目錄

第 1 篇　Python

第 2 篇　樹莓派

第 3 篇 OpenCV

Python 1

1-1 介紹與安裝

Python 是一個誕生於 1991 年的直譯式程式語言。雖然不是一個新的語言，但是最近這幾年卻紅透半邊天，以 TIOBE 2020 11 月的資料，Python 在所有程式語言中受歡迎程度為第二名，而 PYPL 的排名資料更是高達第一名。紅的原因不僅僅是語法高階、簡單、不艱澀，還有一大部分紅的原因是在各個不同領域有非常多的函數庫可以讓我們下載使用，其中不乏超重量級函數庫，例如 OpenCV、NumPy、Matplotlib、SciPi、Keras、Tensorflow…等多到要列完幾乎是不可能的任務。有了這些函數庫的加持，當我們想要完成一些複雜的工作，可能只要短短數十行甚至幾行程式碼就能完成了，您可以想像一個語音辨識系統只要不到十行程式碼就寫完了嗎？

Python 是一個高度跨平台的語言，除了 Windows 外，macOS 以及其他的 UNIX、Linux 系統上都可以執行 Python 程式，甚至，各位可能想像不到，現在連單晶片都可以用 Python 來寫相當於韌體等級的程式了，例如 PyBoard、ESP8266 或 ESP32。

雖然直譯式語言的執行速度比編譯式語言來的慢，例如 C 語言，但是有很多的函數庫核心是用 C 或是 C++ 撰寫，Python 只是一個介接界面而已，例如 OpenCV 或是 NumPy，因此，用 Python 完成的這類型專案，執行效能並不一定比純粹用 C 或是 C++ 寫出來的系統慢，但開發速度卻比 C 或是 C++ 來的快。

Python 目前有兩個主要的發行版本，一個是 Python 2.x 版，另外一個是 Python 3.x 版。Python 跟其他大部分的程式語言在版本歷程上有點不太一樣，其他的語言有新版本推出後，舊版本就慢慢被淘汰，而用舊版本開發的專案由於新版本可以向前相容，因此執行上不會有太大的問題。當然也有些語言比較強勢，雖然向前相容，但幾個版本後就變的「不太相容」了，所以工程師就要想辦法用新版本改寫原系統，這是不容易的一件事情。Python 3 並沒有向前相容 Python 2，所以用

Python 2 開發的系統自然就不太有人想要用 Python 3 重新改寫，反正 Python 2 與 Python 3 的直譯環境可以並存於電腦中。直譯式語言有一個好處，就是一個系統中的兩個 Python 檔案，其中一個檔案用 Python 2 去執行，另外一個檔案用 Python 3 去執行，並不會有什麼問題。

雖然 Python 2 已經停止更新，我們學習的時候自然會選擇使用 Python 3，但是有些系統已經內建 Python 2（例如 macOS），或是兩個版本都有（例如樹莓派）。請讀者不要隨意移除 Python 2，因為有很多程式還是依賴 Python 2 才能順利執行，移除掉的結果恐怕連作業系統都要重灌了。

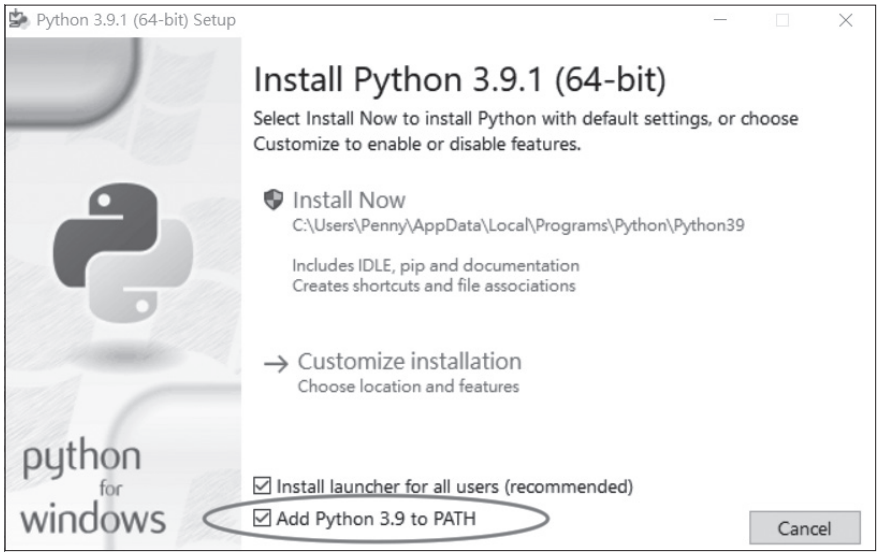
當系統中有兩個 Python 版本並存時，一般來說執行 python 指令是使用 Python 2，如果要使用 Python 3，必須輸入 python3。請讀者務必先確定您的電腦裡面的 Python 版本編號，以及用哪一個指令啟動他。

1-1-1 安裝

我們以 MS-Windows、macOS 與樹莓派三個平台來說明 Python 的安裝方式：

MS-Windows

至 Python 官網（https://www.python.org）下載 3.X 最新版。注意在安裝過程的第一個畫面最下方要勾選「Add Python to PATH」選項，這樣之後在下指令執行 Python 程式時，作業系統才能夠找到 Python 執行檔在哪個路徑。若忘了勾選，雖然可以用手動的方式將 Python 的安裝路徑加到環境變數中，但建議重裝一次比較快。



macOS

雖然 macOS 已經內建了 Python 2，但不要移除他以免一些需要 Python 2 的系統服務無法正確執行。我們額外安裝 Python 3 就好了，先到 Python 官網（https://www.python.org）下載 Python 3.X 最新版後安裝即可。安裝完畢後請開起終端機，執行以下指令更新憑證。

```
% sudo "/Applications/Python3.X/Install Certificates.command"
```

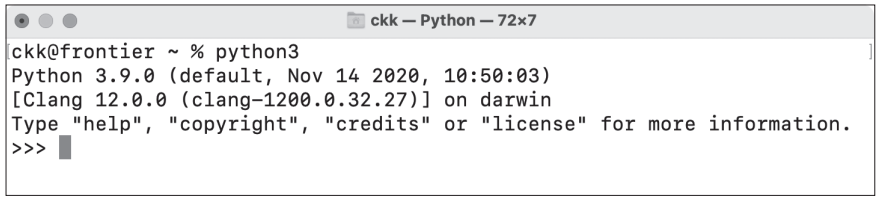
現在您的電腦中同時存在兩套 Python 版本，指令 python 是執行 Python 2，指令 python3 是執行 Python 3。

樹莓派

作業系統已經內建 Python 2 與 Python 3，因此不需要再安裝。指令 python 是執行 Python 2，指令 python3 是執行 Python 3。

1-1-2 驗證 Python 是否安裝成功

要測試是否安裝成功，請開啟命令提示字元（MS-Windows）或是終端機（macOS）或是使用 MobaXterm 或 putty 遠端 ssh 連線到樹莓派，然後輸入指令「python」或「python3」。如果看到類似下方的內容（最後出現三個「>>>」符號），就表示 Python 環境已經正確安裝完畢了，以此畫面為例，電腦中的 Python 3 版本為 3.9.0。連線樹莓派請參考 2.1 節第 7 步。



「>>>」代表此時已經進入 Python 直譯環境，輸入 Python 程式指令就可以執行了。照慣例，試試以下程式碼。

```
>>> print('hello world')
hello world
```

這樣我們就完成了第一支 Python 程式。要離開 Python 直譯環境輸入 quit() 即可。

接下來建議讀者根據您使用的作業系統安裝一個順手的程式編輯器，例如 Notepad++（只有 MS-Windows 才有）、Visual Studio Code（MS-Windows 與 macOS 都有）、BBEdit（只有 macOS 才有）或是所有 UNIX 系統應該都內建的 vi。本書並不會特別介紹許多豪華且功能超強的 IDE（整合開發環境）如何使用，因此，寫程式都是在基本的編輯器中撰寫，然後在命令提示字元或是終端機中下指令執行。

1-2 基礎概念

本節將介紹 Python 3 常用且重要的語法。執行方式為使用編輯器撰寫程式，存檔時副檔名為 .py，例如 test.py。執行時在命令提示字元或是終端機中輸入 python3 test.py 就可以執行了。

Windows 的讀者請使用 python test.py 就可以了，不需要使用 python3，電腦中應該也沒有 python3 這個指令。

1-2-1 註解

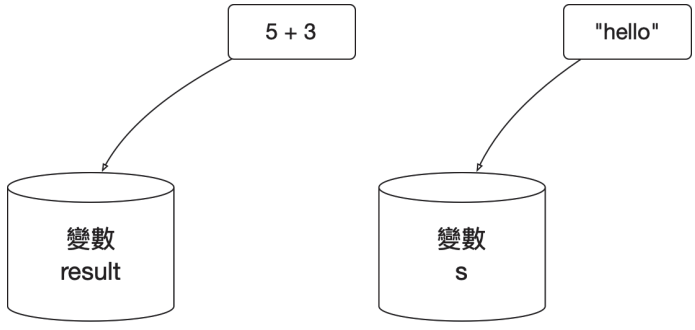
Python 註解有兩種形式，一種為單行註解以「#」號開頭；另一種為多行註解，以三個單引號「'」或三個雙引號「"」前後夾起來的多行文字。例如：

```
# 此為單行註解
print('hello')  # 註解

'''
這是
多行註解
'''

"""
用三個雙引號也可以
是多行註解
"""
```

註解可以幫助我們理解程式碼的用途，否則一些複雜的程式邏輯，往往一段時間後就忘了當初為什麼要這樣寫，沒有註解的話會讓後續程式閱讀不易也造成維護困難。

1-2-2 變數與資料型態

變數是用來儲存資料的地方。如果我們從某個地方得到一個值，例如使用者在鍵盤輸入了一個數字，我們就需要使用一個變數來儲存使用者在鍵盤上輸入的這個數字，好讓之後可以很方便的使用程式針對這個變數所儲存的內容進行各種運算與分析。所以變數，顧名思義，就是內容可以隨時變動的一個記憶體儲存空間，而我們為這個儲存空間取了一個名字。



有些程式語言會規定使用變數前必須要先「宣告」，這個步驟相當於告訴作業系統請他分配一個記憶體空間給這個變數，如果沒有事先宣告就要使用這個變數，是會得到錯誤訊息的。但 Python 不需要事先宣告，變數隨叫就隨用了。這樣的好處是，我們不用理會宣告是怎麼一回事，也不需要在宣告的時候來決定資料型態（稍後會提），反正想用的時候隨手就拿個變數來用，此時型態也不是那麼的重要，變數裡面放什麼，型態就是什麼。但缺點就是，如果變數拼錯字就會發生不可預期的錯誤。由於拼錯字的變數相當於是一個新的變數，跟之前正確拼法的變數一點關係都沒有，執行時只有在某些情況下才會產生錯誤訊息，大部分的情況 Python 就順利執行下去但運算結果卻是錯的，若沒有事後檢查就不會發現這個錯誤。所以撰寫程式碼時必須小心謹慎，隨時要留意是否打錯字了。

資料必伴隨著型態，所謂的型態代表了這個變數有哪些功能可以做哪些事情，例如數字型態的變數可以做數學的四則運算，文字型態的變數

可以做關鍵字搜尋，當然文字型態的變數是不能做四則運算的。Python 有四種純量資料型態，分別是整數 int、浮點數 float、字串 str 與布林 bool，分述如下。

int

儲存的內容必須為整數。就大部分的程式語言而言，整數值是有範圍限制的，也就是我們不能儲存一個位數非常大的數字，以 Python 2 且在 64 位元的電腦上而言，數字範圍必須在 -2^{63} 到 $2^{63}-1$ 之間。以 Python 3 而言，int 範圍並沒有限制，只要記憶體夠大，可以儲存相當於無限制的整數位數。

float

儲存的是有小數的數字。有些程式語言在浮點數的地方還分為單精確與倍精確兩種型態，表示能夠儲存浮點數範圍。Python 就相對簡單許多，只有單一型態：float。範圍為 $2.2250738585072014e^{-308}$ 到 $1.7976931348623157e^{308}$，顯而易見這範圍非常大。

str

用來儲存字串。可容納的字串長度沒有上限，電腦所裝的記憶體能夠儲存多少字就多少字。

bool

布林變數，裡面放的值只有兩種，不是 True 就是 False，注意 T 與 F 要大寫。這種變數專門用儲存邏輯運算後的結果，或是純粹表示兩種不同的狀態。

當我們要將值放到變數裡作法很簡單，設定一個變數名稱等於某個值就完成了，如下：

```
n = 10
f = 3.14
s = 'hello world'
b = True
```

上述四行程式碼的等號左邊 n、f、s 與 b 分別代表四個變數，而他們的型態分別是整數、浮點數、字串與布林。在 Python 中，字串的前後必須以單引號或是雙引號夾起來，若開頭為單引號結尾就必須單引號，若開頭為雙引號結尾也就要雙引號。

前頭說過，變數型態代表這個變數可以做哪些事情，例如有個變數是整數型態（例如 n = 10），另外一個變數是文字型態（例如 s = 'abc'），如我們想要將這兩個變數相加（例如 n + s），這時候到底要怎麼「加」呢？加完的結果又是什麼？我想這時大家應該都覺得困惑，是的，當兩個變數型態不一致的時候，通常是無法放在一起處理的，因為電腦也不知道該怎麼辦。我們再看一個例子，假如 n = 10，s = '20'，注意這裡的 20 前後加了單引號，因此 20 為字串而非整數，因此變數 s 的型態為 str。若我們希望將 n 與 s 相加並得到 30 的結果，我們就必須對 s 進行「型別轉換」，要將字串的 '20' 轉成整數的 20 或小數的 20.0 才能跟 n 進行四則運算。否則硬加的結果，會得到錯誤訊息，如下：

```
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

如果我們希望進行四則運算的「+」也就是結果為 30，就必須轉換變數 s 的型態為 int 或 float（這裡我們轉成 int 比較適當，因為變數 n 的型態為 int，這樣「+」號左右兩邊的型態就一致了）。將型態 str 轉 int 只要使用 int() 即可，如下：

```
n = 10
s = '20'
ans = n + int(s)
```

這時運算結果 30 會放到變數 ans 中，並且 ans 的型態為 int 整數型態。若我們希望加完的結果是 1020 而不是 30，這代表的是我們要將兩個字串連接（concatenate）起來，這時就要將變數 n 的型態轉成 str 型態，對 str 型態而言，「+」號代表將兩個字串連接起來，如下：

```
n = 10
s = '20'
ans = str(n) + s
```

此時變數 ans 就是字串 1020 了。有沒有發現，若等號右邊的程式碼要能正確無誤的運作，每個變數的型態都必須一致才行，雖然有些程式語言會很聰明的自動調整型態，但我們還是養成良好的程式撰寫習慣，手動調整型態比較妥當。補充說明一下，Python 允許 float 型態與 int 型態可以放在一起進行四則運算，例如 3 + 5.0，運算結果為 float 型態的 8.0。事實上，有些強型別語言是不可以這樣做的，例如 Swift。

Python 的四個資料型態彼此間都可以互相轉換，當然如果要把字串 'abc' 轉成 int 型態，那肯定會得到一個錯誤訊息，不信您試試看。

```
# 字串轉整數
n = int('10')

# 數字轉字串
s1 = str(25)
s2 = str(1.37)

# 轉成浮點數
f1 = float(10)
f2 = float('3.14')

# 轉成布林的 True
b1 = bool('True')
b2 = bool(1)

# 轉成布林的 False
b3 = bool('False')
b4 = bool(0)
```

1-2-3 標準輸入輸出

輸入輸出指的是變數值從哪裡來然後要到哪裡去，加上「標準」兩字，代表的就是鍵盤輸入螢幕輸出。以下述程式碼為例，執行後會讓使用者透過鍵盤輸入兩個數字，相加後將結果顯示在螢幕上。注意標準輸入得到的值其型態一律是 str。

```
n1 = input('請輸入第一個數字:')
n2 = input('請輸入第二個數字:')
ans = float(n1) + float(n2)
print(ans)
```

透過函數 print()，我們就可以在螢幕上看見變數 n1 與 n2 相加後的結果。但如果只顯示一個數字，使用者往往不知道這個數字代表的意思，我們要多輸出一點訊息給使用者，例如 10 + 20 = 30。我們可以把程式碼修改成如下，這樣的輸出結果就比較理想了。

```
n1 = input('請輸入第一個數字:')
n2 = input('請輸入第二個數字:')
ans = float(n1) + float(n2)
print('{} + {} = {}'.format(n1, n2, ans))
```

字串中的三個 {} 分別代表了在這三個地方要依序換成字串後面 format() 函數中的三個參數內容。我們也可以將上述程式碼修改的更簡潔，如下：

```
n1 = float(input('請輸入第一個數字:'))
n2 = float(input('請輸入第二個數字:'))
print('{} + {} = {}'.format(n1, n2, n1 + n2))
```

使用 float() 來做型別轉換時，我們沒有考慮使用者不是輸入數字的情況，也就是當使用者故意輸入英文字母，這時要轉換成 float 型態時就會出現無法轉換的錯誤而導致程式當掉。錯誤處理部分請讀者翻閱本書 ch 1.2.7 錯誤處理一節。

跳脫字元

我們現在已經知道字串是由兩個單引號或是兩個雙引號前後夾住的文字、數字或文字數字混和的一種資料型態，若使用 print() 函數將這個字串顯示在螢幕上時，我們就可以看到這個字串的原始內容。但 print() 函數並不僅僅只能將字串的原始內容顯示到螢幕上，若字串中包含了倒斜線「\」(稱為跳脫字元)，倒斜線後面所接的字元會變成一個特殊指令，這時就可以讓顯示在螢幕上的資料以某種形式呈現，例如在字串中加上 \n 就可以讓字串在這個地方換行顯示。

```
print('hello\nworld')
```

上面這一行程式碼會讓螢幕上所顯示的 hello world 分成兩行。除了 \n 之外，\t 代表的意思是鍵盤的 TAB 鍵，也就是相當於在這個位置按了一個 TAB 鍵，例如：

```
print('a\tb\tcde')
```

輸出結果如下，可以看到 a 與 b 之間以及 b 與 c 之間空了很大的距離，這就是 \t 造成的效果，相當於這裡按了一個 TAB 鍵。

```
a    b    cde
```

如果字串中本來就要輸出 \n 或 \t 的話該如何處理呢？有兩種作法，一種是連打兩個 \\，例如 print('hello\\nworld')，這樣輸出結果會是 hello\nworld 並且不換行。另外一種作法是在字串前加上「r」修飾子，告訴 print() 函數不要將「\」當成跳脫字元，例如 print(r'hello\nworld')。還有哪些常用且有趣的指令呢？如下表：

跳脫字元指令	說明
\a	發出 beep 一聲
\n	換行
\t	TAB
\'	單引號
\"	雙引號

格式化字串

除了字串中可使用跳脫字元「\」外，還可以使用「{}」讓輸出的字串有更多樣化的顯示格式。例如如下的程式碼：

```
# 僅僅顯示 5
print(5)

# 顯示 005（三位數缺項補零）
print('{:03d}'.format(5))
```

{:03d} 代表的是這個位置會用一個整數取代（d 代表整數），該整數若不足三位數的話，缺少的部分會補零，字串後方使用 .format() 來承接真正要顯示的參數，例如數字 5。若有兩個數字需要顯示，則在 format 中放兩個參數即可，如下：

```
# 顯示 005 003
print('{:03d} {:03d}'.format(5, 3))
```

{:06.3f} 表示全部顯示位數為六個字包含小數點，小數部分要顯示三位數，因此整數部分顯示兩位數，整數與小數部分缺項都要補零，例如以下程式碼顯示的結果為 03.140：

```
print('{:06.3f}'.format(3.14))
```

還有哪些其他的格式就請讀者自行參考官方文件說明了，網址如下：

https://docs.python.org/3/library/string.html#string-formatting

輸出結果不換行

各位讀者應該會發現，print() 所顯示的資料會在結尾處自動換行，如果您連打兩行 print() 就可以發現輸出的資料會換行。如果我們想要輸出結果不換行，就必須在 print() 中加上 end=''，等號右邊為連續兩個單引號或連續兩個雙引號（意即將結尾的換行符號用空字串取代）。

```
print('A', end='')
print('B', end='')
```

這樣的輸出結果為 AB 連在一起不換行。

多行文字輸出

除了使用 \n 讓輸出的文字換行外，還可以使用連續三個單引號或連續三個雙引號來顯示多行字串，如下：

```
print('''
Hello
    world
    Hi
''')
```

1-2-4 流程控制

流程控制指的是改變程式執行流程。一般而言，程式碼都是由上往下依序執行，但如果我們希望某些程式碼在特定情況下才要執行或者不要執行，這時候就需要使用「條件判斷」來改變原本程式由上而下「依序」執行這樣的流程。此外，如果有些程式碼需要重複執行，也就是已經執行過的程式碼還要再執行一次，雖然用複製貼上數次也可以達到重複執行的效果，但這樣的程式碼就不漂亮也不好維護了。為了達到重複執行的需求，迴圈這樣的語法結構就應運而生，利用迴圈，我們可以讓程式回頭再執行一次。首先，我們來看條件判斷的語法。

條件判斷（或稱 if 判斷式）

If 判斷式用來決定哪些程式碼要執行，哪些程式碼不要執行，if 後面接一個邏輯判斷式，如果判斷式成立（True）就執行 if 區段中的程式碼，如果不成立（False）就執行 else 區段中的程式碼。區段 else 部分也可以加上 if 進行額外的判斷，使用的語句是 elif。以下面程式碼為例，先從標準輸入裝置（鍵盤）輸入一個字串，如果這個字串中有「開」這個字，就印出現在燈開了以及早安這兩行字串，否則如果有「關」這個字就印出現在燈關了以及晚安，如果既沒有「開」也沒有「關」就印出命令錯誤，如下：

```
text = input('請輸入命令：')
if '開' in text:
    print('現在燈開了')
    print('早安')
elif '關' in text:
    print('現在燈關了')
    print('晚安')
else:
    print('命令錯誤')
```

注意 if、elif 與 else 最後要加上冒號「:」代表程式區段開始，類似其他語言的左大刮號「{」或是「BEGIN」。除此之外，Python 沒有程式區段的結束符號，因此用來決定程式區段範圍純粹靠的是縮排。縮排可以使用 TAB 鍵縮排也可以使用 SPACE 鍵縮排，不論縮排幾個字都可以只要有縮排就好。但要特別注意的是相同程式區段中每行的縮排格式必須一致，例如程式區段的第一行使用 SPACE 鍵縮排 2 字元，同一區段中的其他行也都要使用 SPACE 鍵縮排 2 字元；或者第一行用 TAB 鍵縮排，之後的每一行都要用 TAB 鍵縮排。如果縮排格式不一致，就會得到一個錯誤訊息，如下。

```
IndentationError: unindent does not match any outer indentation level
```

縮排是 Python 語法的一部份也是這個語言的一項特色，對那些經常程式碼左側切齊或是隨意縮排的程式設計師們，這會是一個養成良好程式撰寫風格的絕佳機會。

If 語句中的 elif 或者 else 區段都可以依據實際需求來決定是否要添加，例如只有 if 區段的程式如下：

```
n = int(input(' 請輸入一個數字 '))
if n < 0:
    n = -n
```

上述程式會讓輸入值為負的時候改為正的，也就是取絕對值的意思。If、elif 或是 else 區段中當然可以再有 if 區段以形成巢狀條件判斷，只要縮排格式正確即可，如下：

```
text = input(" 請輸入一個字串： ")
if " 開 " in text:
    if " 廚房 " in text:
        print(" 現在廚房開燈了 ")
    if " 客廳 " in text:
        print(" 現在客廳開燈了 ")
```

運算子與邏輯運算子

兩個數值或變數間要怎麼運算靠的是運算子，例如常見的加減乘除。而在 if 判斷式中，兩個變數與數值間的運算結果必須是布林值才能讓 if 決定是否要執行 if 區段中的程式碼，因此我們需要知道哪些運算子可以產生布林值。

運算子	範例	說明
==	a == 10	若變數 a 的內容等於 10 傳回 True
>	a > 10	若變數 a 的內容大於 10 傳回 True
<	a < 10	若變數 a 的內容小於 10 傳回 True
>=	a >= 10	若變數 a 的內容大於等於 10 傳回 True
<=	a <= 10	若變數 a 的內容小於等於 10 傳回 True
!=	a != 10	若變數 a 的內容不等於 10 傳回 True
is	a is None	若變數 a 的內容等於 None 傳回 True
is not	a is not None	若變數 a 的內容不等於 None 傳回 True
in	'開' in text	字串 text 中有「開」這個字傳回 True

None 是 Python 的一個特殊值，代表「空的」或是「什麼都沒有」的意思，相當於其他語言的 null 或是 nil。判斷變數是否為 None 要用 is 或是 is not，不可以用「==」或「!=」。

If 後方接著的條件判斷式並不一定只能有一個判斷式，如果有兩個以上的判斷式，就需要使用邏輯運算子把兩個判斷式連接起來。Python 的邏輯運算子有三個 and、or 與 not，其中 and 與 or 就是用來連結兩個判斷式的，例如：

```
if a >= 10 and a < 20:
    print('範圍在 10 到 20 之間')
```

上述程式碼代表變數 a 的值如果大於等於 10 並且不到 20 的話，就會進入 if 區段。注意邏輯運算子的優先順序是不一樣的，and 的優先順序比 or 大，例如：

```
window = 'close'
temp = 25
humi = 70

if window == 'close' or temp > 28 and humi > 80:
    print('把冷氣打開')
```

上述這段程式碼的意思是如果窗戶關閉或者氣溫超過 28 度且濕度超過 80% 時就把冷氣打開。由於 and 的優先順序比 or 高，因此 and 會先運算，運算完的結果再跟 or 運算。如果希望 or 先運算，可以使用小刮號來改變優先順序，例如下述程式碼，這個意思是只要濕度不到 80% 時，不論氣溫多高或是窗戶是否緊閉，冷氣都不會打開。

```
if (window == 'close' or temp > 28) and humi > 80:
    print('把冷氣打開')
```

邏輯運算子的真值表如下所示，其中 A 與 B 都代表一個獨立的判斷式，例如 A 表示 x >= 10，B 表示 y > 0：

邏輯且（and）

A	B	A and B
True	True	True
True	False	False
False	True	False
False	False	False

邏輯或（or）

A	B	A or B
True	True	True
True	False	True
False	True	True
False	False	False

否定（not）

A	not A
True	False
False	True

For 迴圈

For 迴圈的語法結構如下：

```
for 變數 in [陣列]:
    要重複執行的程式碼
```

雖然陣列在稍後章節才會正式介紹，這裡我們就先簡單理解一下陣列的語法就是用中刮號將一堆資料綁在一起的一種資料結構就好了，例如：

```
zoo = ['長頸鹿', '獅子', '老虎', '斑馬']
```

變數 zoo 中放了四筆資料（較正式的用詞為元素，element），這四筆資料的型態為字串因此用單引號夾住。資料與資料間用逗號「,」分隔，頭尾用中刮號「[…]」夾起來，因此變數 zoo 就是一個陣列。對陣列而言，裡面放的資料型態不一定只能是字串，數字或是其他任何資料型態都可以，我們先瞭解到這樣即可。陣列中的資料數量就代表了這個 for 迴圈要重複執行幾次，每執行一次都會依序從陣列中由左至右取出一筆資料放在 for 後面的變數中，舉例如下：

```
zoo = ['長頸鹿', '獅子', '老虎', '斑馬']
for animal in zoo:
    print('動物園有{}'.format(animal))
```

這段程式的輸出結果是：

```
動物園有長頸鹿
動物園有獅子
動物園有老虎
動物園有斑馬
```

另外一個要特別注意的地方是 for 迴圈第一行最後的「:」號，這代表了一個程式區段的開始，然後在該區段中的所有程式碼都必須要縮排，而且縮排格式必須要一致，這樣所有被縮排的程式碼就是要重複執行的部分了。

接下來試想一個狀況，如果我們需要 for 迴圈重複執行 100 次，難到要手動準備好一個擁有 100 個元素的陣列嗎？當然不是，Python 沒有這麼難搞。函數 range() 永遠是 for 迴圈的好朋友，range() 傳回 for 迴圈所需要的陣列，例如我們想要計算 0+1+2+⋯+100 時，我們使用 range(101) 產生一個 [0, 1, 2, ⋯, 100] 共 101 個元素的陣列。請參考本節補充說明。

```
sum = 0
for i in range(101):
    sum += i
print(sum)
```

呼叫 range(101) 後產生的陣列剛好給 for 迴圈使用，因此 for 迴圈區段中的程式碼就會執行 101 遍。

函數 range() 除了可以接受一個參數外，也可以接受兩個參數，例如 range(5, 10)，代表陣列的內容為 5, 6, 7, 8, 9（不包含 10）；也可以接受三個參數，例如 range(5, 10, 2) 代表從 5 開始一次增加 2，但不超過 10，因此會產生資料為 5, 7, 9 的陣列。

在迴圈中，我們還可以使用 continue 與 break 來改變迴圈的執行流程。如果執行時遇到 continue，程式會立刻回到 for 迴圈的開頭位置，並且從陣列中挑選下一筆資料來執行，例如：

```
for i in range(10):
    if i == 5:
        continue
    print(i)
```

if 語法表示 i 如果等於 5 的話就要執行 continue，這時候 continue 之後的程式碼都不會執行，程式立刻回到 for 的開頭處，並且從陣列中取出下一筆資料，也就是 6 然後繼續執行 for 迴圈的程式區段。因此，當這段程式碼跑完，螢幕上會顯示 0 1 2 3 4 6 7 8 9，少了 5。

若執行到 break，程式會立刻跳出迴圈不論該迴圈還有多少遍需要跑，例如：

```
for i in range(10):
    if i == 5:
        break
    print(i)
```

這時候輸出結果僅僅為 0 1 2 3 4。

當迴圈結束時我們可以在 else 區段中撰寫一些離開迴圈前會執行的程式碼，例如：

```
for i in range(10):
    print(i)
else:
    print('迴圈即將結束')
print('迴圈已結束')
```

注意 else 跟 for 是同一層（縮排層級一樣）。這裡 else 區段中的程式碼只有在 for 迴圈「正常」結束時才會執行，意思是如果用 break 跳離迴圈的話，else 區段是不會執行的，但 continue 會。

While 迴圈

除了 for 迴圈外，while 迴圈是 Python 的另外一種迴圈形式，語法是 while 後方接一個邏輯判斷式，如果該判斷式傳回 True 則進入迴圈區段中執行，例如要產生費式數列（Fibonacci）中的前 10 個，程式碼如下：

```
n = 0
a, b = 1, 1
while n < 10:
    print (a, end=' ')
    a, b = b, a + b
    n += 1
# 輸出結果為 1 1 2 3 5 8 13 21 34 55
```

我們在 for 迴圈中看到的 break、continue 與 else 等用法，在 while 迴圈中也可以使用，用法跟 for 迴圈一模一樣。

補充說明

在 Python 3 中，range() 函數傳回結果嚴格講起來並不是陣列而是一個物件，我們可以用 type() 函數來證明這樣的結果（為方便起見，這裡直接在 Python 直譯環境中輸入程式碼），如下：

```
>>> range(10)
range(0, 10)
>>> type(range(10))
<class 'range'>
```

如果想要得到 range() 所產生的陣列，可以使用 list() 函數做型別轉換，如下：

```
>>> list(range(10))
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

1-2-5 資料結構

Python 具有四大資料結構，分別是 list、tuple、set 與 dictionary，分述如下：

List

串列，也是陣列。雖然在有些程式語言中，串列跟陣列不一樣，不過在 Python 裡這兩者沒什麼差別，範例如下：

```
arr = ['台北', '台中', '高雄']
```

語法是用中刮號夾住每一個元素。在 Python 中，陣列的每一個元素的資料型態不用一致，例如下述陣列中的元素包含了字串、數字與另一個陣列。

```
arr = ['台北', 30, 0.7, [1, 2, 3]]
```

存取陣列內容是透過陣列索引，例如要取得上述陣列中的 0.7 這個值，或是將台北改為花蓮，程式碼如下：

```
print(arr[2])  ## 印出 0.7
arr[0] = '花蓮'
```

Ptyhon 的陣列索引值一律從 0 開始一直到陣列元素個數減 1，換句話說陣列中的第一個元素其索引值為 0，其實幾乎每個語言都是如此。但是 Python 的陣列索引值除了有「正」索引值之外，還有「負」索引值，對映方式如下表：

```
陣列 arr = ['台北', '新竹', '台中', '嘉義', '台南', '高雄']
```

陣列內容	台北	新竹	台中	嘉義	台南	高雄
正索引	0	1	2	3	4	5
負索引	-6	-5	-4	-3	-2	-1

由此可知，如果要存取陣列中最後一筆資料，可以使用 arr[-1] 即可，非常方便。例如取出高雄除了可以用 arr[5] 之外，也可以使用 arr[-1]。

常用的函數有：

- len(list)：取得串列元素個數。

```
arr = ['台北', '台中', '高雄']
print(len(arr)) ## 印出 3
```

- append(element)：從陣列尾端附加一個元素。

```
arr = ['台北', '台中', '高雄']
arr.append('屏東')
## ['台北', '台中', '高雄', '屏東']
```

- insert(index, element)：從指定位置插入一個元素，例如將「基隆」放在陣列開頭位置。

```
arr = ['台北', '台中', '高雄']
arr.insert(0, '基隆')
## ['基隆', '台北', '台中', '高雄']
```

- remove(element)：刪除第一個符合的元素。

```
arr = ['台北', '台中', '高雄']
arr.remove('台中')
## ['台北', '高雄']
```

補充說明

若要刪除所有符合條件的元素，程式碼如下：

```
arr = ['基隆', '台北', '台中', '高雄', '基隆']
arr = [c for c in arr if c != '基隆']
## ['台北', '台中', '高雄']
```

- sort([reverse=False])：排序。

```
arr = [1, 5, 2, 3]
arr.sort()
## [1, 2, 3, 5]
```

```
arr = [1, 5, 2, 3]
arr.sort(reverse=True)
## [5, 3, 2, 1]
```

- sorted(list[, reverse=False])：排序。與 sort() 不同處在於 sorted() 不會修改原本的陣列內容，而是將排序後的結果放到新的陣列中。

```
arr = [1, 5, 2, 3]
new = sorted(arr, reverse=True)
## arr = [1, 5, 2, 3]
## new = [5, 3, 2, 1]
```

- copy()：陣列複製。先看一段程式碼。下方程式碼中，當修改陣列 b 的內容時，同時間也是動到陣列 a 的內容。

```
a = [1, 2, 3, 4]
b = a
b[2] = 10
print(a)
## [1, 2, 10, 4]
print(b)
## [1, 2, 10, 4]
```

若不希望修改 b 時 a 也跟著改，就需要將 a 的內容複製給 b。

```
a = [1, 2, 3, 4]
b = a.copy()
b[2] = 10
print(a)
## [1, 2, 3, 4]
print(b)
## [1, 2, 10, 4]
```

Tuple

Tuple 跟陣列非常類似，差別在於一旦將值放到 tuple 後就無法修改其內容了，換句話說，tuple 相當於「常數」陣列。語法範例如下，注意使用小括號：

```
a = (1, 2, 3, 4)
```

取得 tuple 中的元素內容，跟陣列一樣，使用索引值，範例如下，注意這裡使用中括號框住索引值：

```
a = (1, 2, 3, 4)
print(a[2]) ## 印出 3
```

由於無法修改 tuple 的內容，因此常用的函數只有一個：

- len(tuple)：取得 tuple 中的元素個數。

```
a = (1, 2, 3, 4)
print(len(a)) ## 印出 4
```

Set

Set 為集合，特性是集合中的元素具有唯一性，並且也沒有順序性，使用的符號為大括號，語法範例如下：

```
s = {'淡水', '日月潭', '墾丁'}
```

通常使用運算子 in 或是 not in 來檢查某個元素是否在集合中，例如：

```
s = {'淡水', '日月潭', '墾丁'}
print('淡水' in s)       ## True
print('野柳' not in s)   ## True
```

常用的函數有：

- len(set)：取得集合中元素個數。

```
s = {'淡水', '日月潭', '墾丁'}
print(len(s))  ## 印出 3
```

- add(element)：將元素加到集合中，如果集合中已經存在此元素，不會報錯，也不會多一個。

```
s = {'淡水', '日月潭', '墾丁'}
s.add('關山')
## {'關山', '淡水', '日月潭', '墾丁'}
```

- remove(element)：從集合中刪除某元素。

```
s = {'淡水', '日月潭', '墾丁'}
s.remove('日月潭')
## {淡水', '墾丁'}
```

- issubset(S)：判斷是否為 S 的子集合。

```
s1 = {1, 2, 3, 4}
s2 = {2, 3}
print(s2.issubset(s1))  ## True
```

- issuperset(S)：判斷是否為 S 的超集合。

```
s1 = {1, 2, 3, 4}
s2 = {2, 3}
print(s1.issuperset(s2))  ## True
```

- isdisjoint(S)：判斷是否與 S 的交集為空集合。

```
s1 = {1, 2, 3, 4}
s2 = {5}
print(s1.isdisjoint(s2))  ## True
```

- union(S)：聯集運算。

```
s1 = {1, 2, 3, 4}
s2 = {4, 5}
print(s1.union(s2))
## {1, 2, 3, 4, 5}
```

- intersection(S)：交集運算。

```
s1 = {1, 2, 3, 4}
s2 = {4, 5}
print(s1.intersection(s2))
## {4}
```

- difference(S)：差集運算。

```
s1 = {1, 2, 3, 4}
s2 = {4, 5}
print(s1.difference(s2))
## {1, 2, 3}
print(s2.difference(s1))
## {5}
```

Dictionary

字典結構，為因應 json 格式而衍生出來的資料結構。字典結構有點類似集合與陣列的混和體。字典結構中的每一個元素包含了 key 與 value 這兩個部分，key 可以是字串、數字或是布林型態，value 的型態沒有限制，除此之外，每一個字典結構中 key 不可以重複。存取方式跟陣列一樣，只不過索引值的部分換成 key 即可。語法範例如下：

```
d = {'uid': 'A01', 'name': '王大明'}
print(d['name'])     ## 印出 王大明
d['uid'] = 'B01'     ## 將 uid 的內容 A01 改為 B01
print(d['uid'])      ## 印出 B01
```

要在字典中增加一筆資料，只要直接給一個新的 key 值就可以了，如果字典中已經存在這個 key，就相當於修改該 key 的值。

```
d = {'uid': 'A01', 'name': '王大明'}
d['age'] = 30
print(d['age'])      ## 印出 30
```

刪除 key 可用 del 運算子，如下：

```
d = {'uid': 'A01', 'name': '王大明'}
del d['name']
## {'uid': 'A01'}
```

常用的函數有：

- len(dict)：取得字典中 key-value 配對的個數。

```
d = {'uid': 'A01', 'name': '王大明'}
print(len(d))    ## 印出 2
```

- keys()：取得字典中所有的 key 並且以陣列形式傳回。

```
d = {'uid': 'A01', 'name': '王大明'}
for k in d.keys():
    print(k)
```

- values()：取得字典中所有的 value 並且以陣列形式傳回。

```
d = {'uid': 'A01', 'name': '王大明'}
for v in d.values():
    print(v)
```

- items()：取得字典中所有的 key-value 配對並且以陣列形式傳回。

```
d = {'uid': 'A01', 'name': '王大明'}
for p in d.items():
    print(p)
## ('uid', 'A01')
## ('name', '王大明')
```

1-2-6 命令列參數

當我們在執行 Python 程式時除了可以透過標準輸入方式取得使用者輸入的資料外，也可以使用命令列參數的方式取得。所謂的命令列參數就是讓使用者將資料直接放在命令列，例如在 UNIX 中的「ls -l」查詢目錄下所有檔案的指令，其中「-l」就是命令列參數。

由於不知道使用者會在命令列放幾個參數，因此所有的參數都會存放在一個特定的陣列中，而該陣列被定義在 sys 這個函數庫，因此使用命令列參數時必須要先匯入這個函數庫，然後再透過陣列索引值來取得每一個參數值。

步驟與說明

1. 編輯 args.py（檔名任取）。

```
import sys
print(sys.argv)
```

2. 執行時輸入以下參數。

```
$ python3 args.py 1 2 3
```

3. 執行看看。

```
['args.py', '1', '2', '3']
```

補充說明

從命令列參數取得的值，其資料型態一律為 str。

4. 將參數加總起來。

```
ans = int(sys.argv[1]) + int(sys.argv[2]) + int(sys.argv[3])
print(ans)    ## 印出 6
```

1-2-7 錯誤處理

當程式遇到執行不下去的時候會產生執行時錯誤，然後當掉，例如分母為零的除法運算。為避免程式當掉，所以我們要處理執行時錯誤，也就是偵測、攔截、處理。在 Python，我們用 try except 語法來完成錯誤處理程序。

步驟與說明

1. 設計一個除法運算程式。

```
x = input('請輸入分子')
y = input('請輸入分母')
r = float(x) / float(y)
print(r)
```

2. 這段程式碼至少會產生兩種不同的錯誤：一種是分母輸入 0；另一種是輸入的不是數字，導致轉成 float 時出現無法轉換的錯誤。分母為 0 的錯誤訊息為 ZeroDivisionError: float division by zero，型態錯誤的錯誤訊息為 ValueError: could not convert string to float: 'abc'。

3. 將會造成執行時錯誤的程式碼放到 try 區段中。

```
x = input('請輸入分子')
y = input('請輸入分母')
try:
    r = float(x) / float(y)
    print(r)
except ZeroDivisionError:
    print('分母不可為零')
except ValueError:
    print('請輸入數字')
finally:
    print('不論有沒有錯都會執行到這裡')
```

4. 執行看看。

補充說明

我們也可以用這樣的程式碼來攔截所有的錯誤與問題（包含按 Ctrl-C 中斷程式）。

```
import sys
try:
    x = input('請輸入分子')
    y = input('請輸入分母')
    r = float(x) / float(y)
    print(r)
except Exception as e:
    # 分母為零或是輸入的不是數字
    print('錯誤：{}'.format(e))
except:
    # 按下 Ctrl+C 中斷程式
    print('其餘問題：{}'.format(sys.exc_info()[0]))
finally:
    print('不論有沒有錯都會執行到這裡')
```

1-3 字串處理

Python 的字串型別具有豐富的字串處理函數可以讓我們很容易的操作字串，例如子字串處理、搜尋、取代、轉大小寫等。

1-3-1 字串長度

計算字串長度使用 len() 函數，英文以字元中文以字為單位。

步驟與說明

1. 計算 hello world 的長度。

```
s = 'hello world'
print(len(s))
```

2. 執行看看，結果為 11。

1-3-2 子字串

要取得一個字串中的部分字串，就把該字串當作字元陣列，然後用陣列方式來處理即可。比較特殊的地方是 Python 的陣列索引值除了有正索引之外，還有負索引。

字串	p	y	t	h	o	n
正索引	0	1	2	3	4	5
負索引	-6	-5	-4	-3	-2	-1

步驟與說明

1. 設定一個字串。

```
s = 'python'
```

2 透過陣列索引值取得字串的各個部分。

```
# 取索引 2 的字
print(s[2])      # t

# 取索引 2 到 4 但是不包含 4
print(s[2:4])    # th

# 取索引 1 到索引 -2 但不包含 -2
print(s[1:-2])   # yth

# 開頭取 4 個字
print(s[:4])     # pyth

# 結尾取 3 個字
print(s[-3:])    # hon
```

3 執行看看。

1-3-3 搜尋

字串搜尋的目的是找到特定字串在某個字串中的位置，找到位置後才能進一步的處理，例如把特定長度的字串取出來。Python 的字串搜尋有兩個函數：index() 與 find()。這兩個函數的參數與用法幾乎一模一樣，唯一的差別在於如果找不到特定字串的時候，index() 會丟出 exception，而 find() 會傳回 -1。這裡以 index() 為範例，find() 就不再贅述。

步驟與說明

1 在 hello world 字串中搜尋字串 o，如果有找到會傳回 o 所在的位置，如果找不到會丟出 ValueError 的 exception，所以用 try except 攔截這個錯誤就可以知道是否有找到。

```
s = 'hello world'
try:
    index = s.index('o')
    print(index)
except:
    print('not found')
```

2 執行後會印出 4，因為 o 在第 5 個字，其陣列索引值為 4。

3 若要找出字串中所有 o 的位置（hello world 中有兩個 o），則需要靠 index() 函數中的第 2 與第 3 個參數。第 2 個參數是設定從哪個位置開始搜尋，預設是 0 代表從頭開始搜尋；第 3 個參數是設定搜尋到那個位置，預設是最後一個字。所以這兩個參數的目的就是設定搜尋範圍。

```
s = 'hello world'
try:
    index = -1
    while True:
        index = s.index('o', index + 1, len(s))
        print(index)
except:
    if index == -1:
        print('not found')
```

4 執行看看。會印出 4 與 7。

1-3-4 頭尾去空白

當某個變數中的字串頭尾都有空白的時候，這裡指的「空白」除了空白鍵之外還包含了 TAB 鍵與換行符號，若這些空白會影響資料正確性的時候，就需要將他們移除。

步驟與說明

1 一個頭尾都有空白的字串。

```
s = '    \t\nhello world\n\r    '
```

2 去空白處理有三個函數：lstrip()、rstrip() 與 strip()。

```
t = s.lstrip()
print('左側去空白：[{}]'.format(t))

t = s.rstrip()
print('右側去空白：[{}]'.format(t))

t = s.strip()
print('頭尾去空白：[{}]'.format(t))
```

3 執行看看。

```
左側去空白：[hello world
    ]
右側去空白：[
hello world]
頭尾去空白：[hello world]
```

1-3-5 取代

取代的目的是將字串中特定的字換成別的字，例如將字串 abc 中的 b 換成 5 變成 a5c。

步驟與說明

1 設定一個字串。

```
s = 'hello world'
```

2 將小寫的 l 換成大寫 L。

```
t = s.replace('l', 'L')
print(t)
```

3 執行看看。

```
heLLo worLd
```

補充說明

replace() 函數可以接受第 3 個參數，該參數用來指定要取代幾個，例如：

```
t = s.replace('l', 'L', 1)
```

這樣執行完的結果會是 heLlo world，只換掉第一個 l。

1-3-6 分割

若字串中有一些特定的字串或符號，我們就可以從這個地方把原本的字串分割開來，分開後的結果以陣列形式傳回。

步驟與說明

1. 一個以「:」分隔三個欄位資料的字串。

```
s = 'A01:王大明:臺中市台灣大道1號'
```

2. 從「:」處將字串分割成三段字串，這三段字串以陣列形式傳回。

```
arr = s.split(':')
print(arr)
```

3. 執行看看。

```
['A01', '王大明', '臺中市台灣大道1號']
```

補充說明

split() 函數接受第二個參數，該參數設定要分割幾次，例如：

```
arr = s.split(':', 1)
```

這樣執行結果會是：

```
['A01', '王大明:臺中市台灣大道1號']
```

1-3-7 特定開頭與結尾

當我們想要知道字串是否由某個特定的字串開頭，或是某個特定的字串結尾時，就可以用這個方式來判定，如果有就會傳回 True，否則傳回 False。

步驟與說明

1. 設定字串。

```
s = '溫度：20 度 C'
```

2. 判定是否有某個字串開頭用 startswith() 函數。

```
print(s.startswith('溫度'))
```

3. 判定是否有某個字串結尾用 endswith() 函數

```
print(s.endswith('F'))
```

4. 執行看看。

```
True
False
```

1-3-8 轉大小寫

這裡的函數當然只對英文有效，如果是中文字，自然沒有大小寫之分。

步驟與說明

1. 設定字串。

```
s = 'hello WORLD'
```

2 轉大寫、轉小寫與首字大寫。

```
print('全部轉大寫：{}'.format(s.upper()))
print('全部轉小寫：{}'.format(s.lower()))
print('首字大寫：{}'.format(s.capitalize()))
```

3 執行看看。

```
全部轉大寫：HELLO WORLD
全部轉小寫：hello world
首字大寫：Hello world
```

1-3-9 運算子 in

如果只是想要知道字串中是否包含了某個特定的字串，不一定需要使用 index() 或 find() 函數，只要使用運算子 in 就可以了。in 會傳回 True 或 False 來表示該字串中是否包含了某特定字串。

步驟與說明

1 設定一個字串。

```
s = 'hello world'
```

2 檢查某個字串是否包含於其中。

```
if 'wo' in s:
    print('包含')
else:
    print('未包含')
```

3 執行看看。結果會印出「包含」。