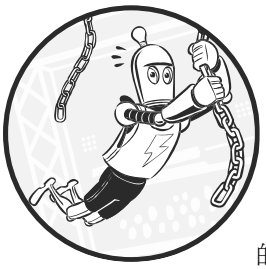
簡介



哈囉，大家好！還記得在 1990 年代，我還是個年少的程式設計師，夢想著要成為駭客，當時被「2600:The Hacker Quarterly」駭客季刊最新一期的主題深深吸引。有一天，我終於鼓起勇氣參加了該雜誌每個月在我居住城市所舉辦的定期聚會，那時對其他看起來很有學問的參加者感到敬畏（後來才發現那些人大多並沒有那麼厲害）。在聚會中我只能當個聽眾，對談時只能一直點頭稱是。離開聚會後，我決定投入更多時間來研究電腦、程式設計和網路安全等主題，好讓我在參加下個月的聚會時能參與討論和對話。

在下一次聚會中，與其他人相比，我還是只能繼續點頭稱是，覺得自己還是不足。因此，我再次下定決心學習，讓自己變得「足夠聰明」，好追上別人的步伐，這樣一個月又一個月的累積，我雖增長了許多的知識，但開始意識到電腦相關領域的龐大，擔心自己學到的不夠多。

我雖然比同期高中時代的朋友在程式設計的知識上要了解得多，但我還不能勝任軟體開發的工作。在 1990 年代，Google、YouTube 和維基百科都還沒出現。

但就算有這些資源，我也不知道怎麼使用；也不確定接下來要學些什麼內容。我學會如何用不同的程式語言編寫「Hello, world!」程式，但仍覺得自己沒有什麼真正的進步。我不知道學會基礎知識後的下一步該怎麼辦？

軟體開發的知識中除了迴圈和函式外，還有很多要學的。在我們完成了入門課程或讀完程式設計入門書後，找到的其他指南怎麼還是只會引領自己到另一個「Hello, world!」的課程。程式設計師通常把這段時期稱為絕望沙漠期（desert of despair）：我們花了不少時間漫無目的地遊蕩於不同的學習材料上，但感覺自己並沒有進步。雖然比初學者強很多，但對複雜的主題卻又沒有什麼經驗。

在絕望沙漠期中的人會有強烈的冒名頂替症候（Impostor syndrome）。您不會覺得自己是個「真正的」程式設計師，也不知道怎麼樣才能像「真正的」程式設計師那樣設計編寫程式碼。我寫這本書的目的就是針對這樣的讀者群，如果您已學過 Python 的基礎知識，那麼本書應該能幫助您成為更有能力的軟體開發人員，並擺脫這種絕望的感覺。

本書的適用對象和特點

本書的目標是針對那些已學過 Python 基礎課程，然後還想要學會更多知識的讀者。這些基礎課程像是我所寫的前一本書「Python 自動化的樂趣：搞定重複瑣碎&單調無聊的工作－第二版（2020/08 碁峰出版）」，或是 Eric Matthes 所著的「Python 程式設計的樂趣：範例實作與專題研究的 20 堂程式設計課－第二版（2020/04 碁峰出版）」，又或是其他的線上課程。

上述的這些課程教材可能會讓您迷上程式設計，但是您仍需要學習更多的技能。如果您覺得自己還不算是專業的程式設計師，但又不知道怎麼達到這樣的水準，那麼本書就是您最理想的選擇。

或者，您也許是從 Python 以外的另一種程式語言來學習程式設計的，而您想直接進入 Python 及其工具相關生態系統，不想要重讀相同的「Hello, world!」基礎知識。如果是這樣，則無須重讀數百頁的基本語法內容；請直接瀏覽 https://learnxinyminutes.com/docs/python/ 中「Learn Python in Y Minutes」，或是 Eric Matthes 網站 https://ehmatthes.github.io/pcc/cheatsheets/README.html 上的

「Python Crash Course—Cheat Sheet」頁面，這樣就足以掌握 Python 基礎，讓您能進入本書更進一步的應用。

關於本書

本書所涵蓋的內容不僅是更深入進階的 Python 語法，還說明怎麼使用命令提示模式和專業開發人員使用的命令行工具，例如 code formatters（程式碼格式化工具）、linter 和版本控制。書中解釋了怎麼讓程式碼更有可讀性，以及如何編寫出乾淨的程式碼（clean code）。本書精選了一些程式設計專題，因此讀者可以從中學到怎麼把這些原理應用到實際的軟體中。雖然本書並不定位為電腦科學的教科書，但還是有解說大 O 演算法分析和物件導向設計等內容。

光靠單一本書的內容很難讓某個人直接轉變成專業的軟體開發人員，但本書能提升您的知識，讓您向這個目標邁進。書中介紹的幾個主題包含了筆者所累積的難得經驗，是讀者很難發掘且零碎的知識點。讀完本書後，您的根基將更為穩固，因此更有能力應對新的挑戰。

我建議讀者按章節順序閱讀本書的內容，但讀者其實可以隨時跳到您感興趣的任何一章節主題：

Part I：起程

Chapter 1：處理錯誤和尋求協助　本章介紹如何有效率地提出問題與找出答案。也講解怎麼閱讀錯誤訊息以及在網路上尋求協助應有的禮節。

Chapter 2：環境設定和命令提示模式　本章解釋怎麼駕馭命令提示模式、設定開發環境與 PATH 環境變數。

Part 2：最佳實務、工具和技能

Chapter 3：使用 Black 進行程式碼格式化　本章介紹 PEP 8 風格指南以及如何格式化程式碼，讓它們更具可讀性。讀者將學會如何使用 Black 程式碼格式化工具來進行自動化處理。

Chapter 4：選用易懂的命名　本章講述怎麼為變數和函式選用好的名稱，讓程式碼更好讀易懂。

Chapter 5：找出程式碼的異味（Code smell）　本章列出了在程式碼中存在錯誤所潛藏的幾個危險信號。

Chapter 6：寫出 Pythonic 風格的程式碼　本章詳細介紹了幾種編寫出慣用 Python 程式碼的方法，以及寫出 Pythonic 風格有何影響。

Chapter 7：程式設計的行話　本章介紹在程式設計領域所用的技術用語，以及怎麼辨識會讓大家混淆的用語。

Chapter 8：常見的 Python 誤解和陷阱　本章解說 Python 語言中常見的混淆和錯誤來源，並說明如何修正及避免的編寫策略。

Chapter 9：少為人知的 Python 奇異之處　本章介紹一些大家可能不會注意到的 Python 奇異之處，例如字串駐留和 antigravity 彩蛋。藉由弄清楚為何某些資料型別和運算子會導致某種意外情況，我們對 Python 的運作原理會有更深入的了解。

Chapter 10：寫出有效率的函式　本章詳細說明如何建構出擁有最大實用性和可讀性的函式。讀者會學到 * 和 ** 引數語法、怎麼權衡大型和小型函式、以及像 lambda 等函式語言程式設計的技術。

Chapter 11：注釋、文件字串和型別提示　本章介紹了程式碼中非程式部分的重要性，以及這些內容怎麼影響程式的可維護性。本章談到了應該要多久編寫一次注釋（comments）和文件字串（docstrings），以及怎麼讓這些內容具備傳達資訊的特質。本章還討論了型別提示（type hints）以及如何使用像 Mypy 這類靜態分析工具來檢測錯誤。

Chapter 12：使用 Git 來組織程式碼專案　本章說明使用 Git 版本控制工具來記錄原始程式碼的修改變更歷史記錄、備份復原先前版本，或追蹤首次出現錯誤的時間，還介紹了如何使用 Cookiecutter 工具來組建程式碼專案的檔案。

Chapter 13：評測效能和大 O 演算法分析　本章講解如何使用 timeit 和 cProfile 模組客觀地評測程式碼的速度。此外還介紹了大 O 演算法分析以及如何預測隨著要處理資料量的增長而讓程式碼效能下降的原因。

Chapter 14：實務專案　本章透過編寫兩個命令提示模式遊戲來運用在這個部分中所學到的技術。第一個是河內塔，把盤子從某個塔柱移動到另一個塔柱的益智遊戲；第二個是兩位玩家對奕的經典四子棋遊戲。

Part 3：物件導向的 Python

Chapter 15：物件導向程式設計與類別　本章定義物件導向程式設計（OOP）的角色和作用，因為這項技術經常被人誤用。許多開發人員過度使用 OOP 技術，緣由是他們覺得別人也都這樣做，但這樣會導致原始程式碼太過複雜。本章將教讀者如何設計編寫類別（class），但更重要的是教讀者不應該使用類別的原因。

Chapter 16：物件導向程式設計與繼承　本章解釋類別繼承及其在程式碼重複使用的實用性。

Chapter 17：Pythonic 風格的 OOP－property 與 dunder 方法　本章介紹了在物件導向程式設計中特別屬於 Python 的功能，例如 property、dunder 方法和運算子多載。

您的程式設計旅程

從菜鳥到程式老手的過程有點像是想要從消防栓來喝水，有太多可選擇的資源要學，您可能會擔心自己是否浪費時間在次優的程式設計指南上。

閱讀完本書後（或是在閱讀本書的同時），我建議讀者藉由閱讀以下的資料來當作補充教材：

- *Python Crash Course* (No Starch Press, 2019)，作者為 Eric Matthes，中文版書名為「Python 程式設計的樂趣：範例實作與專題研究的 20 堂程式設計課」（碁峰資訊，2020），是一本入門的專書，但書中有專題導向的章節，對程式老手也很有幫助，其中 Python 的 Pygame、matplotlib 和 Django 等都有專案主題解說。

- *Impractical Python Projects* (No Starch Press, 2018)，作者為 Lee Vaughan，這本是以專案為基礎的寫作方式來讓讀者學習 Python 技巧。跟著書中指引來寫出程式是很有趣的，同時也是很棒的實務練習。
- *Serious Python* (No Starch Press, 2018)，作者為 Julien Danjou，本書講述了從個人專案研發的愛好者變成軟體開發高手所需要採取的步驟，這本書所講述的軟體開發高手是指能遵循業界最佳實務作法，並編寫出可擴充程式碼的專業人員。

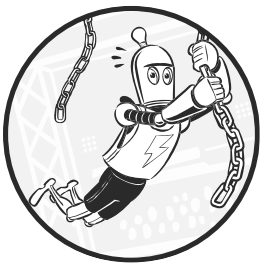
Python 的技術層面只是其優勢之一，這套程式語言吸引了各式各樣的社群，他們負責建立許多友善、可存取的文件和支援，這是其他程式語言生態系統所無法比擬的。一年一度的 PyCon 研討會以及各地區域性的 PyCons 研討會舉辦了適合各種程度的講座。PyCon 主辦單位讓這些講座也可從 https://pyvideo.org/ 網站免費線上觀看，網頁上各標籤連結能讓我們輕鬆找到感興趣的話題所對應的講座。

為了能更深入地了解 Python 語法和標準程式庫的進階功能，我建議大家閱讀以下好書：

- *Effective Python* (Addison-Wesley Professional, 2019)，作者為 Brett Slatkin，這是一本令人印象深刻的 Python 式最佳實務和語言特性的彙整集合。
- *Python Cookbook* (O'Reilly Media, 2013)，作者為 David Beazley and Brian K.Jones，此書提供了很全面的程式碼片段，能提升 Python 新手的能力。
- *Fluent Python* (O'Reilly Media, 2021)，作者為 Luciano Ramalho，這本書是深入探索 Python 語言各種功能的傑作，雖然近 800 頁的厚度可能令人害怕，但很值得在這本書上花點功夫。

祝福您的程式設計旅程一切順利，我們起程吧！

第 1 章
處理錯誤和尋求協助



請不要把電腦太過擬人化，這樣會讓您很傷腦筋。當電腦對您顯示錯誤訊息時，並不是因為您冒犯了它，電腦是我們大多數人都會與之互動的複雜工具，但是它就只是工具而已。

即便如此，我們還是很容易錯怪這些工具，因為很多程式的知識都是需要自我導向學習的，就算您已學過 Python 很多個月了，您可能還是需要每天多次查詢網路求解，感到失敗沮喪也很正常。即使軟體開發專家也還都需要在網路上搜尋或查閱相關文件來解決他們可能碰到的程式設計問題。

除非您經濟或資源上許可，能聘請私人老師來回答您的程式設計問題，不然您是擺脫不了電腦和網路搜尋引擎，需要自己堅強面對。幸運的是，您遇到的問題網路可能早就有人問過。身為一位程式設計師，能夠自己找答案這項能力比學習演算法或資料結構知識更為重要。本章將會指導您開發這項關鍵技能。

怎樣了解 Python 錯誤訊息？

在遇到錯誤訊息顯現一大片含糊的技術文字時，大多數程式設計師的第一個直覺和衝動就是完全忽略它，但是在這些錯誤訊息中卻有程式出了什麼問題的解答。找到答案的過程分為兩步：檢查 traceback 內容和在網路上搜尋錯誤訊息。

檢查 traceback 內容

當程式碼引發一個 except 陳述句無法處理的例外時，Python 程式就會停掉。發生這種情況時，Python 顯示該例外的訊息和一個 **traceback 回溯內容**，這也稱為**堆疊追蹤（stack trace）**。這個 traceback 內容顯示了程式中發生例外的位置，以及函式呼叫的軌跡。

為了能實際體會閱讀 traceback 內容的過程，請輸入如下有錯誤的程式碼，並將它儲存成 abcTraceback.py 檔。程式左側的行號僅提供參考使用，不是程式碼的內容：

```
   1. def a():
   2.     print('Start of a()')
❶  3.     b()  # Call b().
   4.
   5. def b():
   6.     print('Start of b()')
❷  7.     c()  # Call c().
   8.
   9. def c():
  10.     print('Start of c()')
❸ 11.     42 / 0  # This will cause a zero divide error.
  12.
  13. a()  # Call a().
```

在這段程式中，a() 函式呼叫了 b()，而 b() 呼叫了 c()。在 c() 裡面，「42 / 0」表示式會引發除以 0 的錯誤。當我們執行這支程式時，輸出結果會是：

```
Start of a()
Start of b()
Start of c()
Traceback (most recent call last):
  File "abcTraceback.py", line 13, in <module>
    a()  # Call a().
  File "abcTraceback.py", line 3, in a
    b()  # Call b().
```

```
  File "abcTraceback.py", line 7, in b
    c()  # Call c().
  File "abcTraceback.py", line 11, in c
    42 / 0  # This will cause a zero divide error.
ZeroDivisionError: division by zero
```

讓我們逐行查看 traceback 內容，第一行是：

```
Traceback (most recent call last):
```

這行訊息告知我們以下是 traceback 的內容，「most recent call last」指的是每個函式呼叫會依序列出，由第一個函式呼叫開始到最後呼叫的函式結尾。

接下來是 traceback 內容的第一個函式呼叫：

```
  File "abcTraceback.py", line 13, in <module>
    a()  # Call a().
```

這兩行是**框架摘要（frame summary）**，顯示框架物件內部的資訊，當呼叫函式時，區域變數的資料以及函式呼叫結束後要返回的程式碼是儲存在**框架物件（frame object）**中。框架物件儲存了區域變數和與函式呼叫相關的其他資料，框架物件是在呼叫函式時建立，並在函式返回時銷毀。traceback 內容會顯示引起程式崩掉的每一個相關的框架摘要。我們可以看到它列出 abcTraceback.py 第 13 行的這個函式呼叫，而且 <module> 告知我們此行是位於全域範圍內，第 13 行程式內容的顯示有內縮兩個空格。

後續 4 行是接下來的 2 個框架摘要：

```
  File "abcTraceback.py", line 3, in a
    b()  # Call b().
  File "abcTraceback.py", line 7, in b
    c()  # Call c().
```

「line 3, in a」文字告知我們在 a() 函式第 3 行呼叫了 b()，導致在 b() 函式的第 7 行又呼叫了 c()。請留意，在第 2、6 和 10 行上的 print() 呼叫不會顯示在 traceback 內容中，就算它在函式呼叫之前就執行了也一樣不會顯示。traceback 內容中僅顯示含有引發例外的函式呼叫內容。

最後一個框架摘要顯示了導致無法處理例外所在的那一行，緊接著是函式的名稱和例外訊息：

```
  File "abcTraceback.py", line 11, in c
    42 / 0  # This will cause a zero divide error.
ZeroDivisionError: division by zero
```

請注意，traceback 列出的行號是 Python 最終檢測到錯誤的地方，這個錯誤的真正源頭有可能在該行之前。

錯誤訊息很明顯而簡短，且不好理解：「division by zero」三個字對您來說沒有什麼意義，除非您知道數學上某個數除以零是不可能的，這也是一個很常見的軟體錯誤。查看框架摘要中的程式碼行，可以清楚地看到「42 / 0」這行程式碼所引發的除以 0 錯誤。

接著讓我們看一個更難察覺的情況。請在文字編輯器中輸入以下程式碼，並將其儲存成 zeroDivideTraceback.py 檔：

```
def spam(number1, number2):
    return number1 / (number2 - 42)

spam(101, 42)
```

當您執行程式，其輸出結果如下：

```
Traceback (most recent call last):
  File "zeroDivideTraceback.py", line 4, in <module>
    spam(101, 42)
  File "zeroDivideTraceback.py", line 2, in spam
    return number1 / (number2 - 42)
ZeroDivisionError: division by zero
```

錯誤訊息是相同的，但是「return number1 / (number2 - 42)」的除以零錯誤卻不是很明顯。我們推斷出 / 運算子發生了除法運算，而且表示式「(number2 - 42)」必須計算為 0。這樣會引導您得出結論，只要將 number2 參數設為 42，spam() 函式就會發生錯誤。

有時候 traceback 內容可能會指出錯誤真正原因所在後面的程式行發生錯誤，例如，在以下程式中，第 1 行少了右括號：

```
print('Hello.'
print('How are you?')
```

當錯誤訊息卻指出第 2 行有問題：

```
  File "example.py", line 2
    print('How are you?')
        ^
SyntaxError: invalid syntax
```

原因是 Python 直譯器在讀取到第 2 行才注意到語法錯誤。traceback 內容可以指出問題出在哪裡，但這並不一定是錯誤實際發生原因的所在。如果框架摘要中沒有提供足夠的訊息讓您找出錯誤，或者如果錯誤的真正原因在 traceback 所列的上一行，並沒有顯示出現，則必須使用 debugger 逐步檢查程式，或是檢查所有日誌記錄訊息來找出原因，這樣會很花時間。在網路上搜尋這條錯誤訊息可能會更快地幫您找出有關解決方案的關鍵線索。

搜尋錯誤訊息

錯誤訊息通常很簡短，甚至不是完整的句子。由於程式設計師經常遇到這些錯誤訊息，因此，這些簡短訊息的目的只是提醒而不是給與完整的解釋。如果遇到沒看過的錯誤訊息，可將其複製並貼上到網路搜尋引擎中搜尋，通常都能找到很詳細的說明、列出錯誤的含意以及引發的可能原因。圖 1-1 秀出搜尋引擎搜尋「python "ZeroDivisionError: division by zero"」的結果。在錯誤訊息左右加上引號有助於找到最確切的詞組，加上 python 這個單字也能縮小搜尋範圍。

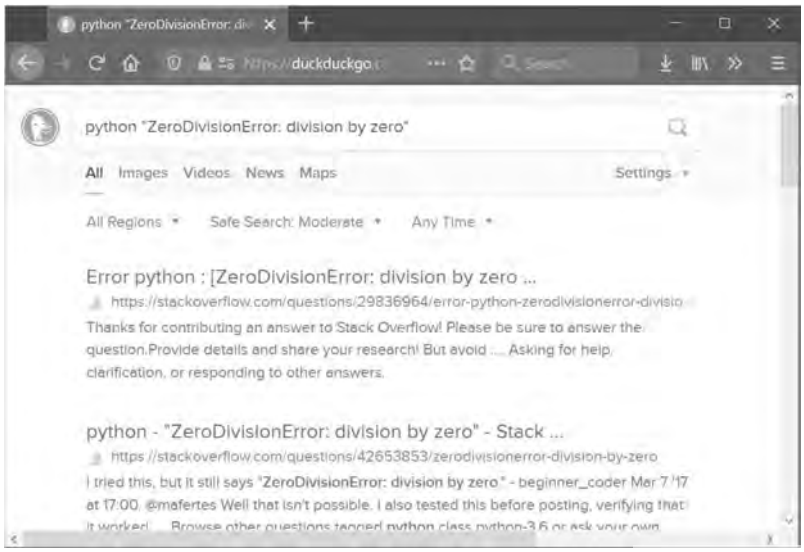


圖 1-1　把錯誤訊息複製貼上到網路中搜尋能找出相關解釋與解決方案

搜尋錯誤訊息並不是什麼投機的技巧，沒有人能記住某種程式語言中的各種可能的錯誤訊息，軟體開發專家也會每天在網路上搜尋程式相關的解答。

搜尋時可能要排除錯誤訊息中特定於程式碼的部分，舉例來說，請看下列錯誤訊息：

```
    >>> print(employeRecord)
        Traceback (most recent call last):
          File "<stdin>", line 1, in <module>
❶   NameError: name 'employeRecord' is not defined
    >>> 42 - 'hello'
        Traceback (most recent call last):
          File "<stdin>", line 1, in <module>
❷   TypeError: unsupported operand type(s) for -: 'int' and 'str'
```

上述範例中的變數 employeRecord 在輸入時有個錯字，因而引發錯誤❶。由於「NameError: name 'employeRecord' is not defined」訊息中未定義的 employeRecord 是程式碼中特有的部分，因此在搜尋時可以用「**python "Name Error: name" "is not defined"**」來搜尋。範例最後一行錯誤訊息中的「'int' and 'str'」似乎指的是 42 和 'hello' 的值❷，截斷錯誤訊息用「**python "TypeError: unsupported operand type(s) for"**」來搜尋可避開特定於程式碼的部分。如果這些搜尋結果沒有有用的資訊，再嘗試把完整的錯誤訊息拿來搜尋。

利用 Linters 來避免錯誤

解決錯誤的最好方法是不要出錯。Lint 軟體或 linters 這些應用程式能分析原始程式碼，能對任何潛在錯誤提出警告。這個名字是引用了乾衣機內過濾細小纖維和碎屑的棉絮濾網（lint trap）。雖然 linters 沒辦法捕捉所有的錯誤，但是**靜態分析**（不執行原始程式碼的情況下進行檢查）可以識別出因為錯別字引起的常見錯誤（第 11 章會探討如何使用型別提示來進行靜態分析）。許多文字編輯器和整合式開發環境（IDE）中都含有一個在後端背景執行的 linters，能即時指出問題所在，如圖 1-2 所示。

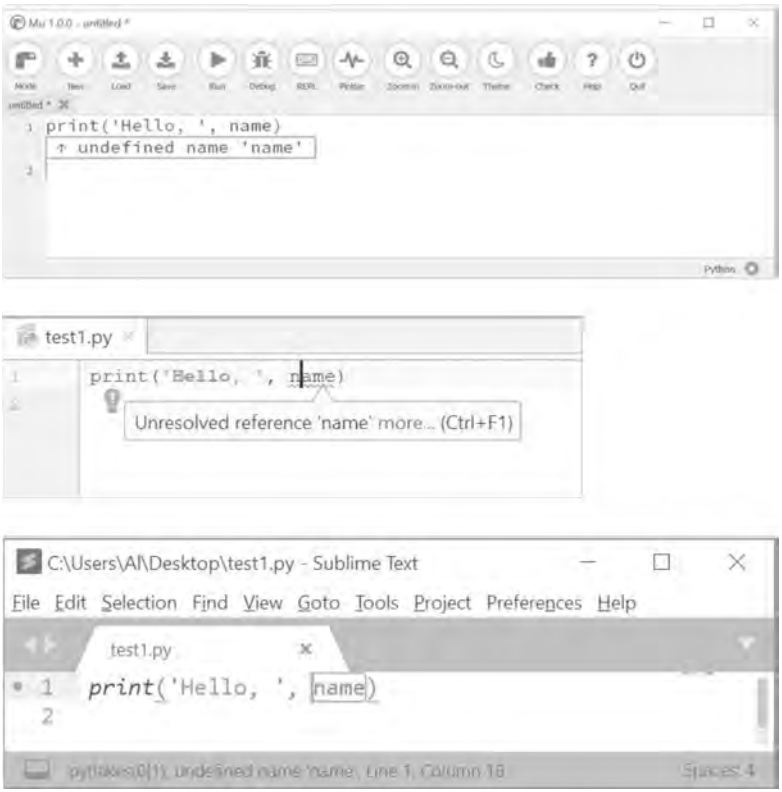


圖 1-2　Mu（上）、PyCharm（中）和 Sublime Text（下）中能指出未定義變數的 liter 功能

Linter 所提供的即時通知提醒能大幅提升程式設計編寫的效率。如果沒有這樣的通知提醒，則必須在執行程式後觀察程式的崩潰出錯，閱讀 traceback 內容後，再到原始程式碼中去修正拼寫錯誤的那行程式。而且，如果程式拼錯了很多個字，那麼一個執行修復週期只能找出一個錯別字來修訂。Linting 功能可以一次指出多個錯誤，並且直接在編輯器中就指出，我們在編寫程式時馬上就能看到發生錯誤的那行程式。

您用的編輯器或 IDE 可能不具備 lint 功能，但如果有支援 plug-in，則幾乎肯定能使用 linter，一般來說 plug-in 外掛所使用的 linting 模組名稱是 Pyflakes，或者也可能用其他模組來進行分析。請連到 https://pypi.org/project/pyflakes/ 來安裝 Pyflakes，也可以透過執行「pip install --user pyflakes」命令來進行安裝。很值得花點時間來安裝使用。

NOTE

在 Windows 系統中可以執行 python 和 pip 命令，但在 macOS 和 Linux 上，這些命令名稱僅適用於 Python 2 版本，因此需要執行 python3 和 pip3 命令來替代。每次在本書中看到使用 python 或 pip 命令時，請留意這一點。

IDLE 這個 Python 隨附的 IDE 就沒有 linter，也不能安裝這樣的功能。

程式問題要怎麼提問才正確？

當網路搜尋和 linters 不能解決您的程式設計問題時，可以在網路上提問尋求幫助，而遵循禮節可以讓您更有效快速地得到建議和解答。如果有經驗的軟體開發專家願意免費回答您的問題，最好是能有效利用他們的寶貴時間。

最好把向陌生人尋求程式設計上的協助當作是最後的選擇。這可能要經過幾個小時或幾天才能有人回答您發布的問題，想要得到完整的回覆可能要等待一段時間。在網路上搜尋已經提問過的類似問題並閱讀別人給的答案，這樣的速度要快得多。線上文件和搜尋引擎的出現減少了必須以人來回答問題的工作。

但如果已用盡其他方法也找不到答案，最後必須向網友詢問您的程式設計問題時，請避開以下常見的錯誤：

- 問別人是否可以提問卻不直接把真的問題提出。
- 暗示性提出問題，而不直接詢問。
- 問錯地方，在不相關的討論區或網站提問。
- 發文標題或 email 主旨寫得不明確，例如只寫「我有個問題」、「請幫幫我」之類。
- 只說「我的程式執行不出我要的」，卻不解釋想要讓程式怎麼執行。
- 沒有提供完整的錯誤訊息。
- 沒有提供程式碼。
- 提供的程式碼格式編排很差，讓人看不下去。
- 沒有解釋您已試過哪些處理。

- 沒有提供作業系統或版本資訊。
- 要求別人寫出程式給您。

避開上述「不要做」的清單是一種禮節，因為這些壞習慣會讓別人幫不了您。要幫您的人第一步是要執行您的程式並試著重現問題。要做到這一點，需要很多資訊來判斷，包括程式碼、電腦系統和您的意圖。一般最常見的狀況是提供的資訊太少而無法回應問題。接下來的幾個小節會探討怎麼防止這些常見錯誤。我先假設讀者是在線上論壇上發布問題，不過這些準則也適用於把問題以電子郵件發送給某個人或郵件論壇的情況。

一次提供所有的資訊，避免來回補充

如果親自面對某個人時提出「我能問您一個問題嗎？」，看看他是否有空，這是種簡短而愉快的方式。但在線上論壇，別人可能會推遲回覆，或是真的都很閒的時候才會回覆。由於兩個回覆之間可能要花上幾個小時，因此最好在最初的發文中提供所有可能需要的資訊，而不是徵求許可才提出問題。如果大家沒有回覆，您可以將此資訊複製並貼到其他論壇試試。

以真實問題的形式來陳述問題

我們很容易假設別人都聽得懂我們解釋問題時想要表達的，但是程式設計的領域範疇很廣闊，大家可能對特定領域的問題並沒有經驗。因此，以真實問題的形式來陳述問題是很重要。雖然句子開頭用「我想要...」或「程式碼不起作用」能暗示問題是什麼，但請明確的陳述問題：在字面上講清楚，句子以問號結尾。否則大家可能不清楚您要問什麼。

在合適的網站上提問

在 JavaScript 論壇上詢問 Python 問題或在網路安全郵件論壇上詢問演算法問題可能得不到回應。一般來說，郵件論壇和線上論壇都有常見問題解答（FAQ）文件或說明頁面，這些頁面都會解釋有哪些主題適合討論。舉例來說，python-dev 郵件論壇的討論是針對 Python 語言的設計功能，並不是一般 Python 問題的討論。https://www.python.org/about/help/這個網頁能將我們指引到適當的地方來詢問遇到的任何 Python 問題。

問題摘要放在標題上

把問題發布到線上論壇的好處是，將來別人有相同問題時都能在網路搜尋時找到該問題及其答案。請務必把問題摘要放在標題主旨上，這樣方便搜尋引擎進行組織整理。一般性的標題如「請幫忙」或「為什麼行不通？」這種語句太模糊了。如果您用電子郵件來提問，有意義的主旨標題能告知想要幫您的人，他們在掃描收件箱時主旨一眼就能看出。

解釋您想要讓程式怎麼運作

「我的程式為什麼不起作用？」這樣的問題省略了您希望程式怎麼執行的關鍵細節。這對想幫您回答問題的人沒作用，因為不知道您的意圖是什麼。就算問題只是「為什麼會出現這種錯誤？」，這都還能指出程式的最終目標是什麼。在某些情況下，網友可能會告訴您試著用完全不同的方法來處理，放棄這個問題不要浪費時間來解決。

放上完整的錯誤訊息

確定有複製貼上整個錯誤訊息，包括 traceback 內容。僅描述錯誤（例如「我遇到了超出範圍的錯誤」）是沒有足夠的細節來讓別人幫您找出錯誤所在。另外，請指出是否一直會遇到此錯誤，或者是間歇性出現。如果您已確定發生錯誤的具體情況，請放入這些詳細資訊。

放上完整的程式碼

除了完整的錯誤訊息和 traceback 內容，連同整支程式的原始程式碼也一起提供。如此一來，想幫您的人就能用他們的電腦來執行程式，並在 debugger 中檢測發生的狀況。最好提供一個最小、完整且可重現（minimum、complete 和 reproducible，縮寫為 MCR）的範例，用它來重現您所遇到的錯誤。

MCR 這個術語來自 Stack Overflow 網站，在 https://stackoverflow.com/help/mcve/ 上有詳細討論。最小（minimum）表示程式範例盡可能短，同時仍能重現您遇到的問題。完整（complete）表示程式範例要包含能重現該問題所需的完整內容。可重現（reproducible）表示程式範例能確實地重現您所描述的問題。

如果您的程式放在一個檔案中，那發送給想要幫您的人是很簡單。只需確定其格式正確就行了，下一段會說明格式的重要性。

Stack Overflow 網站與建立答案歸檔

Stack Overflow 是一個很受歡迎的網站，該網站能回答程式設計相關問題，但是許多程式新手在使用這個網站時還是感到沮喪甚至是害怕。Stack Overflow 的版主以嚴格聞名，會關掉不符合規定的問題討論。Stack Overflow 網站的管理如此嚴格是有理由的。

Stack Overflow 的目的不在於回答問題，而是要建立程式設計問題與答案的歸檔。因此，他們想要的問題是具體、唯一的，而不是有選擇性的。問題會詳細說明，以便讓搜尋引擎的使用者可以輕鬆找到。(在有 Stack Overflow 之前，程式設計師在網路上找答案的情況有時會像 XKCD 漫畫網 https://xkcd.com/979/ 上「Wisdom of the Ancients」這篇的嘲諷。) 一直重複輸入 30 個類似的問題，不僅會重複浪費網站上的志願者和專家的解答時間，也會讓搜尋引擎使用者找到更多混淆的結果。問題需要有具體且客觀的答案，像「什麼是最好的程式語言？」這種問題是一個選擇性的見解，答案可能引起不必要的爭論 (反正我們都已經知道 Python 是最好的程式語言了)。

從需求和尋求協助的角度來看，若只有您發文的問題被關掉是滿尷尬的。我建議您先仔細閱讀本章內容和 Stack Overflow 網站「How do I ask a good question?」中的說明指南，網址是 https://stackoverflow.com/help/how-to-ask/。其次，如果您害怕問出「蠢」問題，那就使用化名來發問吧。Stack Overflow 不需要使用其帳號的真實姓名。如果您想在比較沒那麼嚴格的地方發問，可以考慮到 https://reddit.com/r/learnpython/ 網站提問，這裡接受的提問較為寬鬆。不過在提交問題之前，還是請您一定要閱讀其發布指南。

以適當的格式化讓程式碼具有可讀性

提供程式碼的目的是使幫您解答的人可以執行程式並重現您所遇到的錯誤。他們不僅需要程式碼，也需要有正確的格式。請確定幫您的人能輕鬆複製您的原

始程式碼並按其原樣執行。如果要在電子郵件中複製和貼上原始程式碼時，請注意，許多電子郵件客戶端可能會刪除內縮編排，這樣會導致程式碼變成下列所示這般：

```
def knuts(self, value):
if not isinstance(value, int) or value < 0:
raise WizCoinException('knuts attr must be a positive int')
self._knuts = value
```

別人要花費很長的時間重新處理程式碼每一行的內縮格式，而且對程式行要內縮多少也很含糊。為了確保程式碼的編排格式正確，請把程式碼複製貼到 pastebin 這類網站上來處理，這類網站有 https://pastebin.com/ 或 https://gist.github.com/，該網站會把程式碼儲存在簡短的公用 URL，例如 https://pastebin.com/XeU3yusC。分享這個 URL 比使用檔案附件更容易。

如果要把程式碼發布到像 https://stackoverflow.com/ 或 https://reddit.com/r/learnpython/ 的網站中，請確定您是使用其文字方塊所提供的格式設定工具。一般來說，內縮四個空格的程式行會用等寬的程式碼字型，這樣會更易於閱讀。我們還可以用反引號（`）字元把文字括起來，將其置於等寬程式碼字型中。這些網站通常都有提供格式資訊的連結。若沒有使用前述這些技巧，貼上的程式碼可能會亂掉，全部會擠在一行中，如下所示：

```
def knuts(self, value):if not isinstance(value, int) or value < 0:raise
WizCoinException('knuts attr must be a positive int') self._knuts = value
```

此外，不要使用螢幕截圖或圖片的方式發送圖像來提供程式碼。因為這樣是無法複製和貼上圖片中的程式碼，通常也不好閱讀。

告知幫您的人已做了哪些嘗試

發布問題時，請告知幫您的人已做了哪些嘗試以及得到什麼樣的結果。這些資訊所提供的線索能讓人免去重試這些錯誤的時間，也表明了您有努力想要解決自己的問題。

此外，這些資訊說明了您是尋求協助，而不是只在要求別人幫您寫程式而已。不幸的是，電腦相關科系的學生常常要求線上的陌生人幫他們寫作業，也有些企業還要求別人免費幫他們建立「很快能用的 App」，這些怪現象很常見。但程式設計論壇的協助並不是用來做這種事情的。

描述您的設定

您電腦的特殊設定可能會影響程式的執行方式，也可能導致錯誤的發生。為確保幫您的人能在他們的電腦上重現您的問題，請提供關於您電腦的下列資訊：

- 作業系統和版本，例如「Windows 10 Professional Edition」或「macOS Catalina」。
- 執行程式的 Python 版本，例如「Python 3.7」或「Python 3.6.6」。
- 您的程式所使用的任何第三方模組及其版本，例如「Django 2.1.1」。

我們可以透過執行「pip list」命令來查看已安裝的第三方模組的版本。按照慣例，將模組的版本包含在 __version__ 屬性中，如下列在互動式 Shell 模式中所示的操作：

```
>>> import django
>>> django.__version__
'2.1.1'
```

很有可能這些資訊並不是必須的。但為了減少來回詢問的次數，無論如何，請在您的最初發文的帖子中提供此資訊。

提問的實例

以下是遵循上一節內容所示範的正確提問：

> **Selenium webdriver: How do I find ALL of an element's attributes?**
>
> In the Python Selenium module, once I have a WebElement object I can get the value of any of its attributes with `get_attribute()`:
>
> `foo = elem.get_attribute('href')`
>
> If the attribute named `'href'` doesn't exist, `None` is returned.
>
> My question is, how can I get a list of all the attributes that an element has? There doesn't seem to be a `get_attributes()` or `get_attribute_names()` method.
>
> I'm using version 2.44.0 of the Selenium module for Python.

Selenium webdriver: 我要怎麼找出所有的元素屬性

在 Python Selenium 模組中，一旦有了 WebElement 物件，就可以用 get_attribute() 取得物件任何屬性的值：

```
foo = elem.get_attribute('href')
```

如果屬性名稱 'href' 不存在會返回 None。

我的問題是，我要怎麼取得所有的元素屬性而不是只列出一個？好像不是用 get_attributes() 或 get_attribute_names() 方法。

我使用的是 2.44.0 版的 Python Selenium 模組。

這個問題來自 https://stackoverflow.com/q/27307131/1893164/。標題用一個句子摘要問題的重點。提問是以問題的形式來陳述，並以問號結尾。將來若是某個人在網路上的搜尋結果中閱讀到此標題，就會立即知道這個標題是否與他的問題相關。

這個提問是用等寬程式碼字型來格式化其程式碼，並把文字分成多個段落。很明顯可看出這篇文章中的提問內容：甚至用了「我的問題是」來開頭。其中也列出了 get_attributes() 或 get_attribute_names() 可能不是答案，這表示提問者已嘗試用這兩個方法來解決方案，同時暗示了這個問題真正答案的樣子。提問者還寫上 Selenium 模組版本的資訊，以方便別人參考。提供的資訊不怕寫太多，就怕寫的不夠。

總結

對於程式設計的問題，有能力自己找答案是程式設計師必須學習的最重要技能。由網路上很多程式設計師建立的大量參考資源，能提供您需要的答案。

但首先，您必須要解析 Python 引發的隱密錯誤訊息。如果您看不懂錯誤訊息的文字，那也沒關係，可以把這段文字提交給搜尋引擎，查詢錯誤訊息較白話的解釋以及可能的原因。錯誤訊息的 traceback 內容會指示錯誤在程式中哪個位置發生的。

當我們在編寫程式碼時，即時的 linter 功能可以指出拼寫和潛在的錯誤。Linter 功能非常有用，現代軟體開發實際上很需要它們。如果您用的文字編輯器或 IDE 沒有 linter 或不能外掛 linter 功能，請考慮換一個吧。

本章談到怎麼提出一個明確具體的問題，提供完整的原始程式碼和錯誤訊息的詳細資訊，解釋您已嘗試過的內容，並告知您正在使用什麼作業系統和 Python 版本。發布的答案不僅可以解決了您的問題，也幫助了將來遇到相同問題的其他程式設計師，他們能搜尋到您發文的帖子。

如果您一直還是需要上網找答案並尋求協助，請不要灰心。程式設計的領域很廣泛，沒有人能一次掌握所有的東西。即使是經驗豐富的軟體開發專家，他們也會每天查詢線上文件和找尋解決方案。多多熟練怎麼找出解決方案，讓您慢慢成為一位精明的 Python 高手。