程式，您就有能力發揮現今這個最強大的工具了，這樣您也會很有成就感。程式設計不是什麼腦科手術，不用那麼精準，對業餘人士來說，設計程式完全可以犯點錯誤和一試再試。

本書是從零開始講述程式設計的基本知識，但您的問題可能超出本書的講解範圍，請記得如何有效正確地提出問題，知道如何去尋找答案對您的程式設計之旅會是無價的工具。

讓我們一起出發吧！

第 1 章
Python 基礎



Python 程式語言有很多的語法結構、標準函式庫和互動式的開發環境等功能。別擔心太難學，您可以略過大多數的內容，只需學習其中一小部分，就能編寫出方便好用的小程式。

不過在您開始編寫程式之前，還要學一些基本的程式設計概念。就像參加魔法學校訓練一樣，您可能認為這些基本概念有點神秘又單調，但只要學會了這些知識和實作方法，就可以像拿著魔法棒一樣指揮電腦完成超讚的工作。

本章有一些實例讓讀者可在互動式 Shell 模式中輸入一些指令和程式，這些稱為 PERL（讀取－求值－輸出循環），執行後馬上可看到結果。利用互動式 Shell 模式來體會學習 Python 的基本指令是很好的開始，請跟著書中的例子動手試一試。邊看書邊動手實作會比只看書中內容更有學習效果。

在互動式 Shell 模式中輸入表示式

我們可以透過啟動 Mu 編輯器來執行互動式 Shell 模式，在閱讀本書「簡介」這章中的設定說明時應已下載和安裝 Mu 編輯器。若在 Windows 中，點開「開始」功能表，輸入「Mu」，然後啟動 Mu 應用程式。若在 macOS 內，開啟「應用程式」檔案夾，然後連按二下 **Mu**。按下 **New** 按鈕，然後把空白的檔案另存為 blank.py。按下 **Run** 按鈕或按 F5 鍵執行此空白檔案時，會開啟互動式 Shell 模式，這個 Shell 會以新的窗格開啟，並顯示在 Mu 編輯器視窗的底部，此時您應該會在互動式 Shell 模式中看到 >>> 提示符號。

請在提示符號輸入 **2 + 2**，讓 Python 做些簡單的運算。Mu 視窗的 Shell 窗格內看起來應該會是下列這般：

```
>>> 2 + 2
4
>>>
```

在 Python 中，2 + 2 稱為「表示式（expression）」，它是語言中最基本的程式指令結構。表示式包括「值」（例如 2）和「運算子」（例如+），然後運算求解（歸併）成單一個值，也就是說，在 Python 程式碼中能用表示式的地方，也一樣可以用單個值來陳述。

在前述的例子中，表示式 2 + 2 運算求值為單個值 4。沒有用運算子的單個值也被認定為表示式，單個值在運算求解時其實也是它自己，像下面的實例：

```
>>> 2
2
```

出現 Error 也 OK！

如果程式中有電腦無法理解的程式碼內容就會當掉，在 Python 中遇到這種情況會顯示錯誤（Error）訊息。錯誤訊息並不會弄壞電腦啦，所以不要怕犯錯。「當機（Crash）」的意思只是程式在不預期的情況下停止運作了而已，不用太擔心。

> 假如讀者想對錯誤訊息有更深入的了解，可把錯誤訊息文字丟上 Google 之類的網站來搜尋，就能找到更多關於這個錯誤訊息的內容連結。另外也請連到 https://nostarch.com/automatestuff2/ 這裡的連結資源，其中有收集了常見的 Python 錯誤訊息和其解釋說明的清單列表。

Python 表述示中還可使用其他很多的運算子，例如，表 1-1 就列出了 Python 的所有數學運算子。

表 1-1　優先等級從高到低的數學運算子

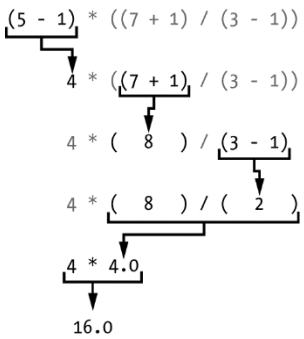
運算子	運算功能	實例	運算結果...
**	指數	2 ** 3	8
%	模數/餘數	22 % 8	6
//	整除/商數取整	22 // 8	2
/	除法	22 / 8	2.75
*	乘法	3 * 5	15
-	減法	5 - 2	3
+	加法	2 + 2	4

Python 數學運算子的運算順序（也稱為優先等級）和數學是一樣的。** 運算子會第一個先運算，接下是 *、/、// 和 % 運算子，從左而右運算。+ 和 - 運算子最後運算，也是從左而右。如果有需要可用括號來改變其優先順序，運算子和值之間的空格在 Python 中並沒有影響（在每行最前面空格縮排除外），請動手在互動式 Shell 中輸入以下的表示式：

```
>>> 2 + 3 * 6
20
>>> (2 + 3) * 6
30
>>> 48565878 * 578453
28093077826734
>>> 2 ** 8
256
>>> 23 / 7
3.2857142857142856
>>> 23 // 7
3
```

```
>>> 23 %  7
2
>>> 2      +          2
4
>>> (5 - 1) * ((7 + 1) / (3 - 1))
16.0
```

在前述的例子中，您扮演著程式設計師角色輸入表示式，而 Python 則完成較難的運算求解工作，將表示式運算歸併成單一個值。Python 遇到複雜的表示式時會持續對表示式各部分先運算，直到歸併成單一個值為止。如下圖所示。



把運算子和值放在表示式裡也有規則，是 Python 程式語言最基本的部分，就像我們言語溝通時所用的文法規則一樣。舉例來說：

This is a grammatically correct English sentence.

This grammatically is sentence not English correct a.

第二行的句子很難懂，因為它不合英語的文法規則。同樣地，如果輸入錯誤的 Python 指令，Python 也會看不懂，因此就會顯示錯誤訊息，像下面的例子：

```
>>> 5 +
  File "<stdin>", line 1
    5 +
      ^
SyntaxError: invalid syntax
>>> 42 + 5 + * 2
  File "<stdin>", line 1
    42 + 5 + * 2
             ^
SyntaxError: invalid syntax
```

我們可在互動式 Shell 中輸入某一指令，檢查該指令是否能運作。別擔心會弄壞電腦，最差的情況只是 Python 會顯示錯誤訊息而已。就算是專業的軟體開發工程師在編寫程式碼時，一般也都常會遇到錯誤訊息。

整數、浮點數和字串資料型別

請再記一次，表示式是由值和運算子組成，然後運算歸併成為單一個值。「資料型別（data type）」是「值」的分類，每個值都只屬於某一類的資料型別。表 1-2 列出了 Python 最常見的資料型別，例如，值 -2 和 30 屬於「整數」型別。整數型別（int）表示該「值」為整數，有小數點的數，如 3.14 就歸類為「浮點數」型別（floats）。請留意一點，雖然 42 是整數型別，但 42.0 則是浮點數型別哦。

表 1-2　常見的資料型別

資料型別	實例
整數	-2, -1, 0, 1, 2, 3, 4, 5
浮點數	-1.25, -1.0, --0.5, 0.0, 0.5, 1.0, 1.25
字串	'a', 'aa', 'aaa', 'Hello!', '11 cats'

Python 程式也可以有文字值，就是「字串」（string 或 strs，發音為"stirs"）。字串通常都會單引號（'）括住，例如 'Hello' 或 'Goodbye cruel world!'，這樣 Python 就能知道字串的起始和結尾。另外還可以在兩個單引號之間不放東西，像這樣 ''，這就是「空字串」。關於字串更詳細的說明會第 4 章討論。

假如您看到錯誤訊息 SyntaxError: EOL while scanning string literal，可能是忘了字串尾端要加單引號，如下面的實例：

```
>>> 'Hello world!
SyntaxError: EOL while scanning string literal
```

字串的連接與複製

運算子在運算處理不同資料型別的值時，其意義可能會改變。舉例來說，+ 運算子在運算處理兩個整數或浮點數型別時，它的意義是相加運算子。但是 + 運算子在運算處理兩個字串時，其意義是將兩個字串連接起來的意思，+ 就變成「字串連接」運算子。請在互動式 Shell 環境中輸入以下內容：

```
>>> 'Alice' + 'Bob'
'AliceBob'
```

這個表示式會將兩個字串的文字歸併成為單一個新的字串。不過，若您將一個字串和一個整數型別的值用 + 運算子來操作時，Python 就不知道要怎麼處理了，因此會顯示錯誤訊息。

```
>>> 'Alice' + 42
Traceback (most recent call last):
  File "<pyshell#0>", line 1, in <module>
    'Alice' + 42
TypeError: can only concatenate str (not "int") to str
```

錯誤訊息 can only concatenate str (not "int") to str 告訴我們要用 str 字串型別而不是用 int 整數型別，因為 Python 認為我們試圖將整數 42 連接到字串 'Alice'，這樣並不一致。程式碼的編寫必須先將整數轉換成字串，因為 Python 不會自動轉換。（本章後面「解析您的程式」單元會介紹 str()、int() 和 float() 函式）

對兩個整數或浮點數型別進行運算處理時，* 運算子的作用就是乘法。但 * 運算子用在一個字串和一個整數值時，就變成「字串複製」的作用。請在互動式 Shell 模式中輸入如下的例子，看看結果是什麼：

```
>>> 'Alice' * 5
'AliceAliceAliceAliceAlice'
```

這個表示式會運算求值成為單個字串值，該值是會以式子中的整數值為複製次數對原始字串進行複製。字串的複製是很好用的技巧，但並沒有像字串連接那麼常被大家使用。

* 運算子只能用在兩個數值（乘法），或一個字串和一個整數（字串複製）。不然，Python 會顯示錯誤訊息，如下列這般：

```
>>> 'Alice' * 'Bob'
Traceback (most recent call last):
  File "<pyshell#32>", line 1, in <module>
    'Alice' * 'Bob'
TypeError: can't multiply sequence by non-int of type 'str'
>>> 'Alice' * 5.0
Traceback (most recent call last):
  File "<pyshell#33>", line 1, in <module>
    'Alice' * 5.0
TypeError: can't multiply sequence by non-int of type 'float'
```

上述的例子 Python 是看不懂也不能處理，因為兩個字串不能相乘，對字串進行複製的次數也沒辦法有小數點、幾分之幾的複製方式。

將值存放到變數中

「變數（variable）」就像電腦記憶體中的盒子，可用來存放「值（value）」。如果程式在稍後會用到某個表示式運算的結果，那麼就可以將該結果儲存到變數之中。

指定陳述式

我們可以利用「指定陳述式（assignment statement）」將值儲存到變數中。指定陳述式含有一個變數名稱、一個等號（指定運算子），和要存放的值。如果輸入指定陳述式 spam = 42，那麼變數 spam 會存放整數值 42。

把前述的例子想像成一個貼有標籤的盒子，而其中放了值，如圖 1-1 所示。

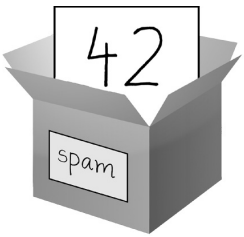


圖 1-1　spam = 42 這個陳述式像是告訴程式「spam 變數現在放了整數 42 在其中」

舉例來看，請在互動式 Shell 中輸入以下內容：

```
❶ >>> spam = 40
   >>> spam
   40
   >>> eggs = 2
❷ >>> spam + eggs
   42
   >>> spam + eggs + spam
   82
❸ >>> spam = spam + 2
   >>> spam
   42
```

一開始將值存入變數其中的動作，就稱之為變數的初始化（或建立）❶，隨後就可以在表示式中與其化值或變數一起使用❷。如果變數被指定了新的值，原來的就會被蓋過去❸。這就是為什麼上述例子在執行完之後，spam 變數存放

的是 42 而不是原來初始的 40。這稱之為「蓋過或覆寫（overwriting）」此變數。接下來再輸入程式碼，試一下對 spam 變數以新字串蓋過去的例子：

```
>>> spam = 'Hello'
>>> spam
'Hello'
>>> spam = 'Goodbye'
>>> spam
'Goodbye'
```

如圖 1-2 的例子，原有 spam 這個盒子放的是 'Hello' 字串，隨後以 'Goodbye' 取代它。

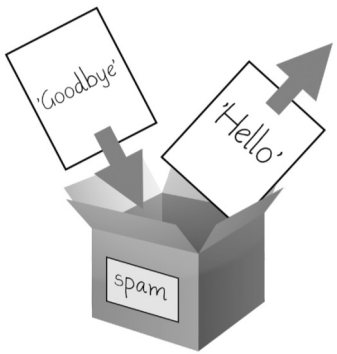


圖 1-2　如果以新值指定到變數中，那變數內舊的值就會被蓋過去

變數名稱

好的變數名稱本身已把該變數存放資料的內涵描述出來了。假設您搬家時，紙箱標的標籤都只寫「東西（stuff）」，那您要找想要的東西時就很困難了。本書所用的例子和許多 Python 的文件中，使用 spam、eggs 和 bacon 等當作一般的變數名稱（主要是受了 Monty Python 所編的 Spam 短劇所影響），但在您自己的程式中，取名時用具有描述性的名稱會有助於提高程式碼的可讀性。

雖然可以為變數取任何名稱，但 Python 在取名上還是有一些限制的。表 1-3 列出了一些合法與不合法的變數名稱使用實例。為變數取任何名字都可以，但要遵守以下 3 條規則：

1. 只能是一個字詞。
2. 只能用英文字母、數字和底線。
3. 不能以數字開頭。

表 1-3　合法與不合法的變數名稱

合法的變數名稱	不合法的變數名稱
current_balance	current-balance (不能用連字符號 -)
currentBalance	current balance (不能有空格)
account4	4account (不能以數字開頭)
_42	42 (不能只是數字)
TOTAL_SUM	TOTAL_$UM (不能用 $ 這種特殊字元)
hello	'hello' (不能用 ' 這種特殊字元)

變數名稱是有區分英文字母大小寫的，也就是說，spam、SPAM、Spam 和 sPaM 是 4 個不同的變數哦。為變數取名時使用小寫字母開頭是 Python 程式風格的慣例。

本書所用的變數名稱方式是以「駝峰式大小寫（camelcase）」命名，而不是「底線式（underscores）」命名。舉例來說，是使用 lookLikeThis，而不是 look_like_this。有些程式老手可能會說，官方的Python程式碼風格是用PEP8，也就是底線式的命名風格。嗯，沒錯，我就是喜歡駝峰式，必竟 PEP8 本身的使用指南就有一段內容就提到「傻傻堅持一致性是頭腦簡單的怪物（A Foolish Consistency Is the Hobgoblin of Little Minds）」:

> 「編寫程式時風格一致性很重要，但更重要的是知道什麼時候不適用。當有疑慮的時候，請相信自己的判斷。」
>
> "Consistency with the style guide is important. But most importantly: know when to be inconsistent—sometimes the style guide just doesn't apply. When in doubt, use your best judgment."

您的第一支程式

雖然在互動式 Shell 中對一次執行一行指令的方式很好用，但若要編寫長一點完整的 Python 程式碼時，最好還是在文字編輯器中先輸入。file editor 和記事本與 TextMate 這類文字編輯器功能差不多，但「file editor」還是有一些特別的功能可以使用。在 Mu 中想要編寫新的程式，可在 Mu 最上方的工具按鈕中按下 **New** 按鈕。

隨即顯示一個新的空白視窗，其中有一閃動的游標等著我們輸入，不過此視窗和互動式 Shell 不同，在 Shell 中只要按 Enter 鍵就會執行 Python 指令。file editor 視窗可讓我們輸入多行指令，也能儲存成檔案，和執行輸入的所有指令。以下兩點可讓我們區分其不同：

- 互動式 Shell 視窗會有 >>> 提示符號。
- file editor 視窗則沒有 >>> 提示符號。

接著是建立您的第一隻程式的時候了。請在 file editor 視窗開啟後，輸入以下內容：

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

❷ print('Hello world!')
   print('What is your name?')    # ask for their name
❸ myName = input()
❹ print('It is good to meet you, ' + myName)
❺ print('The length of your name is:')
   print(len(myName))
❻ print('What is your age?')    # ask for their age
   myAge = input()
   print('You will be ' + str(int(myAge) + 1) + ' in a year.')
```

輸入完以上所有程式碼後將它存起來，這樣以後開啟 Mu 時就不用重新輸入。請按下 **Save** 按鈕，在另存新檔對話方塊的檔案名稱方塊中輸入 hello.py，然後按下**存檔**按鈕。

在編寫輸入程式時，記得要隨時存檔，以免當機或不小心退出 Mu 或其他編輯器時，所輸入的程式碼都不見了。在 Windows 和 Linux 上有存檔的快捷鍵 Ctrl-S，在 macOS 上的快捷鍵為 ⌘-S。

檔案存好後，接著來執行。不管是在 Mu 或是在 Python 的 file editor 中，按下 **F5** 鍵程式就會執行，若是以 Python 的 file editor 按下則會顯示互動式 Shell 來執行。請留意，是在 file editor 視窗中按下 **F5** 鍵，而不是在互動式 Shell 中按下。程式執行後會提示輸入，此時請輸入您的名字。在互動式 Shell 中程式執行的輸出會像下面這樣：

```
Python 3.7.0b4 (v3.7.0b4:eb96c37699, May 2 2018, 19:02:22) [MSC v.1913 64 bit
(AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ================================ RESTART ================================
>>>
```

```
Hello, world!
What is your name?
Al
It is good to meet you, Al
The length of your name is:
2
What is your age?
4
You will be 5 in a year.
>>>
```

如果沒有更多程式碼要執行，Python 就會「中止（terminates）」，也就是停止執行。（也可以說是 Python 程式跳出了）

我們可以按下檔案視窗的 X 鈕關閉 file editor 視窗。若要重新開啟已存檔的程式碼，在 Mu 中可按下 **Load** 鈕開啟（若在 Python 的 file editor 則可選取 **File→Open…**指令），這樣會顯示開啟舊檔對話方塊，點選之前存好的程式碼檔 hello.py，再按下「開啟舊檔」鈕。之前存檔的 hello.py 就會在 file editor 視窗開啟並顯示出來。

您可以使用 http://pythontutor.com/ 的 Python Tutor 視覺化工具來查看程式的執行情況。另外也可以在 https://autbor.com/hellopy/ 上看到上面這支程式逐步的執行，請按下 Next 按鈕以瀏覽程式執行的每個步驟，這裡還能夠看到變數值和輸出的變化情況。

解析您的程式

在 file editor 視窗開啟上面這支新程式後，讓我們很快瀏覽一下這程式有用到的 Python 指令，一行一行來解析這支程式的內容。

註釋

下面這一行就稱之為「註釋（comments）」。

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

Python 在執行時會略過註釋，其功能大多用於編寫程式碼的註解，或是提醒我們所編寫的程式碼是用來做什麼的。上述這行中，# 符號之後的所有文字就是註釋。

程式設計師有時在測試程式碼時，會在某一行程式最前面加上 # 符號，臨時取消這行程式，這稱之為「註釋掉程式碼（commenting out code）」，當我們在測試或試誤程式時還滿有用的。如果想還原該行程式時，把前面的 # 符號刪掉即可。

Python 也會略掉註釋之後的空行，在程式中適當地加入空行來區隔，會讓程式碼更容易閱讀。空行的作用就像書本文章中的段落一樣。

print() 函式

print() 函式的功能是會將括號內的字串顯示在螢幕上。

```
❷ print('Hello world!')
  print('What is your name?') # ask for their name
```

print('Hello world!') 這一行是指把 'Hello world!' 字串印出來。Python 執行到這行程式時，會呼叫 print() 函式，並把字串值傳給（pass）函式，這個傳給函式的值稱之為引數（argument，也就是實際參數）。請留意，' 引號並沒有顯示在螢幕上，引號僅用來標出字串的起始和結尾，並不是字串值的一部分。

NOTE

可利用 print 函式在螢幕上印出空行，只要呼叫 print() 即可，括號內空著就可以了。

想要分辨是否為函式名稱，看到名稱後的左右括號是最好認的方法。這就是為什麼本書是寫 print()，而不是僅寫 print。第 2 章的內容會更深入討論函式這個議題。

input() 函式

input() 函式的功能是等待使用者從鍵盤輸入一些文字和按下 Enter 鍵接收。

```
❸ myName = input()
```

呼叫此函式後會取得使用者輸入的字串，而前面的程式碼是要把這個字串值指定給變數 myName。

您可以把呼叫 input() 函式當成表示式，用來取得使用者輸入的任意字串。假如使用者輸入的是 'A1'，那麼這個表示式可看成 myName = 'A1'。

如果呼叫 input() 後出現錯誤訊息，像 NameError: name 'Al' is not defined，則是因為用了 Python 2 而不是 Python 3 來執行這支程式。

印出使用者的名字

接下的 print() 例子中，在括號之間放了表示式 'It is good to meet you, ' + myName。

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

請記得一點，表示式運算求值後會歸併成單一個值。如果前面 myName 變數❸存放的是 'A1'，那麼這裡的表示式運算求值後會歸併成 'It is good to meet you, A1' 這個字串值，然後此字串值會傳給 print() 印到螢幕上。

len() 函式

我們可以把一個字串（或存有字串的變數）傳給 len() 函式，這個函式就會幫我們求出該字串的字元個數，是一個整數值。

```
❺ print('The length of your name is:')
   print(len(myName))
```

請試著在互動式 Shell 中輸入以下內容：

```
>>> len('hello')
5
>>> len('My very energetic monster just scarfed nachos.')
46
>>> len('')
0
```

就像前述的這些例子，len(myName) 會求得一個整數，然後把它傳給 print() 印在螢幕上。請留意，傳入 print() 的可以是整數值或字串，但如果在互動式 Shell 中輸入以下內容時，會顯示錯誤訊息：

```
>>> print('I am ' + 29 + ' years old.')
Traceback (most recent call last):
  File "<pyshell#6>", line 1, in <module>
    print('I am ' + 29 + ' years old.')
TypeError: can only concatenate str (not "int") to str
```

導致錯誤的原因不是 print() 函式，而是傳給 print() 的那個表示式。如果在互動式 Shell 中輸入這個表示式時，也一樣會顯示錯誤訊息：

```
>>> 'I am ' + 29 + ' years old.'
Traceback (most recent call last):
  File "<pyshell#7>", line 1, in <module>
    'I am ' + 29 + ' years old.'
TypeError: can only concatenate str (not "int") to str
```

Python 顯示錯誤的原因是，+ 運算子只能運算兩個整數，或用來連接兩個字串，整數和字串不能混在一起相加，這樣不符合 Python 的語法。可將整數轉成字串的方式來修正此錯誤。下一小節會討論其作法。

str()、int() 和 float() 函式

如果想連接整數（如 29）和字串再傳給 print() 印出，就需要取得 '29' 字串值，這是指 29 的字串形式。str() 函式可以把傳入的整數值轉換成字串值的形式，如下所示：

```
>>> str(29)
'29'
>>> print('I am ' + str(29) + ' years old.')
I am 29 years old.
```

由於 str(29) 可取得 '29' 字串值，所以 'I am ' + str(29) + ' years old.' 這個表示式會變成以 'I am ' + '29' + ' years old.' 來運算求值，其結果是 'I am 29 years old.'，這也是要傳到 print() 函式的字串值。

str()、int() 和 float() 函式的功用分別是將傳入值變成字串、整數和浮點數型式。請試著在 Shell 中利用這些函式來轉換一些值，看看會有什麼樣的結果：

```
>>> str(0)
'0'
>>> str(-3.14)
'-3.14'
>>> int('42')
42
>>> int('-99')
-99
>>> int(1.25)
1
>>> int(1.99)
1
>>> float('3.14')
3.14
>>> float(10)
10.0
```

前述的實例呼叫了 str()、int() 和 float() 函式，分別傳入不同資料型別的值，結果也分別取得字串、整數和浮點數形式的值。

若想要把整數或浮點數和字串連接起來，str() 函式就很好用。假如有些字串值想拿來進行數學運算，那麼就要用到 int() 函式的功能了。舉例來說，input() 函式的功用是返回字串值，就算使用者輸入的是數字也一樣。在互動式 Shell 中輸入 spam = input()，然後輸入 101。

```
>>> spam = input()
101
>>> spam
'101'
```

儲存在 spam 變數中的值不是整數 101，而是 '101' 字串形式。如果想要利用 spam 中的值來進行數學運算，那麼就要再用 int() 函式把字串值轉換成整數值再存回 spam 中。

```
>>> spam = int(spam)
>>> spam
101
```

此時就可以把 spam 變數當成整數來運算，因為它已不再是字串了。

```
>>> spam * 10 / 5
202.0
```

請留意，如果您不能轉換成整數的值放入 int() 中進行轉換，那麼 Python 會顯示錯誤訊息。

```
>>> int('99.99')
Traceback (most recent call last):
  File "<pyshell#18>", line 1, in <module>
    int('99.99')
ValueError: invalid literal for int() with base 10: '99.99'
>>> int('twelve')
Traceback (most recent call last):
  File "<pyshell#19>", line 1, in <module>
    int('twelve')
ValueError: invalid literal for int() with base 10: 'twelve'
```

如果想對浮點數進行取整數運算，也可用 int() 函式來處理。

```
>>> int(7.7)
7
>>> int(7.7) + 1
8
```

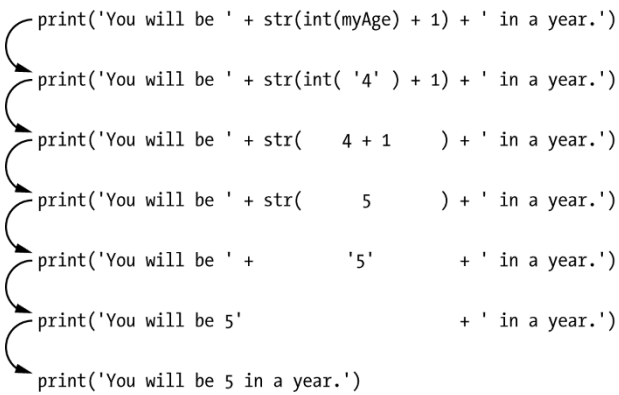
在前一節輸入存檔的 hello.py 程式範例中，最後 3 行使用了 int() 和 str() 函式，分別取得合適的值來運算和呈現。

```
❻ print('What is your age?') # ask for their age
   myAge = input()
   print('You will be ' + str(int(myAge) + 1) + ' in a year.')
```

myAge 變數會存放 input() 函式取得的值。由於 input() 函式取得返回的是字串值（就算輸入的是數字），因此可利用 int(myAge) 將字串值轉換成整數值。這個整數值會在表示式 int(myAge)+1 中運算，也就是加 1。

相加的結果再傳給 str() 函式轉成字串：str(int(myAge) + 1)，這個字串值會與 'You will be ' 和 ' in a year.' 字串連接起來，歸併一個更長的字串值。這個更長的字串最後傳到 print() 內，即可顯示印出在螢幕上。

以前述的例子來看，如果使用者輸入字串 '4'，並存到 myAge 變數中。字串 '4' 會轉換成整數，因此加 1 後結果為整數值 5，隨後 str() 函式又把此結果轉換為字串值，如此一來就能和第二個字串 'in a year.' 連接，產生最終的訊息。這個例子的運算求值過程如下圖所示。



文字與數字的相等運算

數字以字串值形式呈現，並不等於其整數值和浮點數值，不過其整數值卻和浮點數值是相等的。

```
>>> 42 == '42'
False
>>> 42 == 42.0
True
>>> 42.0 == 0042.000
True
```

Python 這樣子的區分方式是因為把字串視為文字，而整數值和浮點數值則都視為數字。

總結

我們可以在小算盤中是以數學式來運算，在文書處理軟體中則是輸入文字字串，甚至進行文字字串的複製貼上等處理。但在程式設計中的程式，其基本的建構區域是表示式和組成的值（包括運算子、變數和函式的呼叫），能夠搞定這些元素，就可以利用 Python 編寫程式來操控大量的資料。

請記住本章中所介紹的各種運算子（+、-、*、/、//、% 和 ** 是數學運算子，另外 + 和 * 也可當成字串運算子），以及 3 種資料型別（整數型、浮點數型和字串型）。

本章也介紹了幾個不同的函式。print() 和 input() 函式能處理簡單的文字輸出（到螢幕）和輸入（由鍵盤）。len() 函式可放入字串，它會算出該字串的字元個數。str()、int() 和 float() 函式可以把放入的值轉換成字串、整數和浮點數的形式。

下一章的內容會介紹程式的流程控制，讓 Python 判斷要執行那些程式碼，又在什麼情況下要跳過，甚至符合某些條件時重複執行某些程式碼。學會流程控制，就能編寫出具有判斷能力的聰明程式碼哦。

習題

1. 下面哪些是運算子，哪些是值？

```
*
'hello'
-88.8
-
/
+
5
```

2. 下列哪個是變數，哪個是字串？

```
spam
'spam'
```

3. 請列出 3 種資料型別。

4. 表示式（expression）是由什麼構成？所有表示式的最後結果是什麼？

5. 本章介紹了指定陳述式（statement），如 spam = 10。表示式（expression）和陳述式（statement），有什麼不同？

6. 下列陳述式執行後，變數 bacon 的值什麼？

```
bacon = 20
bacon + 1
```

7. 下列這兩個表示式運算結果是什麼？

```
'spam' + 'spamspam'
'spam' * 3
```

8. 為什麼 eggs 是合法的變數名稱，而 100 卻不合法？

9. 哪 3 個函式可將值轉換成整數型、浮點數型和字串型？

10. 為何下列的表示式會發生錯誤？怎麼修正？

```
'I have eaten ' + 99 + ' burritos.'
```

延伸題：請上網搜尋 len() 函式的 Python 說明文件。該文件在標題為「Build-in Functions」的網頁中。瀏覽一下 Python 還有什麼其他的函式，查一下 round() 函式的功用，並在互動式 Shell 互動環境中試一試。