簡介

對於怎麼學習寫出第一支程式，每位程式設計師都有屬於自己的故事。當我還是個小孩子時就開始學習程式設計了，那時候我的父親在迪吉多（Digital Equipment Corporation）工作，在當時這是家很先進的電腦公司。我使用一台由我父親在地下室組裝出來的電腦寫出了第一支程式，這台電腦沒有機殼，只有主機板和鍵盤連接，顯示器還是露出的陰極射線管（CRT）螢幕。我寫出的這支程式是很簡單的猜數字遊戲，輸出的畫面如下這般：

```
I'm thinking of a number! Try to guess the number I'm thinking of: 25
Too low! Guess again: 50
Too high! Guess again: 42
That's it! Would you like to play again? (yes/no) no
Thanks for playing!
```

看著家人在玩我寫出來的遊戲，而遊戲也照我的預期執行，這樣的成就感和滿足感是我難以忘懷的。

早年的這種體驗一直影響著我，每當我設計寫出程式來解決某個問題時，心裡都是真實的滿足感。相較於童年，我現在設計的軟體滿足了更大的需求，但這份成就感與滿足感和童年時期幾乎是一樣的。

本書適合的讀者

本書的目標是讓讀者能快點學會 Python，可以寫出能正確執行的程式，例如電玩遊戲、資料數據的視覺化處理和 Web 應用程式等，並在學習的過程中同時能掌握程式設計必學的基礎知識。本書適合所有年齡層的讀者，不要求有任何 Python 的經驗，沒學過程式設計也適用。如果您想快速掌握基本的程式設計知識，然後專注在開發您感興趣的專題應用上，並想藉由解決實際問題來檢驗您對學習新概念的理解程度，這本書就是針對您所設計編寫的。本書適用於各種程度的教學課程安排，書中專案導向的專題應用實作會引導學生學習程式設計的基礎。如果您正在上大學的課程，並且想要得到比課程上教科書更易讀好學的 Python 解說教材，那麼這本書會讓您在上課時更輕鬆容易。如果你想轉職，本書也能協助你過渡到更理想的職場中。本書適用的讀者群十分廣泛。

本書能學到什麼？

本書的目標是希望讀者能成為一流的程式設計人員，尤其是成為優秀的 Python 程式設計師。經由閱讀本書，讀者可以快速掌握程式設計的概念，打下好的基礎，並養成好的程式設計習慣。讀過本書之後，您就有能力學習更進階的 Python 技術，並能更輕鬆地去學習其他程式語言。

在本書的 Part I 中，讀者將學到編寫 Python 程式時要熟悉的基礎程式設計觀念，這些都是在讀者剛接觸任何程式語言時一定要學會的基本概念。讀者可以學到各種資料類型，以及在程式中儲存資料的技巧。您將學會利用串列和字典來建構資料的集合，也學到多種快速遍訪這些集合的處理方式。書中會講解使用 while 和 if 陳述句來檢測條件，並在條件滿足時執行某部分的程式碼，而在條件不滿足時執行另一部分的程式碼－這些技能對自動化處理有很大的作用。

讀者可從書中學到怎麼取得使用者輸入的資料，讓程式能與使用者互動，並在使用者沒有停止輸入時維持執行的狀態。讀者會從範例中一起探索如何編寫函

式來讓程式的各個部分可以重複使用，這樣在編寫執行某項工作的程式碼後，想要重複使用幾次都可以。隨後書中會教您使用類別來擴充延伸，好能實作出更複雜的行為，並讓很簡單的程式也能處理各種不同的狀況。讀者會在書中學到怎麼編寫能好好處理常見錯誤例外的程式。有了這些基本知識後，就能寫出越來越複雜的程式來解決一些特定的問題。讀者在這個 Part 的最後會學習邁向中級程度的程式設計，學習如何為程式碼編寫測試，以便在未來修改擴增程式時不用擔心會引入 bug。Part I 基礎必修章節所介紹的知識能讓讀者開發更大、更複雜的專案應用程式。

在 Part II 專題應用的專案開發中，讀者會用到 Part I 所學到的知識來開發三個應用專題。讀者可依據自己的需要，以最適合的順序來學習完成這三個大型專案，或是選擇只完成其中某些內容。在第一個專題（第 12~14 章）中，您會建立一個像小蜜蜂射擊的電玩遊戲，名稱為「外星人入侵」，此專案含有多個難度不斷增加的遊戲關卡。在您完成這個專案後，就有能力自己動手設計開發 2D 電玩遊戲了。就算您不想成為一名遊戲程式設計師，這個專案的內容也是讓您把 Part I 所學到的知識整合在一起的運用實作。

第二個專題（第 15~17 章）則是介紹資料的視覺化處理。資料科學家的目標是利用各種視覺化技術，以易讀好懂的方式來呈現大量的資訊。讀者在這裡會學習如何處理由程式碼生成、已從網路下載，或是程式自動下載的資料集合。完成這個專案後，讀者就有能力寫出對大型資料集合進行篩選的程式，並以視覺化的方式來把資料呈現出來。

在第三個專題（第 18~20 章）中，讀者會建置一個名為「Learning log（學習日誌）」的小型 Web 應用程式。這個專案應用程式能讓使用者對某個特定主題的學習歷程、概念、心得記錄下來。讀者能分別記錄不同的學習主題，還可讓其他人建立帳號，並開始記錄屬於他自己的學習日誌。讀者會學到如何部署這個專案應用程式到線上的伺服器，讓網路上的所有人都能使用這個系統。

線上資源

請連到 *https://nostarch.com/python-crash-course-3rd-edition* 或是作者主持維護的網站 *https://ehmatthes. github.io/pcc_3e* 取得本書的相關補充資源，其資源如下所示：

第 2 章

變數和簡單資料型別

本章您將學到在 Python 程式中運用各種不同的資料型別，也將會學習如何使用變數在程式之中存放和表示不同的資料。

執行 hello_world.py 時程式做了什麼？

我們一起來瞧瞧執行 hello_world.py 時，Python 做了哪些事？事實上，就算只是執行很簡單的小程式，Python 也做了相當多的工作：

⬇ hello_world.py

```
print("Hello Python world!")
```

執行這行程式碼時，會顯示下列的輸出：

```
Hello Python world!
```

在執行 hello_world.py 檔時，副檔名 .py 已指出這是個 Python 程式，因此編輯器會以 Python 直譯器來執行它，直譯器會巡遍整支程式的內容，並確定程式中每個單字的意義，例如，當直譯器看到 print 單字時，不管括弧中的內容是什麼，都會印出到螢幕上。

當您在編寫程式時，編輯器會以顏色來標示程式碼不同的語法內容，舉例來說，它認出 print() 是個函式名稱，會以用一種屬於 Python 程式碼的顏色來標示，它認出 "Hello Python world!" 不是 Python 程式碼，就以另外一種顏色來標示，這樣的功能稱為「**語法突顯（Syntax highlighting）**」，在剛開始學習編寫程式時會很有用。

變數

讓我們試著在 hello_world.py 中使用變數吧！在這個檔案的開頭新增一行程式碼，並修改第 2 行程式碼：

⬇ hello_world.py

```
message = "Hello Python world!"
print(message)
```

執行這支程式看看會發生什麼事情。您應該會看到和前面一樣的輸出結果：

```
Hello Python world!
```

我們在這裡新加入一個名稱為 message 的**變數**，每個變數都會存放一個**值（value）**，這個值是與變數相關聯的資訊，在這個範例中，值就是「Hello Python world!」。

加入變數會讓 Python 直譯器多做一些工作，當它在處理第 1 行指令時，會把「Hello Python world!」和 message 變數關聯起來，而在處理第 2 行時，會把 message 變數關聯的值印在螢幕上。

接著再對 hello_world.py 修改擴充，讓它印出第二條訊息文字。在 hello_world.py 加上一行空行，然後輸入二行新的程式指令：

```
message = "Hello Python world!"
print(message)
```

```
message = "Hello Python Crash Course world!"
print(message)
```

現在執行程式就會看如下兩行輸出：

```
Hello Python world!
Hello Python Crash Course world!
```

在程式中我們可以隨時修改變數的值，而 Python 也會記錄下它現在最近的值。

變數的命名和使用

當您在 Python 中使用變數，需要遵守幾項規則和準繩，若違反這些規則會引起錯誤的發生，而準繩指引會讓您編寫出來的程式更易讀和易懂。請一定要記住下列的變數規則：

- 變數名稱只能有英文字母、數字和底線。變數名稱可以字母和底線開頭，但不能以數字起頭，例如，變數名稱可以使用 message_1，但不能用 1_message。
- 變數名稱裡不能有空格，但可以用底線來做單字的分隔，例如，變數名稱可以用 greeting_message，但不能用 greeting message，否則會引發錯誤。
- 避免使用 Python 的關鍵字和內建函式名稱來當作變數的名字，換句話說，就是不要使用 Python 留作特殊程式功能的單字，例如 print 單字（詳情請參考附錄 A 的「Python 關鍵字和內建函式」）。
- 變數名稱最好是簡短又具備描述性，舉例來說，取名為 name 會比 n 好，student_name 也比 s_n 好，而 name_length 則比 length_of_persons_name 好。
- 小心使用小寫的 *l* 和大寫的 *O*，因為它們跟數字 *1* 和 *0* 很像，容易搞混。

想要取好變數名稱還需要多一點練習和實作，特別是當您編寫的程式愈來愈複雜和有趣時，更要留意變數名稱的重要。當您編寫設計愈多程式和看過更多別人所寫的程式碼後，您就愈來愈會取出好的變數名稱。

NOTE

現階段建議您使用小寫英文字來取 Python 變數名稱，雖然用大寫字母取名不會引發錯誤，但具有特別的意義，後面的章節會講解。

使用變數時避免 NameError

編寫程式時難免會出錯，而且每天都會發生。好的程式設計師雖然也會犯錯，但他們知道並能有效快速地回應處理這些錯誤。接下來讓我們看一下大家可能犯的錯誤，並學習怎麼修復。

我們會故意編寫一些有錯的程式碼當例子，請輸入如下的內容，包括其中以粗體顯示但拼錯的單字 mesage：

```
message = "Hello Python Crash Course reader!"
print(mesage)
```

當程式中發生錯誤，Python 直譯器會盡量幫您找出問題所在。當程式執行失敗時直譯器會提供 traceback 訊息。**traceback** 是一條記錄（record），記下直譯器執行程式碼時在什麼位置遇到麻煩。下面是不小心打錯變數名稱，執行時 Python 直譯器所提供的 traceback 訊息：

```
   Traceback (most recent call last):
❶   File "hello_world.py", line 2, in <module>
❷     print(mesage)
            ^^^^^^
❸ NameError: name 'mesage' is not defined. Did you mean: 'message'?
```

在輸出報告❶中指出 hello_world.py 檔的第 2 行出現錯誤，直譯器會秀出這行內容，協助我們能快速找出錯誤❷，並且也會告知發生什麼樣的錯誤❸。在這個例子中顯示的是 NameError 的錯誤，回報要印出的變數 mesage 沒有定義，Python 無法辨識這個變數名稱。NameError 錯誤通常是我們使用某個變數前卻忘了對變數設定值，或是在輸入變數名稱時打錯字。如果 Python 在程式中找到一個與拼錯變數名稱相似的變數名稱，它會詢問這是否是您要使用的名稱。

當然，這個例子是我們在第二行故意對 message 變數名稱少打一個 s，Python 直譯器雖不會對程式碼進行拼字檢查，但會要求在設定值和使用上變數名稱要一致。舉例來說，如果程式碼中指定值的 message 也打錯成 mesage，那執行結果會怎樣呢？

```
mesage = "Hello Python Crash Course reader!"
print(mesage)
```

在這種情況下，程式執行會順利成功！

```
Hello Python Crash Course reader!
```

程式語言很嚴謹，但它卻不會管單字拼寫的好與壞，因此，在建立變數名稱和編寫程式碼時，不需要考量英文的拼寫和文法。

很多程式設計上的錯誤其實都很簡單，只是某行程式中的某個字元打錯了而已。如果您花了很多時間在找這種錯誤，那表示您跟大家一樣，很多程式設計老手也會花上數小時來尋找這種微小的錯誤，很好笑，對吧！當您在程式設計生涯的路上繼續往前進時，您就會知道這種鳥事有多常發生。

變數是個標籤

變數通常都被描述成箱子，可以讓我們存放東西進去。這個概念在一開始使用變數時會很有幫助，但並不能準確描述變數在 Python 內部所代表的意義。最好把變數視為可以指定值的標籤（label）。您也可以把變數看成是參照引用了某個值。

在您的初始程式中，上述的區別看起來可能無關緊要，但這個觀念還是值得早點學習。在以後的某個時間點上，您可能會突然看到變數的行為與您所想的不同，若能準確了解變數的工作方式會有助於您識別程式碼中正在發生的事情。

NOTE

想要理解某個新的程式設計觀念，最好的方式就是把它用在您的程式中。如果您在實作本書的練習題時遇到困難，先停下來試試別的東西，如果還有困難，就再複習書中的相關內容，如果情況依舊，則請參考附錄 C 的建議。

實作練習

請完成下列的練習題，在實作時都要編寫一支獨立的程式，儲存程式時要符合標準的 Python 對檔案名稱命名的慣例，使用小寫英文字和底線，例如 simple_message.py 或 simple_messages.py 等。

2-1. 一條簡單的訊息：將一條訊息文字指定到變數中，再把它印出。

2-2. 多條簡單的訊息：將一條訊息文字指定到變數中，把它印出；再把變數的值修改為另一條新訊息文字，再把它印出。

字串

因為大多數的程式都會定義和收集某些資料再拿來運用，所以對資料進行分類是有必要的，我們在這裡討論的第一種資料型別是字串，初看之下字串好像很簡單，但卻有很多不同的運用方式。

字串（String）其實是一系列的字元所組成，在 Python 中以引號括起來的都是字串，引號可以是單引號或雙引號，如下列所示的實例：

```
"This is a string."
'This is also a string.'
```

兩種引號運用的彈性可讓我們在字串中也能放入不同的引號：

```
'I told my friend, "Python is my favorite language!"'
"The language 'Python' is named after Monty Python, not the snake."
"One of Python's strengths is its diverse and supportive community."
```

接下來介紹一些字串的應用方式。

使用方法來變更字串的大小寫

對字串進行變更最簡單的操作就是變更其字母的大小寫，請看下列程式碼範例，並試著判斷結果如何：

⬇ name.py

```
name = "ada lovelace"
print(name.title())
```

把這支程式儲存成 name.py 再執行，其輸出結果如下：

```
Ada Lovelace
```

在這個範例中，name 變數指到小寫字串 "ada lovelace"，在 print() 陳述句中變數的後面出現了 title() 方法。**方法（method）**是 Python 對某段資料所進行的操作，在 name.title() 中位在 name 後面的句點（.）是用來告知 Python 對 name 變數進行 title() 的操作。每個方法後面都會跟著一對括號，因為方法通常都需要額外的資訊來搭配一起完成工作，額外的資訊會放在括號內，由於 title() 方法不需要額外的資訊配合，所以括號中是空的。

title() 方法的功用是讓字串的單字以標題首字大寫的方式呈現，在常需要把名字當成某段資訊時這個方法還滿有用的，例如，我們希望程式把 Ada、ADA 和 ada 等輸入值都視為同一個名字，並都顯示為 Ada。

這裡還有幾個有用的方法可處理字串的大小寫，舉例來說，我們可以把字串都改成大寫字母或小寫字母，如下所示：

```
name = "Ada Lovelace"
print(name.upper())
print(name.lower())
```

這段程式碼執行結果會顯示為：

```
ADA LOVELACE
ada lovelace
```

lower() 方法在儲存資料上特別有用。在大多數情況下，我們不用依靠使用者在輸入時用了正確大小寫，因此把字串都先轉換為小寫再儲存，這樣就能確保資料的一致性。日後需要輸出顯示這些資訊時，再轉換適當的大小寫方式來顯示即可。

在字串中使用變數

在某些情況下，您需要在字串中使用變數。舉例來說，您可能需要用兩個變數來表示名字的姓和名，當在顯示時才組合起來讓別人看到完整的名字：

⬇ full_name.py

```
   first_name = "ada"
   last_name = "lovelace"
❶ full_name = f"{first_name} {last_name}"
   print(full_name)
```

若想要把某個變數的值插入字串中，可用字母 f 緊接在引號之前❶來括住變數。在字串中要使用的任何一個或多個變數名稱是需要在兩邊加上大括號。當在顯示字串時，Python 會把變數替換為其指到的「值」。

這樣的字串稱為 **f-strings**。其 **f** 是指 format（格式）的簡寫，因為 Python 會把大括號中的變數名稱替換為其指到的值，以此來格式化字串，所以前面程式碼的輸出結果為：

```
ada lovelace
```

f-strings 的用途很多，舉例來說，我們可以用 f-strings 把變數中所關聯的資訊連接起來組成完整的訊息。讓我們來看個範例：

```
   first_name = "ada"
   last_name = "lovelace"
   full_name = f"{first_name} {last_name}"
❶ print(f"Hello, {full_name.title()}!")
```

在❶這裡的句子使用了完整名字來問候，並用了 title() 方法把完整姓名設成標題式首字大寫的格式呈現，這段程式執行後返回經過格式美化的問候語句：

```
Hello, Ada Lovelace!
```

我們可以利用 f-strings 把訊息組合起來再指定到一個變數內：

```
   first_name = "ada"
   last_name = "lovelace"
   full_name = f"{first_name} {last_name}"
❶ message = f"Hello, {full_name.title()}!"
❷ print(message)
```

這段程式碼也會顯示「Hello, Ada Lovelace!」訊息，在❶這裡是把整段訊息字串指定到一個變數內，這樣讓 print() 陳述句變簡潔許多❷。

使用定位符號或換行來增加空白

在編寫程式的過程中，**空白（whitespace）**是指不會印出來的各種字元，例如空格、定位符號和換行符號等。我們可以利用空白來組織編排輸出的呈現，讓顯示的內容更容易閱讀。

若想要加個定位空白到字串中，可使用字元連接 \t，如下所示：

```
>>> print("Python")
Python
>>> print("\tPython")
    Python
```

若想要加個換行符號到字串中，可使用字元連接 \n，如下所示：

```
>>> print("Languages:\nPython\nC\nJavaScript")
Languages:
Python
C
JavaScript
```

我們也可以把定位符號和換行符號放在同個字串中，「\n\t」字串會讓 Python 先換行，然後在新行開頭加入定位符號空白。下列的範例展示了怎麼以一行字串產生四行輸出：

```
>>> print("Languages:\n\tPython\n\tC\n\tJavaScript")
Languages:
    Python
    C
    JavaScript
```

在下二章的內容中，當我們以很少行的程式碼來產生多行輸出時，定位符號和換行符號將會發揮很大功用。

刪除空白

在程式中多餘的空白可能會引起混亂，對設計程式的人來說，以下的 'python' 和 'python ' 看起來好像沒什麼多大差別，但對程式來說，這是兩個完全不同的字串。Python 會偵測到 'python ' 中的空白，並視這個空白是有其意義的，除非您告知它並不是這麼一回事。

在比對兩個字串判斷是否相同時，空白也很重要，例如，像使用者登入網站時要比對檢查其輸入的帳號名稱，這就是重要的應用實例。多餘的空白在一些更簡單的情況中也會產生混淆，好在 Python 要刪除使用者輸入資料中多餘的空白是很簡單的事。

Python 能找出某個字串其左側和右側是否有多的空白。要確定某個字串其右側沒有空白，可使用 rstrip() 方法。

```
❶ >>> favorite_language = 'python '
❷ >>> favorite_language
  'python '
❸ >>> favorite_language.rstrip()
  'python'
❹ >>> favorite_language
  'python '
```

關聯到 favorite_language 變數內的字串結尾處有多了空白❶，當我們在終端對話中向 Python 要求顯示此變數的值時，就會看到這個值結尾處有多了空白❷，當我們對 favorite_language 變數使用 rstrip() 方法❸，結尾的空白就會去掉了，不過這只是暫時的刪除而已，接著再次輸入 favorite_language 要求 Python 顯示其值時，您會發現此字串和當初是一樣的，還是有多餘的空白在結尾❹。

想要永久刪掉字串的空白，則要把刪除空白後的字串再指定回變數中：

```
  >>> favorite_language = 'python '
❶ >>> favorite_language = favorite_language.rstrip()
  >>> favorite_language
  'python'
```

若想要刪除掉字串的空白，要先呼叫 rstrip() 刪除結尾的空白後，再將其結果關聯指定到變數中❶。在編寫和設計程式的過程中，修改變數的值並將新的值指定回原本的變數中是很常見的處理，這就是為什麼變數中存放的值會隨著程式的執行或使用者的回應而產生變化。

我們也可以使用 lstrip() 刪除字串左側開頭的空白，或是使用 strip() 同時把左右兩側的空白都刪掉：

```
❶ >>> favorite_language = ' python '
❷ >>> favorite_language.rstrip()
  ' python'
❸ >>> favorite_language.lstrip()
  'python '
❹ >>> favorite_language.strip()
  'python'
```

在範例中，一開始就放了一個左右都有空白的字串關聯到 favorite_language 變數內❶，接著分別把右側❷、左側❸和兩側❹的空白都刪除。好好活用這幾個刪除函式會讓我們能夠靈活操作字串的內容。在實際的應用中，這些刪除函式最常用來處理使用者的輸入，刪除整理好輸入的內容後才儲存起來。

刪除前置內容

在運用字串時，另一個常見的處理是刪除前置內容。請思考某個帶有通用前置內容「https://」的 URL 網址。我們想要刪除這個前置內容，只關注使用者在網址欄中需要輸入的 URL 部分。以下是實作的方法：

```
>>> nostarch_url = 'https://nostarch.com'
>>> nostarch_url.removeprefix('https://')
'nostarch.com'
```

輸入變數名稱後跟一個句點（.），然後是方法 removeprefix()。在括號內，輸入要從原始字串中刪除的前置內容。

與刪除空格的方法一樣，removeprefix() 會保留原始字串不變。如果要把刪除前置內容的新值保留下來，請將它重新指定到原來的變數，或是將它指定給新的變數：

```
>>> simple_url = nostarch_url.removeprefix('https://')
```

當您在網址欄中看到某個 URL 並沒有顯示 https://，這表示瀏覽器可能在幕後使用了類似 removeprefix() 的方法來進行處理。

字串的使用要避免產生語法錯誤

語法錯誤（syntax error）是一種很常會碰到的錯誤，當 Python 沒辦法識別程式中所含有的不合法程式碼時，就會告知有語法錯誤的發生。舉例來說，在用單引號括住的某個字串中又放了單引號時，就會發生語法錯誤。這是因為 Python 把括住字串的第一個單引號和字串中的第一個單引號搞混，以致於引號不能相配合，讓剩下的文字被當成 Python 程式碼而引發語法錯誤。

接下來示範正確使用單引號和雙引號，範例會存成 apostrophe.py 檔並執行：

⬇ apostrophe.py

```
message = "One of Python's strengths is its diverse community."
print(message)
```

字串中的單引號（ ' 撇號）是放在兩個雙引號之間，因此 Python 直譯器能正確識別這是字串的一部分，執行結果也順利顯示：

```
One of Python's strengths is its diverse community.
```

不過，如果我們把改用單引號來括住字串，則 Python 就無法正確識別字串結尾的位置：

```
message = 'One of Python's strengths is its diverse community.'
print(message)
```

當我們執行這段程式，就會產生如下的錯誤訊息：

```
  File "apostrophe.py", line 1
    message = 'One of Python's strengths is its diverse community.'
                                                                ❶ ^
SyntaxError: unterminated string literal (detected at line 1)
```

在輸出中您會看到錯誤定在最後的單引號❶位置上，這個 SyntaxError 指出直譯器無法識別程式中的某些不合法程式碼，告知可能是字串的引號使用出問題。錯誤的發生可能源自於很多不同的因素，這裡我只點出一些常見的。在您學習編寫程式時可能會常碰到語法錯誤，這種錯誤是最不具體也難以分辨的一種，有時很難發覺和修改，如果您在程式設計的過程中陷入這樣的泥淖中，可參閱本書附錄 C 所提供的建議。

NOTE

編寫和設計程式時，文字編輯器的語法突顯（顏色標示）功能可幫您快速找出某些語法錯誤，若看到編輯器把 Python 程式碼以普通文字的顏色顯示時，或者把普通文字以程式碼的顏色來標示的話，就有可能是其中使用的引號不能配合。

實作練習

在實作每個練習時都請都編寫成獨立的程式檔，儲存程式時檔名取名為 name_cases.py 之類的檔案。如果實作過程遇到困難，請先休息一下或參閱附錄 C 的建議。

2-3. 個人訊息：使用變數來代表某個人的名字，並對那個人印出個人訊息。顯示的訊息文字要簡單明瞭，如 "Hello Eric, would you like to learn some Python today?"。

2-4. 英文名字的大小寫：使用變數來代表個人名字，再以全都小寫、大寫和字首大寫等方式顯示出來。

2-5. 名言：找出您所欣賞名人的名言，印出名人的名字和名言，其印出的格式如下所示，包括引號也在內：

> Albert Einstein once said, "A person who never made a mistake never tried anything new."

2-6. 名言 2：重複 2-5 的實作練習，但請用 famous_person 變數來代表名人的名字，再組合要顯示的名言訊息，一起指定關聯到 message 變數內，然後印出這個訊息。

2-7. 刪除名字中的空白：使用變數來代表某個人的名字，這個名字的前後有加了空白字元，空白字元至少要使用 "\t" 和 "\n" 一次。

先印出含有空白的名字。然後分別使用 lstrip()、rstrip() 和 strip() 函式來整理名字，並將結果印出來。

2-8. 副檔名：Python 有一個 removesuffix() 方法，其功能和處理方式與 removeprefix() 完全相同。把值 "python_notes.txt" 指定給名為 filename 的變數，然後使用 removesuffix() 方法顯示沒有副檔名的檔案名稱，就像某些檔案瀏覽器所顯示的那樣。

數值

在程式設計中很常使用「數值」來記錄遊戲的分數、呈現視覺化的資料、儲存 Web 應用程式的資訊等。Python 會依據程式使用數字的方式來進行不同的處置，由於使用上很簡單，我們就一起來看看 Python 是如何管理整數的應用。

整數

在 Python 中可以對整數進行 +（加）、-（減）、*（乘）、\（除）的運算。

```
>>> 2 + 3
```

```
5
>>> 3 - 2
1
>>> 2 * 3
6
>>> 3 / 2
1.5
```

在終端對話模式中，Python 會直接返回顯示運算的結果。Python 使用 **（兩個乘號）來表示次方的運算：

```
>>> 3 ** 2
9
>>> 3 ** 3
27
>>> 10 ** 6
1000000
```

Python 也支援運算順序（由左而右、先乘除後加減等順序），因此可以在一個表示式中使用多個運算子，我們還可以用括號來改變運算的順序，讓 Python 依照我們指定的順序來運算，舉例來說：

```
>>> 2 + 3*4
14
>>> (2 + 3) * 4
20
```

在這幾個範例中可得知表示式中的空格不影響 Python 運算，空格協助我們在閱讀表示式時能快速確定運算子的執行順序。

浮點數

Python 把帶有小數點的數值都稱為**浮點數（float）**，這個詞在大多數的程式語言中都有用到，它指出了小數點可放在數字的任一位置這樣的事實。每種程式語言都要小心設計妥善管理浮點數，好讓小數點不管出現在什麼位置，數值都是正確的。

在大多數的情況下，使用浮點數並不用擔心它們有什麼影響，只需輸入想要使用的數值，Python 都會以我們所期待的方式來運算和處理：

```
>>> 0.1 + 0.1
0.2
```

```
>>> 0.2 + 0.2
0.4
>>> 2 * 0.1
0.2
>>> 2 * 0.2
0.4
```

但請注意您所得到的運算結果，其小數位數有可能不是那麼精確：

```
>>> 0.2 + 0.1
0.30000000000000004
>>> 3 * 0.1
0.30000000000000004
```

所有程式語言都有這樣的問題，不需要太擔心。Python 會盡可能找出最精確的方式來表示結果，但受限於電腦內部表示數值的方式，所以有時會很難十分精確。以現階段來看，可暫時忽略多餘的小數位置，等到本書 Part 2 的專題實作範例中，就能學到處理多餘小數位置的方法。

整數與浮點數

當我們對兩個數值進行除法運算時，就算兩個數值都是整數且能整除，其運算的結果仍會以浮點數呈現：

```
>>> 4/2
2.0
```

如果在其他的運算式中混和了整數與浮點數進行運算，其結果也是浮點數：

```
>>> 1 + 2.0
3.0
>>> 2 * 3.0
6.0
>>> 3.0 ** 2
9.0
```

Python 在使用浮點數的任何操作中，就算輸出結果是整數，預設還是會以浮點數來呈現。

在數字中使用底線

當我們使用長位數的數值時，可用底線來進行三位劃分，讓長數字更好閱讀：

```
>>> universe_age = 14_000_000_000
```

當我們印出用了底線劃分位置的數值時，Python 只會印出數值，沒有底線：

```
>>> print(universe_age)
14000000000
```

當我們在存放這種加了底線的數值時，Python 會忽略掉底線。就算數字不是以三位來加底線劃分，也不會有影響。對 Python 來說，1000、1_000 和 10_00 都是一樣的，這項功能可用在整數和浮點數上。

多重指定

我們可以僅用一行語法就能將多個值指定給多個變數。這樣能縮減程式行數，也讓程式更容易閱讀。這項技巧大都用在一組數值的初始化處理時。

舉例來說，下列是對 x、y 和 z 變數以 0 值進行初始化的處理：

```
>>> x, y, z = 0, 0, 0
```

變數和值都是以逗號來分隔，Python 會依其位置順序來進行值指定到變數中。變數的數量和值的數量要一致，不然 Python 會顯示錯誤訊息。

常數

常數（constant）也是個變數，其中所存放的值在整支程式中都會維持相同。Python 沒有內建的常數型別，但是 Python 程式設計師會以全都是大寫字母的變數當作常數，且都不更改其常數內的值：

```
MAX_CONNECTIONS = 5000
```

當您想在程式碼中把變數當成常數時，請將該變數的名字全都改用大寫字母來表示。

實作練習

2-9. 數字 8：編寫 4 個表示式，分別使用加、減、乘和除法運算，讓運算的結果都為 8，使用 print 陳述句來顯示結果，表示式要放在 print() 的括號內，也就是要編寫設計出像下列這樣的 4 行程式碼：

```
print(5+3)
```

輸出要分成 4 行，其中每行的結果都只能為數字 8。

2-10. 最愛的數字：使用一個變數來表示您最愛的數字，再搭配使用這個變數製作一條訊息說明這是您最愛的數字，然後將這條訊息印出。

注釋

在大多數的程式語言中「注釋」是很有用的功能。本書前面所編寫的程式中都只放了 Python 程式碼，但隨著程式愈來愈長、愈來愈複雜，就需要在程式碼中加上說明注釋，大致描述程式解決問題的方法。**注釋（comment）**，或譯**註解**可以讓我們以口語的文字在程式中加入說明。

如何編寫注釋？

在 Python 中，井字符號（#）標識出注釋的內容，井字符號後面的內容都會被 Python 直譯器忽略，例如：

⬇ comment.py

```
# Say hello to everyone.
print("Hello Python people!")
```

Python 直譯器會忽略第 1 行，只執行第 2 行的指令。

```
Hello Python people!
```

要寫出什麼樣的注釋呢？

編寫注釋的主要理由是用來解釋程式碼是要做什麼的，並說明程式是怎麼運作的。當您身處在開發專案中，此時也許很能掌握理解專案的各部分並知道怎麼整合在一起，但時間一久，再回到專案時有些細節可能就忘了，雖然能透過研讀程式碼來重拾對整個專案的了解，但若在程式碼中編寫了好的注釋，以清楚的口語來描述專案的內容，這樣可以讓我們省下很多時間。

如果您想要成為專業的程式設計師，或與其他程式設計師一起協同工作，就要編寫出有意義的注釋。現在大多數的軟體應用程式都是很多人協同合作所開發出來的，設計和編寫的人可能是同一家公司的多位員工，也可能是開放原始碼專案中一起合作的一群人。程式設計老手大都希望程式碼中含有注釋，因此您最好從現在開始就養成在程式中加入描述性的注釋說明。以程式設計初學者來說，最值得養成的習慣之一就是在程式碼中編寫清楚、簡潔的注釋說明。

當您無法決定是否要編寫注釋時，先回頭問問自己在找到最合理的解決方案之前是否有考慮了多種方法；如果是，請寫下這解決方案的相關注釋說明。刪除多餘寫好的注釋會比回過頭為簡短的程式再編寫注釋要容易得多了。從現在開始，我會在整本書的範例中加上注釋來協助解釋程式碼內容。

實作練習

2-11. 新增注釋：選兩支您所編寫的程式，在程式中都至少新增一條注釋說明文字。如果程式太簡單沒什麼重點要說明，就在程式的開端加上您的姓名和日期的注釋，再用一句話描述程式的功用。

Python 之禪

有經驗的 Python 程式設計師都會建議您不要弄得太複雜，並盡可能簡化目標。Python 社群的哲學都放在 Tim Peters 的「Python 之禪（The Zen of Python）」中。若想要取得寫出一流 Python 程式碼的這套原則說明，可在終端對話中輸入 import this，直譯器就會顯示出來。在這裡我不再重述整個「Python 之禪」的內容，但會和大家分享其中幾條原則，協助您了解為什麼這些原則對程式初學者很重要。

```
>>> import this
The Zen of Python, by Tim Peters
Beautiful is better than ugly.
```

Python 程式設計師信奉程式碼可以寫得漂亮而優雅。程式設計能解決問題，程式設計師對於設計精良、高效率且優雅的解決方案是很推崇的，當您學會更多關於 Python 的功用，並用它來編寫更多程式碼時，也許某一天有人可能會在您身後說：「哇，這段程式碼寫得真漂亮！」

```
Simple is better than complex.
```

如果有一個簡單和一個複雜的解決方案都能用，請選擇簡單的那個吧！這樣一來，您的程式碼會更好維護，日後您或他人想要擴充使用這些程式碼時也會更容易。

```
Complex is better than complicated.
```

現實的世界沒那麼簡單，有時也沒有簡單的路可走，在這種情況下，就盡量選最簡單可行的那個解決方案吧！

```
Readability counts.
```

就算您的程式是複雜的，也要讓它們易讀好懂，當您在處理一個含有複雜程式碼的開發專案時，要記得為這些程式碼加上有用的注釋說明。

```
There should be one-- and preferably only one --obvious way to do it.
```

如果有兩位 Python 程式設計師被要求去解決同個問題，那麼他們所提出的解決方案應該大致相容。別說程式設計沒有創新的空間，其實剛好相反！但是在更大型、更有創新空間的專案中，大部分的程式設計工作也都還是用一般常見的

解決方案來處理簡單的問題。在您的程式中最基本且重要的部分，對其他 Python 程式設計師來說也是要合理且講得通的。

```
Now is better than never.
```

花太多時間來學習 Python 的所有複雜內容和程式技巧，可能永遠不會完成什麼專案。別試著寫出一切完美的程式，應該先寫出程式再說，然後再看看是否能進一步改進這支程式，或是轉去重寫新的程式。

當您繼續下一章的內容，開始研究更深入的課題時，請記住「簡潔和清晰」這個禪意，如此一來，就算是程式設計老手也會對您所編寫設計的程式產生敬意，並且樂於給您回應，與您合作您感興趣的專案。

實作練習

2-12. Python 之禪：在 Python 的終端對話中輸入 **import this**，然後按下 Enter，瀏覽一下列出的這些原則。

總結

本章我們學會了變數的使用，知道怎麼取個好的變數名稱，也學到了如何修正 NameError 名稱錯誤和 SyntaxError 語法錯誤；還學到什麼是字串，以及如何將字串的英文字母以小寫、大寫和字首大寫等格式顯示出來；學會使用空白來顯示分隔輸出顯示的內容，以及如何刪除字串中多餘的空白；學會如何使用整數和浮點數，以及處理數值資料型別的多種方式；同時也學到編寫注釋說明，讓程式碼內容更容易閱讀和好懂。最後還介紹了編寫程式時要盡量簡單的禪意。

在第 3 章，我們將要學習怎麼在「串列」的資料結構中儲存一組資訊，以及學習遍訪整個串列來存取操控其中的資訊。