前言

從書名可以知道，本書將介紹世上最流行的程式語言之一：Python。本書的目標讀者是程式初學者，以及學過其他語言，但也想要學習 Python 的資深程式員。

程式語言通常比人類語言更容易學習，因為你不需要背那麼多模糊的規則，以及例外的情況，而 Python 是最一致且最簡潔的語言之一，它在易學性、易用性和表達性之間取得很好的平衡。

電腦語言是**資料**（就像口語的名詞）與**指令**或**程式碼**（就像動詞）組成的，兩者都不可或缺。你會在各個章節中學到 Python 的基本程式與資料結構，瞭解如何結合它們，並且建構更進階的程式與資料。在過程中，你看到的和寫出來的程式將愈來愈長且愈來愈複雜。用木工來比喻，我們會從錘子、釘子和木片開始學起，本書的後半部介紹較專門的元件，包括相當於車床和其他電動工具的元件。

屆時你不但可以學會這種語言，也知道如何使用它。我們將從 Python 語言和它的內建標準程式庫開始看起，但我也會介紹如何尋找、下載、安裝和使用一些優秀的第三方程式包。我不會談到冷門的主題或複雜的技巧，而是介紹我在 10 年以上的 Python 開發經驗中，認為真正有用的東西。

雖然這是入門書籍，但它也有一些進階的主題，因為我想要讓你認識它們。本書仍然會介紹資料庫與 web 之類的領域，但是技術變遷的速度很快，當今的 Python 程式員或許也想知道雲端計算、機器學習或事件串流之類的東西，本書也會介紹它們。

Python 有一些特殊的功能，它們的效果比其他語言的寫法更好，例如，使用 `for` 與**迭代器**（*iterator*）來製作迴圈比親自遞增某個計數變數更直截了當。

碁峰 www.gotop.com.tw

當你學習新東西時，你很難區分哪些詞彙是具體的而不是非正式的，以及哪些概念確實很重要，換句話說，「它會考嗎？」，我會特別強調有特殊意義的、重要的 Python 詞彙及概念，但不會一次介紹太多。本書會儘早展示真正的 Python 程式碼，也會經常展示。

如果有些內容比較難以理解，或是有更符合 *Python* 風格的做法，我會特別說明。

Python 不是完美的語言。我會告訴你比較奇怪或不應該做的事情，並且告訴你可以改用哪種方案。

有時對於某些主題，我的看法會與一般人不同（例如物件繼承，或 web 的 MVC 和 REST 設計），請自行斟酌。

對象

本書適合每一位想要學習世上最流行的程式語言的人，無論你有沒有學過程式設計。

第二版的修改

本書改了第一版的哪些內容？

- 增加大約一百頁，包括貓的照片。
- 加入兩倍的章數，現在每一章都比較短。
- 用前面的章節介紹資料型態、變數與名稱。
- 新的 Python 標準功能，例如 *f-strings*。
- 新的或改善過的第三方程式包。
- 全部使用新的範例程式。
- 為程式新手加入一個附錄來介紹硬體與軟體的基本知識。
- 為有經驗的程式員加入一個附錄介紹 *asyncio*。
- 涵蓋「新堆棧」：容器、雲端、資料科學和機器學習。
- 提示如何獲得 Python 程式設計工作。

碁峰 www.gotop.com.tw

哪些沒有改？使用糟糕的詩歌和鴨子的那些簡例，它們都是永恆的。

大綱

第一部分（第 1–11 章）將介紹 Python 的基本知識，請按照順序閱讀。我會從最簡單的資料與程式結構說起，逐步將它們結合成比較詳細且實際的程式。第二部分（第 12–22 章）介紹如何在特定的應用領域中使用 Python，例如 web、資料庫、網路等；你可以按照你喜歡的順序閱讀。

接下來是各章與附錄的簡介，包括你將會在裡面看到的術語：

第一章，*初嘗 py*

電腦程式和你在日常生活中看到的指令沒有太大的不同，本章會用一些小型的 Python 程式來讓你大概知道這種語言的長相、功能，以及在真實世界的用途。你會知道如何在 Python **互動式解譯器**（或 *shell*）裡面，或是用存在電腦的文字**檔**執行 Python 程式。

第二章，資料：型態、值、變數與名稱

電腦語言混合了資料與指令。電腦用不同的方式來儲存與對待不同**型態**的資料。它們可能允許它們的值被改變（**可變**），也可能不允許（**不可變**）。在 Python 程式中，資料可能是**常值**（像 `78` 這種數字，或是像 `"waffle"` 這種文字**字串**），也可能是用有名字的**變數**來表示的。Python 對待變數與對待**名稱**一樣，這一點與許多其他語言不同，並且有很重要的結果。

第三章，數字

本章介紹最簡單的 Python 資料型態：**布林**、**整數**與**浮點數**，並教你基本的數學運算。本章的範例會將 Python 的互動式解譯器當成計算機來使用。

第四章，用 *if* 來選擇

接下來的幾章會在 Python 的名詞（資料型態）與動詞（程式結構）之間來回切換。Python 程式通常一次執行一行，從程式的最開始跑到最後面。`if` 結構可讓你根據某些資料比較結果來執行不同的程式行。

第五章，文字字串

回到名詞，以及文字**字串**的世界。瞭解如何建立、結合、改變、取回與印出字串。

碁峰 www.gotop.com.tw

第六章，用 *while* 與 *for* 來執行迴圈

再次回到動詞，用兩種方式來製作**迴圈**：`for` 與 `while`。本章也會介紹一種 Python 核心概念：**迭代器**（*iterator*）。

第七章，*tuple* 與串列

現在是時候介紹第一種 Python 的高階內建資料結構了：`list` 與 `tuple`。它們是一系列的值，很像用來建立複雜許多的資料結構的樂高積木。你可以用**迭代器**（*iterator*）來遍歷它們，以及用**生成式**（*comprehension*）來快速建立 list。

第八章，字典與集合

字典（亦稱 *dict*）與**集合**可以讓你用值來儲存資料，而不是用位置來儲存資料。這是非常方便的功能，也將是你最喜歡的 Python 功能之一。

第九章，函式

將前面幾章的資料與程式結構組織起來，進行比較、選擇與重複。將程式包在**函式**裡面，並且用**例外**來處理錯誤。

第十章，喔喔：物件與類別

物件這個術語有點模糊，但是在許多電腦語言中都很重要，包括 Python。如果你用其他語言寫過**物件導向程式**，Python 用起來比較簡單。本章介紹如何使用物件與類別，以及何時使用替代品比較好。

第十一章，模組、程式包與好東西

本章介紹如何擴展成更大規模的程式結構：**模組**、**程式包**與**程式**（*program*），告訴你應該將程式碼和資料放在哪裡、如何放入與取出資料、處理選項、瀏覽 Python Standard Library，以及看看外面還有什麼東西。

第十二章，玩轉資料

學會像專家一樣玩轉資料。本章全部都在討論文字與二進制資料，享受 Unicode 字元帶來的樂趣，以及搜尋 *regex* 文字。本章也介紹資料型態 *bytes* 與 *bytearray*，它們是與 string 不一樣的型態，裡面是原始的二進制值，而不是文字字元。

第十三章，日曆與時鐘

日期與時間有時很難處理，本章探討常見的問題與實用的解決方案。

碁峰 www.gotop.com.tw

第十四章，檔案與目錄

基本的資料儲存體都使用**檔案**與**目錄**。本章告訴你如何建立與使用它們。

第十五章，時間中的資料：程序與並行處理

本章初次討論核心系統，本章的主題是及時資料—如何使用**程式**、**程序**與**執行緒**在一段時間內做更多事情（**並行**）。本章也會提到 Python 最近新增的 *async*（**非同步**），附錄 C 會詳細介紹它。

第十六章，盒子資料：持久保存

我們可以用基本的平面檔案和檔案系統中的目錄來儲存與取回資料。當你使用 CSV、JSON 與 XML 等常見的文字格式時，它們可以得到一些結構。隨著資料愈來愈大且愈來愈複雜，你必須用**資料庫**提供的服務，包含傳統**關聯**資料庫，以及一些較新的 *NoSQL* 資料儲存體。

第十七章，空間中的資料：網路

用各種**服務**、**協定**與 *API* 在**網路**的空間中傳送你的程式碼和資料。本章範例包含低階的 TCP **通訊端**、**傳訊**程式庫、排隊（queuing）系統，及**雲端**部署。

第十八章，網路，解開

web 有獨立的一章—用戶端、伺服器、API 與框架。你將**爬抓**網站，接著用**請求**參數與**模板**來建構實際的網站。

第十九章，成為 *Python* 鐵粉

本章包含 Python 開發訣竅—用 `pip` 與 `virtualenv` 來安裝、使用 IDE、測試、除錯、log、原始碼控制和文件註釋。本章也會協助你找到與安裝實用的第三方程式包、包裝你自己的程式碼以便重複使用，以及瞭解要去哪裡取得更多資訊。

第二十章，*Py* 藝術

很多人在藝術領域使用 Python 來做很酷的事情，包括製作繪圖、音樂、動畫與遊戲。

第二十一章，*Py* 上工

Python 在商業領域有特定的用途：資料視覺化（繪圖、圖表和地圖）、安全防護和管制（regulation）。

碁峰 www.gotop.com.tw

第二十二章，*Py* 科學

近年來，Python 已經成為科學領域的頂尖語言了，包括數學與統計學、物理科學、生物科學與醫學。**資料科學**與**機器學習**都是它的顯著優勢。本章將簡介 NumPy、SciPy 與 Pandas。

附錄 *A*，硬體和軟體入門

如果你是很菜的程式新人，本附錄為你說明硬體和軟體實際上如何運作，並且介紹一些常見的術語。

附錄 *B*，安裝 *Python 3*

如果你還沒有在電腦安裝 Python 3，這個附錄告訴你如何安裝它，無論你使用的是 Windows、macOS、Linux 或某些 Unix 的變體。

附錄 *C*，全然不同的東西：非同步

Python 已經在許多版本加入非同步功能了，它們都不容易瞭解。我會在各章提到它們，但是特別用這個附錄來詳細討論它們。

附錄 *D*，習題解答

本附錄有章末習題的解答，請先完成習題之後再來看這一章，否則你不會變聰明。

附錄 *E*，備忘錄

這個附錄是可供快速參考的備忘錄。

Python 版本

電腦語言會隨著開發者加入新功能和修正錯誤而不斷改變，本書的範例是用 Python 3.7 版來編寫與測試的。3.7 版在我寫這本書時是最新的版本，我也會討論它值得注意的新增功能。Python 計畫在 2019 年年底正式發表 3.8 版，我也會介紹它可能加入的一些東西。如果你想要知道 Python 加入哪些東西，以及何時加入，可以試著參考 What's New in Python 網頁（*https://docs.python.org/3/whatsnew*）。它是一份技術文件，對身為 Python 初學者的你來說或許有點難懂，但是將來如果你要讓程式在使用各種 Python 版本的電腦上運作時，它會有很大的幫助。

碁峰 www.gotop.com.tw

第一章

初嘗 Py

只有醜陋的語言才會流行。Python 是特例。

—Don Knuth

謎

我們從兩個小謎題和它們的解答談起，你認為這兩行文字的意思是什麼？

```
(Row 1): (RS) K18,ssk,k1,turn work.
(Row 2): (WS) Sl 1 pwise,p5,p2tog,p1,turn.
```

它看起來很技術性，就像某種電腦程式。事實上，它是**針織圖案**，具體來說，它描述了如何編織襪子的腳跟部分，就像圖 1-1 那樣。

圖 1-1　針織襪子

碁峰 www.gotop.com.tw

我理解這段文字的程度，差不多跟家裡的貓理解紐約時報的數獨的程度一樣，但是我的太太完全看得懂它。如果你會編織，你也看得懂。

我們再來看一段神秘的文字，這是在索引卡找到的。或許你不知道它最終的成果是什麼，但你應該馬上就可以知道它的目的：

```
  1/2 c. butter or margarine
  1/2 c. cream
  2 1/2 c. flour
  1 t. salt
  1 T. sugar
  4 c. riced potatoes (cold)

Be sure all ingredients are cold before adding flour.
Mix all ingredients.
Knead thoroughly.
Form into 20 balls.  Store cold until the next step.
For each ball:
  Spread flour on cloth.
  Roll ball into a circle with a grooved rolling pin.
  Fry on griddle until brown spots appear.
  Turn over and fry other side.
```

就算你不會煮飯，你也應該知道這是一份**食譜**[1]：它先列出食材，再說明烹調方式。它的成品是什麼？*Lefse*，一種類似墨西哥玉米餅的挪威美食（圖 1-2）。你可以塗上奶油與果醬，或你喜愛的任何東西，將它捲起來，好好享受。

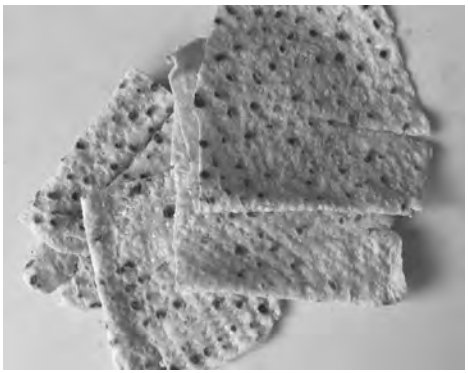

圖 1-2　Lefse

1　通常只會在料理書籍和犯罪小說裡面看到。

碁峰 www.gotop.com.tw

編織圖案與食譜有一些相同的地方：

- 有常用的**單字**、縮寫，與符號。你可能看過其中的一些，但其他的則讓你一頭霧水。
- 規定哪些可以說，以及可以在哪裡說的規則，也就是**語法**。
- 一系列需要**依序執行的動作**。
- 有時要重複某些動作（**迴圈**），例如油炸每一片 lefse 的做法。
- 有時會引用一系列其他的動作（電腦術語稱為**函式**）。在食譜中，你可能要參考其他的食譜來瞭解如何壓碎馬鈴薯。
- 假設讀者知道**背景**知識。食譜假設你知道什麼是水，以及如何將它煮開。編識圖案假設你懂得編織，不會經常刺傷自己。
- 它會使用、建立或修改某種**資料**—馬鈴薯與紗線。
- 用來處理資料的**工具**—鍋子、攪拌機、烤箱、編織棒。
- 有一個預期的**結果**。在我們的例子中，它是讓你的腳穿的東西，以及裝進你的胃的東西。不要將它們混在一起。

無論你如何稱呼它們（使用俗語、行話、方言），你到處都可以看到它們。行話可以幫已經知道它們的人節省時間，但是對其他的人而言則神秘難懂。如果你不懂橋牌，你可以試著解讀關於橋牌的報紙專欄，或者，如果你不是科學家，試著解讀科學論文（如果你是科學家，那就試著解讀不同領域的論文）。

小程式

你可以在電腦程式中看到上述的所有概念，它們本身就像小型語言，專門用來告訴電腦該做什麼事情。我用了針織圖案和食譜來告訴你程式設計沒有那麼神秘，它大部分都是關於學習正確的單字和規則。

我們的大腦一次只能記得那麼多東西，如果單字和規則不多，而且不需要一次學會許多單字及規則，對我們來說有很大的幫助。

最後，我們來看一段真正的電腦程式（範例 1-1）。你認為這段程式在做什麼事情？

範例 1-1　countdown.py

```
for countdown in 5, 4, 3, 2, 1, "hey!":
    print(countdown)
```

碁峰 www.gotop.com.tw

如果你猜得到它是一段印出這個結果的 Python 程式：

```
5
4
3
2
1
hey!
```

你就知道，Python 比食譜或編織圖案更簡單。而且你可以舒適且安全地坐在桌前練習 Python 程式，遠離熱水與刺針的威脅。

Python 有一些特殊的單字與符號—for、in、print、逗號、冒號、括號等等，它們都是很重要的語法（規則）元素。好消息是，與大多數的電腦語言相比，Python 的語法更好，而且需要記憶的東西較少。它讀起來比較自然，幾乎就像一份食譜。

範例 1-2 是另一段 Python 小程式，它會從 Python **串列**中選出一句哈利波特咒語並將它印出來。

範例 *1-2*　*spells.py*

```
spells = [
    "Riddikulus!",
    "Wingardium Leviosa!",
    "Avada Kedavra!",
    "Expecto Patronum!",
    "Nox!",
    "Lumos!",
    ]
print(spells[3])
```

每一句咒語都是 Python **字串**（被放在引號裡面的一串文字字元）。它們都被逗號分開，並且被放在一個 Python **串列**裡面，串列是用方括號來定義的（[與]）。spells 這個字是個**變數**，它是串列的名稱，讓我們可以用它來做一些事情。在這個例子中，程式會印出第四個咒語：

```
Expecto Patronum!
```

為什麼我們用 3 來取得第四個？ spells 這種 Python 串列是一系列的值，為了取得裡面的值，我們必須使用從串列開頭算起的 *offset*（**偏位值**），它的第一個值在 offset 0，第四個值在 offset 3。

碁峰 www.gotop.com.tw

人類習慣從 1 開始算起，所以從 0 算起有點奇怪，從 offset 而非位置的角度來思考會讓你更容易理解。沒錯，這是為了展示電腦程式與一般的語言有些不同的例子。

串列是常見的 Python **資料結構**，第 7 章會介紹如何使用它們。

範例 1-3 的程式會印出 Three Stooges 喜劇中的一句話，不過這段程式用說那句話的人來引用，而不是用那句話在串列內的位置。

範例 1-3　quotes.py

```
quotes = {
    "Moe": "A wise guy, huh?",
    "Larry": "Ow!",
    "Curly": "Nyuk nyuk!",
    }
stooge = "Curly"
print(stooge, "says:", quotes[stooge])
```

執行這段小程式會印出：

```
Curly says: Nyuk nyuk!
```

`quotes` 是作為 Python **字典**的名稱的變數，字典是由許多唯一的**鍵**（在這個例子裡面是 Stooge 的名字）及其相關的**值**（在這裡是那位 Stooge 的名言）組成的。使用字典時，你可以用名稱來進行儲存與查找，它通常是很實用的串列替代方案。

`spells` 範例使用中括號（`[` 與 `]`）來製作 Python 串列，`quotes` 範例使用大括號（`{` 與 `}`）來製作 Python 字典，它也在字典裡面使用冒號（`:`）來建立每一個鍵和它的值的關係。第 8 章會更詳細地介紹字典。

希望我沒有一次教你太多語法，在接下來幾章，你還會遇到這些小規則，每次都不會太多。

大一些的程式

接下來要介紹完全不同的東西了：範例 1-4 是執行一系列比較複雜的任務的 Python 程式。先不用期望你現在就可以瞭解這段程式如何運作，那是這整本書的目的！這個範例的目的是讓你看一下、感受一下典型的而且有點難度的 Python 程式。如果你知道其他的電腦語言，可以拿它來比較一下 Python。雖然你還不認識 Python，你可不可以在閱讀

碁峰 www.gotop.com.tw

程式後面的解釋之前，大概說出每一行程式在做什麼事情？你已經看過 Python 串列與字典的例子了，這段程式加入一些其他的功能。

在本書的第一版中，這段範例程式會連接 YouTube 網站，並取得評分最高的影片的資訊，例如「Charlie Bit My Finger」。但是就在第二版的墨跡乾掉時，它無法運作了，因為 Google 不支援這項服務了，所以 marquee 範例程式無法動作了。我們的新範例 1-4 會前往另一個網站，它的存活時間應該會比較長—在 Internet Archive（*http://archive.org*）的 *Wayback Machine*，這個免費的服務在過去的 20 年來，已經保存了數十億個網頁（與電影、電視節目、音樂、遊戲和其他數位產品）。你會在第 18 章看到其他關於這個 *web API* 的範例。

程式會要求你輸入一個 URL 與一個日期，接著詢問 Wayback Machine 有沒有該網站在那個日期左右的複本。如果有，它會回傳資訊給這個 Python 程式，程式會印出 URL，並在你的瀏覽器中顯示它。我的目的是展示 Python 如何處理各種工作—取得你輸入的東西、透過網際網路與網站溝通、取回內容、從中提取 URL，以及說服你的瀏覽器顯示該 URL。

如果我們取回一般的網頁，裡面全部都是 HTML 格式的文字，我們就要設法知道如何顯示它，這是一件繁重的工作，所以我們樂於信任瀏覽器，交給它處理。我們也可以試著取出我們想要的部分（詳情見第 18 章的**爬網**）。這兩種做法都要進行更多工作和編寫更大型的程式，但 Wayback Machine 回傳 *JSON* 格式的資料。JSON（JavaScript Object Notation）是一種人類看得懂的文字格式，它可以描述裡面的資料的類型、值與順序。它是另一種小型語言，很多人都用它在不同的電腦語言和系統之間交換資料。第 12 章會進一步介紹 JSON。

Python 程式可以將 JSON 文字轉換成 Python **資料結構**（你會在接下來幾章看到它們），彷彿你寫了一個程式來自行建立它們一般。我們的小程式只選擇一小部分（從 Internet Archive 網站取得的舊網頁的 URL）。再次聲明，這是你可以自行執行的完整 Python 程式。為了保持程式的簡短，我們只加入少量的錯誤檢查。行數不是程式的一部分，它們是為了方便你在後面的內容中參考程式用的。

範例 1-4　archive.py

```
1 import webbrowser
2 import json
3 from urllib.request import urlopen
4
5 print("Let's find an old website.")
6 site = input("Type a website URL: ")
```

碁峰 www.gotop.com.tw

```
7 era = input("Type a year, month, and day, like 20150613: ")
8 url = "http://archive.org/wayback/available?url=%s&timestamp=%s" % (site, era)
9 response = urlopen(url)
10 contents = response.read()
11 text = contents.decode("utf-8")
12 data = json.loads(text)
13 try:
14     old_site = data["archived_snapshots"]["closest"]["url"]
15     print("Found this copy: ", old_site)
16     print("It should appear in your browser now.")
17     webbrowser.open(old_site)
18 except:
19     print("Sorry, no luck finding", site)
```

這一小段 Python 程式用一些易懂的程式做了很多事情。雖然你還不知道所有的詞彙，但接下來的幾章會讓你認識它們。以下是每一行做的事情：

1. 匯入（讓這段程式可用）Python 標準程式庫模組 webbrowser 的所有程式。
2. 匯入 Python 標準程式庫模組 json 的所有程式。
3. 只從標準程式庫模組 urllib.request 匯入 urlopen 函式。
4. 空的一行，因為不希望讓程式看起來很擁擠。
5. 在螢幕上印出一些初始文字。
6. 印出詢問 URL 的文字，讀取你輸入的文字，並將它存入一個稱為 site 的變數。
7. 印出另一個問題，這一次讀取年、月與日，並將它存入一個稱為 era 的變數。
8. 建立一個稱為 url 的字串變數，來讓 Wayback Machine 可用來尋找該網站在你輸入的日期的複本。
9. 連接該在 URL 的 web 伺服器，並請求特定的 *web* 服務。
10. 取得回應資料，並將它指派給 contents 變數。
11. 將 contents 解碼成 JSON 格式的字串，再指派給 text 變數。
12. 將 text 轉換成 data，即 Python 資料結構。
13. 檢查錯誤：try（試著）執行接下來的四行，如果有任何失敗，就執行最後一行程式（在 except 後面的）。
14. 如果我們得到符合日期的網站，從三層的 Python 字典取得它的值。注意這一行與下三行是縮進來的，Python 用縮排來代表它們屬於上面的 try。

碁峰 www.gotop.com.tw

15. 印出我們找到的 URL。
16. 印出下一行執行之後會發生什麼事。
17. 在你的瀏覽器顯示我們找到的 URL。
18. 如果在前四行裡面有任何失敗，Python 就會跳到這裡。
19. 如果它失敗了，印出一個訊息與我們尋找的網站。這一行內縮，因為它只能在上面的 except 執行時執行。

當我在終端機視窗執行它時，我輸入一個網站 URL 與一個日期，得到這段輸出文字：

```
$ python archive.py
Let's find an old website.
Type a website URL: lolcats.com
Type a year, month, and day, like 20150613: 20151022
Found this copy: http://web.archive.org/web/20151102055938/http://www.lolcats.com/
It should appear in your browser now.
```

圖 1-3 是我的瀏覽器的畫面。

圖 1-3　來自 Wayback Machine

碁峰 www.gotop.com.tw

在上面的例子中，我們用了一些 Python **標準程式庫**模組（一種程式，當它被安裝之後，就會被納入 Python），但它們沒有什麼神秘之處。Python 有大量優秀的第三方軟體。範例 1-5 是使用一種稱為 requests 的外部 Python 軟體包來訪問 Internet Archive 網站的另一種寫法。

範例 *1-5*　*archive2.py*

```
import webbrowser
import requests

print("Let's find an old website.")
site = input("Type a website URL: ")
era = input("Type a year, month, and day, like 20150613: ")
url = "http://archive.org/wayback/available?url=%s&timestamp=%s" % (site, era)
response = requests.get(url)
data = response.json()
try:
    old_site = data["archived_snapshots"]["closest"]["url"]
    print("Found this copy: ", old_site)
    print("It should appear in your browser now.")
    webbrowser.open(old_site)
except:
    print("Sorry, no luck finding", site)
```

新版本比較短，我認為它對大部分的人來說比較易懂。第 18 章會更詳細介紹 requests，第 11 章則廣泛地介紹外部創作的 Python 軟體。

真實世界的 Python

那麼，Python 值得你付出時間和精力來學習嗎？ Python 在 1991 年就出現了（比 Java 老，比 C 年輕），一直以來都在最流行的電腦語言排行榜的前五名。很多人受僱撰寫 Python 程式，那些程式都是你每天都會用到的正式作品，例如 Google、YouTube、Instagram、Netflix 與 Hulu。我也在許多領域中，用它來製作產品 app。Python 以極高的生產力而聞名，吸引了發展快速的組織。

你可以在許多電腦環境中找到 Python，包括：

- 監視器或終端機視窗裡面的命令列
- 圖形化使用者介面（GUI），包括 web
- web，包括用戶端與伺服器端

碁峰 www.gotop.com.tw

- 支援大型熱門網站的後端伺服器
- **雲端**（由第三方管理的伺服器）
- 行動裝置
- 嵌入式裝置

Python 程式包括一次性的腳本（例如在本章出現過的那些），和上百萬行的系統。

在 2018 年的 Python Developers' Survey（*https://oreil.ly/8vK7y*）裡面，你可以看到代表 Python 目前在電腦世界的地位的數據和圖表。

我們將會看到它在網站、系統管理和資料操作之中的應用。在最後幾章，我們將會看到 Python 在藝術、科學和商業領域中的具體用途。

Python vs. 行星 X 語言

Python 和其他語言相較之下的表現如何？在何時 / 何處該選擇其中一項而非另一項？本節將展示其他語言的程式，讓你瞭解這場競賽的實況。如果你沒有用過這些語言，我不期待你瞭解它們。（當你看到最終的 Python 範例時，或許還會慶幸自己沒有用過其中的一些語言。）

接下來的每一段程式都會印出一個數字，並且介紹一下那種語言。

如果你使用終端機，或終端機視窗，讀取你輸入的訊息、執行它，並顯示結果的程式稱為 *shell* 程式。Windows shell 稱為 `cmd`（*https://en.wikipedia.org/wiki/Cmd.exe*），它執行副檔名為 `.bat` 的**批次**（*batch*）檔。Linux 與其他的類 Unix 系統（包括 Mac OS X）都有許多 shell 程式，最流行的稱為 `bash`（*https://www.gnu.org/software/bash*）或 `sh`。shell 有簡單的功能，例如簡單的邏輯，以及將萬用符號（例如 `*`）擴展為檔名。你可以先將指令存入稱為 *shell script* 的檔案中，稍後再執行它們。它們可能是身為程式設計師的你第一次遇到的程式。問題在於，shell script 不能超過上百行程式，否則就會比其他語言慢很多。下面是一小段 shell 程式：

```
#!/bin/sh
language=0
echo "Language $language: I am the shell. So there."
```

將這些程式存為 `test.sh` 檔，並且用 `sh test.sh` 來執行它之後，你會在螢幕看到：

```
Language 0: I am the shell. So there.
```

碁峰 www.gotop.com.tw

經久耐用的 C（*https://oreil.ly/7QKsf*）與 C++（*https://oreil.ly/iOJPN*）都是非常低階的語言，適合在速度非常重要的情況下使用。你的作業系統與它的許多程式（包括在你的電腦上的 python 程式）可能都是用 C 或 C++ 寫成的。

這兩種語言的學習和維護難度較高。你必須注意許多難以診斷的細節（例如**記憶體管理**），它們可能造成程式當機和其他問題。這是一小段 C 程式：

```
#include <stdio.h>
int main(int argc, char *argv[]) {
    int language = 1;
    printf("Language %d: I am C! See? Si!\n", language);
    return 0;
}
```

C++ 具有 C 家族的特徵，但也演化出一些獨特的功能：

```
#include <iostream>
using namespace std;
int main() {
    int language = 2;
    cout << "Language " << language << \
       ": I am C++!  Pay no attention to my little brother!" << \
       endl;
    return(0);
}
```

Java（*https://www.java.com*）與 C#（*https://oreil.ly/1wo5A*）是 C 和 C ++ 的後繼者，它們避免了其前輩的一些問題（尤其是記憶體管理），但也有些繁雜。這是一段 Java：

```
public class Anecdote {
    public static void main (String[] args) {
        int language = 3;
        System.out.format("Language %d: I am Java! So there!\n", language);
    }
}
```

如果你沒有用這些語言寫過程式，可能會想：它們到底在**幹嘛**？其實我們只想要印出一行文字。有些語言背負著沉重的語法包袱。第 2 章會更深入說明這個部分。

C、C++ 與 Java 都是**靜態語言**，你必須幫電腦設定一些低階的細節，例如資料型態。你可以在附錄 A 看到，整數這類的資料型態在電腦裡面有特定數量的位元，並且只能做可用整數來做的事情。相較之下，**動態語言**（也稱為**腳本語言**）不會要求你必須宣告變數型態才能使用它們。

碁峰 www.gotop.com.tw

多年來，Perl（*http://www.perl.org*）一直是萬用的動態語言，它有強大的功能，以及廣泛的程式庫。但是它的語法有點彆扭，近年來，它的動能似乎已經被 Python 和 Ruby 超愈了。這是一段美妙的 Perl：

```
my $language = 4;
print "Language $language: I am Perl, the camel of languages.\n";
```

Ruby（*http://www.ruby-lang.org*）是比較現代的語言，它借鑑一些 Perl 的概念，近來之所以流行，在很大程度上是因為 *Ruby on Rails*（一種 web 開發框架）。它有許多應用領域和 Python 重疊，人們往往是根據個人喜好或特定應用領域的程式庫多寡，在兩者之間做出選擇。這是一段 Ruby：

```
language = 5
puts "Language #{language}: I am Ruby, ready and aglow."
```

PHP（*http://www.php.net*）在 web 開發領域非常流行，因為它可以輕鬆地結合 HTML 和程式碼。但是 PHP 語言本身有許多缺陷，在 web 領域之外尚未成為通用語言。這是它的長相：

```
<?PHP
$language = 6;
echo "Language $language: I am PHP, a language and palindrome.\n";
?>
```

Go（*https://golang.org*）（或 *Golang*，如果你試著用 Google 搜尋它的話）是最近出現的語言，試圖同時具備效率和方便性：

```
package main

import "fmt"

func main() {
  language := 7
  fmt.Printf("Language %d: Hey, ho, let's Go!\n", language)
}
```

C 和 C++ 的另一種現代替代方案是 Rust（*https://doc.rust-lang.org*）：

```
fn main() {
    println!("Language {}: Rust here!", 8)
```

還有誰沒介紹？噢，別忘了 Python（*https://python.org*）：

```
language = 9
print(f"Language {language}: I am Python. What's for supper?")
```

碁峰 www.gotop.com.tw

為何選擇 Python？

其中一個原因，但不一定是最重要的，是它的普遍性。從各種層面的評量，Python 是：

- 成長速度最快（*https://oreil.ly/YHqqD*）的主要程式語言，見圖 1-4。
- 2019 年 6 月 TIOBE 指數（*https://www.tiobe.com/tiobe-index*）的編輯說：「Python 本月的 TIOBE 指數再次創下歷史新高的 8.5%。如果 Python 繼續保持這種發展速度，在 3 至 4 年內，它很可能會取代 C 和 Java，成為世界最流行的程式語言。」
- 2018 年的年度程式語言（TIOBE），在 IEEE Spectrum（*https://oreil.ly/saRgb*）和 PyPL（*http://pypl.github.io/PYPL.html*）也得到第一名。
- 美國頂尖大學的電腦科學入門課程最喜歡的語言（*http://bit.ly/popular-py*）。
- 法國高中的官方教學語言。

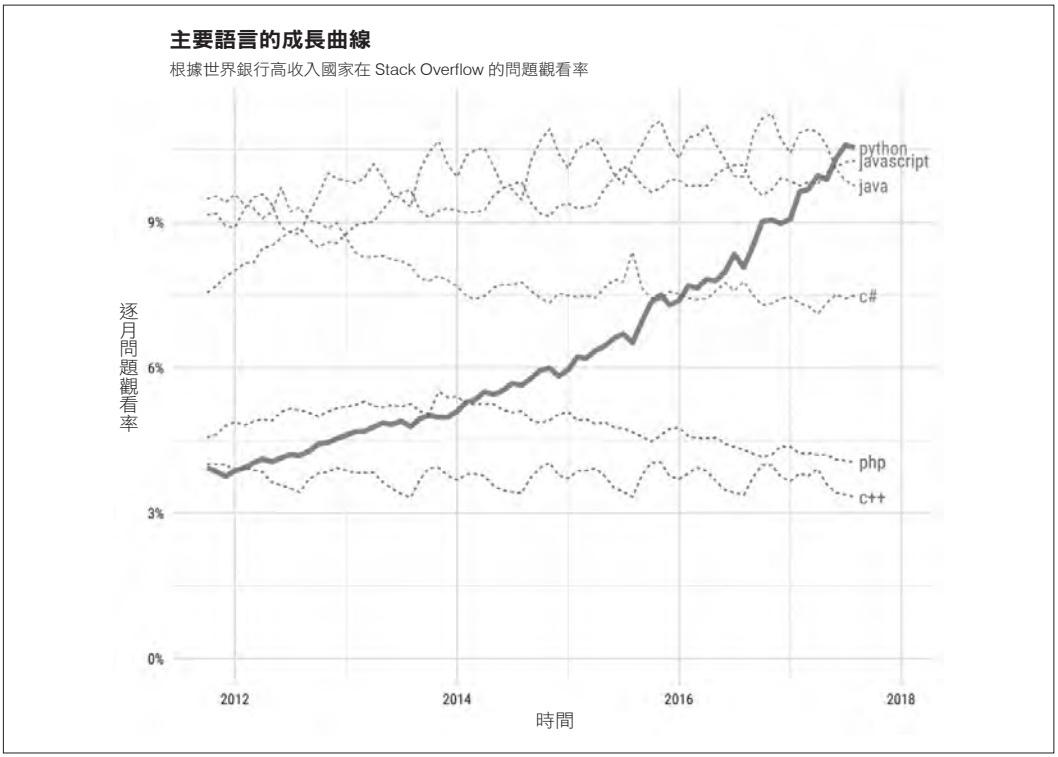

圖 1-4　Python 是成長速度最快的主要程式語言

碁峰 www.gotop.com.tw

最近，它在資料科學和機器學習領域中非常流行。如果你想要在你喜歡的領域找到一份高薪的程式工作，Python 是個不錯的選擇。如果你正在招聘人員，有經驗的 Python 開發者的數量也在持續增加。

不過，**為什麼**它這麼流行？程式語言本身是沒有什麼魅力的，根本的理由是什麼？

Python 是很棒的通用、高階語言。它的設計讓它非常易讀，這件事比你想像的重要。每一個電腦程式都只被撰寫一次，但是會被閱讀和修改多次，通常有很多人參與其中。易讀也讓它易學和易記，因此易寫。相較於其他流行的語言，Python 的學習曲線很溫和，可讓你快速具備生產力，但它也具備深度，可讓你在獲得經驗之後進一步探索。

比較簡潔的 Python 也可以讓你寫出比其他靜態語言更精簡的程式。有一些研究指出，無論程式員使用哪一種語言，他們每天寫出來的行數都差不多，所以只要用一半的行數就可以完成一項功能時，你的生產力就會翻倍。很多公司都認為這件事很重要，而 Python 正是它們的一項「不太機密」的利器。

當然，Python 是 free 的，無論是在價格上，或是自由度上。你可以用 Python 寫出任何東西，在任何地方使用它，全部都免費。不會有人在看了你的 Python 程式說「你寫了一段很棒的程式，但是它會不會因為授權問題而造成遺憾？」

Python 可以在幾乎任何地方運行，而且「內建電池」——它的標準程式庫有一大堆實用的軟體。本書將展示許多標準程式庫和實用的第三方 Python 程式碼。

但是，使用 Python 的最佳理由或許是你想像不到的：大家通常都很**喜歡**用它來寫程式，而不是把它當成完成工作的必要之惡。它不會妨礙你。套句你熟悉的格言，它「和你心靈相通」。當開發人員必須使用其他的語言時，他們經常想念 Python 的某些設計。這就是 Python 與多數其他語言的區別。

為什麼不使用 Python？

Python 不是適合各種狀況的最佳語言，並不是任何一個地方在預設的情況下都安裝它。如果你還沒有在電腦安裝 Python，附錄 B 會告訴你做法。

對大部分的應用來說，它的速度已經夠快了，但是對更要求速度的應用而言，它可能還不夠快。如果你的程式要花大部分的時間來進行計算（術語稱為 *CPU-bound*，CPU 受限），用 C、C++、C#、Java、Rust 或 Go 寫出來的程式通常跑得比 Python 等效程式更快，不過不是絕對如此！

碁峰 www.gotop.com.tw

你可以採取這些解決方案：

- 有時在 Python 中使用較佳的**演算法**（階段性解決方案）勝過在 C 中使用沒效率的演算法。因為 Python 可以提升開發速度，所以你有更多時間來試驗別的做法。
- 在許多應用領域中（尤其是在 web），當程式需要等待網路伺服器的回應時，通常會進入閒置狀態。CPU（中央處理單元，執行所有計算的電腦**晶片**）幾乎不參與其中，因此，靜態與動態程式的端對端（end-to-end）時間很接近。
- Python 標準解譯器是用 C 寫成的，可用 C 程式來擴充。我會在第 19 章稍微討論這個主題。
- Python 解譯器更快了，Java 在它的起步階段非常緩慢，許多機構為了提升它的速度而投入大量的研究與資金。 Python 不屬於任何一家企業，所以它的改善是漸進式的。在第 468 頁的「PyPy」中，我會探討 *PyPy* 專案與它的影響。
- 你的應用程式可能有很高的要求，不論你怎麼寫，Python 都無法滿足你的需求。通常大家會用 C、C++ 與 Java 來取代 Python。或許你也可以研究一下 Go（*http://golang.org*）（感覺起來很像 Python，但跑起來像 C）或 Rust。

Python 2 vs. Python 3

Python 有兩種版本，這件事有點複雜。Python 2 已經出現很久了，並且被預先裝在 Linux 和 Apple 電腦之中。它一向是個傑出的語言，但世上沒有完美的東西，如同許多其他領域，在電腦語言中，有些錯誤是表面的，很容易修正，有些則很難處理，處理它們的修正版本與之前是**不相容**的：用它們寫出來的新程式沒辦法在舊的 Python 系統上運作，在修正之前寫出來的舊程式也沒辦法在新的系統上運作。

所以 Python 的創造者（Guido van Rossum（*https://www.python.org/~guido*））和其他人決定將難處理的修正綁在一起，在 2008 年發表它們，稱之為 Python 3。Python 2 已經過去了，未來是 Python 3 的時代。Python 2 的最後一版是 2.7，它還會存在一段時間，但它已經到達生命的終點，將來不會出現 Python 2.8 了。Python 在 2020 年 1 月停止支援 Python 2 語言，再也不提供安全防護和其他修正，許多著名的 Python 程式包也不支援 Python 2 了（*https://python3statement.org*）。作業系統會移除 Python 2，或預先安裝 Python 3。許多流行的 Python 軟體已經被逐漸轉換成 Python 3 了，不過我們會跨過這個分界，用 Python 3 來進行所有新的開發。

碁峰 www.gotop.com.tw

本書探討 Python 3，它看起來幾乎與 Python 2 一模一樣。最明顯的改變在於 `print` 在 Python 3 是函式，所以當你呼叫它時，必須幫引數加上括號。最重要的改變是 *Unicode* 字元的處理，第 12 章會說明。我會在其他明顯的差異出現時指出它們。

安裝 Python

為了避免造成這一章的混亂，我用附錄 B 來說明 Python 3 的安裝細節。如果你沒有 Python 3，或不確定有沒有，可翻到那一個附錄，看看你需要在電腦做些什麼。是的，這項工作不輕鬆，但是你只要做一次就好。

執行 Python

當你安裝可執行的 Python 3 之後，你就可以用它來執行本書以及你自己的 Python 程式了。該如何執行 Python 程式？主要的方法有兩種：

- Python 內建的**互動式解譯器**（也稱為它的 *shell*）很適合用來試驗小程式。你可以在裡面逐行輸入指令，並且立刻看到結果。你可以透過緊湊的打字和查看節奏來快速進行試驗。我會用互動式解譯器來展示語言的功能，你也可以在自己的 Python 環境中輸入相同的指令。
- 將 Python 程式存入文字檔，通常使用 *.py* 副檔名，並且輸入 `python` 加上檔名來執行它們。

我們來試一下這兩種做法。

使用互動式解譯器

本書大部分的範例程式都使用內建的互動式解譯器。如果你輸入與範例一樣的指令，並且取得一樣的結果，那就代表你正走在正確的道路上。

你只要在電腦輸入主 Python 程式的名稱就可以啟動解譯器了：它是 `python`、`python3` 或類似的東西。在接下來的內容，我們假設它叫做 `python`，如果你使用不同的名稱，當你在範例程式中看到 `python` 時，請輸入該名稱。

互動式解譯器的運作方式幾乎與 Python 處理檔案的方式一模一樣，但有一個例外：當你輸入有個值的東西時，互動式解譯器會自動印出它的值。這個功能不屬於 Python 語言，它只是為了讓你不需要每次都輸入 `print()` 而提供的功能。例如，當你啟動 Python，並

碁峰 www.gotop.com.tw

且在解譯器中輸入數字 27 時，它會被 echo 到你的終端機（如果你在檔案裡面加入一行 27，Python 不會生氣，但是當你執行程式時，它不會印出任何東西）：

```
$ python
Python 3.7.2 (v3.7.2:9a3ffc0492, Dec 24 2018, 02:44:43)
[Clang 6.0 (clang-600.0.57)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> 27
27
```

在上面的例子中，$ 是系統提示符號，提示你在終端機視窗裡面輸入 python 之類的指令。這一本書的範例程式都會使用它，不過你的提示符號可能不一樣。

對了，當你想要印出某個東西時，print() 也可以在解譯器裡面執行：

```
>>> print(27)
27
```

如果你用互動式解譯器來嘗試這些例子，並且看到同樣的結果，那就代表你已經執行真正的 Python 程式了（雖然很小）。在接下來的幾章，你會從一行程式開始，慢慢寫出更長的 Python 程式。

使用 Python 檔案

如果你只在一個檔案裡面放入 27 並且用 Python 執行它，它可以執行，但不會印出任何東西。在一般的非互動式 Python 程式中，你必須呼叫 print 函式才能印出東西：

```
print(27)
```

我們來做一個 Python 程式檔，並執行它：

1. 開啟文字編輯器。
2. 輸入 print(27) 這一行，跟上面一樣。
3. 將它存成一個稱為 *test.py* 的檔案。務必將它存為一般的文字檔，而不是 RTF 或 Word 等「豐富」格式。你不一定要讓 Python 程式檔使用 *.py* 副檔名，但它可以協助你認出它們是什麼。
4. 如果你使用 GUI（幾乎每個人都是），打開終端機視窗 [2]。

2 如果你不知道這是什麼意思，可參考附錄 B 來瞭解各種作業系統的詳情。

碁峰 www.gotop.com.tw

5. 輸入下列指令來執行程式：

```
$ python test.py
```

你應該會看到這一行輸出：

```
27
```

可以正常動作嗎？如果可以，恭喜你已經執行第一個獨立的 Python 程式了。

接下來要做什麼？

你會在實際的 Python 系統內輸入指令，它們必須遵守正確的 Python 語法。我們不會將所有語法規則一次塞入你的腦中，而是會用接下來的幾章慢慢地介紹它們。

要開發 Python 程式，最基本的做法是使用純文字編輯器與終端機視窗。這本書會使用純文字，有時會展示互動式終端機對話，有時會用一段的 Python 檔。你要知道的是，坊間也有許多優秀的 Python **整合式開發環境**（IDE）。它們可能有 GUI 與進階的文字編輯功能和協助畫面。第 19 章會介紹其中一些 IDE 的細節。

你的禪修時刻

每一種程式語言都有其風格。在前言中，我提過你通常可以用 Python 風格來表達自我。Python 內建一篇散文，它簡潔地闡述 Python 的哲學（就我所知，Python 是唯一有這種彩蛋的語言）。你只要在互動式解譯器中輸入 `import this`，再按下 Enter 鍵，就可以體驗片刻的禪意：

```
>>> import this
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
```

碁峰 www.gotop.com.tw

```
There should be one--and preferably only one--obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea--let's do more of those!
```

我會在這本書舉例說明這些意境。

次章預告

下一章將討論 Python 資料型態與變數，幫你為後續的章節做好準備，那些章節將更深入探討 Python 的資料型態與程式結構。

待辦事項

本章介紹了 Python 語言，包括它可以做什麼，長怎樣，可在電腦世界中扮演什麼角色。我會在每一章的結尾提供一些迷你的專案來幫助你記得看過的內容，並且協助你為接下來的內容做好準備。

1.1 如果你還沒有在電腦中安裝 Python 3，現在就去安裝。你可以閱讀附錄 B 來瞭解關於電腦系統的細節。

1.2 啟動 Python 3 互動式解譯器。附錄 B 有詳細的說明。它應該會印出幾行介紹自己的文字，接著有一行以 `>>>` 開頭的訊息，它就是讓你輸入 Python 指令的提示符號。

1.3 稍微操作一下解譯器。將它當成計算機，輸入：`8 * 9`。按下 Enter 鍵來查看結果。Python 應該會印出 `72`。

1.4 輸入 `47` 這個數字，並按下 Enter 鍵。它會在下一行印出 `47` 嗎？

1.5 現在輸入 `print(47)` 並按下 Enter 鍵。它也會在下一行印出 `47` 嗎？

碁峰 www.gotop.com.tw