我想透過這本書，協助你成為 Python 高手。本書把重點放在 *Python* 的「一行程式碼」，也就是把一些簡潔實用的程式，打包成 Python 一行程式碼。我們把重點聚焦於一行程式碼，應該可協助你更快讀懂並寫出更簡潔的程式碼，也可以增進你對語言的理解。

另外還有五個理由，讓我相信 Python 一行程式碼的學習可協助你成長，而且確實值得好好研究。

第一，你可藉此提高 Python 的核心技能，進而克服許多讓你退縮的程式設計弱點。如果你對基礎沒有深刻理解，恐怕很難在技術上有所進展。而一行程式碼正是任何程式的基本構成元素。只要好好瞭解這些基本構成元素，就可以協助你掌握更高層次的複雜度，而不至於感到不知所措。

第二，你可以學習到如何善用一些廣受歡迎的 Python 函式庫（例如資料科學、機器學習相關的函式庫）。本書共有六個章節，從正則表達式到機器學習，可說是各大領域皆有所涉略。這樣的做法可以讓你很快對 Python 應用有個整體的概念，而且我們也會教你如何善用各種功能強大的函式庫。

第三，你會學到如何寫出更符合 Python 風格的程式碼。大部分 Python 初學者（尤其是學過其他程式語言的初學者）常會寫出不符合 Python 風格的程式碼。我們會介紹一些 Python 特有的概念（例如解析式列表、同時指定多值與切取片段的做法），這些概念全都是為了協助你，寫出同領域其他程式設計者更容易閱讀與分享的程式碼。

第四，學習 Python 一行程式碼，一定會迫使你採用更清晰簡潔的方式進行思考。如果你很在意每一行程式碼所用到的字數，自然就不會寫出鬆垮垮、沒有重點的程式碼。

第五，當你學會一整套一行程式碼的技巧之後，就有能力更快速瀏覽其他複雜的 Python 程式碼，而且你的程式碼也會讓朋友或其他檢視者留下深刻的印象。你可能會發現，運用一行程式碼來解決一些很有

碁峰 www.gotop.com.tw

挑戰性的程式設計問題，本身就很有趣、很有成就感。而且你並不孤單：網路上各種充滿活力的 Python 技客社群，經常有許多人在競爭，看誰能寫出最高度壓縮、最具 Python 風格的解法，以解決各種實際（或沒那麼實際）的問題。

Python 一行程式碼範例

本書的中心論點是，學習 Python 一行程式碼既是理解高級程式碼的基礎，也是提高技能的絕佳工具。如果你想確實理解好幾千行的程式碼，搞清楚其中究竟發生什麼事，就必須瞭解每一行程式碼的含義。

我們先來快速檢視一下，什麼是 Python 一行程式碼。如果你目前還無法完全理解，也請不要擔心。到了第 6 章，你就能完全掌握這裡的一行程式碼了。

```
q = lambda l: q( ❶[x for x in l[1:] if x <= l[0]]) + [l[0]] + q([x for x in l if x > l[0]]) if l else []
```

這一行程式碼針對著名的 Quicksort（快速排序）演算法進行了壓縮，構成了一種優美而簡潔的寫法，不過對於許多 Python 初學者與中等程度的人來說，其中的含義恐怕很難理解。

Python 一行程式碼通常會根據之前的基礎逐步建構起來，因此本書一行程式碼的複雜度也會逐步提升。我們會先從簡單的一行程式碼開始，而這一行程式碼又會變成更複雜一行程式碼的基礎。舉例來說，前面所提到的快速排序一行程式碼看起來又難又長，但其中「解析式列表」（list comprehension）❶ 則是相對比較容易的一個基礎概念。下面有個比較簡單的解析式列表，可用來建立一個平方數的列表：

```
lst  = [x**2 for x in range(10)]
```

我們可以把這一行程式碼拆成好幾行更簡單的程式碼，其中包含一些重要的 Python 基礎寫法，例如指定變數值、數學運算、資料結構、

碁峰 www.gotop.com.tw

```
for i in range(10):
    squares.append(i**2)

print(squares)
# [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

在這段程式碼中，我們需要四行程式碼，才能建立 10 個平方數的列表，並在 shell 界面列印出結果。不過，一行程式碼的解法顯然好很多，因為它可以用一種更容易閱讀、更簡潔的方式，達到完全相同的效果：

```
# 精簡後
print([i**2 for i in range(10)])
# [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

輸出的結果完全相同，但一行程式碼所採用的解析式列表，更符合 Python 的風格。程式碼不僅更容易閱讀，而且也更加簡潔。

不過，Python 一行程式碼也有可能變得很難理解。在某些情況下，用 Python 一行程式碼來解決問題，反而會讓程式碼變得比較不容易讀懂。不過，就像西洋棋大師一樣，他們在判斷下哪個棋子最好之前，都必須先想好所有可能的棋步；你也應該瞭解各種運用程式碼來表達想法的所有做法，這樣才能判斷哪一種是最好的做法。找出**最漂亮的**解決方式，並不是件輕而易舉的事；這可說是 Python 體系裡的一個核心概念。「*Python* **的禪意**」不是也說了：「優美總比醜陋好」。

本書是寫給誰看的？

你是能力介於初級到中級的 Python 程式設計者嗎？和許多程度相近的人一樣，你的程式設計能力或許正陷入進展緩慢的困境。這本書可以協助你脫離困境。你應該已經在網路上閱讀過許多程式設計教程。你也已經編寫過自己的原始程式碼，成功交付過小型的專案。或許你還修過一些程式設計基礎課程，閱讀過一兩本程式設計教科書。你甚至

碁峰 www.gotop.com.tw

都屬於異常值。以我們的資料來說，只要是落在 [9,11] 這個區間以外的值，全都屬於異常值。

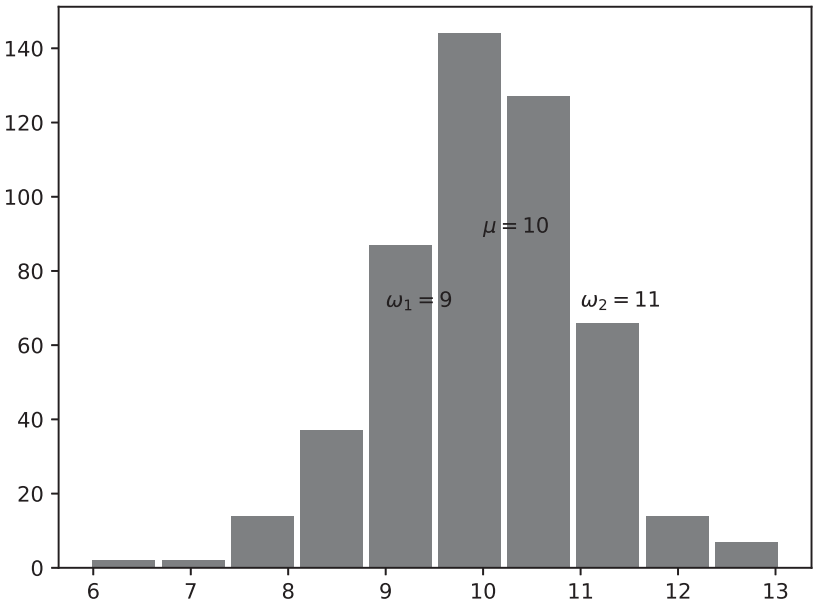

圖 3-3：一大堆資料值的直方圖

列表 3-25 顯示的就是我們用來生成圖形的簡單程式碼。你能找出其中定義平均值與標準差的程式碼嗎？

```
import numpy as np
import matplotlib.pyplot as plt

sequence = np.random.normal(10.0, 1.0, 500)
print(sequence)

plt.xkcd()
plt.hist(sequence)
plt.annotate(r"$\omega_1=9$", (9, 70))
plt.annotate(r"$\omega_2=11$", (11, 70))
plt.annotate(r"$\mu=10$", (10, 90))
plt.savefig("plot.jpg")
plt.show()
```

列表 3-25：用 Matplotlib 函式庫畫出直方圖

碁峰 www.gotop.com.tw

簡單的關聯性分析：購買 X 的人也購買了 Y

你是否曾購買過 Amazon 演算法所推薦的商品？推薦演算法通常是以一種叫做**關聯分析**（*association analysis*）的技術做為其基礎。我們在本節就會學習到關聯分析的基本概念，一腳踏入推薦系統的深淵之中。

基礎

關聯分析主要是根據客戶的歷史資料，例如 Amazon 就有「購買 *x* 的人也購買了 *y*」這樣的資料。這種不同商品之間的關聯性，可說是一種強而有力的市場行銷概念，它不僅可以把相關與互補的商品聯繫起來，還可以為客人提供「**社群掛保證**」的元素，因為知道有別人買過某商品，你在購買該商品時通常就會更放心。這可說是行銷人員絕佳的工具。

我們就來看一下圖 3-4 的實際範例。

Alice	✗	✓	✓	✓	✗
Bob	✓	✗	✓	✗	✓
Louis	✗	✓	✓	?	✗
Larissa	✓	✓	✗	✓	✗

圖 3-4：「商品 - 客戶」矩陣：哪個客戶購買了哪個商品？

Alice（愛麗絲）、Bob（鮑勃）、Louis（路易斯）與 Larissa（拉里莎）這四個客戶各自購買了以下幾種商品的不同組合：書籍、遊戲、足球、筆記型電腦、耳機。想像一下，假設你已經知道其他四個人都購買了哪些商品，唯獨不知道 Louis 有沒有購買筆記型電腦。你會怎麼想：Louis 有可能會購買筆記型電腦嗎？

碁峰 www.gotop.com.tw

關聯分析（或協作篩選 *collaborative filtering*）可以針對這個問題提供解答。基本的假設是，如果有兩個人過去曾表現出類似的行為（例如購買了類似的商品），這兩個人未來就比較有可能繼續表現出類似的行為。Louis 的購買行為與 Alice 很類似，而 Alice 確實有購買筆記型電腦。因此，推薦系統應該可以預測 Louis 也可能會想購買筆記型電腦。

下面的程式碼，把這個問題進行了簡化。

程式碼

請考慮以下問題：有百分之幾的客戶，會同時購買兩本電子書？只要根據這樣的資料，推薦系統或許就可以在客戶原本只打算購買一本書的情況下，向客戶提供「合購」其他書籍的建議。參見列表 3-29。

```
## 依賴的模組套件
import numpy as np


## 資料：每一橫行對應一個客戶的購買記錄
## row = [ 課程 1, 課程 2, 電子書 1, 電子書 2]
## 1 這個值代表曾購買過該項目。
basket = np.array([[0, 1, 1, 0],
                   [0, 0, 0, 1],
                   [1, 1, 0, 0],
                   [0, 1, 1, 1],
                   [1, 1, 1, 0],
                   [0, 1, 1, 0],
                   [1, 1, 0, 1],
                   [1, 1, 1, 1]])


## 一行程式碼
copurchases = np.sum(np.all(basket[:,2:], axis = 1)) / basket.shape[0]


## 結果
print(copurchases)
```

列表 3-29：這一行程式碼運用到切取片段的做法、axis 參數、shape 屬性，並用撒播機制執行了一些陣列基本運算

www.gotop.com.tw

接著可以把直線與訓練資料畫在同一個圖形中，如圖 4-5 所示。

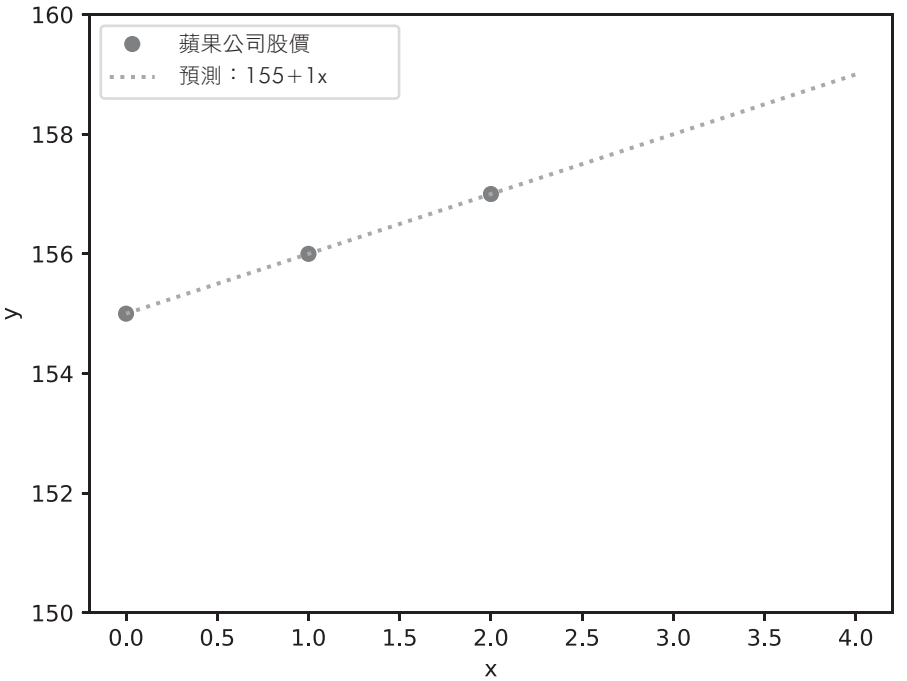

圖 4-5：運用我們的線性迴歸模型，所製作出來的預測直線

完美套入！直線（代表模型的預測）與訓練資料之間的平方距離為零，代表你已經找到可以讓誤差最小化的模型了。只要運用此模型，你就可以預測出任何 x 值所對應的股價。舉例來說，假設你要預測 $x = 4$ 時相應的股票價格。你只需要用模型計算一下 $f(x) = 155.0 + 1.0 \times 4 = 159.0$。第 4 天的預測股價就是 159 美元。當然，這種預測能否準確反映現實世界的情況，那又是另一回事了。

以上就是相關概念的介紹。接著我們就來仔細看看如何在程式碼中實現這些概念。

碁峰 www.gotop.com.tw

在這裡的範例中，訓練過的神經網路可用來偵測出圖片中的動物。實際上，你會把圖片中的每個像素全都對應到一個輸入神經元，以構成一個輸入層。這樣可能會導致有好幾百萬個輸入神經元，與好幾百萬個隱藏神經元相連。而每個輸出神經元通常都只負責整體輸出的一個位元。舉例來說，如果要檢測出兩種不同的動物（例如貓和狗），你就只會在輸出層中使用單一個神經元，以模擬兩種不同的狀態（0 = 貓，1 = 狗）。

其構想就是，如果有某個輸入脈衝被送到神經元，神經元就會被「激活」（activated）或「啟動」（fired）。每個神經元都會根據輸入脈衝的強度，獨立判斷是否被啟動。這樣一來，你就可以模擬人的大腦，因為大腦的神經元也是透過脈衝彼此互相激活。輸入神經元會透過網路陸續判斷是否要進行激活，就這樣不斷往後傳播直到抵達輸出神經元為止。其中有一些輸出神經元會被激活，有一些則不會。最後輸出神經元激活狀況所形成的特定模式，就成為人工神經網路最終的輸出（或預測）。在你的模型中，激活的輸出神經元可編碼為 1，而未激活的輸出神經元則可編碼為 0。這樣一來，你就可以訓練神經網路，藉以預測任何可編碼成一系列 0 與 1 的事物（也就是電腦可表示的一切事物）。

我們就來詳細瞭解一下神經元運作的數學原理，如圖 4-19 所示。

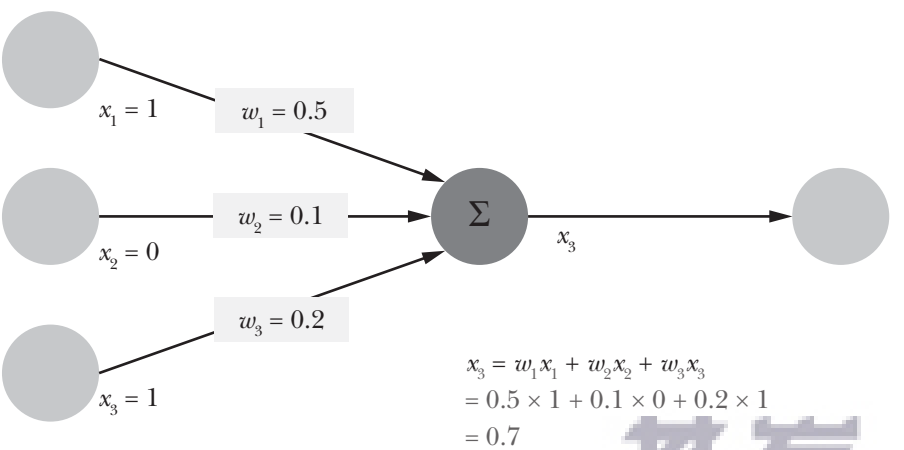

圖 4-19：單一神經元的數學模型：輸出是三個輸入的一個函數。

碁峰 www.gotop.com.tw

WEEK：過去七天內，你看了幾小時的 Python 程式碼？

YEARS：你是從幾年前開始學習資訊科學？

BOOKS：你的書架上有幾本程式設計的書籍？

PROJECTS：在實際的專案中，你在 Python 花費多少百分比的時間？

EARN：你每個月可以靠你的技術技能（廣義上來說）賺多少錢（四捨五入到 1000 美元）？

我們再次站上巨人的肩膀，運用 scikit-learn（`sklearn`）函式庫進行神經網路迴歸處理，如列表 4-5 所示。

```
## 依賴的模組套件
from sklearn.neural_network import MLPRegressor
import numpy as np

## 問卷資料 (WEEK, YEARS, BOOKS, PROJECTS, EARN, RATING)
X = np.array(
    [[20,  11,  20,  30,  4000,  3000],
     [12,   4,   0,   0, 1000,  1500],
     [2,   0,   1,  10,   0,  1400],
     [35,   5,  10,  70,  6000,  3800],
     [30,   1,   4,  65,   0,  3900],
     [35,   1,   0,   0,   0, 100],
     [15,   1,   2,  25,   0,  3700],
     [40,   3,  -1,  60,  1000,  2000],
     [40,   1,   2,  95,   0,  1000],
     [10,   0,   0,   0,   0,  1400],
     [30,   1,   0,  50,   0,  1700],
     [1,   0,   0,  45,   0,  1762],
     [10,  32,  10,   5,   0,  2400],
     [5,  35,   4,   0, 13000,  3900],
     [8,   9,  40,  30,  1000,  2625],
     [1,   0,   1,   0,   0,  1900],
     [1,  30,  10,   0,  1000,  1900],
     [7,  16,   5,   0,   0,  3000]])
```


碁峰 www.gotop.com.tw

```
## 一行程式碼
neural_net = MLPRegressor(max_iter=10000).fit(X[:,:-1], X[:,-1])

## 結果
res = neural_net.predict([[0, 0, 0, 0, 0]])
print(res)
```

列表 4-5：神經網路分析的一行程式碼

一般人恐怕不太可能預測出正確的輸出結果，但你想嘗試一下嗎？

原理說明

我們在前幾行程式碼建立了資料集。scikit-learn 函式庫裡的機器學習演算法，都是使用這樣的輸入格式。每一橫行都是具有多個特徵的單一筆觀察結果。越多橫行，就表示越多的訓練資料；越多縱列，就表示每次觀察的特徵越多。在這裡的例子中，每一筆訓練資料都有五個輸入特徵，輸出值則只有一個特徵。

只要運用 `MLPRegressor` 這個物件類別的建構函式，就可以用一行程式碼建立一個神經網路。我把 `max_iter = 10000` 當做參數送入，主要是因為若使用默認的迭代次數（`max_iter = 200`），訓練並不會收斂。

之後再調用 `fit()` 函式，以決定神經網路的參數。調用 `fit()` 之後，神經網路就可成功完成初始化的工作。`fit()` 函式可接受一個多維的輸入陣列（每一橫行代表一次的觀察結果，每一縱列則代表一個特徵），以及一個一維的輸出陣列（其大小就等於觀察結果的數量）。

最後剩下的唯一工作，就是針對特定的輸入值調用 predict 函式：

```
## 結果
res = neural_net.predict([[0, 0, 0, 0, 0]])
print(res)
# [94.94925927]
```

碁峰 www.gotop.com.tw

請注意，由於函式的不確定性與不同的收斂行為，你所得到的實際輸出可能會與這裡略有不同。

簡而言之：如果 ...

- ... 你在上週訓練了 0 個小時，
- ... 你是 0 年前才開始學習資訊科學，
- ... 你的書架上有 0 本程式設計的書籍，
- ... 你花費了 0% 的時間來實作真正的 Python 專案，而且
- ... 你靠著自己的程式設計技能，賺到了 $ 0，

根據神經網路的估計，你的技術水準應該是**非常**低的（Finxter 評分為 94，表示你連 `print("hello, world")` 這樣的 Python 程式碼都很難理解）。

因此，我們就來改變一下：如果你在一週內花費 20 個小時學習，然後在一周之後重新檢視神經網路，會得到什麼樣的結果呢：

```
## 結果
res = neural_net.predict([[20, 0, 0, 0, 0]])
print(res)
# [440.40167562]
```

還不錯，你的技能有了明顯的提升！不過你對這個評分還是不太滿意，對吧？（高於平均水準的 Python 程式設計者，在 Finxter 的評分至少會在 1500–1700 左右。）

沒問題。去買 10 本 Python 書吧（你已經買了本書，所以還差 9 本）。我們再來看看你的評分發生了什麼變化：

碁峰 www.gotop.com.tw

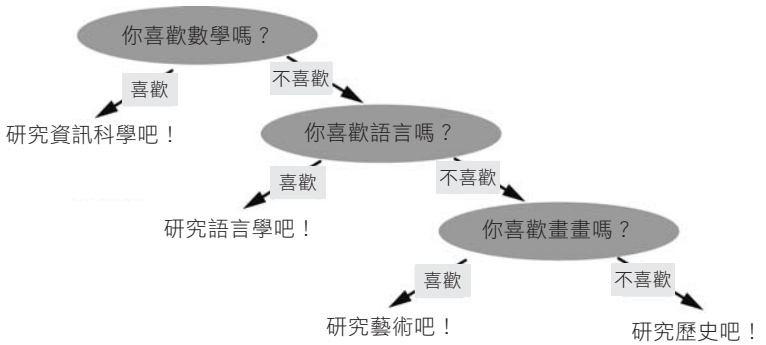

圖 4-20：推薦研究主題的簡化決策樹

決策樹可用於分類問題，例如「根據我的興趣，我應該鑽研哪一門學科？」首先從最頂部開始。接下來可能要反覆回答一些問題，並選出最能夠描述你個人特徵的選項。最後你會到達樹狀結構的**葉子節點**（*leaf node*），也就是下面已經沒有子節點了。這個節點就是根據你所選擇的個人特徵，最後所推薦的類別。

決策樹學習可能會有許多細微的差異。譬如在前面的範例中，第一個問題就比最後一個問題具有更大的權重。如果你喜歡數學，決策樹就永遠不會推薦藝術或語言學。這個特性很有用，因為對於分類決策來說，某些特徵確實有可能比其他特徵重要得多。舉例來說，假設有一個可預測你目前健康狀況的分類系統，它確實有可能一開始就利用你的性別（特徵），排除掉許多可能的疾病（類別）。

因此，決策節點的順序確實會影響最佳化的表現：我們應該把一些對於最終分類有重大影響的特徵，盡可能放到比較接近頂部的位置。在決策樹學習中，你也可以把一些對於最終分類影響很小的問題整併起來，如圖 4-21 所示。

碁峰 www.gotop.com.tw