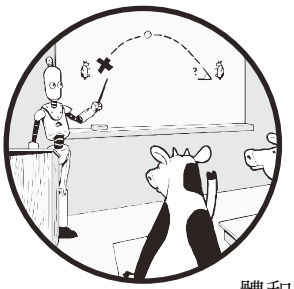
簡介



我們會使用電腦來完成工作和解決問題。舉例來說，也許您曾經使用文書處理軟體來寫過一篇文章或書信；或是曾經使用試算表軟體來打理您的財務資料；又或是使用過影像編輯軟體來修飾圖片。很難想像在現今生活中不使用電腦來處理這些事情。我們從文書處理軟體、試算表軟體和影像編輯軟體得到很多助益。

這些軟體程式被設計成通用的工具來完成各式各樣的工作。但話雖如此，它們都還是別人設計編寫的程式，而不是由我們自己設計編寫來的。當現有的程式不能完全滿足需求時，我們該怎麼辦呢？

在本書中，我們的目標是超越終端使用者只會運用電腦中現有的程式來完成工作。我們要學會編寫出符合自己需要的程式。但不會設計編寫文書處理軟體、試算表或影像編輯軟體，設計這些軟體程式算是大型的艱鉅工作，幸運的是已有人完成並提供這類工具。我們在本書將學習如何設計編寫小型程式來解

決一些現有工具不能搞定的問題。我會協助讀者學習怎麼對電腦下達指令（instructions）；這些指令所統合起來的計劃就是告知電腦怎麼執行來解決問題的藍圖。

為了要向電腦下達指令，我們會用**程式語言（programming language）**設計編寫程式碼（code）。程式語言有統一的規則和語法來讓我們設計編寫程式碼，並指示電腦如何回應這些程式碼。

我們會學習使用 Python 語言來進行程式的設計和編寫。這是讀者會從本書中學到的一項具體技能，將來可以把這項技能寫在您的履歷上。此外我們還會學到使用電腦解決問題所需要的思維型態，而不僅僅只是會用 Python 語言。程式語言種類很多，不斷新舊交替、來來去去，而解決問題的思維方法則是固定不變的。我希望這本書能幫助讀者從終端使用者變成程式設計師，並讓讀者在探索各種可能性的過程中獲得樂趣。

線上資源

本書隨附的線上資源包括可下載使用的範例程式碼檔案和額外的練習題目，請連到 https://nostarch.com/learn-code-solving-problems/ 網站查閱取用。

本書適用對象

本書適用於想要學習如何編寫電腦程式來解決問題的所有讀者。特別是下列這三種類型的讀者更適合。

第一種是可能聽過 Python 語言並想學習如何運用 Python 編寫程式碼的讀者。我會在下一小節解釋為什麼 Python 是學習程式語言很好的選擇。讀者能在本書學到很多關於 Python 的必要知識，有了這個基礎，往下一步時就能夠閱讀其他 Python 進階的參考書籍。

第二種是沒有聽過 Python 或者只是想了解程式設計的相關內容，請不要擔心：這本書也很適合這類讀者！本書會教我們如何在程式設計中思考和分析。程式設計師具有特殊的思維方式可以把問題分解成可管理的小部分，並用程式碼來

表達這些小部分的解決方案。在這個層面上，使用什麼程式語言並不是重點，因為程式設計師的思維方式與特定的程式語言並無關係。

最後一種是可能對學習其他程式語言也感興趣的讀者，例如 C++、Java、Go 或 Rust 等程式語言。當讀者在學習其他程式語言時，學過 Python 的好處是有很多基礎知識都是通用共有的。另外，Python 語言本身就十分好用也很值得學習。接下來讓我們談談為什麼 Python 這麼好用。

為什麼要學 Python？

多年的程式設計入門教學經驗向我證明了 Python 是程式語言的第一優選。與其他語言相比，Python 程式碼更具結構化和可讀性。一旦讀者習慣了 Python 語法，就會發現 Python 的某些部分幾乎讀起來像英文口語一樣簡單！

Python 還具有許多其他語言所沒有的功能特性，包括可用來操控和儲存資料的強大工具。我們會在本書中介紹和使用其中的許多功能。

Python 不僅是可用來教學程式設計的出色語言，它也是目前世界上最受歡迎的程式語言之一。程式設計師使用它來編寫出 Web 應用程式、遊戲軟體、視覺化應用、機器學習軟體等。

有了 Python 之後，讀者就有了非常適合教學的語言，也帶來專業的優勢。我不能要求更多了！

安裝 Python

在可以用 Python 進行程式設計之前，需要先安裝好。以下是安裝的方法。

Python 有兩個主要版本：Python 2 和 Python 3。Python 2 是舊版本，也不再支援了。在本書中，我們使用的 Python 3，因此請在電腦中安裝 Python 3。

Python 3 是繼 2 版之後的重大更新，而 3 這個版本仍在不斷更新中。Python 3 的第一個版本是 Python 3.0，之後發布了 Python 3.1，然後是 Python 3.2，依此一直更新。在撰寫本書時，Python 3 的最新版本是 Python 3.9。不過本書使用

Python 3.6 之類的舊版本就已足夠，但我還是鼓勵讀者安裝並使用最新版本的 Python。

請按照適用於您電腦作業系統的操作步驟來安裝 Python。

Windows

預設的情況下，Windows 是沒有內建 Python 的。需要下載安裝，請連到 https://www.python.org/ 並點按 **Downloads** 連結，這裡應該會提供下載適用於 Windows 系統的最新版本 Python 選項。點按連結下載 Python，然後執行安裝程式。在安裝過程的第一個畫面中，勾選 **Add Python 3.9 to PATH** 或 **Add Python to environment variables**；這樣可以在執行 Python 時變得更容易（如果是更新 Python 版本，則可能需要點按「Customize installation」才能找到這個選項）。

macOS

預設的情況下，macOS 沒有內建 Python 3。需要另外安裝，請連到 https://www.python.org/ 並點按 **Downloads** 連結。這裡應該會提供下載適用於 macOS 系統的最新版本 Python 選項。點按連結下載 Python，然後執行安裝程式。

Linux

Linux 已內建 Python 3，但它可能是 Python 3 的舊版本。安裝指南會根據您使用的 Linux 發行版本而有所不同，但您應該能夠使用最喜歡的套件管理工具來安裝最新版本的 Python。

如何閱讀本書

一口氣從頭到尾讀完這本書，這種讀法能教給您的東西比較少。這種方式就像聘請某人到你家彈鋼琴幾個小時，請他們回去後，您調暗燈光，唱個小夜曲就覺得自己學會彈鋼琴了。這不是以動手實作的方式來學習。

以下是閱讀本書的幾項建議：

騰出一段學習時間。斷斷續續的少量課程時間，效果遠不如騰出一段學習時間來實作練習。當您學習過程感到疲倦時，休息一下再出發。沒有人能告訴休息後要學習多久時間才有效果，也沒人能告訴您讀完本書需要多長時間，這取決於您自己的身心狀態。

暫停一下來測試理解的程度。閱讀與學習後會讓我們產生已掌握和理解的錯覺。實際應用一下能讓我們知道理解的程度是否對等。出於這個原因，在每一章的關鍵點會有「觀念回顧」的多選擇題來測試您的理解程度。請認真對待這些習題！仔細閱讀題目並回答，無需使用電腦來檢測問題內容。隨後再閱讀我的答案和解釋。這樣的確認能讓讀者走在正確的學習軌道上。如果您回答錯誤或回答正確但理解錯誤，請在繼續學習之前花一點時間回顧之前的內容。這裡的對應方式可能涉及很多書中正在討論的相關 Python 功能特性，您需要重新閱讀書本中的素材或連到網路上搜尋觀看其他解釋和範例。

實作演練程式設計。在閱讀時進行預測會有助於鞏固我們對關鍵概念的理解。但要成為熟練的問題解決者和程式設計師，我們還需要進行更多的實作練習。需要實際使用 Python 來解決新的問題，而這些練習問題的解決方案不在書中。每章最後會以「本章習題」當作結尾。請盡可能多多動手實作完成這些習題。

學會程式設計需要花上一點時間。如果進步緩慢或犯了一些錯誤，請不要氣餒。不要被網路上的虛張聲勢或酸言酸語嚇倒。找出能協助自己學習的網友和資源。

使用程式競賽解題系統

我是以國際程式競賽解題系統網站上的題目來建構這本書。程式競賽解題系統網站提供程式競賽的題庫，可以供全世界的程式設計師來競賽解題。可提交您的解答（您的 Python 程式碼）到網站上來執行測試。如果您的程式碼對每個測試用例都能生成正確的答案，那麼您的解決方案是正確的。相反地，如果您的程式碼在某個或多個測試用例生成錯誤的答案，那就表示您的解答程式碼是錯的，需要進行修訂。

我認為競賽解題系統網站很適合學習程式設計有以下幾個原因：

快速反饋。在學習程式設計的早期階段，能快速並有針對性的反饋是十分重要的。當我們提交程式碼後，競賽解題系統會立即提供反饋。

高品質的題庫。我發現程式競賽解題系統的題目品質都很高。許多問題最初來自國際程式競賽所出的題目。很多題目都是由與程式競賽解題系統相關的個人或想幫助他人學習的網友所編寫提供。關於這些題目的來源，請參閱附錄「問題出處與貢獻」的內容。

問題數量。競賽解題系統的題庫含有數百個題目。這本書只選擇了一小部分來解析和說明。如果您需要更多練習，程式競賽解題系統中的題庫可以滿足您的需要。

社群功能。競賽解題系統網站的使用者可以閱讀和回應留言。如果您遇到問題，請瀏覽相關留言來獲取他人的經驗分享。如果還是卡住，請考慮發布問題的留言以尋求協助。成功解開了某個題目，您的學習過程還不算完成！許多程式競賽解題系統允許您觀看他人所提交的程式碼。挖掘一些他人的提交內容，觀看和比較別人與您的解答有何不同。解決問題的方法有很多，也許您現在採用的方法是最直觀的，但學習更多其他可能性是邁向高手的重要一步。

在競賽解題系統的網站上建立帳號

我們在整本書中會使用到幾個競賽解題系統的網站。那是因為不同的程式競賽解題系統會有別人所沒有發掘的題目；我需要多個競賽解題系統的問題來涵蓋本書選用的所有學習主題。

以下是我們所使用的程式競賽解題系統：

解題系統	URL
DMOJ	https://dmoj.ca/
Timus	https://acm.timus.ru/
USACO	http://usaco.org/

競賽解題系統的網站都會要求先建立一個帳戶，然後才能提交解答程式碼。接下來讓我們完成建立帳戶的處理，並在進行時了解關於解題系統的資訊。

DMOJ 競賽解題系統

DMOJ 競賽解題系統是本書中最常使用的程式競賽解題系統。與任何其他程式競賽解題系統相比，很值得您花一些時間瀏覽 DMOJ 網站並了解該系統提供的內容。

若想要在 DMOJ 解題系統上建立帳號，請連到 https://dmoj.ca/ 網站並點按右上角的「**註冊（Sign up）**」連結。在顯現的註冊頁面上，輸入您的使用者名稱、密碼和電子郵件。此頁面還允許設定預設的程式語言（Default language）。我們在本書中專門使用的是 Python 程式語言，因此這裡建議點選「**Python 3**」。完成後點按「**Register!**」鈕建立帳號。註冊之後，您就可以利用使用者名稱和密碼來登錄 DMOJ。

書中的每一道題目都有列出出處是來自哪一個解題系統網站，以及用來存取該題目的編號代碼。舉例來說，我們在第 1 章中的第一個問題是來自 DMOJ 網站，題目編號代碼是 dmopc15c7p2。若想要在 DMOJ 中找到此題目，請點按頁面上方的「**問題（Problems）**」連結，並在右側的「**Problem search**」方塊中輸入「**dmopc15c7p2**」，然後點按「**搜索（Go）**」鈕。您應該會找到唯一的結果。如果點按問題標題，就能進入問題本身的專屬頁面。

當您準備好要提交問題的解答 Python 程式碼時，請先進入該問題專屬頁面並點按右側的「**Submit solution**」鈕。在結果頁面中，把您的 Python 程式碼複製貼上到文字方塊內，然後按下「**提交！**」鈕。隨後系統就會判讀您的程式碼並顯示結果。

Timus 競賽解題系統

若想要在 Timus 解題系統中建立帳號，請連到 https://acm.timus.ru/ 網站並點按「**Register**」連結。在出現的註冊頁面中輸入您的姓名、密碼、電子郵件和其他要求的資訊，點按「Register」鈕來建立您的帳號。隨後請檢查您的電子郵件信箱，找出自 Timus 提供給您的 judge ID 資訊。每當您提交 Python 程式碼時，您都需要用到您的 ID。

目前無法設定預設的程式語言類型，因此無論何時在提交 Python 程式碼，請記得先選擇可用的 Python 3 版本。

本書只在第 6 章中用過 Timus 解題系統，所以就不在此贅述。

USACO 競賽解題系統

若想要在 USACO 解題系統上建立帳號，請連到 http://usaco.org/ 網站並點按「**Register for New Account**」。在出現的註冊頁面中輸入您的使用者名稱、電子郵件和其他要求的資訊。按下「Submit」鈕建立您的帳號。隨後請檢查您的電子郵件信箱中是否有來自 USACO 提供給您密碼的訊息。取得密碼後就可以使用您的使用者名稱和密碼來登錄 USACO 網站。

目前無法設定預設使用的程式語言類型，因此無論何時在提交 Python 程式碼，請記得先選擇可用的 Python 3 版本。此外還需要選擇上傳含有 Python 程式碼的檔案，而不是把程式碼以複製貼上的方式放到文字方塊中。

到本書第 7 章才會用到 USACO 解題系統，所以在這裡就不多贅述。

關於本書

本書各章的內容都是由國際程式競賽解題系統網站的幾個題目來驅動。實際上，在教授新的 Python 語法之前，我都會以問題來當作章節內容的起始！我這樣做的目的是激發讀者去學習解決問題所需的 Python 功能特性。如果您在閱讀題目描述後還是不確定如何解決問題，也請不要擔心（還不能解開問題，表示您閱讀本書是正確的選擇）。理解了問題的要求，就已經離答案不遠了。接著學會 Python，一起把問題解決掉。章節後續的問題可能會介紹更多的 Python 功能特性，或要求我們延伸前面問題所學到的知識。每章最後都有習題，請動手解開習題，試一試前面所學到的東西。

以下是本書各章的內容簡述：

第 1 章：入門起步。在使用 Python 解決問題之前，我們需要先學會一些基本必要的概念。本章內容會講述這些概念，包括輸入 Python 程式碼、字串和數值的處理、變數的使用、讀入和寫出。

第 2 章：作出決定。在本章將會學習 if 陳述句的使用，此語法會讓程式根據特定條件是 True 或 False 來決定接下來要做什麼。

第 3 章：重複執行：有確定次數的迴圈。只要還有工作要處理程式就會持續執行。在本章中，我們會學習 for 迴圈，讓程式處理各個輸入的內容，直到工作完成。

第 4 章：重複執行：不確定次數的迴圈。有時候我們事先並不知道程式應該重複某些特定動作的次數。for 迴圈就不適用於這類問題。在本章中，我們將學習 while 迴圈，只要特定條件為 True，迴圈中的程式碼就會一直執行。

第 5 章：使用串列來管理多個值。Python 串列讓我們只用一個名稱就能引用整組資料序列。使用串列能幫助我們組織管理資料，可以配合 Python 提供的強大的串列操控功能（例如排序和搜尋）來處理資料。本章會講述關於串列的所有內容。

第 6 章：使用函式來設計程式。如果我們不能好好地組織管理含有大量程式碼的大型程式，則這支程式可能會變得笨拙而冗長。在本章中，我們會學習函式，它能幫助我們設計劃分出小型、內含程式碼區塊所組成的程式。使用函式能讓程式更易理解和修改。我們還會學到「由上而下（top-down）」的設計概念，這種設計方式會讓程式中使用函式。

第 7 章：檔案的讀取和寫入。檔案便於對程式提供資料，而程式處理的資料也能很容易地存入檔案中。在本章中，我們將學到如何從檔案讀取資料以及怎麼把資料寫入檔案。

第 8 章：使用集合與字典來管理多個值。當我們開始要處理越來越有挑戰性的問題時，考量資料如何儲存是很重要的關鍵點。在本章中，我們會學習兩種 Python 儲存資料的新方式：集合與字典。

第 9 章：使用完全搜尋法來設計演算法。程式設計師不會急著從頭開始解決每個問題。相反地，他們會思考是否有通用的解決方案模式來解開問題，例如使用某個現成的演算法來解決它。在本章中，我們會學習用來解決各種問題的完全搜尋演算法。

第 10 章：大 O 符號與程式效能。有時候我們設計和編寫出的某支程式，它雖然可以正確完成工作，但因為速度太慢而不實用。在本章中，我們將學習怎麼改善程式的效能，學會使用能設計出更高效率程式碼的工具。

第 1 章
入門起步

程式設計所牽涉的就是設計和寫出程式碼來解決問題。此，我會與您一起走過解決問題的所有過程。也就是說，我們不是學習一個又一個的 Python 功能和概念之後再來解決問題，而是利用問題來決定需要學習什麼樣的知識和概念。

在本章中，我們會處理兩個問題：確定一行中的單字的數量（類似文書處理軟體中的字數統計功能）和計算圓錐體的體積。解開這兩個問題需要了解相當多的 Python 知識和概念。您可能覺得需要更多詳細資訊才能完全理解我在這裡介紹的某些內容，以及怎麼與 Python 程式設計的過程整合在一起。請別擔心：我們會在後面的章節內容中重新回顧和闡述最重要的知識和概念。

我們要做什麼

如簡介中所介紹的，我們會使用 Python 語言來解開國際程式競賽的題目。國際程式競賽的問題都能在競賽解題系統的網站上找到。我假設讀者已經按照「簡介」章節中的說明內容，安裝好 Python 並建立了解題系統網站的帳號。

對於書中的每個問題，我們都會設計寫出程式來解決它。問題的要求中會指定程式所提供的輸入類型，以及預期的輸出（或結果）類型。如果程式能接受任何合法的輸入並產生正確的輸出結果，就表示這支程式能正確解決問題。

一般來說，輸入內容的可能性有數百萬或數十億種。每個這樣的輸入被稱為一個**問題實例**。舉例來說，在我們要解決的第一個問題中，要求是輸入是一行文字，像 hello there 或 bbaabbb aaabab 之類。我們的工作是輸出這行文字中的單字數量。程式設計中最強大的思維之一是，利用少量通用的程式碼就可以來解決看似無窮無盡的問題實例。一行文字中是 2 個還是 3 個或 50 個單字都不影響程式的運作。我們所設計的程式每次都能處理得很好。

這支程式要處理下列三件工作：

讀取輸入：我們需要確定正要解決問題的具體實例，因此先讀取提供的輸入內容。

處理：處理輸入的內容來找出正確的輸出結果。

寫入輸出：解決問題之後要生成需要的輸出內容。

這些步驟之間的界限並不一定分得那麼清楚，舉例來說，我們可能一邊處理一邊產生一些輸出，步驟有可能是交錯在一起的，但請記住整支程式離不開這三個主要步驟。

您可能一直都在使用遵循著「輸入－處理－輸出」模式的程式。請想一下電腦程式的處理過程：輸入公式（輸入），程式計算數值（處理），然後程式顯示答案（輸出）。或者請思考某個網路搜尋引擎的處理過程：輸入要查詢的內容（輸入），搜尋引擎確定最相關的結果（處理），並顯示這些結果（輸出）。

將這類程式與互動式程式進行對比，它們都有輸入、處理和輸出的過程。舉例來說，我正在使用文字編輯器輸入這本書的內容。當我輸入一個字元時，編輯器會把這個字元新增到我的文件中。它不會等我輸入整份文件才顯示；在我建構時就以互動式顯示出輸入的字元。我們不會在本書中設計和編寫互動式程式。如果您學習了這本書的內容後，對編寫這樣的程式感興趣，您會很高興地發現 Python 真的能勝任這項工作。

每個問題的陳述說明都可以在書中和線上的解題系統網站內找到。但是，陳述說明文字不完全相同，因為我為了整本書的一致性而作了一些修飾。請別擔心：我所改寫的內容與官方問題陳述所傳達的資訊是相同的。

Python Shell 模式

對於書中的每個問題都會編寫一支程式並儲存成檔案，但前提是我們要知道設計編寫什麼樣的程式！對於本書中的許多問題，在解決之前，我們還需要學習一些 Python 的新功能特性。

在 Python shell 模式中試驗 Python 功能是最好的選擇。這是個互動式的環境，您可以在這個模式中鍵入一些 Python 程式碼並按 ENTER 鍵來執行，Python 會顯示執行結果。一旦我們學會了足夠的知識來解決目前的問題，就可以停用 shell 模式並開始在文字檔中輸入解決方案的程式碼。

一開始請在電腦系統的桌面上建立一個名為「programming」的新資料夾。我們會使用這個資料夾來存放書中所完成的所有範例程式檔。

現在請切換這個「programming」資料夾並啟動 Python shell 模式。若想要啟動 Python shell，可依照您的作業系統執行以下步驟。

Windows

若是在 Windows 系統中，請依照下列步驟來操作：

1. 按住 SHIFT 鍵後，再以滑鼠右鍵點按「**programming**」資料夾。
2. 在展開的功能中點選「**在此處開啟 PowerShell 視窗**」，如果沒有這個命令選項，請點選「**在此處開啟命令視窗**」。

3. 接著在顯示的視窗底端會出現大放（>）符號和跳動的游標。這是作業系統的輸入提示字元，表示等待您輸入命令。請輸入作業系統的命令而不是 Python 程式碼。在輸入命令後記得按 ENTER 鍵執行。

4. 在開啟的視窗中已進入「programming」資料夾，可輸入 dir 命令來查看目前所在目錄路徑的內容。

5. 現在輸入 **python** 來啟動 Python shell 模式。

當我們啟動了 Python shell 模式後，會顯示如下的訊息：

```
Python 3.9.2 (tags/v3.9.2:1a79785, Feb 19 2021, 13:30:23)
[MSC v.1928 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

這裡的訊息中最重要的是第一行所顯示的至少是 3.6 以後的 Python 版本。如果您使用的是過舊的版本，尤其是 2.x，或者根本無法載入 Python，請按照前面「簡介」章節中的安裝說明，先下載安裝最新版本的 Python。

在此視窗的底端，您將看到 >>> 這個 Python 提示符號。這是鍵入 Python 程式碼的地方。不用自己鍵入 >>> 符號，而是在符號後面輸入程式碼。在試驗完成之後，可以按 CTRL-Z 鍵，再按 ENTER 鍵退出這個 shell 模式。

macOS

在 macOS 系統中，請依照下列步驟來操作：

1. 請開啟「終端機」。可按下 COMMAND-空白鍵，再輸入「**終端機**」或「**terminal**」，然後連按下二下這個結果。

2. 在開啟的視窗中會看到一行以金錢符號（$）結尾的提示行。這就是作業系統的提示符號，表示等待您輸入命令。請輸入作業系統的命令而不是 Python 程式碼。在輸入命令後記得按 ENTER 鍵執行。

3. 輸入「**ls**」命令來查看目前所在目錄路徑的內容。您的 Desktop 目錄會顯示在這裡。

4. 輸入「**cd Desktop**」命令切換到 Desktop 資料夾。cd 命令是 change directory 的意思，directory（目錄）的另一個別稱是 folder（資料夾）。

5. 輸入「**cd programming**」命令切換到 programming 資料夾。

6. 現在輸入「**python3**」啟動 Python shell 模式（您也可以只輸入 **python** 而沒有加 3，這樣啟動的是 python 2。但這個舊版本並不適用於本書的內容）。

當 Python shell 模式啟動後會看到如下訊息：

```
Python 3.9.2 (default, Mar 15 2021, 17:23:44)
[Clang 11.0.0 (clang-1100.0.33.17)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

這裡的訊息中最重要的是第一行所顯示的至少是 3.6 以後的 Python 版本。如果您使用的是過舊的版本，尤其是 2.x，或者根本無法載入 Python，請按照前面「簡介」章節中的安裝說明，先下載安裝最新版本的 Python。

在此視窗的底端，您將看到 `>>>` 這個 Python 提示符號。這是鍵入 Python 程式碼的地方。不用自己鍵入 `>>>` 符號，而是在符號後面的完成程式碼。在試驗完成之後，可以按 CTRL-D 鍵，再按 ENTER 鍵退出這個 shell 模式。

Linux

在 Linux 系統中，請依照下列步驟來操作：

1. 請以滑鼠右鍵點按「programming」資料夾。

2. 在展開的功能表中點選「**Open in Terminal**」。（您也可以直接開啟 Terminal 後再切換到 programming 資料夾中）

3. 在開啟的視窗中會看到一行以金錢符號（$）結尾的提示行。這就是作業系統的提示符號，表示等待您輸入命令。請輸入作業系統的命令而不是 Python 程式碼。在輸入命令後記得按 ENTER 鍵執行。

4. 現在應該已在 programming 資料夾中，可輸入「**ls**」命令來查看目前所在路徑的內容。您應該看不到任何內容，因為我們還沒有建立任何檔案。

5. 現在輸入「**python3**」啟動 Python shell 模式（若只輸入 **python** 而沒有加 3，這樣啟動的是 python 2。但這個舊版本並不適用於本書的內容）。

當 Python shell 模式啟動後會看到如下訊息：

```
Python 3.9.2 (default, Feb 20 2021, 20:57:50)
[GCC 7.5.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

這裡的訊息中最重要的是第一行所顯示的至少是 3.6 以後的 Python 版本。如果您使用的是過舊的版本，尤其是 2.x，或者根本無法載入 Python，請按照前面「簡介」章節中的安裝說明，先下載安裝最新版本的 Python。

在此視窗的底端，您將看到 >>> 這個 Python 提示符號。這是鍵入 Python 程式碼的地方。不用自己鍵入 >>> 符號，而是在符號後面的完成程式碼。在試驗完成之後，可以按 CTRL-D 鍵，再按 ENTER 鍵退出這個 shell 模式。

問題#1：計算字數（Word Count）

現在是處理第一個問題的時候了！我們會使用 Python 設計編寫一支小型的計算字數程式。我們將學習如何從使用者那裡讀取輸入內容，再處理輸入內容來解開問題，並輸出結果。我們還要學習如何在程式中處理文字和數值、活用內建的 Python 運算子，以及在解決方案的過程中儲存中間結果。

這個問題在 DMOJ 網站的題庫編號為 dmopc15c7p2。

挑戰

計算提供的英文單字數量。這個問題所指的英文單字是指任何小寫字母所組成的文字序列。例如，hello 是一個單字，但像 bbaabbb 這樣的無意義的英文也算一個單字。

輸入

輸入的內容是一行文字，由小寫字母和空格組成。每對單字之間是以空格分開，第一個單字之前和最後一個單字之後是沒有空格的。

一行的最大長度為 80 個字元。

輸出

輸出的內容為算出輸入行有多少個單字的數量。

字串

值（value）是 Python 程式的基本組成區塊。每個值都有一個**型別（type）**，型別決定了這個值可以執行的操作和處理。在計算單字數量的問題中，我們處理的是一行文字。文字在 Python 中儲存為字串值，因此需要了解**字串（string）**的相關知識。為了解開這個問題，計算出文字中的單字數量，我們還需要學習數值（numeric value）。首先讓我們從字串開始。

字串的表示

字串是用來儲存和處理文字的 Python 型別。若想要寫出一個字串值，需要以單引號括住字元。請在 Python shell 中進行下列的操作：

```
>>> 'hello'
'hello'
>>> 'a bunch of words'
'a bunch of words'
```

Python shell 會回應我們輸入的字串。

若字串中含有單引號這個字元時會發生什麼狀況呢？

```
>>> 'don't say that'
File "<stdin>", line 1
'don't say that'
     ^
SyntaxError: invalid syntax
```

「don't」內的單引號 ' 變成字串左側的結尾符號，也就是「'don'」被視為一個字串，但這行文字的其餘部分「t say that'」就變得沒有意義，這就是產生語法錯誤的原因。**語法錯誤（syntax error）**是指違反了 Python 的語法規則，表示我們設計寫出不合法的 Python 程式碼。

若想要修正這個問題，可以用雙引號 " 來括住字串，因為雙引號也可以當作括住字串的符號：

```
>>> "don't say that"
"don't say that"
```

本書的慣例是以單引號 ' 來括住字串，除非題目中的字串內有用到單引號，否則不會在書中使用雙引號來表示字串。

字串運算子

我們可以使用一個字串來存放想要計算其單字數量的文字。若想要計算單字數量或者用字串處理任何其他事情，我們都需要學習運用字串的相關處理。

字串有很多可以搭配使用的運算子。有一些是在運算元之間使用特殊的符號來表示。例如，+ 運算子可用來處理字串的連接：

```
>>> 'hello' + 'there'
'hellothere'
```

哇！忘了在兩個單字之間加上空格了。讓我們在第一個字串的尾端加上空格：

```
>>> 'hello ' + 'there'
'hello there'
```

另外還有一個 * 運算子，可以讓字串重複很多次：

```
>>> '-' * 30
'------------------------------'
```

上述實例中的 30 是個整數值，稍後我會說明更多關於「整數」的內容。

觀念檢測

下列這行程式碼的執行結果為何？

_____

```
>>> '' * 3
```

_____

A. ''''''
B. ''
C. 顯示語法錯誤（不合法的 Python 程式碼）

答案：B。'' 是個空字串，也還是字串中沒有字元。這題是讓空字串重複 3 次，結果還是空字串。

字串方法

方法（**method**）是特定於某種型別的值才有的操作處理。字串有很多方法可用。例如，upper 是個很常用的方法，它能讓英文字串變成大寫的版本：

```
>>> 'hello'.upper()
'HELLO'
```

我們從方法中取得的資訊稱為方法的**返回值**（**return value**）。舉例來說，以前面的範例來說，upper 方法會返回字串 'HELLO'。

對某個值執行方法的這個操作稱為「呼叫方法」。呼叫方法是在值和方法名稱之間要放上句號運算子（.），它還需要在方法名稱後面加上括號。對於某些方法，括號中是空的，就像呼叫 upper 方法的操作處理。

但有些方法，則可能需要在括號中放入某些資訊。若沒有放入必需的資訊，這些方法就根本無法運作。我們在呼叫方法時所引入的資訊稱為方法的**引數**（**argument**）。

舉例來說，字串中有個 strip 方法，在呼叫時若括號沒有放入引數，strip 方法就會把要處理的字串中前後的空格都去掉：

```
>>> '   abc'.strip()
'abc'
>>> '   abc       '.strip()
'abc'
>>> 'abc'.strip()
'abc'
```

但我們也可以搭配字串引數來呼叫。如果這樣做，這個字串引數就是用來決定要刪除哪些字元的：

```
>>> 'abc'.strip('a')
'bc'
>>> 'abca'.strip('a')
'bc'
>>> 'abca'.strip('ac')
'b'
```

接著再介紹另一個字串方法：count。如果傳入一個字串引數，則 count 會告知要處理的字串中有多少個符合該字串引數的數量：

```
>>> 'abc'.count('a')
1
>>> 'abc'.count('q')
0
>>> 'aaabcaa'.count('a')
5
>>> 'aaabcaa'.count('ab')
1
```

如果出現字串引數重疊的情況，則只算一個計數：

```
>>> 'ababa'.count('aba')
1
```

與其他方法不同，這個 count 方法對我們的「計算字數（Word Count）」問題能直接發揮作用。

假設有個「'this is a string with a few words'」這樣的字串。請留意，這裡的每個單字後面都有個空格。事實上，如果我們必須手動計算單字的數量，那就會知道每個單字的結尾位置有個空格。如果我們計算字串中的有多少個空格數量會是怎麼樣的情況呢？我們可以傳入單個空格字元的字串到 count 方法來讓它計數。這個範例看起來像下列這般：

```
>>> 'this is a string with a few words'.count(' ')
7
```

我們得到的計數結果為 7。這不是我們要計算出的正確單字數量（因為這個範例中的字串有 8 個單字），但答案已經很接近囉。為什麼我們得到 7 而不是 8 呢？

原因是上面的範例字串中，除了最後一個單字之外，每個單字後面都有一個空格。因此，計算空格時少算了最後的單字。為了解決這個問題，我們還需要學習如何處理數值資料。