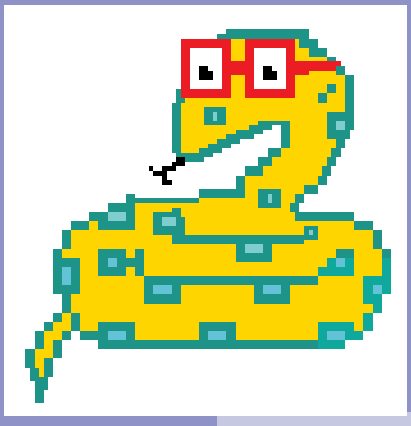
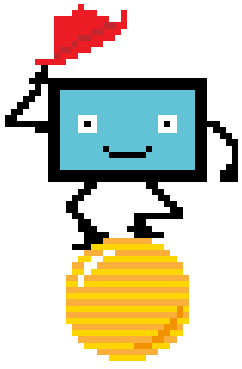
Python
新手教學



碁峰
www.gotop.com.tw

什麼是「寫程式」?

我們常聽到的電腦程式設計師（computer programmer）或是「寫程式的人」（coder），都是指那些寫出指令，讓電腦一步步執行工作的人，他們能讓電腦進行計算、製作音樂、讓機器人在空間裡四處移動或是讓火箭飛向火星。



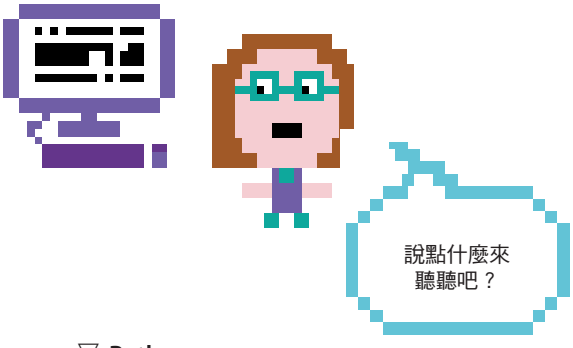
△ **訓練寵物**
學會寫程式就能自己設計程式，讓電腦幫我們完成想做的事。這有點像養了一隻電子寵物後，要訓練它會一些才藝！

無聲的箱子

電腦不會主動做任何事，除非有人告訴它確實要執行的工作，否則就只是一個靜置在空間裡、不會發出任何聲響的箱子。由於電腦不會主動思考，只會等著人來告訴它要做什麼，因此，寫程式的人必須代替電腦思考，仔細將電腦要執行的指令寫下來。

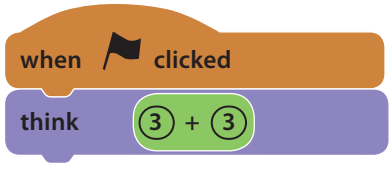
程式語言

為了告訴電腦我們想讓它做的事，就必須學習能和電腦溝通的程式語言。圖形化程式語言對初學者來說容易學習，專業的程式設計師則會使用以文字指令為基礎的程式語言。本書內容採用目前最熱門的文字式程式語言—Python。



▽ **Scratch**
Scratch 屬於圖形化程式語言，它的強項是創作遊戲、動畫和互動式故事。在 Scratch 的開發環境下，寫程式就是把「積木指令」拼在一起。

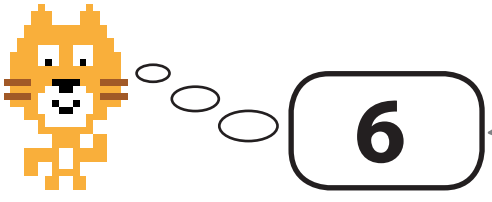
▽ **Python**
Python 是以文字指令為基礎的程式語言。在 Python 的開發環境下，程式設計師寫程式是用英文單字、縮寫、數字和符號，以電腦鍵盤輸入程式指令。



左右兩邊程式碼做的事都一樣。

```
>>> 3 + 3
6
```

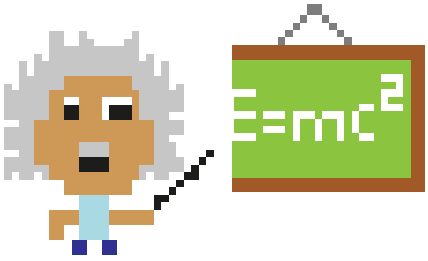
按下『enter / return』鍵就能看到數字相加的計算結果。



數字相加的計算結果會顯示在螢幕上的「思考」泡泡裡。

人人都能寫程式

想成為「寫程式的人」，你只需要學幾個基本規則和指令，就能開始寫一些能跟自身技能和興趣配合的程式。例如，從事科學研究的人可以開發應用程式，將實驗結果繪製成圖表；具備美術能力的人可以設計一個外星世界，創造一款屬於自己的電子遊戲。

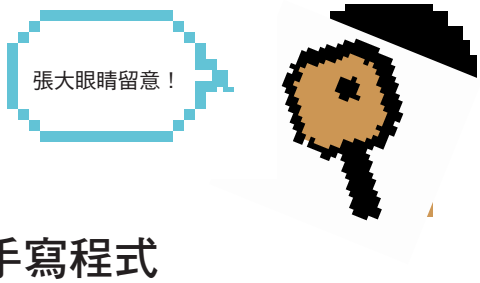


▽ 邏輯思考

「寫程式的人」必須具有邏輯思考能力和細心的態度，才能寫出好程式。如果程式碼裡的指令不完全正確或步驟的順序錯了，程式就無法運作。因此，請仔細思考每個步驟，確定它們發生的順序符合邏輯，這就像你會先穿內褲，才穿上褲子，不是嗎？

▽ 注意細節

如果你很擅長玩「大家來找碴」這類的解謎遊戲，或許能成為優秀的程式人，這是因為寫程式時，一項必備的重要技巧是找出程式碼裡的錯誤。這些錯誤通常稱為「臭蟲」（Bug），即使是程式碼裡十分微小的錯誤也會導致嚴重的問題。眼力好的程式人能憑指令的邏輯或順序，挑出程式碼裡的拼字錯誤或設計上的缺失。為程式除錯是非常棘手的工作，但想提升程式能力，從錯誤中學習是最棒的方法。



知識補給站

程式裡的臭蟲（Bug）

「程式裡的臭蟲」是指存在程式碼裡的錯誤，造成程式的表現異常。之所以稱為臭蟲，是因為早期的電腦有時會因為昆蟲跑進電路裡而發生故障。

英文小教室

本書幽默地將「bug hunt」設計成雙關語：抓出程式錯誤／抓「臭蟲」。

動手寫程式

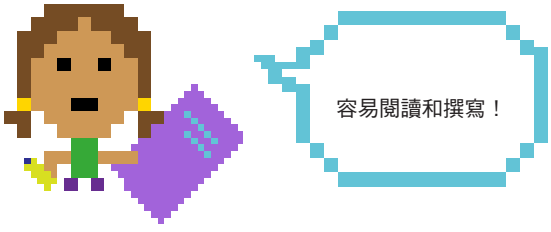
許多人一聽到寫程式，就覺得是一項艱難的任務，但學習寫程式其實相當容易，秘訣就是投入熱情，動手去做。本書設計的程式教學內容是從簡單的範例起始，逐步引導你寫程式的方法，只要跟著書中精心編排的步驟實做，你也能立刻創作出遊戲、應用程式和數位藝術。

認識 Python

Python 是全世界最熱門的電腦程式語言之一，自 1990 年代釋出第一版以來，Python 逐漸開發出百萬個應用程式、遊戲和網站。

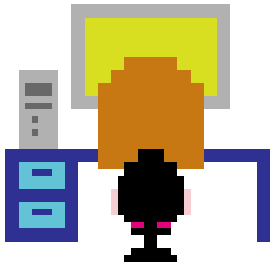
本書選擇 Python 的理由

對剛開始學習電腦程式設計的初學者來說，Python 是很棒的入門語言。許多學校和大學都已經採用 Python 作為程式設計課程的教學內容，以下是 Python 會如此實用的幾個原因。



△ 容易閱讀與撰寫

Python 是以文字指令為基礎的電腦程式語言，這些指令是由英文單字、標點符號、記號和數字所組成。這樣的特性使得 Python 語言的程式碼更容易讓人閱讀、撰寫和理解。



△ 隨處可用

Python 程式的可攜性高，意思是我們能在各種不同的電腦上撰寫和執行 Python 的程式碼，同樣的程式碼在個人電腦、Mac 系統、Linux 環境的機器和迷你電腦 Raspberry Pi 上都能正常運作，不論在哪臺機器上，程式表現的效果都一樣。

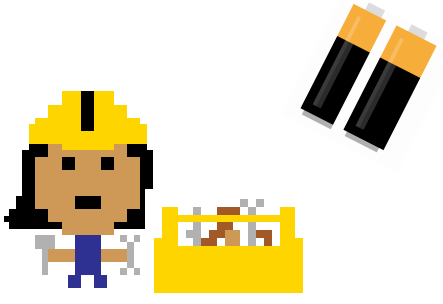
知識補給站

Python 命名小故事

Python 的原意是一種蛇——巨蟒，但 Python 語言的創始人 Guido van Rossum 選中『Python』作為程式語言的名稱其實和原意無關，真正的原因是因為他是英國搞笑團體「Monty Python's Flying Circus」的忠實粉絲，喜歡他們獨特另類的幽默，才以此命名。Python 程式設計師經常把這個搞笑團體的笑話和有名的語錄用在程式碼裡，以此致敬。

▽ 備有電池

程式設計師說 Python 是一個「備有電池」的程式語言，這是因為 Python 本身功能完善，安裝之後就能立即開始進行程式設計。

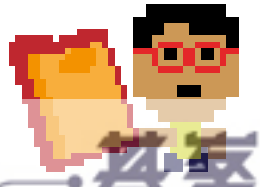


△ 便利的工具

Python 搭載了大量的便利工具和一些現成就能使用的程式碼，我們稱為標準函式庫（Standard Library）。有了這些便利的工具，人人都能更輕鬆、快速地建立自己的程式。

▷ 完善的支援

Python 提供了完善的說明文件，引導初學者入門，包含檢索相關知識的參考資料和大量的範例程式碼。



現實生活中的 Python

Python 不只用於程式教育，它強大的程式能力還能用在許多令人興奮和有趣的工作上，例如，商業、醫療、科學和娛樂媒體等領域，甚至還能用來控制家中的燈光和暖器設備。

程式高手秘笈

直譯器（Interpreter）

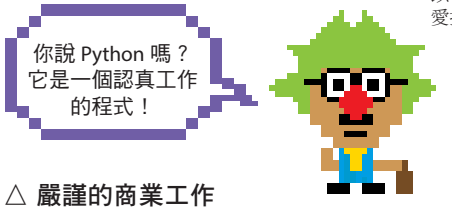
有些程式語言會利用直譯器這種程式，將一種程式設計語言轉換成另一種語言。每當我們執行 Python 程式時，Python 內建的直譯器就會將每一行 Python 程式碼轉換成電腦能理解的特殊程式碼，稱為機器碼。

▽ 在網路上搜尋資訊

Python 現已廣泛地應用在網路上，Google 搜尋引擎的部分程式碼就是以 Python 撰寫而成，連 YouTube 也是使用 Python 建構大部分的程式碼。

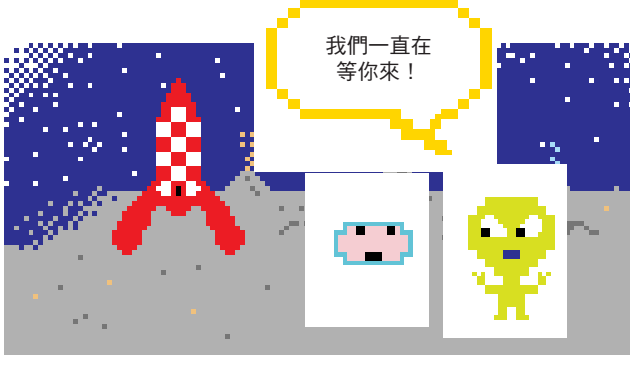
英文小教室

Python 的原意是大蟒蛇，所以本書加入了一些蟒蛇的可愛插圖。



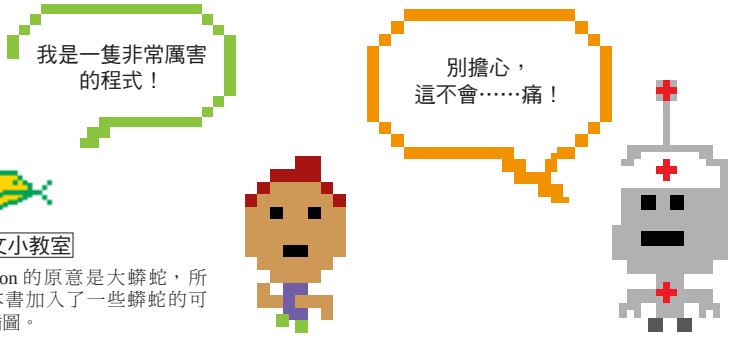
△ 嚴謹的商業工作

Python 能協助銀行追蹤每個銀行帳戶底下的錢，幫助大型連鎖商店設定每件販售商品的價格。



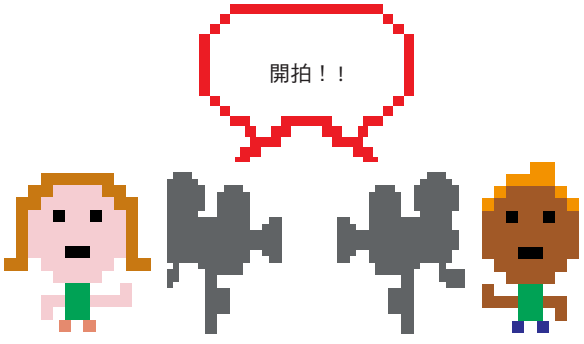
△ 飛出地球

軟體工程師利用 Python 語言，為美國太空總署的任務控制中心開發相關工具，幫助控制中心的工作人員進行每項任務的準備工作與監控每項任務的進度。



△ 創造醫療奇蹟

Python 能用於設計醫療機器人，執行棘手的手術工作。以 Python 語言為架構所設計出來的手術機器人，在執行醫療工作上不僅比人類更有效率而且更精準，能避免一些人為疏失。



△ 製作電影

迪士尼動畫公司利用 Python 程式，自動產生動畫製作過程中重複的部分。原本需要動畫師重複執行相同步驟才能繪製的內容，現在只要利用 Python 程式，就能自動執行這些步驟，省去動畫師大量的工作，大幅縮短影片製作的時間。

安裝 Python

本書所有範例程式都是在 Python 3 的環境下完成，因此，請檢查你從網站上下載的安裝檔案是否為正確版本。請配合你的電腦環境，依照指示安裝 Python。

在 Windows 系統上安裝 Python

在電腦上安裝 Python 3 之前，請先確認電腦使用的 Windows 版本是 32 位元還是 64 位元。以滑鼠左鍵點擊 Windows 的「開始」功能表，然後移到「電腦」上按滑鼠右鍵，會彈出下拉選單，點選「內容」就能看到版本資訊；若有出現選項，請選擇「系統」。

知識補給站

Python 內建的 IDLE 工具

安裝 Python 的同時，還會安裝一個免費應用程式——整合開發環境工具（Integrated Development Environment，簡稱 IDLE）。IDLE 是 Python 專為初學者而設計的工具，包含一個基本的文字編輯器，使用者可以利用這個編輯器撰寫和編輯 Python 的程式碼。

1 請上 Python 官方網站

在瀏覽器網址列輸入以下網址，前往 Python 官網。以滑鼠左鍵點擊首頁的『Downloads』（下載），進入安裝程式的下載頁面。

- https://www.python.org/

2 下載 Python 安裝程式

在安裝程式的下載頁面裡，點擊最新版本的 Python 3 安裝程式，版本號碼的開頭是 3。瀏覽器會自動下載安裝程式的檔案，請選擇適合 Windows 版本的「安裝程式執行檔」（executable installer）。

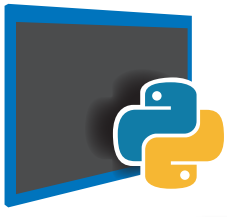
- Python 3.6.0a4 - 2016-08-15
  - Windows x86 executable installer
  - Windows x86-64 executable installer

這是給 32 位元 Windows 使用的安裝程式。

這是給 64 位元 Windows 使用的安裝程式。

3 執行安裝程式

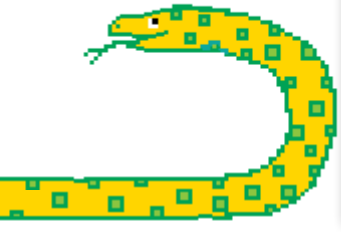
雙擊安裝程式檔，開始安裝 Python。勾選畫面上的『install for all users』（為所有使用者安裝），不要改變安裝的預設選項，只要按『next』（下一步）。



以滑鼠左鍵點擊安裝程式檔。

4 開啟 IDLE

安裝完成後，請確認是否能成功開啟 IDLE 這項程式工具。以滑鼠左鍵點擊 Windows 的「開始」功能表，選擇『所有程式』，然後點選『IDLE』，應該會開啟以下這樣的視窗。



```
Python 3.6.0a4 Shell
IDLE  File  Edit  Shell  Debug  Window  Help
Python 3.6.0a4 (v3.6.0a4:017cf260936b, Aug 15 2016, 00:45:10) [MSC v.1900 32
bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
```

在 Mac 系統上安裝 Python

在 Mac 電腦上安裝 Python 3 之前，請先確認電腦使用的作業系統版本。以滑鼠左鍵點擊螢幕左上方的「蘋果」圖示，選擇下拉選單裡的『關於這台 Mac』，即可取得系統資訊。

1 請上 Python 官方網站

在瀏覽器網址列輸入以下網址，前往 Python 官網。以滑鼠左鍵點擊首頁的『Downloads』（下載），進入安裝程式的下載頁面。

https://www.python.org/

2 下載 Python 安裝程式

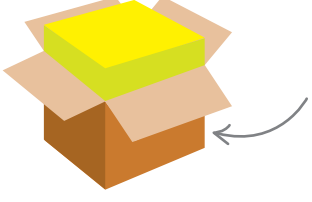
在安裝程式的下載頁面裡，請配合電腦的作業系統點擊最新版本的 Python 3 安裝程式，檔案『Python.pkg』會自動下載到 Mac 電腦裡。

- Python 3.6.0a4 - 2016-08-15
  - Download macOS X 64-bit/32-bit installer

你下載的最新版本不一定和本書使用的版本完全相同，只要確定下載的版本號碼開頭是 3 即可。

3 安裝 Python

在『下載項目』檔案夾裡找到剛剛下載的檔案『Python.pkg』，檔案圖示看起來像是一個打開的包裹，以滑鼠左鍵雙擊檔案就會開始安裝 Python。按下安裝提示畫面上的『Continue』（繼續），使用預設的安裝選項，然後按下『Install』（安裝）。



以滑鼠左鍵點擊「.pkg」檔案，即可執行安裝程式。

4 開啟 IDLE

安裝完成後，請確認是否能成功開啟 IDLE 這項工具程式。開啟『應用程式』檔案夾下的『Python』檔案夾，使用滑鼠左鍵雙擊『IDLE』，應該會出現以下這樣的視窗。

重要

取得安裝軟體的許可

當你想在一臺電腦上安裝 Python 或其他程式，請先取得電腦擁有者或管理者的同意。在安裝程式的過程中，可能還需要請擁有者或管理者提供管理者權限的密碼。

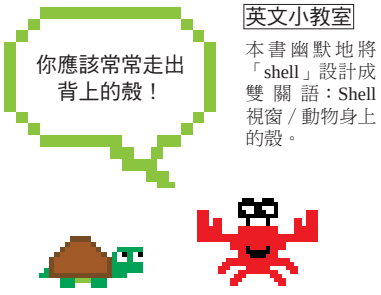
Python 3.6.0a4 Shell

IDLE File Edit Shell Debug Window Help

```
Python 3.6.0a4 (v3.6.0a4:017cf260936b, Aug 15 2016, 13:38:16)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "copyright", "credits" or "license()" for more information.
>>>
```

使用 IDLE 工具

Python 內建的 IDLE 工具有兩種工作視窗。「Shell 視窗」（shell window）能直接執行我們輸入的 Python 指令，「編輯視窗」（editor window）則只能撰寫程式碼，儲存為檔案。



英文小教室

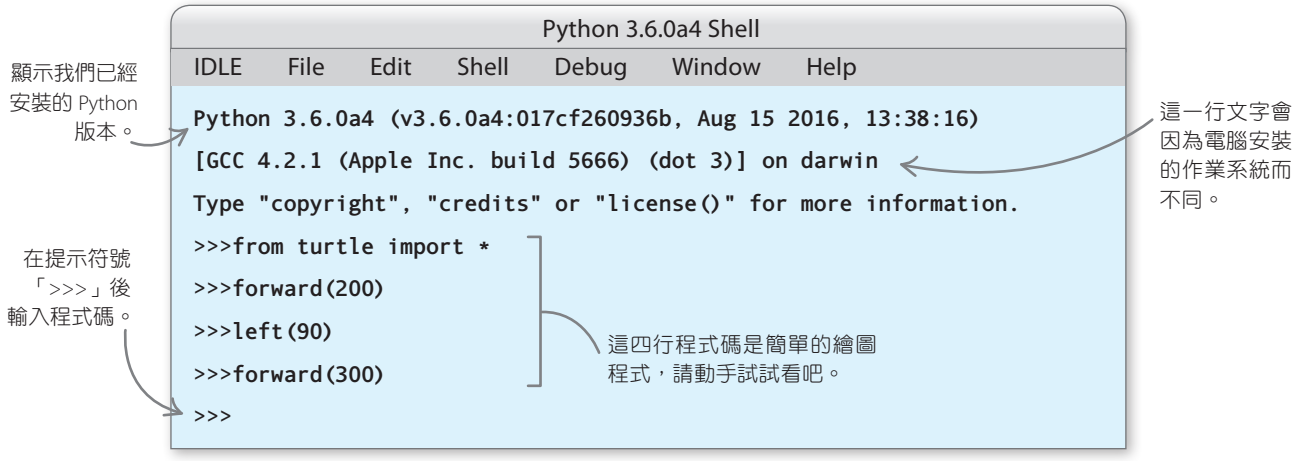
本書幽默地將「shell」設計成雙關語：Shell 視窗 / 動物身上的殼。

Shell 視窗

執行 Python 內建的 IDLE 工具後，會開啟 Shell 視窗。這是最適合初學者的開發環境，剛開始練習時，不必建立任何新檔案，直接在 Shell 視窗下輸入程式碼就能看到結果。

▽ **在 Shell 環境下工作**

Shell 程式會立即執行我們輸入的程式碼，並且顯示訊息或「臭蟲」（錯誤），所以，能利用 Shell 視窗測試一小部分的程式碼，再把它們加到更大的程式裡。



程式高手秘笈

以顏色表示不同的視窗

本書以下列兩種顏色分別表示 IDLE 工具的兩種工作視窗，方便我們知道要在哪一個視窗下輸入程式碼。

Shell 視窗

編輯視窗

▽ **動手試試看**

請在 Shell 視窗下輸入以下這幾行程式碼，每輸入完一行就按下『enter / return』鍵。第一行程式碼會顯示一句訊息，第二行會計算數學式，你知道第三行會做什麼嗎？

```
>>> print('I am 10 years old')
```

```
>>> 123 + 456 * 7 / 8
```

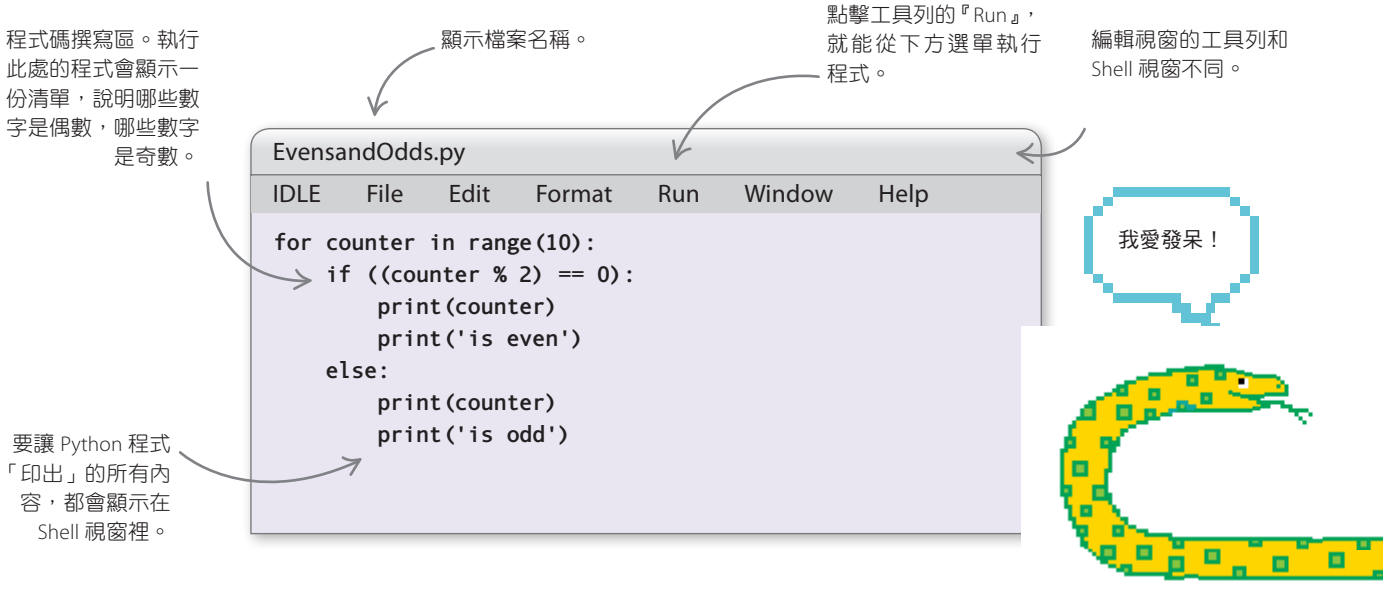
```
>>> ''.join(reversed('Time to code'))
```

編輯視窗

Shell 視窗無法儲存我們輸入的程式碼，因此，關閉視窗後，先前輸入的程式碼會永遠遺失，這就是開發專案時應該選擇編輯視窗的原因。編輯視窗不僅能儲存程式碼，還有一些內建工具可以幫助我們撰寫程式和解決問題。

▽ 開啟編輯視窗

想開啟 IDLE 工具的編輯視窗，請點擊 Shell 視窗上方工具列中的『File』（檔案），選擇『New File』（新增檔案），就會出現一個空白的編輯視窗。我們要在這個視窗下撰寫和執行本書的範例程式。



程式高手秘笈

程式碼的文字顏色

IDLE 工具會自動以不同顏色顯示程式碼裡各個部分的文字，文字顏色不僅能讓我們更容易理解程式碼的內容，還有助於發現錯誤。

◁ 符號和名稱
程式碼中大部分的文字都屬於這一類，顯示為黑色。

◁ 關鍵字
Python 使用的某些特殊關鍵字會顯示為橘色，例如，條件式『`if`』和『`else`』。

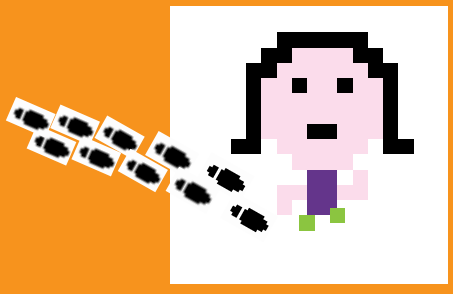
◁ 輸出結果
執行程式碼後產生的任何文字都會顯示為藍色。

◁ 內建命令
Python 內建命令，例如，函式『`print`』會顯示為紫色。

◁ 錯誤訊息
程式碼發生任何錯誤時，Python 會以紅色文字顯示錯誤訊息。

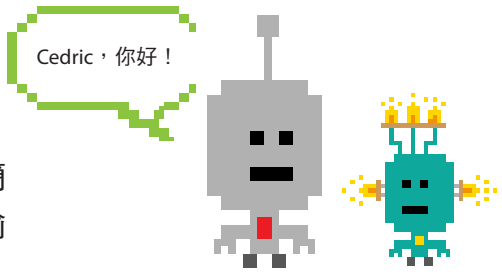
◁ 單引號中的文字
單引號本身和單引號包圍的文字都會顯示為綠色，能幫助我們檢查是否缺少其中一個引號。

Python 新手的第一步



碁峰
www.gotop.com.tw

第一個 Python 程式



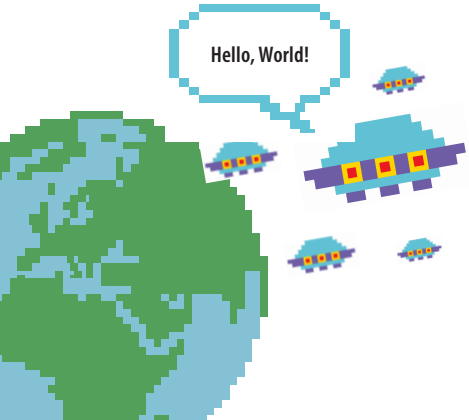
我們在第一章已經學過怎麼安裝 Python 和 IDLE 工具，現在要開始寫第一個 Python 程式。只要跟著以下這些簡單的步驟，馬上就能完成簡單的程式，在畫面上顯示愉快的訊息，向使用者打招呼。

程式技巧

執行範例程式後，我們首先會在畫面上看到訊息「Hello, World!」，接著，程式會問我們叫什麼名字，輸入名字後，程式會再次跟我們打招呼，不同的是，這次它在問候訊息裡加上了名字，這是因為它利用「變數」（variable）這項技巧，記下我們輸入的內容，所以「變數」能用來儲存程式需要的資訊。

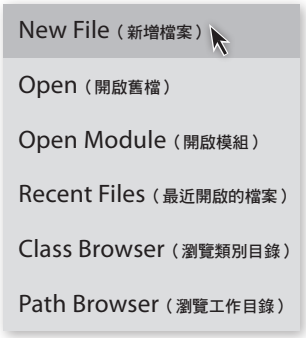
▷ 程式流程圖

程式設計師利用「流程圖」規劃和說明程式的運作方式。流程圖裡的每個方格都是一個步驟，每個步驟之間以箭頭連接。流程圖裡的步驟有時會是一個需要判斷答案的問題，根據結果會有多個箭頭指向不同的步驟。



1 執行 IDLE

執行 IDLE 工具後，會開啟 Shell 視窗，先暫時不要管它，點擊工具列的『File』（檔案），選擇『New File』（新增檔案），開啟空白編輯視窗，在這個視窗裡寫程式。



2 輸入第一行程式碼

請在編輯視窗裡輸入以下這行文字。其中單字『print』是 Python 的內建指令，負責告訴電腦要在螢幕上顯示哪些內容，例如，範例程式裡的文字「Hello, World!」。

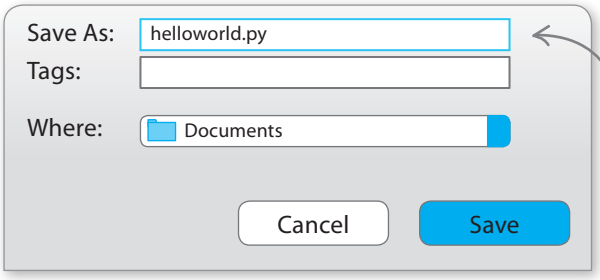
```
print('Hello, World!')
```

3 儲存檔案

執行程式碼之前，必須先儲存成檔案。請點擊『File』（檔案），選擇『Save』（儲存檔案）。

4 儲存 .py 檔

請在跳出的對話框裡輸入程式名稱，例如，「helloworld.py」，然後按下『Save』（儲存檔案）。



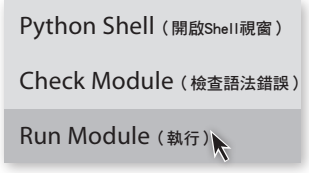
知識補給站

.py 檔

Python 程式名稱的副檔名通常是『.py』，方便我們辨識哪些檔案是 Python 程式。因此，當我們儲存程式檔案時，Python 會自動加上『.py』的副檔名，不用再自己手動輸入。

5 確認程式是否能正常運作

我們現在要執行剛剛寫的第一行程式碼，看看是否能正常運作。請點擊工具列的『Run』（執行），選擇『Run Module』（執行程式），Shell 視窗裡應該會出現一行訊息「Hello, World!」。

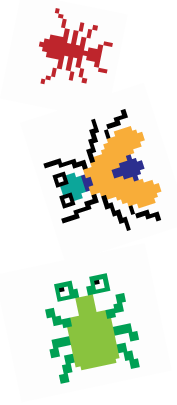



```
>>>
Hello, World!
>>>
```

執行程式後，Shell 視窗裡會出現這行訊息。

6 修正錯誤

如果執行程式之後發現無法正常運作，請先冷靜！每位程式設計師都會犯錯，如果想成為專業的程式設計師，找出藏在程式碼裡的「臭蟲」是非常重要的訓練。所以，請回頭再檢查一次程式碼有沒有打錯字。字串有沒有加上單引號？『print』有沒有拼錯？修正錯誤之後再重新執行程式碼。



程式高手秘笈

快捷鍵（Keyboard shortcut）

在編輯視窗下只要按鍵盤上的『F5』鍵就能執行程式，比用滑鼠點擊『Run』，再選擇『Run Module』要快得多，是非常實用的快捷鍵。

7 多加幾行程式碼

執行成功後，請回到編輯視窗，我們要多加兩行程式碼。請看右邊的程式碼，中間那一行是問使用者叫什麼名字，然後把使用者輸入的名字儲存在變數裡。最後一行是加上使用者的名字，重新顯示問候訊息。你可以隨意將這裡的問候訊息改成喜歡的文字，不管你想要有禮貌還是變粗魯！

```
print('Hello, World!')
person = input('What is your name?')
print('Hello,', person)
```

詢問使用者的名字，並且將輸入的名字儲存在變數『person』裡。

8 最後一步

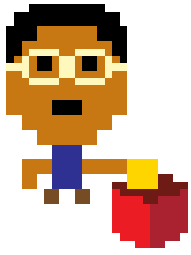
請重新執行程式碼。根據程式指示輸入名字，按下『enter / return』鍵後，Shell 視窗應該會顯示一行經過個人化的文字訊息。執行完這一步後，恭喜你，完成第一個 Python 程式！朝成為厲害程式設計師之路邁出第一步。

```
Hello, World!
What is your name?Josh
Hello, Josh
```

使用者輸入的名字

變數（Variable）

如果想寫出實際能用的程式碼，就一定會需要儲存和標記程式內的部分資訊，這正是變數擅長的工作。變數適用的範圍非常廣泛，從追蹤、記錄玩家在遊戲內的得分，到完成計算和儲存物品清單，變數都是程式碼裡不可或缺的角色。



△ **收納箱**
變數就像貼上名字標籤的收納箱，我們可以把想要儲存的資料放在這個箱子裡，使用資料時，只要用標籤上的名字就能找出資料。

如何建立變數？

建立變數時要先幫它取名字，想個好名字可以提醒我們變數裡儲存的資料是什麼。取好變數名稱後，就要決定變數儲存的內容，也就是變數值。「指定數值」給變數的程式寫法是先輸入變數名稱，然後在名稱後加上等號，最後輸入變數儲存的數值。

1 **指定變數值**
請在 Shell 視窗裡輸入右邊這行程式碼。目的是產生一個變數 **age**，並且指定數值給它。如果你不想用範例中的 12，也可以指定自己的年齡值。

這是我們儲存在變數裡的值。

```
>>> age = 12
```

這是我們幫變數取的名字。

2 **在螢幕上顯示變數值**
請在 Shell 視窗裡輸入右邊這行程式碼，然後按下『enter / return』鍵，就能在螢幕上看到程式執行的結果。

```
>>> print(age)
12
```

顯示變數 **age** 的值。

函式 **print()** 會幫我們把括號裡的變數值顯示在螢幕上。

程式高手秘笈

變數命名

幫變數取個好名字，更容易了解我們寫的程式。例如，不要把變數取名為 **lives** 或 **lr**，改用有意義的名字 **lives_remaining**，一看就知道這個變數是記錄遊戲裡玩家還剩幾條命。變數名稱可以包含字母、數字和「底線」字元，但名稱開頭必須是字母。大家只要跟著這裡所列的命名規則為變數取名字，就不用擔心會出錯。

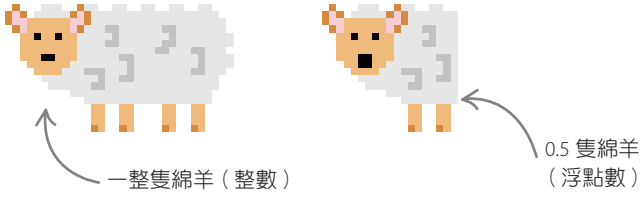
變數命名規則

- 變數名稱的開頭必須是字母。
- 所有的字母和數字都能用在變數名稱裡。
- 不能使用特殊的符號字元，像是 -、/、# 或 @。
- 不能使用「空白」（space）字元。
- 可以使用「底線」（_）字元來代替「空白」字元。
- 在 Python 裡，大小寫是不同的字母，所以「Score」和「score」會當成兩個不同的變數。
- 避免使用 Python 內建命令的名稱，例如，「print」。

知識補給站

資料型態——整數 & 浮點數

寫程式時，我們稱完整的數字為整數（integer），帶有小數的數字則稱為浮點數（float）。程式通常會使用整數來計算事物，浮點數則多半用在測量事物上。



數字運算

變數也能儲存數字，並且進行加總或是搭配數學符號做一些計算，就像我們平常做的數學運算一樣。Python 使用的數學符號類似數學課使用的計算符號，不過，請注意其中兩個符號「乘」和「除」，和我們平常使用的符號長的不太一樣。

運算符號	意義
+	加
-	減
*	乘
/	除

Python 使用的部分數學運算符號

1 簡單的數學運算

請在 Shell 視窗裡輸入右邊的程式碼。這幾行程式碼利用兩個變數 **x** 和 **y** 儲存數字，並且進行簡單的乘法計算。按下『enter / return』鍵就能得到計算的答案。.

```
>>> x = 6
>>> y = x * 7
>>> print(y)
42
```

產生一個新變數 **x**，指定變數值為 6。.

顯示計算結果。

顯示變數 **y** 的值。

將變數 **x** 乘上 7，並且把計算結果儲存到變數 **y**。

2 修改變數值

只要重新指定一個值給變數，就能修改變數值。在右邊的程式碼裡，我們將變數 **x** 的值改為 10，然後再顯示一次計算結果，你認為答案會是什麼？

```
>>> x = 10
>>> print(y)
42
```

改變 **x** 的變數值。

但是計算結果並沒有改變，之後我們會說明原因。

3 更新變數值

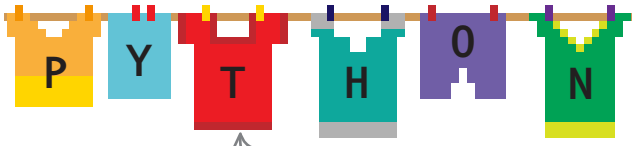
改變 **x** 的變數值後，必須更新變數 **y** 的值，才能得到正確的計算結果。請輸入右邊這幾行程式碼，現在我們已經將新的值指定給變數 **y**。請記住，如果更新程式裡的某個變數值，一定要檢查是否還有其他變數值也要一起更新。

```
>>> x = 10
>>> y = x * 7
>>> print(y)
70
```

更新變數 **y** 的值。

資料型態——字串（String）

寫程式時，我們會把一串字母或字元組成的資料稱為「字串」，所以單字和句子都能儲存成字串。幾乎所有的程式都有用到字串的時候，舉凡我們能用鍵盤輸入的字元，甚至是那些無法輸入的字元，都能存成字串。



字串就是一串字元。

1 利用變數儲存字串

變數也能儲存字串。請在 Shell 視窗裡輸入右邊的程式碼，目的是把字串 **'AllyAlien'** 指定給變數 **name**，然後將字串內容顯示在螢幕上。請注意，字串前後一定要加上單引號。

看到單引號表示變數包含字串。

```
>>> name = 'Ally Alien'
>>> print(name)
Ally Alien
```

按下『enter / return』鍵，字串的內容會顯示在螢幕上。

2 組合字串

當我們需要將不同的字串組合成新字串時，變數是非常好用的技巧。不僅能把兩個字串組合在一起，還能將組合結果儲存為新變數。請動手試試看吧。

記得加上單引號。

```
>>> name = 'Ally Alien'
>>> greeting = 'Welcome to Earth, '
>>> message = greeting + name
>>> print(message)
Welcome to Earth, Ally Alien
```

『+』號可以把兩個字串組合在一起。

在螢幕上顯示字串內容時，不會出現單引號。

程式高手秘笈

字串長度

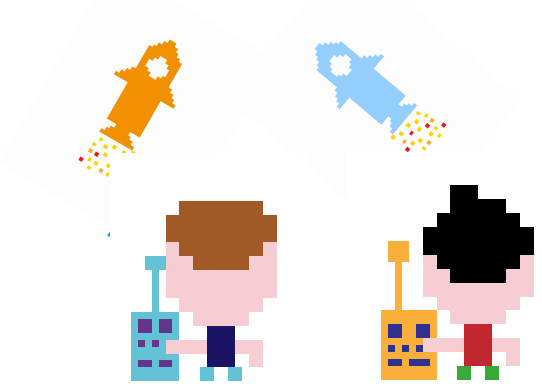
當我們想計算某個字串有幾個字元（包含空白字元），**len()** 是相當實用的技巧。Python 的內建命令 **len()** 就是我們寫程式時會用到的函式，本書之後還會陸續使用更多不同的函式。若想知道字串 **'WelcometoEarth,Ally Alien'** 的字元數，請先產生這個字串，然後輸入以下這行程式碼，再按下『enter / return』鍵，就能知道這個字串的字元數。

```
>>> len(message)
      28
```

函式計算出來的字元數

資料型態——清單（List）

當我們想儲存大量資料，或是資料順序很重要時，就需要用到資料清單。資料清單可以按照順序、同時儲存多個資料值，在 Python 程式裡，清單的每項資料值都有自己的編號，代表各自在清單裡的位置，當然也可以隨時改變清單裡的資料值。



1 使用多個變數

假設我們正在開發一款多人進行的遊戲，需要儲存每個遊戲隊伍裡所有玩家的名字，我們當然可以為每個玩家建立一個變數，就像右邊的程式碼這樣……

```
>>> rockets_player_1 = 'Rory'
>>> rockets_player_2 = 'Rav'
>>> rockets_player_3 = 'Rachel'
>>> planets_player_1 = 'Peter'
>>> planets_player_2 = 'Pablo'
>>> planets_player_3 = 'Polly'
```

每個隊伍裡有三位玩家，所以需要建立六個變數。

2 使用變數儲存資料清單

但如果每隊暴增為六位玩家，同時要管理和更新這麼多變數，我們的工作會變得非常困難，在這種情況下，比較好的做法是使用清單。建立清單時，必須將我們要儲存的所有資料值用中括號（[]）圍起來。請在 Shell 視窗裡練習輸入右邊這些清單。

```
>>> rockets_players = ['Rory', 'Rav',
'Rachel', 'Renata', 'Ryan', 'Ruby']
>>> planets_players = ['Peter', 'Pablo',
'Polly', 'Penny', 'Paula', 'Patrick']
```

清單裡儲存的各個資料值，必須以逗號分開。

以變數 **planets_players** 儲存清單。

3 從清單裡取出資料值

資料儲存到清單後，一切都好辦了。想取出清單裡的某個資料值，寫法是先輸入清單的名稱，然後在名稱後加上中括號（[]），最後在中括號裡填入資料值在清單裡的位置編號。請注意：Python 在數清單裡的資料值時，不是從 1 開始算，而是從 0 開始。現在請動手試試看，從隊伍清單裡取出不同玩家的名字，第一個玩家的名字是在清單裡編號 0 的位置，最後一個玩家則是在編號 5 的位置。

從清單裡編號 0 的位置取出第一個資料值。

```
>>> rockets_players[0]
'Rory'
>>> planets_players[5]
'Patrick'
```

從清單裡編號 5 的位置取出最後一個資料值。

按下『enter / return』鍵，就會顯示取出的資料值。

做決定

我們每天都會問自己各種大大小小的問題，然後根據答案決定下一步要做什麼，例如，「外面正在下雨嗎？」、「我的功課都寫完了嗎？」、「我是一匹馬嗎？」，同樣地，電腦也會問問題，然後決定它下一步要做什麼。

比較性的問題

電腦會問自己問題，通常是在需要比較兩個東西的時候，例如，電腦會問這個數字有沒有比另一個數字大，有的話，就根據這個答案，決定執行某段程式碼，否則就跳過，不執行這段程式碼。

▷ 布林值（Boolean value）

對於電腦問的問題，答案只會有兩個值：True（真）或 False（假），Python 稱這兩個值為布林值。使用布林值時，開頭第一個字母一定要大寫，也能使用變數儲存布林值。

變數

```
>>> answer_one = True
>>> answer_two = False
```

布林值

▽ 邏輯運算子

以下這些符號是告訴電腦進行比較，程式設計師稱這些符號為邏輯運算子，你可能在上數學課時用過其中一些符號。其他像英文單字「and」（且）和「or」（或）也能用在程式碼裡，作為邏輯運算子。

邏輯運算子	意義
==	等於
!=	不等於
<	小於
>	大於

程式高手秘笈

等號

在 Python 裡，單等號『=』和雙等號『==』都能使用，但兩者的意義不太一樣。單等號『=』用於指定變數值，例如，輸入 **age=10**，表示指定變數 **age** 的數值為 10；雙等號『==』則用於比較兩個數值，請看以下的例子。

設定變數值。

```
>>> age = 10
>>> if age == 10:
        print('You are ten years old.')
```

把你的年齡和變數值相互比較。

如果兩者相等，就顯示這段訊息。

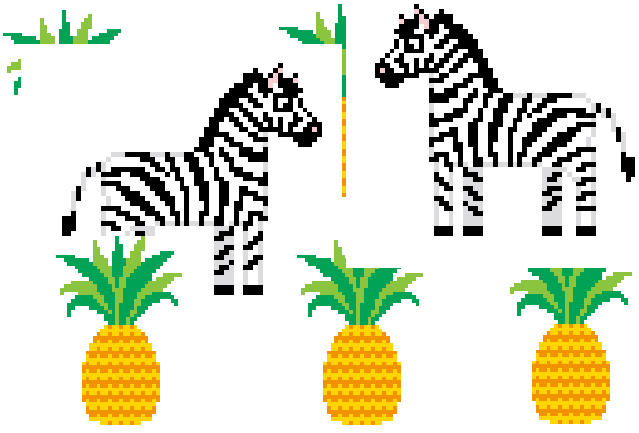
鳳梨和斑馬

現在讓我們在 Shell 視窗下測試一個範例。請在視窗裡輸入以下這兩行程式碼，分別產生變數 **pineapples** 和 **zebras**，表示有五個鳳梨和兩隻斑馬。

```
>>> pineapples = 5
>>> zebras = 2
```

這個變數負責儲存鳳梨的個數。

這個變數負責儲存斑馬的隻數。



▽ ▷ 進行比較

接著，請輸入以下這幾行程式碼，比較前面兩個變數的值。每次輸入完一行程式碼，請按下『enter / return』鍵，Python 就會告訴我們，這行比較式的結果是 True（真）還是 False（假）。

鳳梨的個數大於斑馬的隻數。

```
>>> pineapples > zebras
True
```

```
>>> zebras < pineapples
True
```

斑馬的隻數小於鳳梨的個數。

鳳梨的個數和斑馬的隻數不相等。

```
>>> pineapples == zebras
False
```

知識補給站

布林運算式（Boolean expression）

如果程式碼的陳述式跟變數、數值有關，而且用到邏輯運算子，則運算結果一定會是布林值，例如，True（真）或 False（假）。因此，我們稱這類的陳述式為布林運算式，此處所有用到變數 pineapples 和 zebras 的陳述式都是布林運算式。

變數　　邏輯運算子

```
>>> pineapples != zebras
True
```

布林值　　變數

▽ 組合多個比較式

在 Python 裡，我們還可以利用 **and** 和 **or** 組合多個比較式。使用 **and** 時，必須兩個比較式的結果都正確，陳述式的結果才能為 True（真）；如果使用 **or**，只需要其中一個比較式的結果正確。

```
>>> (pineapples == 3) and (zebras == 2)
False
```

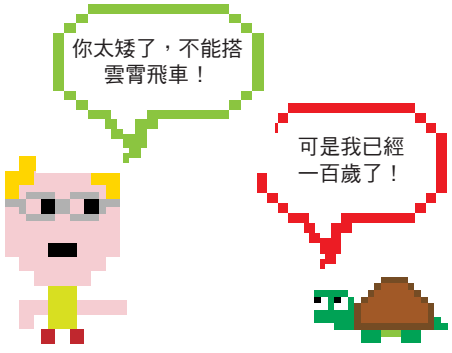
在這行陳述式裡，只要 **(pineapples == 3)** 這個部分是錯的，結果就是 False（假）。

```
>>> (pineapples == 3) or (zebras == 2)
True
```

在這行陳述式裡，只要 (zebras == 2) 這個部分是正確的，結果就是 True（真）。

搭乘雲霄飛車

遊樂園裡的設施說明上寫著，必須滿八歲而且身高 140 公分以上的人才能搭乘雲霄飛車。Mia 現在十歲、身高 150 公分，我們要在 Shell 視窗裡寫程式，檢查她是否可以搭乘雲霄飛車。請輸入以下這幾行程式碼，目的是產生兩個變數儲存 Mia 的年齡和身高，並且將正確的值指定給這兩個變數，然後將搭乘雲霄飛車的規則寫成布林運算式，最後，當然還是記得要按下『enter / return』鍵。



這兩行程式碼是將數值指定給變數。

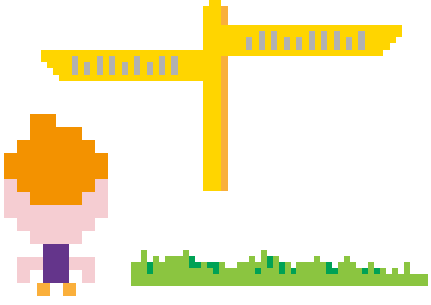
```
>>> age = 10
>>> height = 1.5
>>> (age > 8) and (height > 1.4)
True
```

這行布林運算式表示「必須滿八歲而且身高 140 公分以上」。

運算結果表示 Mia 能搭乘雲霄飛車！

分支（Branching）

電腦經常需要決定要執行哪一部分的程式碼，這是因為大部分程式在設計時，會根據不同的情況做不同的事。從程式裡分裂出來的路線，就像一條路分支成兩條岔路，每一條都通往不同的地方。



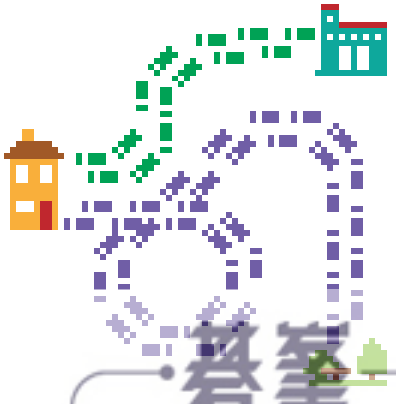
知識補給站

條件式（Condition）

條件式也是布林運算式（確認情況為真或假），當電腦在程式碼裡遇到分岔路時，條件式能幫助電腦決定要走哪一條路。

▷ 往學校還是公園？

想像一下，每天我們都會問自己「今天是平常日嗎？」然後根據問題的答案來決定要走哪一條路。如果今天是平常日，就走往學校的路線；如果不是，則會走往公園的路線。在 Python 下，從程式裡分支出去的不同路線會導向不同區塊的程式碼。每個區塊的程式碼可能只有一行，也可能有好幾行，每一行程式碼的開頭都要縮排，也就是按四個空白鍵。電腦會利用稱為「條件式」的測試方法，指出下一步要執行哪一個區塊的程式碼。



▷ 一個分支

程式裡最簡單的分支命令是 **if** 陳述式，只有一個分支，符合條件時，電腦才會執行這個分支下的程式碼。右邊的範例程式問使用者「外面是不是天黑了？」如果是，電腦會假裝它要睡了！如果條件式『**is_dark=='y'**』的結果為 False，也就是外面沒有天黑，電腦就不會顯示「Goodnight!」（晚安）這段訊息。

這行程式碼是請使用者回答『y』（是）或『n』（不是）。

```
is_dark = input('Is it dark outside? y/n)')
if is_dark == 'y':
    print('Goodnight! Zzzzzzzzzzzzzz....')
```

條件式

符合條件時才執行這段分支程式碼。

這行訊息會顯示在 Shell 視窗裡。

▷ 兩個分支

如果我們希望程式遇到情況為 True（真）和 False（假）時，分別做不同的事，就需要一個有兩個分支的命令，我們稱為 **if-else** 陳述式。右邊的範例程式問使用者是不是有觸手，如果使用者回答「是」，程式會認為使用者一定是章魚（octopus）！如果使用者回答「不是」，程式就認為使用者是人類（human）。程式會根據使用者的決定顯示不同的訊息。

這行程式碼是請使用者輸入答案。

條件式

```
tentacles = input('Do you have tentacles? (n/y)')
if tentacles == 'y':
    print('I never knew octopuses could type!')
else:
    print('Greetings, human!')
```

如果情況為 True（真），就執行這行程式碼。

如果情況為 False（假），則執行這行程式碼。

▷ 多個分支

如果可能發生的情況不只兩個，就要派 **elif** 陳述式（「else-if」的縮寫）上場。右邊的程式請使用者輸入今天的天氣預報：「rain」（下雨）、「snow」（下雪）或「sun」（晴天），然後根據使用者輸入的答案，從三個分支的天氣條件裡，選擇其中一個執行程式碼。

```
weather = input ('What is the forecast for today? (rain/snow/sun)')
if weather == 'rain':
    print('Remember your umbrella!')
elif weather == 'snow':
    print('Remember your woolly gloves!')
else:
    print('Remember your sunglasses!')
```

第一個條件式

如果第一個條件式的結果為 True（真），就執行這行程式碼。

第二個條件式

如果兩個條件式的結果都是 False（假），則執行這行程式碼。

如果第二個條件式的結果為 True（真），就執行這行程式碼。

△ 程式技巧

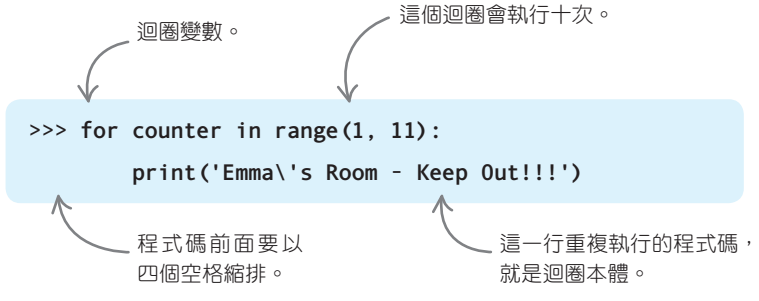
elif 陳述式一定會出現在 **if** 陳述式後面和 **else** 陳述式前面。在上面的範例程式中，只有當 **if** 陳述式的結果是 False（假），程式才會用 **elif** 來檢查輸入的天氣是不是「snow」（下雪）。還可以加入更多的 **elif** 陳述式，檢查更多種類的天氣。

重複執行的迴圈（Loop）

電腦非常擅長做枯燥乏味的工作，而且完全不會抱怨，但程式設計師會，可是他們擅長利用電腦幫他們做重複的工作，只要使用迴圈這項技巧就能辦到。迴圈的功能是重複執行同一段程式碼，接下來我們會介紹幾種不同類型的迴圈。

『For』迴圈

當我們知道同一段程式碼要執行幾次時，可以選擇用『**For**』迴圈。在下面這個範例裡，Emma 寫的程式是在房門的牌子上，印十次「Emma 房間——請勿進入！！！」請在 Shell 視窗裡練習試寫 Emma 的程式碼。輸入第一行程式碼後，請記得按『enter / return』鍵；輸入第二行程式碼時要記得按『backspace』鍵刪除縮排的空格，當然，輸入完畢後別忘了再按一次『enter / return』鍵。



▽ 迴圈變數

迴圈變數的目的是幫助我們追蹤到目前為止已經執行了幾次迴圈。**range(1, 11)** 會產生一份整數清單，執行第一次迴圈時，變數是清單裡的第一個數字，執行第二次迴圈時，變數就變成清單裡的第二個數字，以此類推。當清單裡所有的數字都用完後，程式就會停止執行迴圈。

程式高手秘笈

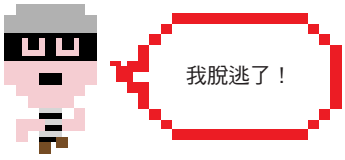
範圍（Range）

在 Python 程式碼裡，內建命令『range』後面的括號裡必須填入兩個參數，表示範圍包含「從第一個數字開始，到第二個數字減 1 為止的所有數字」。所以，**range(1, 4)** 是指數字 1、2 和 3，但不包含 4。在 Emma 寫的「請勿進入」程式裡，**range(1, 11)** 是指數字 1、2、3、4、5、6、7、8、9 和 10。

程式高手秘笈

跳脫字元（Escape character）

程式碼 **Emma\'sRoom** 裡的反斜線（\）是告訴 Python 程式忽略下一個字元——撇號（'），不要把它當成包覆整個字串的單引號之一。我們稱這裡用到的反斜線（\）為跳脫字元，目的是告訴 Python 執行程式時，不論這一行程式碼是合理還是有錯，都不要處理下一個字元。



『while』迴圈

但如果在某些情況下，我們不知道同一段程式碼要執行幾次，該怎麼辦？這時候難道我們要靠水晶球還是其他方法來預知未來嗎？不，我們還可以用『**while**』迴圈。

▷ 迴圈條件

while 迴圈沒有迴圈變數來幫助它設定執行次數的範圍，取而代之的是迴圈條件，這是一個布林運算式，結果不是 True（真）就是 False（假），有點像舞廳門口的保鑣，工作就是負責問你有沒有門票。如果身上有票（True），就可以直衝舞池；但如果沒有（False），保鑣就不會放你進去。在程式設計領域裡，如果不符合迴圈條件（True），程式就不會執行迴圈！

▽ 平衡遊戲

在下面的範例裡，Ahmed 寫的程式目的是追蹤多少隻雜耍河馬互相往上疊，才能堆出一個平衡塔。請看看下面的程式碼，你是否能找出這個程式的運作原理呢？

這個變數負責儲存河馬的隻數。

當程式問「要再加一隻河馬嗎？」，這個變數會負責儲存這個問題的答案。

迴圈條件

這一行程式碼負責顯示訊息，表示平衡塔上目前的河馬總數。

程式根據 Ahmed 的回答，更新變數 **answer** 的值。

往平衡塔上再加一隻河馬。

```
>>> hippos = 0
>>> answer = 'y'
>>> while answer == 'y':
        hippos = hippos + 1
        print(str(hippos) + ' balancing hippos!')
        answer = input('Add another hippo? (y/n)')
```

▷ **程式技巧**

在 Ahmed 的程式裡，**answer=='y'** 就是迴圈條件，表示使用者想多加一隻河馬。迴圈本體的程式會將平衡塔上的河馬隻數加 1，然後問使用者「要再加一隻河馬嗎？」如果使用者輸入的答案是『y』（要），表示迴圈條件是 True（真），程式會再執行一次迴圈；如果使用者輸入的答案是『n』（不要），迴圈條件就是 False（假），程式會離開迴圈本體，不再執行。

『無窮』迴圈

有時候我們會希望只要程式還在運作，就持續不斷地執行『**while**』迴圈，我們稱這種迴圈為『無窮』迴圈。大部分的遊戲程式都會利用無窮迴圈作為遊戲的主迴圈。

因為迴圈條件沒有機會變成 False（假），所以無法離開迴圈。

```
>>> while True:
        print('This is an infinite loop!')
```

△ **陷入無窮迴圈**

當迴圈條件設定為常數值：True（真），就會設計出無窮迴圈。因為這個常數值永遠不會改變，程式當然沒有機會離開迴圈。請在 Shell 視窗裡試著寫寫看上面這個『**while**』迴圈，由於這個迴圈條件沒有機會變成 False（假），只能不停地在螢幕上印出『This is an infinite loop!』（這是一個無窮迴圈！），直到程式停止為止。

▽ **脫離無窮迴圈**

你可以故意拿下面這個無窮迴圈，請使用者輸入他們的答案。這個（煩人的）程式會問使用者是不是覺得厭煩？只要使用者一直輸入『n』，程式就會持續問這個問題，直到使用者覺得受夠了，終於輸入『y』，程式才會說使用者很無禮，然後執行 **break** 命令，離開迴圈！

當迴圈條件是 True（真），表示使用者還沒感到厭煩（使用者回答 **'n'**）。

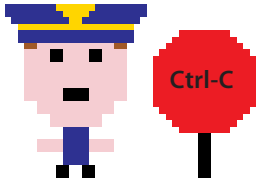
```
>>> while True:
        answer = input('Are you bored yet? (y/n)')
        if answer == 'y':
            print('How rude!')
            break
```

迴圈條件變成 False（假）就會觸發 **break** 命令（使用者回答 'y'）。

程式高手秘笈

停止迴圈

如果你不想設計出無窮迴圈，重點是確定『**while**』迴圈本體有方法能讓迴圈條件變成 False（假）。但如果不小心寫出無窮迴圈，別擔心，還是有方法能中斷迴圈，請同時按下鍵盤上的『Ctrl / control』鍵和『C』鍵，可能需要多按幾次『Ctrl-C』才能停止迴圈。



迴圈中的迴圈

迴圈本體裡還可以寫另一個迴圈嗎？當然可以！我們稱這種結構為巢狀迴圈（nested loop）。就像俄羅斯娃娃，每個娃娃都剛好套在比自己大一點的娃娃裡，巢狀迴圈也是，外層的迴圈中會執行另一個內層迴圈。

hooray_counter 是外迴圈的迴圈變數。

▷ 外迴圈裡的內迴圈

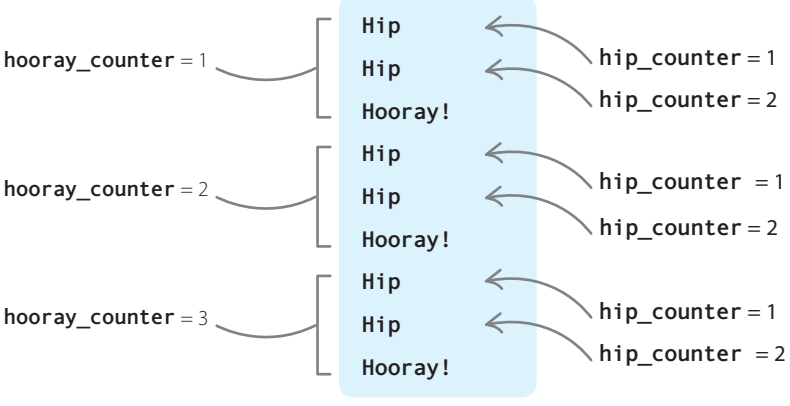
Emma 把之前的「請勿進入」程式改寫成右邊的範例「三次歡呼」，讓程式重複顯示三次『Hip, Hip, Hooray!』（嘿，嘿，萬歲！）。由於每次歡呼都會重複兩次「Hip」，Emma 利用巢狀迴圈幫她重複顯示訊息。

```
>>> for hooray_counter in range(1, 4):
        for hip_counter in range(1, 3):
            print('Hip')
        print('Hooray!')
```

外迴圈本體裡的每行程式碼都要縮排四個空格。

hip_counter 是內迴圈的迴圈變數

內迴圈本體裡的程式碼要再多縮排四個空格。



程式高手秘笈

迴圈本體的縮排規則

迴圈本體的每一行程式碼前面一定要縮排，也就是先敲四個空格再開始寫程式碼，如果沒有縮排，Python 會顯示錯誤訊息，表示無法執行程式碼。使用巢狀迴圈時，迴圈內部的另一個迴圈本體必須再多縮排四個空格。雖然在 Python 迴圈裡，每新增一行程式碼都會自動縮排，不過，最好還是記得檢查迴圈裡每一行程式碼前面的縮排空格數是否正確。

◁ 程式技巧

從上面的範例程式中可以看到，內層的『For』迴圈整個包覆在外層的『For』迴圈裡。因此，每執行一次外層迴圈，就必須執行兩次內層迴圈，也就是說外層迴圈本體執行三次，但內層迴圈本體總共會執行六次。

動物益智問答

你喜歡益智問答遊戲嗎？想自己動手開發嗎？這個程式範例會帶你建立一個動物益智問答的遊戲。雖然範例中所舉的問題都跟動物有關，不過很容易就能修改並且輕鬆套用在任何主題上。

範例說明

這個範例程式會問玩家一些跟動物有關的問題。每一題只有三次答題機會，所以不要出太難的問題！玩家每答對一題就能得到一分，問答遊戲結束後，程式會顯示玩家最後得到的總分。

這就是遊戲整體的外觀，遊戲過程會在 Shell 視窗下進行。

Python 3.5.2 Shell

```
Guess the Animal!
Which bear lives at the North Pole? polar bear
Correct answer
Which is the fastest land animal? cheetah
Correct answer
Which is the largest animal? giraffe
Sorry, wrong answer. Try again. elephant
Sorry, wrong answer. Try again. rhinoceros
The correct answer is blue whale
Your score is 2
```

玩家在這裡輸入答案。

如果玩家猜錯，可以再輸入一次答案。

猜錯三次，程式就會顯示正確答案。

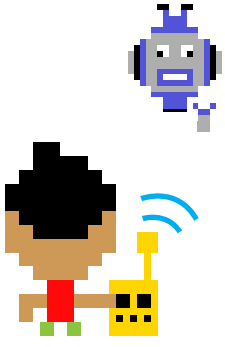
顯示玩家最後獲得的總分。

程式技巧

這個範例使用的程式技巧是「函式」（function）。函式是一段程式碼，有自己專屬的函式名稱，負責完成某個特定工作。當我們想重複使用同一段程式碼，有了函式就不需要每次都重新輸入。雖然 Python 有大量的內建函式，我們還是能設計自己需要的函式。

▷ 呼叫函式

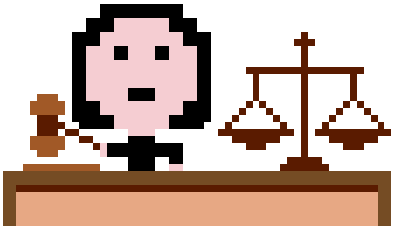
當我們想使用某個函式，只要在程式碼裡輸入函式名稱，就能「呼叫」（call）它出來幫忙。在「動物益智問答」裡，為了知道玩家的答案是否正確，我們設計一個函式，負責比較玩家回答的內容和真正的答案。問答進行過程中，玩家每次回答問題，程式都會呼叫這個函式。



知識補給站

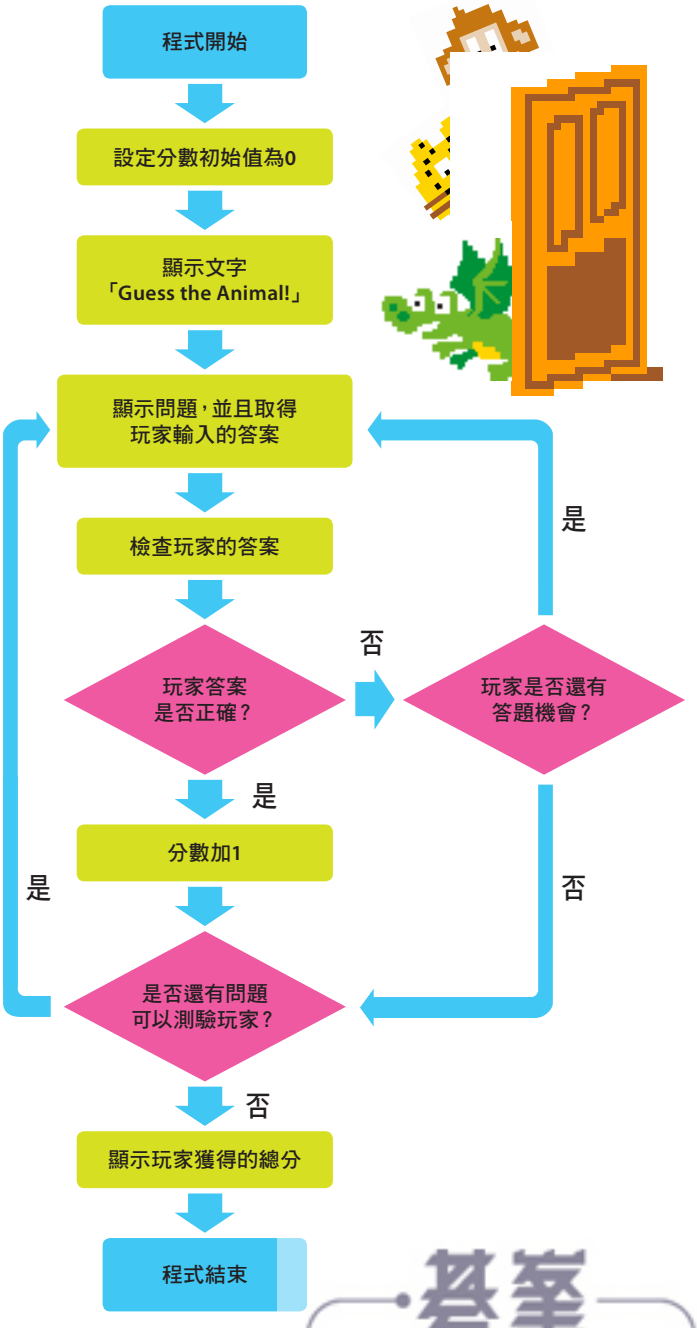
在這個情況下可以忽略大小寫！

比較玩家猜測的內容和正確答案時，可以不用管玩家輸入的字母是大寫還是小寫，重要的是文字的內容都一樣，然而，不是所有程式都能這麼做。例如，如果某個程式檢查密碼時忽略大小寫，就可能會讓密碼更容易被猜出來，因而降低安全性。但是，在「動物益智問答」裡，不論玩家輸入的答案是「bear」還是「Bear」，對我們來說，都是正確答案。



▽ 程式流程圖

遊戲結束前，程式會一直不斷檢查還有沒有問題沒回答到以及玩家是否已經用掉了所有的答題機會。遊戲進行過程中，程式會以變數儲存玩家獲得的分數。當玩家回答完所有的問題，遊戲就會結束。

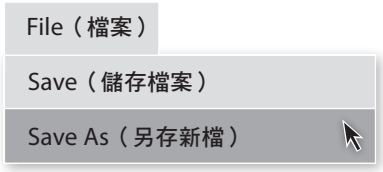


組合所有步驟

現在該開始動手完成這個益智問答的程式了！第一步當然是建立益智問答的題目和檢查答案的機制，然後加入程式碼，讓玩家在回答每一題時都有三次答題機會。

1 建立新檔

開啟 IDLE 工具。點擊工具列上的『File』（檔案），選擇『New File』（新增檔案），將檔案名稱儲存為『animal_quiz.py』。



2 產生變數 score

輸入右邊的程式碼，產生變數 **score**，並且設定變數初始值為 0。

```
score = 0
```

我們會用這個變數來追蹤、記錄玩家的分數。

3 介紹遊戲玩法

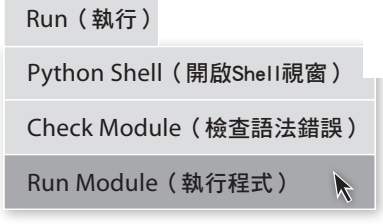
接著建立文字訊息，向玩家介紹遊戲玩法，這是玩家在螢幕上看到的第一個訊息。

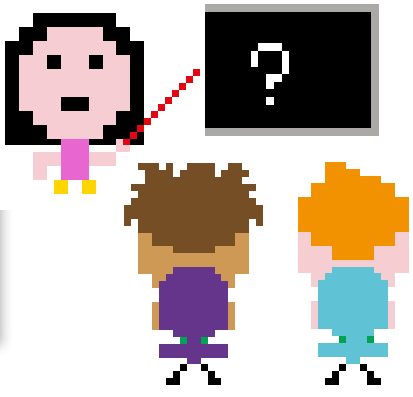
Shell 視窗裡會出現這段訊息。

```
score = 0
print('Guess the Animal!')
```

4 執行程式碼

現在請執行程式碼。點擊工具列的『Run』（執行），選擇『Run Module』（執行程式），看看接下來會發生什麼事？應該會在 Shell 視窗裡看到一行歡迎玩家的訊息。





5 問玩家問題，讓玩家輸入答案

右邊這行粗體字程式碼是問玩家問題，並且等他們輸入答案，玩家輸入的答案會存在變數 **guess1** 裡。請執行程式碼，確定螢幕上會出現題目。

```
print('Guess the Animal!')
guess1 = input('Which bear lives at the North Pole? ')
```

變數 **guess1** 儲存玩家輸入的任何內容。

6 建立檢查函式

接下來的工作就是檢查玩家是不是猜對了。請移動到程式碼的第一行，也就是在程式碼 **score=0** 上面輸入右邊這幾行粗體字程式碼，建立函式 **check_guess()**，目的是檢查玩家猜的答案是否正確。函式括號裡有兩個英文單字，這是函式運作時需要的資訊，我們稱為「參數」(parameter)，呼叫（執行）函式時需要指定（給）參數值。

```
def check_guess(guess, answer):
    global score
    if guess == answer:
        print('Correct answer')
        score = score + 1
score = 0
```

第一行程式碼是為函式命名，並且指定會用到的參數。

這一行程式碼是把變數 **score** 設定成全域變數，當變數有任何變化時，對整個程式都有效。

變數值加 1，表示玩家獲得一分。

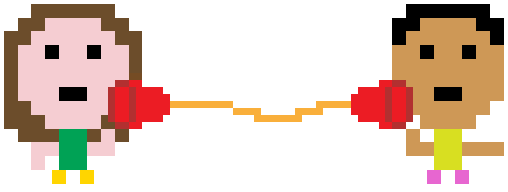
別忘了加上括號。

7 呼叫函式

現在我們要在整個程式碼的最後一行下面，加上右邊這一行粗體字程式碼，目的是呼叫（執行）函式 **check_guess()**，並且告訴函式拿玩家猜的答案作為第一個參數值，正確答案『polar bear』則是第二個參數值。

```
guess1 = input('Which bear lives at the North Pole? ')
check_guess(guess1, 'polar bear')
```

正確答案



8 測試程式碼

請再執行一次程式碼，根據程式指示輸入正確答案，Shell 視窗應該會出現右邊這樣的結果。

```
Guess the Animal!
Which bear lives at the North Pole? polar bear
Correct answer
```

9 多加一點題目

設計益智問答遊戲當然不能只有一個題目！請依照前面這些相同的步驟，在程式裡多加兩個題目，把玩家輸入的答案分別儲存到變數 **guess2** 和 **guess3**。


```
score = 0
print('Guess the Animal!')
guess1 = input('Which bear lives at the North Pole? ')
check_guess(guess1, 'polar bear')
guess2 = input('Which is the fastest land animal? ')
check_guess(guess2, 'cheetah')
guess3 = input('Which is the largest animal? ')
check_guess(guess3, 'blue whale')
```

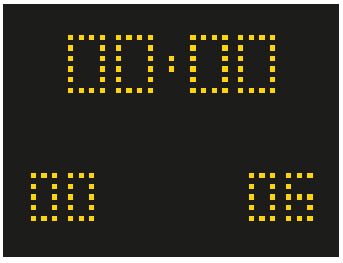
第一題

告訴程式檢查玩家輸入的答案 **guess1**。

告訴程式檢查玩家輸入的答案 **guess3**。

10 顯示分數

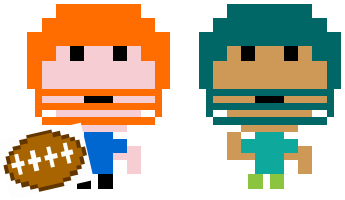
當問答遊戲結束時，下面這一行粗體字的程式碼會以文字訊息顯示玩家獲得的總分。把這一行加在整個檔案的最後一行，也就是最後一個問題的程式碼下面。



```
guess3 = input('Which is the largest animal? ')
check_guess(guess3, 'blue whale')

print('Your score is ' + str(score))
```

產生文字訊息，在螢幕上顯示玩家的總分。



△ 程式技巧

這一步必須利用函式 **str()**，將數字轉換成字串，這是因為在 Python 裡，如果直接將字串和整數放在一起，程式會出現錯誤。

11 忽略大小寫

如果玩家把「Lion」輸入成「lion」，會發生什麼事？還是能獲得一分嗎？不行，目前的程式碼會告訴玩家：答案錯了！為了修正這個情況，我們需要讓程式碼變得聰明一點。Python 的內建函式 **lower()** 能把單字裡所有的字母轉成小寫，請把原本程式碼裡的 **ifguess==answer:** 換成右邊這一行粗體字程式碼。

```
def check_guess(guess, answer):
    global score
    if guess.lower() == answer.lower():
        print('Correct answer')
        score = score + 1
```

修改這一行程式碼。

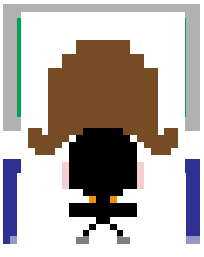
△ 程式技巧

程式檢查正確答案之前，會先把玩家猜的內容和真正的答案全都轉換成小寫。如此一來，不論玩家輸入的內容是用大寫、小寫還是兩者混用，程式都一定能運作。

12 再次測試程式碼

第三次執行程式碼。這次在輸入正確答案時，請試著混用大、小寫字母，看看會發生什麼結果？

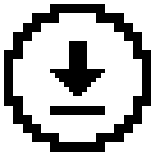
```
Guess the animal!
Which bear lives at the North Pole? polar bear
Correct answer
Which is the fastest land animal? Cheetah
Correct answer
Which is the largest animal? BLUE WHALE
Correct answer
Your score is 3
```



程式這次在判定答案是否正確時，忽略了大、小寫。

13 給玩家更多機會

在目前的程式裡，玩家只有一次答題機會，想讓遊戲更輕鬆一點，可以改成給玩家三次答題機會。請參考以下的粗體字程式碼修改函式 **check_guess()**。



別忘了儲存你的工作成果。

這個變數只有兩種變數值：True（真）或 False（假）。

```
def check_guess(guess, answer):
    global score
    still_guessing = True
    attempt = 0
    while still_guessing and attempt < 3:
        if guess.lower() == answer.lower():
            print('Correct answer')
            score = score + 1
            still_guessing = False
        else:
            if attempt < 2:
                guess = input('Sorry wrong answer. Try again. ')
            attempt = attempt + 1

    if attempt == 3:
        print('The correct answer is ' + answer)

score = 0
```

跳出 **while** 迴圈的條件是已經檢查過三次答案，或是玩家猜到正確答案，只要符合其中一項條件，就不再執行迴圈。

請確定每一行程式碼的縮排都正確。

如果玩家答錯，程式會執行變數 **else** 底下的程式碼，請玩家再輸入一次答案。

玩家用掉的答題次數加 1。

玩家猜錯三次後，這行程式碼負責顯示正確答案。

△ 程式技巧

為了知道玩家是不是已經猜到正確答案，我們需要產生一個變數 **still_guessing**，並且先設定變數值為 True（真），表示玩家還沒猜到正確答案；當玩家猜到正確答案時，就將變數值設定為 False（假）。