前言

這本第三版的書籍將介紹世上最流行的程式語言之一：Python。也許你是程式初學者，或雖然已有經驗，但也想要認識 Python 這種語言。在本書中，我有時會比較 Python 與其他語言，以呈現一般人經常認為它是怎麼運作的，尤其是一些微妙的差異。

計算機語言比人類語言更易學，因為它們更精簡，也更精確。Python 是公認為最容易學習、閱讀、撰寫的電腦語言之一。它由資料（類似口說語言裡的名詞）以及指令或程式碼（類似動詞）組成。你將在各個章節學到 Python 的基本程式與資料結構，瞭解如何結合它們，並建構更高階的程式與資料結構。在過程中，你看到的、寫出來的程式將越來越長、越來越複雜。

你將學會這門語言，以及如何運用它。我們會先認識 Python 的核心語言與「隨機附贈」的標準程式庫，再一路看到如何尋找、下載、安裝、使用各種好用的第三方套件。我的重點是自己在 20 年來、於正式環境中使用 Python 所累積下來的、真正實用的知識，而不是罕見的主題，或複雜的奇技淫巧。

雖然這是一本入門書籍，但仍納入一些進階主題，目的是為了讓你提前接觸它們。本書依然會介紹資料庫、網頁…等領域，但技術會迅速演變，現在的 Python 開發者通常必須瞭解一些機器學習、佇列（queues）、或 Unicode 知識。你也可以在本書中找到它們的詳細說明。

Python 有一些特殊功能，使用它們通常比採用其他語言的語法更好。例如，使用 `for` 與 *iterator* 來撰寫迴圈，比手動遞增計數變數要直接得多。

我們在學習新內容時，經常難以分辨哪些詞彙是專業術語、哪些只是隨口表達，也不容易判斷哪些概念確實很重要，也就是學生常問的：「這會考嗎？」，我會適度標出有明確意義或較重要的 Python 術語和觀念，但不會一次灌輸太多內容。真正的 Python 程式碼從本書開頭就會頻繁出現。

如果內容比較難懂，或是有更 *Pythonic*（符合 *Python* 風格）的寫法，我會加入這一種說明。

Python 並非完美。我會告訴你比較怪異，或應該避免的做法，並且告訴你可以改用哪種方案。

適合的讀者

雖然具備一些程式設計經驗可能有幫助，但我希望初學者也能從本書中受益。Python 非常適合當成入門的電腦語言，而且你不需要通讀或完全理解全書的內容，就能開始使用它。

第三版的變動

雖然本書的整體架構大致沿用第二版，但各頁內容皆已全面更新：

- 移除第 20–22 章，還有附錄 A、C 和 E
- 加入關於 AI、資料科學、效能的章節
- 更廣泛地探討開發環境
- 最新的 Python 功能與修改
- 重視型態提示的使用
- 更新許多範例與較冷僻的小細節（arcane tidbits）[1]
- 將之前的第 19 章（*Be a Pythonista*）擴大為全部的第二部分

1 這好像蠻適合當成樂團或寵物飼料的名字的！？

Python 似乎真的很喜歡那一個樸實的底線字元：

- 以一個下底線（_）開頭的名稱會被 `import` 陳述式視為某種程度的私有（第 12 章會說明）。
- 以兩個底線（__）開頭的名稱，在建立物件類別時，會被特別處理（第 11 章會介紹）。
- 前後皆有雙底線的名稱用於物件類別中的魔術方法（也叫做 *dunder* 方法，第 11 章會提到）。

Python 型態

如你所知，bits、bytes 與多 byte 組合在電腦記憶體或儲存空間中沒有固有的意義，你必須在某處以某種東西來賦予意義，並記得它們，這就是程式語言的型態的作用。

每一種電腦架構（由電腦公司設計的具體架構）都有能力處理特定的 bit 組合，並將其視為不同型態。這些型態包括不同大小的數字、文字字元…等。Python 定義了自己的物件型態來對應這些常見的硬體型態，以及一些純軟體型態，例如複數。表 2-1 是這些內建型態，以下是表格中的各個欄位的意思：

- 「名稱」欄位是它的中（英）文名稱。
- 「型態」欄位是該型態在 Python 中的實際名稱。
- 「可變？」欄位表示該型態的值（不是型態本身！）能不能更改。
- 「範例」欄位是一些用來表達這一種型態值的 Python 語法。
- 「章」欄位則指出你會在哪一章深入瞭解相關內容。

表 2-1　Python 的基本資料型態

名稱	型態	可變？	範例	章
布林（Boolean）	`bool`	否	`True, False`	第 3 章
整數（Integer）	`int`	否	`47`、`25000`、`25_000`	第 3 章
浮點（Floating point）	`float`	否	`3.14`、`2.7e5`	第 3 章
複數（Complex）	`complex`	否	`3j`、`5 + 9j`	第 3 章
文字字串（Text string）	`str`	否	`'alas'`、`"alack"`、`'''a verse attack'''`	第 4 章
串列（List）	`list`	是	`['Winken', 'Blinken', 'Nod']`	第 8 章
Tuple	`tuple`	否	`(2, 4, 8)`	第 8 章
Bytes	`bytes`	否	`b'ab\xff'`	第 5 章

名稱	型態	可變？	範例	章
Bytearray	`bytearray`	是	`bytearray(...)`	第 5 章
集合（Set）	`set`	是	`set([3, 5, 7])`	第 9 章
凍結集合（Frozen set）	`frozenset`	否	`frozenset(['Elsa', 'Otto'])`	第 9 章
字典（Dictionary）	`dict`	是	`{'game': 'bingo', 'dog': 'dingo', 'drummer': 'Ringo'}`	第 9 章

下一章將介紹數字型態：bool、int、float 與 complex。

指定數值

到目前為止，你已經知道數值可以用*字面值*（例如 `5`）或*變數*（例如先前被指派 `5` 的 `x`）來表示了。Python 有一些指定字面值的規則，這些規則與底層型態有關。如你所見，整數是用一串數字來表示的，而 *float*（浮點值，會在下一章介紹）可由一串數字加上小數點組成。

在接下來的幾章中，我會示範如何為 Python 的標準型態指定字面值、將它們指派給變數，以及執行各種運算（例如數字的加法）。

將物件視為記憶體中的透明塑膠盒

你可以把物件想成記憶體書架上的透明塑膠盒。它裡面的一些內容有固定大小（ID、型態、參考計數），其他內容的大小（值本身的 bits）則會變動。所有 Python 資料都是如此。你可能會看到某個 `int` 物件旁邊是 `list` 物件，再旁邊是使用者自訂物件。

Python *程式碼*位於記憶體中的不同位置（另一個書架）。Python 程式碼中的變數可視為貼在資料物件上的便利貼，或是一張綁在細線上的標籤。Python 會自動追蹤所有的程式碼與資料，為我們節省心力。

回顧與展望

本章專門探討變數（在程式裡使用的名稱）與物件。Python 與許多其他程式語言不同，因為變數只是名稱，而它所指的物件有額外的資訊：唯一的 ID、型態、值，以及指向該物件的變數數量（參考計數）。

圖 11-1　鴨子定型（typing）的意思，不是笨拙地伸出兩根食指來打字

魔術方法

你現在已經能建立並使用基本物件了。這一節要教的東西可能會讓你大吃一驚——而且是好的那種吃驚。

當你輸入 `a = 3 + 8` 這種東西時，值為 3 與 8 的整數物件是怎麼知道該如何實作 + 的？又或者，當你輸入 `name = "Daffy" + " " + "Duck"` 時，Python 如何知道這次的 + 代表串接兩個字串？`a` 與 `name` 又是如何透過 = 取得結果的？你可以透過 Python 的特殊方法（或更戲劇化地稱為魔術方法）來存取這些運算子。

這些方法的名稱開頭與結尾都有雙底線（__），為什麼？因為程式開發者不太可能在變數名稱中使用它們。你已經看過一種魔術方法了：`__init__()` 會根據類別定義與傳入的引數來初始化一個新建立的物件。你也看到過雙底線名稱如何用來改寫類別屬性與方法名稱。

假設你有一個簡單的 `Word` 類別，你想要用一個 `equals()` 方法來比較兩個單字，但大小寫視為相同，也就是說，若一個 `Word` 含有 `ha`，而另一個含有 `HA`，兩者應視為相等。

下面的例子是我們的第一次嘗試，使用一般方法 `equals()`。屬性 `self.text` 是 `Word` 物件內的文字字串，而 `equals()` 方法會比較 `self.text` 與 `word2`（另一個 `Word` 物件）的文字字串：

表 11-1　做比較的魔術方法

方法	說明
`__eq__(` *self, other* `)`	*self* `==` *other*
`__ne__(` *self, other* `)`	*self* `!=` *other*
`__lt__(` *self, other* `)`	*self* `<` *other*
`__gt__(` *self, other* `)`	*self* `>` *other*
`__le__(` *self, other* `)`	*self* `<=` *other*
`__ge__(` *self, other* `)`	*self* `>=` *other*

表 11-2　做數學算術的魔術方法

方法	說明
`__add__(` *self, other* `)`	*self* `+` *other*
`__sub__(` *self, other* `)`	*self* `-` *other*
`__mul__(` *self, other* `)`	*self* `*` *other*
`__floordiv__(` *self, other* `)`	*self* `//` *other*
`__truediv__(` *self, other* `)`	*self* `/` *other*
`__mod__(` *self, other* `)`	*self* `%` *other*
`__pow__(` *self, other* `)`	*self* `**` *other*

`+`（魔術方法 `__add__()`）與 `-`（魔術方法 `__sub__()`）等數學運算子並非只能用來處理數字，舉例來說，Python 字串物件可使用 `+` 來做串接，用 `*` 來做重複。此外還有許多魔術方法可用，請參考網路上的「Special method names」文件（*https://bit.ly/pydocs-smn*）。表 11-3 是其中最常見的幾個。

表 11-3　其他的魔術方法

方法	說明
`__str__(` *self* `)`	`str(` *self* `)`
`__repr__(` *self* `)`	`repr(` *self* `)`
`__len__(` *self* `)`	`len(` *self* `)`

除了 `__init__()` 之外，在自訂類別中最常用的應該是 `__str__()`，它決定你的物件該如何印出。`print()`、`str()` 和字串格式化工具（第 17 章會介紹）都會使用它。互動式直譯器則使用 `__repr__()` 函式來輸出變數。如果你沒有定義 `__str__()` 或 `__repr__()`，Python 會使用預設的物件字串表示法：

```
>>> first = Word('ha')
>>> first
```

Browser Use（*https://browser-use.com*）

控制網頁瀏覽器

更進階的做法是將多個代理串接起來，將更複雜的作業流程自動化。例如：

LangChain（*https://oreil.ly/l3HKE*）

這是一款能將 LLM 模型連接到一個或多個代理的 Python 框架。

MCP（*https://oreil.ly/llJ6W*）

來自 Anthropic 的模型內容協定（Model Context Protocol）。

A2A（*https://oreil.ly/kFIMv*）

Google 的 Agent2Agent 協定看似 MCP。它能連接代理，也能協助發現代理，隨著模型遍布全球，這個功能也將日漸重要，這正是 Google 擅長的領域。

效率

事情的真相是……更有效率地執行這些計算的方法已經找到了。電腦硬體與軟體變得更容易擴展與更便宜也有所助益。值得注意的是，GPU 執行平行乘法與加法的效能，也遠勝於 CPU，這些運算正是 AI 的核心工作。雲端運算的興起，也讓規模更勝以往的電腦能夠用來處理這些問題。

建立模型：Python 框架

Python 在 AI 領域的主導地位是自然發展而成的。這個語言免費、開源，並且非常適合處理資料。尤其是，Python 既能拿來建立 AI 模型，也能用來運行 AI 模型。

首先，我們來看看如何用 Python 建立模型。

Python 領域早期且重要的組件包括 NumPy 與 SciPy，它們讓這一款速度不算快的語言，也能處理巨大的運算問題。

SciKit 是一組建構在 SciPy 之上的科學套件。其中的 scikit-learn（*https://scikit-learn.org*）是重要的 ML 套件：它支援建模、分類、分群與多種演算法。其安裝命令是 `pip install scikit-learn`。

回顧與展望

Python 已經是資料科學與 AI 開發領域的主力語言了。AI 已從規則導向的專家系統轉變為資料導向的 LLM 與其他 AI 模型，像 PyTorch 這樣的 Python 框架也隨之出現。如今，我們可以用命令列或 Python 程式碼來取用完整的多模態模型，也能串接代理，以擴展其能力範圍。

本章並未詳細介紹如何使用特定 API 或 Hugging Face 等通用 API 來建立模型，而是以一個框架（Ollama）來示範如何使用不同模型。Ollama 是免費的，更重要的是，它可以在你的本機電腦上運行，讓你不必將資料上傳至雲端服務。

電腦運算正邁入新階段：撰寫程式碼，將不如組合各種服務來發掘資料模式來得重要。

下一章要探討效能的諸多面向，涵蓋演算法到具體工具。

習題

我在第 551 頁的「使用 Ollama」展示了許多不理想的例子。挑選你喜歡的任何一種其他 AI 模型（無論是開源的、商用的、本機的、託管的皆可），試著完成以下任務：

26.1 計算單字 `strawberry` 裡有幾個字母 `r`。

26.2 定義 limerick。

26.3 寫一首 limerick。主題由你挑選。

26.4 寫一個名為 `utc_now_str()` 的 Python 函式，以回傳目前的 UTC 日期與時間字串。

26.5 寫一個 Python 產生器生成式，以回傳 1 到 5 的整數。

26.6 重現範例 1-5。

26.7 重現範例 24-3。