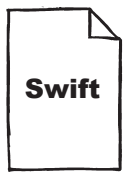
1 Swift 程式設計入門

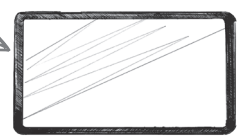
開發應用程式、系統，並且超越極限！

Swift 是能讓你倚賴與信任的程式語言。這個程式語言甚至可靠到你能帶它回去見你的家人，它不僅安全、可靠、快速、友善，而且易於溝通。雖然 Swift **最為人所熟知的用法是應用在 Apple 平台上**，例如 iOS、macOS、watchOS 和 tvOS，不過，有些 Swift 的開放原始碼專案也能在 Linux 和 Windows 上執行，而且逐漸在**系統程式語言**和伺服器上展露頭角。所有你能想到的內容都能利用 Swift 開發出來，從行動應用程式到遊戲、網頁應用程式、框架等等一切都將是你的囊中之物。讓我們開始一起學習吧！

Swift 是萬用程式語言



Swift 可以開發 **macOS、Linux 和 Windows 軟體**。在 macOS 上，利用使用者介面框架能創作出圖形豐富的應用程式，後續章節會再回來探討這項主題。



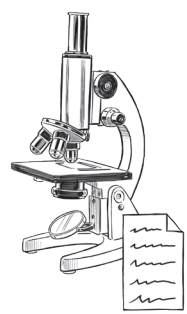
Swift 可以開發適用於 iPad、iPhone 和 iPod touch 環境底下的 **iOS 和 iPadOS 應用程式**，也能開發適用於 Apple TV 上的 tvOS 應用程式、和 Apple Watch 的 watchOS 應用程式。

Swift 可以開發**網頁應用程式**，為你的網站和應用程式的後端 API 驅動後端程式、或是產生靜態的網頁內容，進行同步。

Swift 本身也是開放原始碼。

Swift 這個開放原始碼專案非常活躍、管理完善，而且具備友善與清晰的參與流程。

本書末尾會討論如何參與這項專案。



Swift 可以進行**資料科學**和**機器學習**，以及**設計系統程式**，建置各種科學、模擬等等一切的結構。

不論你的目的是什麼，都可以利用 Swift 達成。

Swift 發展神速。

從原本不起眼的 Swift 1 開始發展，一路走來到今天，放眼未來，Swift 這項程式語言一直不斷成長，年年都會增加新的內容。

Swift 每一項**主要**更新都增加了新的**程式語言功能**，捨棄某些不適切的**語法**元素。自 2014 年公開釋出以來，Swift 一直是程式語言史上，發展最快而且最受歡迎的語言之一。

計算排名有許多不同的認定方式，但某些計算比起其他來說更有意義。然而，不管是在什麼樣的排名列表裡，Swift 都是名列前茅！

過去幾年裡，Swift 經常是**前十大程式語言**的常客，這項語言的使用者與專案數量持續增長。

現今對 Swift 技能的需求很高，所以看完這本書後，請別忘了將它加進你的履歷表中！

雖然 Swift 最初的設計僅是作為 Apple 平台（iOS、macOS、tvOS、watchOS）的語言而生，但自從 2015 年開放原始碼以來，一直不斷成長與擴展它發展的可能性，非常適合系統程式設計、科學、網頁等等超越一切極限的程式。

不論你的目的什麼，Swift 都是最佳的程式語言。

本書絕對無法涵蓋到 Swift 的每個面向，但你會知道你所需要的一切知識，讓你能敏捷地在程式的世界裡四處移動！

重點提示

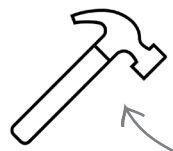
- 新增的**程式語言功能**，像是結構、通訊協定和 SwiftUI view 框架。
- 捨棄的**語法**是指特別的定位和使用元素，像是！、？和括號等等。

Swift 快速進化史

2014

Swift 1

2014 年 6 月，Apple 在開發者大會（WWDC）上大張旗鼓推出 Swift，讓眾人大吃一驚。在這個時間點，Swift 還是一項專用語言，僅用於 Apple 平台上的開發工作，而且屬於封閉原始碼。



Swift 的前幾個版本只支援 macOS 上的 Xcode。

2015

2015年12月，Swift 在版本 2.2 開放原始碼。從那時起，Swift 的內容開發就在這個專案 https://github.com/apple/swift 裡公開進行。

Swift 2

Swift 2 在 WWDC 2015 上公開亮相，此後根據社群回饋，對大量的語法進行修改，並且進化程式語言的功能。其受歡迎的程度和社群接受度也跟著持續成長。

2016

Swift 3

Swift 3 發布於 2016 年 9 月，再次根據社群回饋， 帶來相當大量語法變化。這也預告 Swift 3 進入穩定的新時代，並且允諾未來會減少修改語法。

釋出 Swift 3 的同時，也推出了 iPad 版的 Swift Playgrounds。

本頁內文使用的槌子圖示來自於網站「Noun Project」。

Swift 未來發展

2017

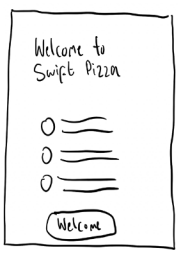
Swift 4

Swift 4 於 2017 年 9 月發布，不僅帶來全新功能，而且相較於之前的版本，穩定性更高。Swift 開放原始碼專案持續展露頭角，語言本身受歡迎的程度也不斷飆升。

2018

Swift 的設計宗旨在於吸納其他程式語言的精華，包含 Objective-C、Rust、Haskell、Ruby、Python、C# 等等語言。

2019



SwiftUI 於 2019 年的年中推出，支援 Swift 的使用者介面框架。

Swift 5

Swift 5 於 2019 年 3 月發布，向世界推出一個完整且穩定的版本，也是你在本書學習時會使用到的主要版本。

2020

光明的未來

Swift 的前景一片光明。Swift 6 很快就會問世，不僅保有 Swift 5 的語法和功能，還會增添令人興奮的全新內容。本書後續也會提到 Swift 5 剛採用的新功能，例如，Actor 模型。

2021

2020 年 2 月，Swift Playgrounds 發布 macOS 版的應用程式。

2022

那你要怎麼開始撰寫 Swift 程式

對，就是你！

每一種程式語言都需要一個工具，協助你設計程式和用來執行你所寫的程式。基本上，你要是不能執行一些 Swift 程式碼，就無法利用 Swift 一展長才。

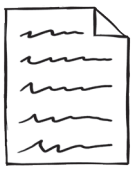
那麼，該怎麼做？本書大部分的內容會採用 Apple 開發的一項應用程式——**Playgrounds**！

Playgrounds 就像大型的舊式文字檔案（集合），不過，事實上，你也可以從 Swift 編譯器執行你所寫的每一行程式碼，和檢視執行的結果。

本書帶你撰寫的 Swift 程式，多數都會用到 Playgrounds。

但並非全部都是如此，本書後續會對此稍作解釋。

使用 Playgrounds 開發 Swift 程式會涉及三個階段：

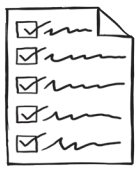


撰寫

❶

就與你在使用其他各種程式語言的情況一樣，Playgrounds 就是**程式碼編輯器**，你可以在這個環境下撰寫 Swift 程式碼。要將所有的程式碼放在同一個檔案裡，或是拆分到多個檔案裡，視你的選擇而定。

本書稍後會討論做法。

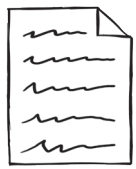


執行

❷

接著是在 Playground 介面下執行你的 Swift 程式碼，可以一次執行全部或是逐行執行。執行過程中發生的錯誤，以及程式碼的輸出結果，都會內嵌在每一行程式碼旁邊，也會顯示在程式碼編輯區域下的控制台裡。

在執行和微調之間切換，最快的方法是利用 Playground 介面，就能對目前進行的情況有清楚的了解。



微調

❸

在 Playground 介面下，撰寫 / 執行程式碼的週期非常快速，所以很容易進行微調、改善程式碼，並且發展成實現程式設計目標所需要的形式。開發速度是 Swift 的主要賣點之一，請聰明地使用！

可是我學習 Swift 是想撰寫**應用程式**，這樣我才能在 App Store 上分享我的想法！既然我現在可以在 **Xcode** 底下撰寫**應用程式**，Playgrounds 的意義何在？

利用 Playground 介面，可以更輕鬆地學習 Swift，效果更好。

在 Playground 介面下工作，表示你不需要擔心樣板程式碼，所有樣板都配合建立一種應用程式，可以將你撰寫的程式碼編譯成應用程式，協助你在 iOS 裝置模擬器和實體裝置上進行測試，以及在所有大同小異的 iOS 和 iPadOS 硬體上排除故障。

讓你專注在 *Swift* 程式碼上，不必擔心建立應用程式（或是撰寫網站後端程式、進行資料科學等等任何你想利用 Swift 完成的事）時，隨之而來的任何其他問題，這就是本書為何要從 Playgrounds 出發的原因。

這項工具會幫你摒除一切的干擾。

從各方面來看，Playgrounds 一點也不遜色：這項工具是真真切切的 Swift 程式碼，你能利用 Swift 開發**應用程式**等等一切的內容，還可以用來撰寫應用程式。

本書後續會討論這一塊。

如果你出現在這裡，是因為你想寫 iOS 應用程式，請透過 Playgrounds 堅持下去，這真的是最好的學習方式。

Xcode vs. Playgrounds

剛學 Swift 程式語言時，真的會很想直接跳進 Xcode 的懷抱裡，尤其是現役的程式設計人員。

如果你只是想學 Swift，但不想在學習過程中體驗 Xcode 環境非常強大又非常複雜的特性，就請從 Playgrounds 起步，它能讓你專注在 Swift 本身的學習上。

學習路徑

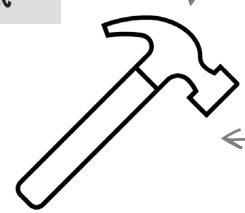
利用 Swift 撰寫應用程式之前，你必須先熟練 Swift 程式設計的方式。

Playgrounds

Playgrounds 是這條學習路徑的起點。適用於 *macOS* 和 *iPadOS*，讓你使用 *Swift* 編譯器寫出真正的 *Swift* 程式碼。在 *Playground* 介面下，可以快速檢視輸出結果、測試新想法，並且將工作切分成不同的邏輯頁面。我們再怎麼強調也不夠，只能說 *Playground* 介面下的所有一切都是真正的 *Swift*。

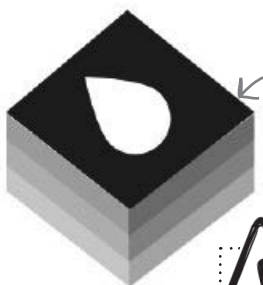
iOS、macOS、tvOS 和 watchOS 應用程式

等到你的 *Swift* 技能夠強大，這些技能就會順勢帶你離開 *Playground*，進入 *Xcode* 的懷抱。在 *Xcode* 底下，你還是能繼續使用 *Swift*，開發所有適用於 *Apple* 平台的應用程式。



在此學習過程中，你會學到 *SwiftUI*，這是 *Apple* 為 *Swift* 提供的使用者介面框架。我們會先在 *Playgrounds* 學習如何使用，後續再轉到 *Xcode* 的環境底下。

網站



在畫下完美的句點前，最後會帶你看看如何利用 *Vapor*、*Publish*、建立網站的框架和 *Swift* 後端程式，將 *Swift* 帶進網頁世界。

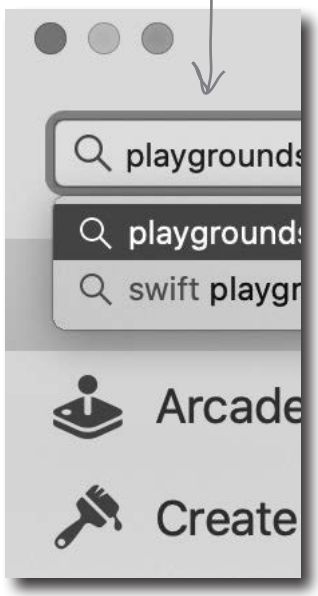
資料科學

直接跳進 Xcode 的懷抱裡可能很吸引人。

你可以自由決定是否要先安裝 Xcode，但現階段不會用到，本書之後才會回到這個部分。

取得 Playgrounds 應用程式

搜尋是最快找到應用程式的方式。



下載 Playgrounds 檔案，並且在 macOS 上安裝

1. 請在電腦上開啟 App Store。找到**搜尋欄**後，在欄位裡輸入「playgrounds」，然後按下鍵盤上的「Return」鍵。
2. 等 App Store 載入搜尋結果頁面後，請點擊結果列表中的項目「**Swift Playgrounds**」（圖示為橘色小鳥）。
3. 請點擊**安裝按鈕**，等待「Playgrounds」下載，並且安裝到電腦上。

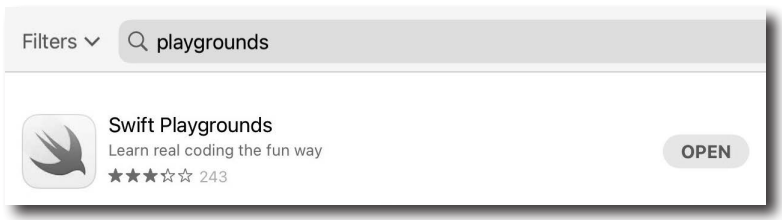
尋找 Swift 的橘色小鳥 LOGO。

安裝按鈕在這裡。

我是行動裝置控，情況允許的話，我會使用 iPad 處理一切事務。我發現 App Store 上有 iPadOS 版本的 Playgrounds，我可以改用這個版本，不要在 Mac 上安裝嗎？

當然可以，本書在 macOS 版 Playgrounds 下進行的一切內容，都可以在 iPadOS 版 Playgrounds 底下完成。

書中需要用到 Playgrounds 的所有操作，都可以在 macOS 或 iPadOS 上完成，也可以在兩者之間混合搭配使用。只要在 iPadOS 上的 App Store 搜尋 Playgrounds，就會帶你找到 iPadOS 版本的 Playgrounds，而且 iCloud 會幫助你同步 macOS 和 iPadOS 兩者之間的內容，所以從技術上來看，你可以同時使用這兩個版本的 Playgrounds ！



請在資料夾「Applications」下搜尋「Playgrounds」。

啟動 Swift Playgrounds 時，請瀏覽資料夾「Applications」，然後雙擊「Swift Playgrounds」的圖示。

如果你偏好用 Launchpad 或 Spotlight 來啟動其他應用程式，也可以用這個方式啟動「Swift Playgrounds」。如果是在 iPadOS 上，請尋找「Playgrounds」的圖示，或是透過 Spotlight 啟動。

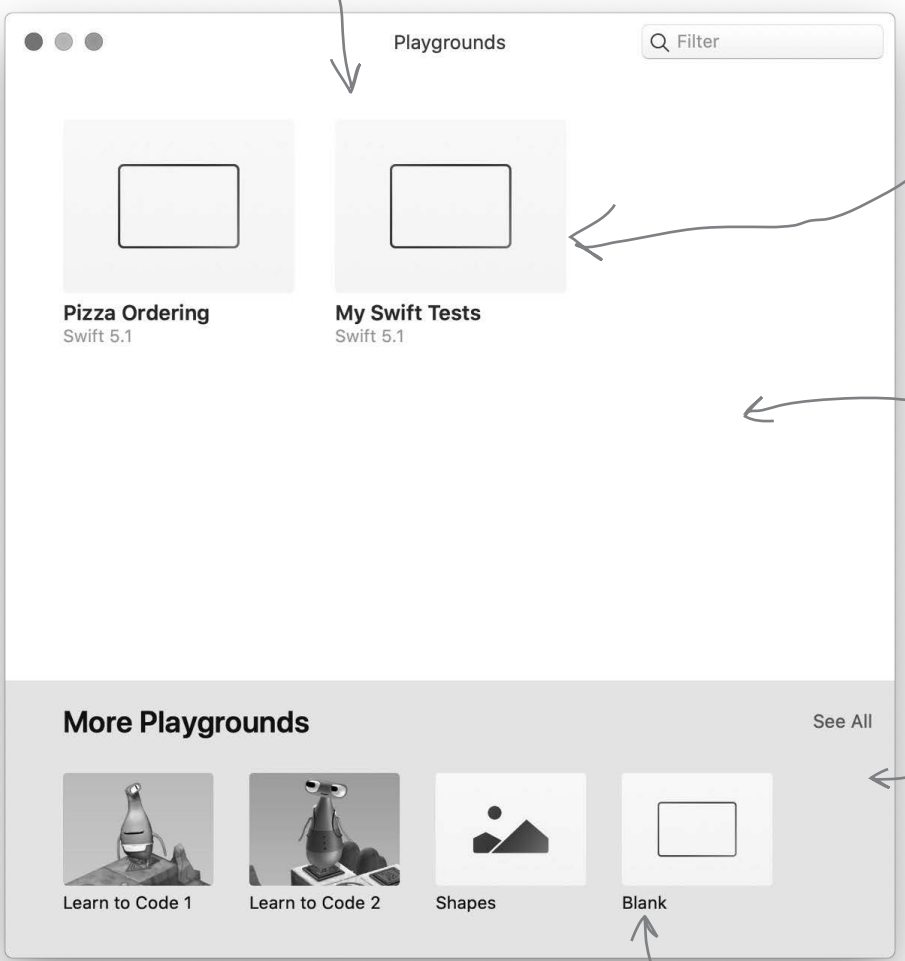
啟動 Playgrounds 的方式有：透過 Spotlight 或是在資料夾「Applications」下尋找。

重點提示

- 撰寫 Swift 程式需要一個或多個不同的工具，目的是讓你撰寫、執行和微調程式碼（再次執行程式碼前，會多次進行微調）。
- Apple 推出的應用程式「Playgrounds」就屬於這樣的工具，適用於 iPadOS 和 macOS。這項工具的進入門檻很低，讓使用者專注在 Swift 程式語言本身，不需要同時學習工具和語言兩者。
- Apple 還有提供另外一個更複雜的工具——Xcode，屬於完全整合的開發環境，類似 Visual Studio 等其他工具，是一個非常大型又非常複雜的工具，具有多種用途。
- 如果你想為 Apple 平台（例如，iOS）撰寫更複雜的應用程式，就必須使用 Xcode，本書後續會介紹 Xcode 的使用方式。
- Playgrounds 是學習 Swift 程式語言的最佳工具，甚至能讓你撰寫應用程式！
- 書中所有專為 Playgrounds 設計的內容，可以在 iPadOS 或 macOS 其中一個、或是同時在兩者上執行，端視你的選擇。

新增你的第一個 Playground 專案

啟動 Swift Playgrounds 後，請新增一個空白的 Playground 專案作為工作專案，或是開啟已經建立的現有 Playground 專案。



使用 Playground 專案撰寫 Swift 程式碼

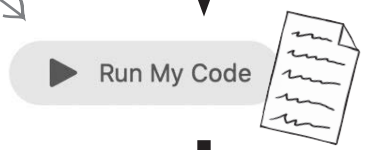
❶ 新增一個 Playground 專案

在 Playgrounds 應用程式裡新增一個 Playground 專案。

❷ 撰寫一些程式碼，然後執行

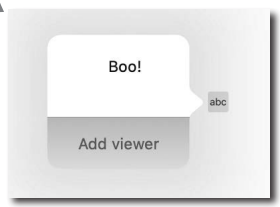
撰寫程式碼（視你的需求而定，寫多寫少都沒關係），然後點擊執行程式碼的按鈕。

如果你使用的 Playgrounds 視窗較小，「Run」這個按鈕可能會變小。

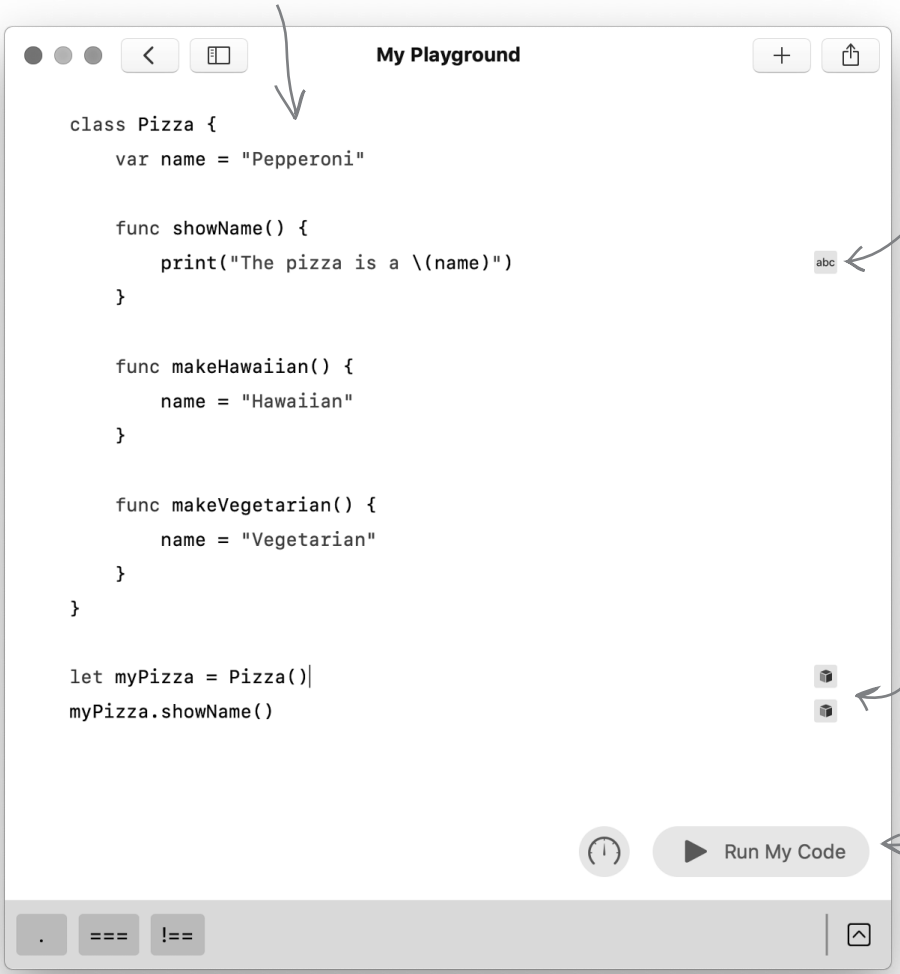


❸ 檢視輸出結果

程式碼執行完畢後，具有輸出值的行數會顯示結果圖示，點擊該圖示可以加入檢視器。



Playground 會以令人舒適的顏色標出程式碼，讓使用者易於閱讀。



如果某塊 Swift 程式碼具有輸出結果，會看到該行程式碼旁邊出現一個小圖示。

點擊結果圖示可以新增一個檢視器，用以顯示特定行數的輸出值或是結果。

使用這個按鈕可以執行程式碼。

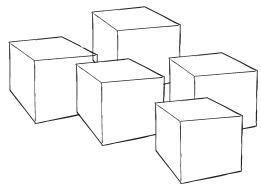
新車試駕

來吧，開始動手囉。請自行建立一個全新的 Swift Playground 專案，並且輸入 Playground 的內容。

然後執行。看看會顯示什麼？使用結果圖示新增檢視器，檢視執行的結果。

建立程式碼的基本組件

本書的 Swift 之旅還會持續一段時間，但我們想先讓你快速掌握基礎知識。我們將快速帶過 Swift 的基礎知識，讓你體驗這項程式語言的許多範疇，之後再回過頭來，以逐章介紹的方式，一一解開 Swift 每項功能的神秘面紗。



在這個地球上，幾乎所有的程式語言都是由某些基本組件建構而成（搞不好連地球以外的任何一種程式語言也是如此，畢竟程式設計可是相當普遍）。

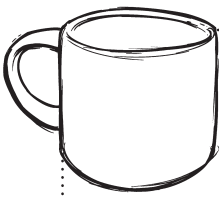
這些建立程式碼的基本組件是由**運算子**、**變數**、**常數**、**資料型態**、**集合**、**控制流**、**函式**、**程式協定**、**列舉**、**結構**和**類別**所組成。

連連看？

雖然本書尚未介紹每一種基本組件，但你可能已經猜到。請將下列每一種建立程式碼的基本組件與其正確的作用進行配對，解答請參見第 27 頁（我們已經先完成一題，讓你有個好的開始）。

運算子（operator）	命名的重複程式碼片段
變數（variable）與常數（constant）	符號或片語，用於檢查、修改或組合資料值
資料型態（type）	以名稱儲存資料的方式
集合（collection）	程式採用的功能藍圖
控制流（control flow）	某種用於儲存或表示資料的類型
函式（function）	儲存群組資料
列舉（enumeration）	將有關聯的資料值和功能性分組
結構（structure）	對有關聯的資料值和功能性進行分組，也能從其他群組繼承這些內容
程式協定（protocol）	多次或在特定條件下執行任務的方式
類別（class）	將有關聯的資料值分組

解答請見第27頁。



放輕鬆

本書不會期望你像專家那樣閱讀 Swift 程式碼、記住 Swift 的發展史，甚至大放厥詞，好像你曾經參與過 Swift 的研發工作。

然而，既然你把辛苦賺來的血汗錢花在這本書上，在你沒有將我們輕鬆分解成一塊塊的 Swift 知識塞進你的腦袋前，我們是不會放你離開。現階段你或許還無法全盤了解這些知識，但隨著時間增長（你或許會說很快地），你就會認知到這些基本的建構組件如何相輔相成。

還記得前幾頁的程式碼嗎？接下來我們要帶你看看這段程式碼，了解它想要做什麼。

建立一個類別，名稱為 Pizza。把 Pizza 的各個相關面向放在同一個群組裡，之後就能建立 Pizza 這個類別的實體（instance），操作這些面向。

```
class Pizza {
    var name = "Pepperoni"

    func showName() {
        print("The pizza is a \(name)")
    }
    func makeHawaiian() {
        name = "Hawaiian"
    }

    func makeVegetarian() {
        name = "Vegetarian"
    }
}

var myPizza = Pizza()
myPizza.showName()
```

建立一個變數，名稱為 name，並且指定變數值為「Pepperoni」。

建立一個函式，名稱為 showName，負責印出名稱，也就是變數 name 的內容。

「print」負責通知 Swift 將某些內容顯示給使用者看，我們很快就會對此有更多介紹。

建立一個函式，名稱為 makeHawaiian，用於將字串「Hawaiian」指定給變數 name。

一樣是建立一個函式，不過是用於將字串「Vegetarian」指定給變數 name。

這個括號前的所有內容都屬於類別 Pizza。

建立一個變數，名稱為 myPizza，包含類別 Pizza 的實體。

以類別 Pizza 所建立的實體（myPizza）來呼叫函式 showName。

不好意思…但是，在我使用過的其他程式語言裡，許多程式語言都必須將所有程式碼包在一個方法裡，例如，main 或是類似的方法？Swift 怎麼處理這個部分？

至少，你不需要寫main方法，程式碼只會在Playgrounds的環境中執行。

你不需要 main 方法或是任何其他符號（例如，分號）。

你可以依照自己喜歡的方法開始寫 Swift 程式碼，在 Playgrounds 的環境裡，目前的做法是從檔案的上方開始執行，然後由上到下依序進行處理。

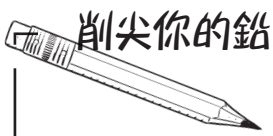
某些基本的建構組件（例如，類別），不會立刻讓你看見它們的執行方式，因為這類基本組件需要依靠實體，而實體會在程式碼的其他地方建立。一般來說，在 Playgrounds 的環境裡，你不需要 main 方法或是任何特定的起點。

之後從 Playgrounds 轉為使用 Swift 建立應用程式時，本書會再介紹應用程式裡可能會有的各種程式碼起點。不過，現在你只需要將這件事從你腦海中抹去。

你也不需要在每一行程式碼的結尾加上分號，空白字元對任何內容完全沒有作用。

如果你是從其他程式語言轉學過來的，可能已經習慣用分號（;）作為程式碼的結尾，或是使用對程式碼邏輯具有某些實質意義的空白字元。在 Swift 程式語言裡，這些做法並不重要，也不適用於此處。

糟糕的是，如果你想念這些來自其他程式語言的分號，還是可以在陳述式的結尾使用，但本書不建議你這麼做。這種做法不會更快，也不會讓你手上正在寫的 Swift 程式碼更穩固。



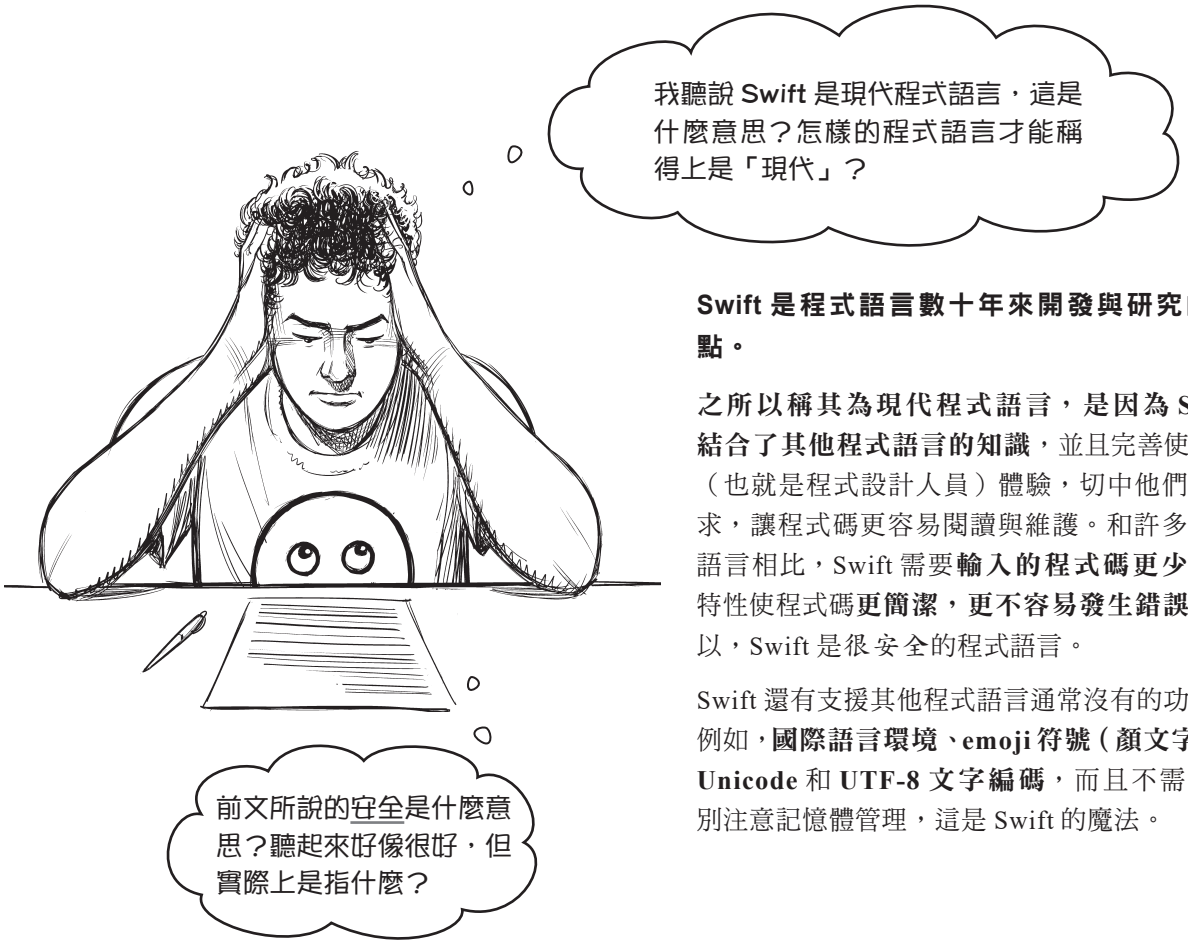
削尖你的鉛筆

雖然你才剛開始學習 Swift，但我們覺得你是個聰明人，或許對某些 Swift 的程式碼有相當的理解，能猜到這些程式碼會發生什麼作用。請檢視以下這份 Swift 程式的程式碼，看看你是不是能搞清楚每一行程式碼在做什麼。在右側的空白行處寫下你的答案，本書已經先幫你完成一行，讓你有個好的開始！如果你想檢查答案或是毫無頭緒，可以在第 19 頁找到完整版本的答案。

```
class Message {
    var message = "Message is: 'Hello from Swift!'"
    var timesDisplayed = 0

    func display() {
        print(message)
        timesDisplayed += 1
    }

    func setMessage(to newMessage: String) {
        message = "Message is: '\(newMessage)'"
        timesDisplayed = 0
    }

    func reset() {
        timesDisplayed = 0
    }
}

let msg = Message()
msg.display()
msg.timesDisplayed
msg.display()
msg.timesDisplayed
msg.setMessage(to: "Swift is the future!")
msg.display()
msg.timesDisplayed
```

宣告一個變數，變數名稱為message，並且指定一個字串值給這個變數。

解答請見第 19 頁。



Swift 是程式語言數十年來開發與研究的頂點。

之所以稱其為現代程式語言，是因為 Swift 結合了其他程式語言的知識，並且完善使用者（也就是程式設計人員）體驗，切中他們的需求，讓程式碼更容易閱讀與維護。和許多程式語言相比，Swift 需要**輸入的程式碼更少**，其特性使程式碼**更簡潔，更不容易發生錯誤**，所以，Swift 是很安全的程式語言。

Swift 還有支援其他程式語言通常沒有的功能，例如，**國際語言環境、emoji 符號（顏文字）、Unicode** 和 **UTF-8 文字編碼**，而且不需要特別注意記憶體管理，這是 Swift 的魔法。

Swift 很安全是因為它會自動處理很多事情，而這些在其他程式語言裡是需要寫程式碼才能處理，或者根本就是挑戰。

Swift 常被人說很安全的理由很多，例如，變數在使用前一定會進行初始化，陣列一定會檢查溢位，還有自動管理程式使用的記憶體。

此外，Swift 相當依賴資料值的型態，採取複製資料型態而非引用，所以當你將資料值挪作他用時，值本身的內容完全不會被任何地方改掉。

Swift 物件永遠不可能出現 nil（空值），有助於避免程式在執行期間發生異常中斷的情況，不過，只要你的程式有需要，還是可以透過 **Optional 型態**使用 nil。

nil 是指空值，你可能已經在其他程式語言裡看過它，或者你看過的是 null。不管是哪種程式語言，程式設計過程中引發異常中斷的一大原因，就是程式試圖存取一個不存在的值；這些值通常就是 nil。

削尖你的鉛筆 解答

題目請見第 17 頁。

你不需要擔心自己現在還無法全盤了解以下這些程式碼！本書會非常詳盡地解釋所有的程式碼，大部分的內容會落在前 40 頁內。如果 Swift 跟你過去使用的程式語言相似，其中一些程式碼對你來說應該很簡單；如果不相似，你也不需要擔心，我們會一起走向成功。

```
class Message {
    var message = "Message is: 'Hello from Swift!'"
    var timesDisplayed = 0

    func display() {
        print(message)
        timesDisplayed += 1
    }

    func setMessage(to newMessage: String) {
        message = "Message is: '\(newMessage)'"
        timesDisplayed = 0
    }

    func reset() {
        timesDisplayed = 0
    }
}

let msg = Message()
msg.display()
msg.timesDisplayed
msg.display()
msg.timesDisplayed
msg.setMessage(to: "Swift is the future!")
msg.display()
msg.timesDisplayed
```

建立一個類別（大括號內的所有內容），名稱為 *Message*。

宣告一個變數，名稱為 *message*，並且指定一個字串值給這個變數。

宣告一個變數，名稱為 *timesDisplayed*，並且指定一個整數值給這個變數。

宣告一個函式，名稱為 *display*。函式會印出變數 *message* 的值，變數 *timesDisplayed* 的值會加 *1*。

宣告一個函式，名稱為 *setMessage*，接受一個字串參數 *newMessage*。函式會更新變數*message* 的字串值，包含參數 *newMessage* 傳進來的內容，還有將變數 *timesDisplayed* 的值設為 *0*。

宣告一個函式，名稱為 *reset*，函式會將變數 *timesDisplayed* 的值設為 *0*。

Message 類別宣告內容的結尾。

建立 *Message* 類別的實體，並且將其儲存在常數 *msg*。

這四行程式碼分別是以 *Message* 類別的實體 *msg* 呼叫 *display* 函式，存取變數 *timesDisplayed*，然後再次呼叫 *display* 函式和存取變數 *timesDisplayed*。

以 *Message* 類別的實體 *msg* 呼叫 *setMessage* 函式，傳入一個字串作為函式的參數。

以 *Message* 類別的實體 *msg* 呼叫 *display* 函式。

這個表達式是以 *Message* 類別的實體 *msg* 存取變數 *timesDisplayed*。

Swift 範例程式

學習 Swift 的一切是很硬的工作，所以我們需要披薩！正巧當地的披薩店聽說你正在學 Swift，於是他們希望你將新技能學以致用。