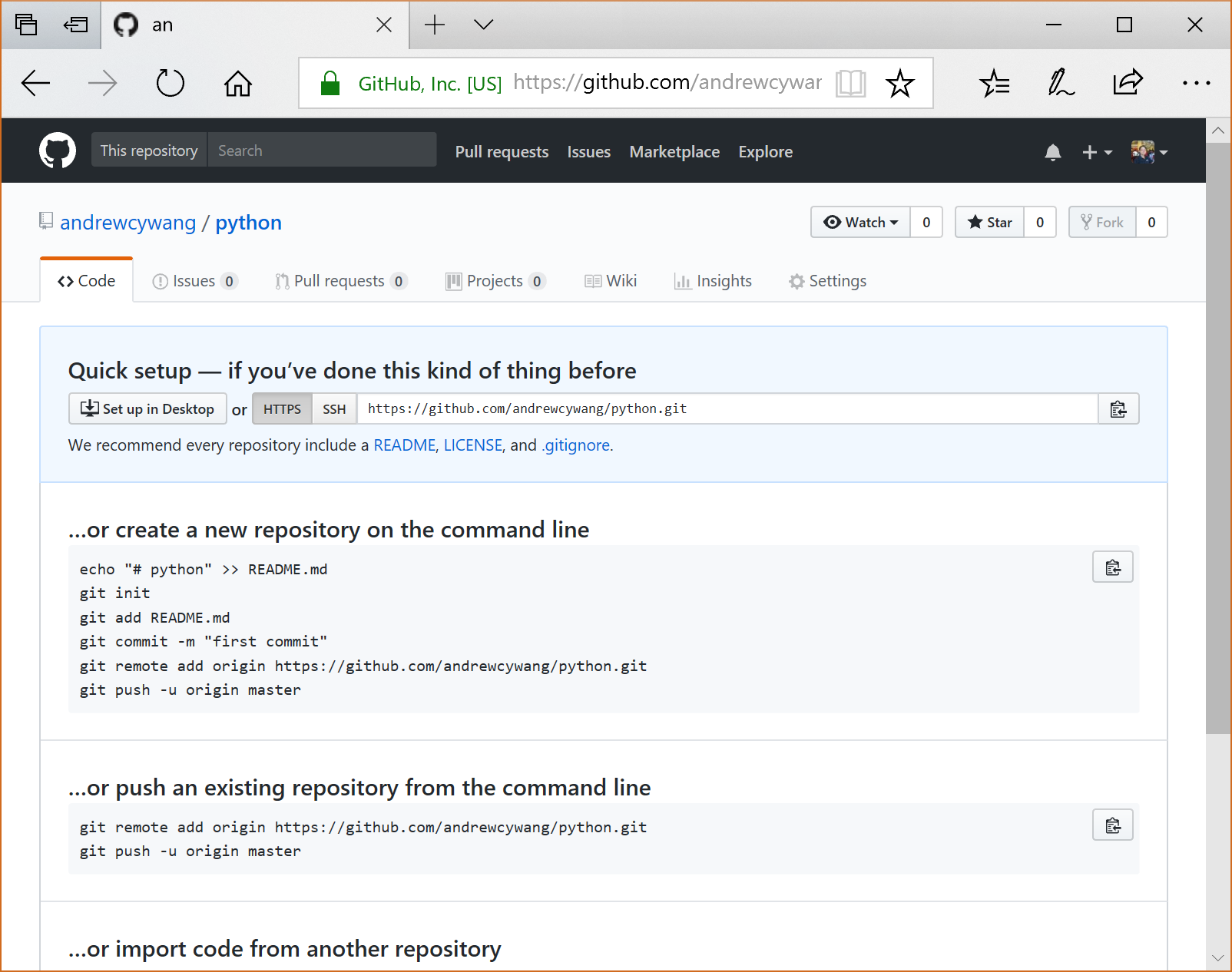
# . Python原來可以這樣玩 .

作者：王昌元

## 課前準備

### GitHub Repository

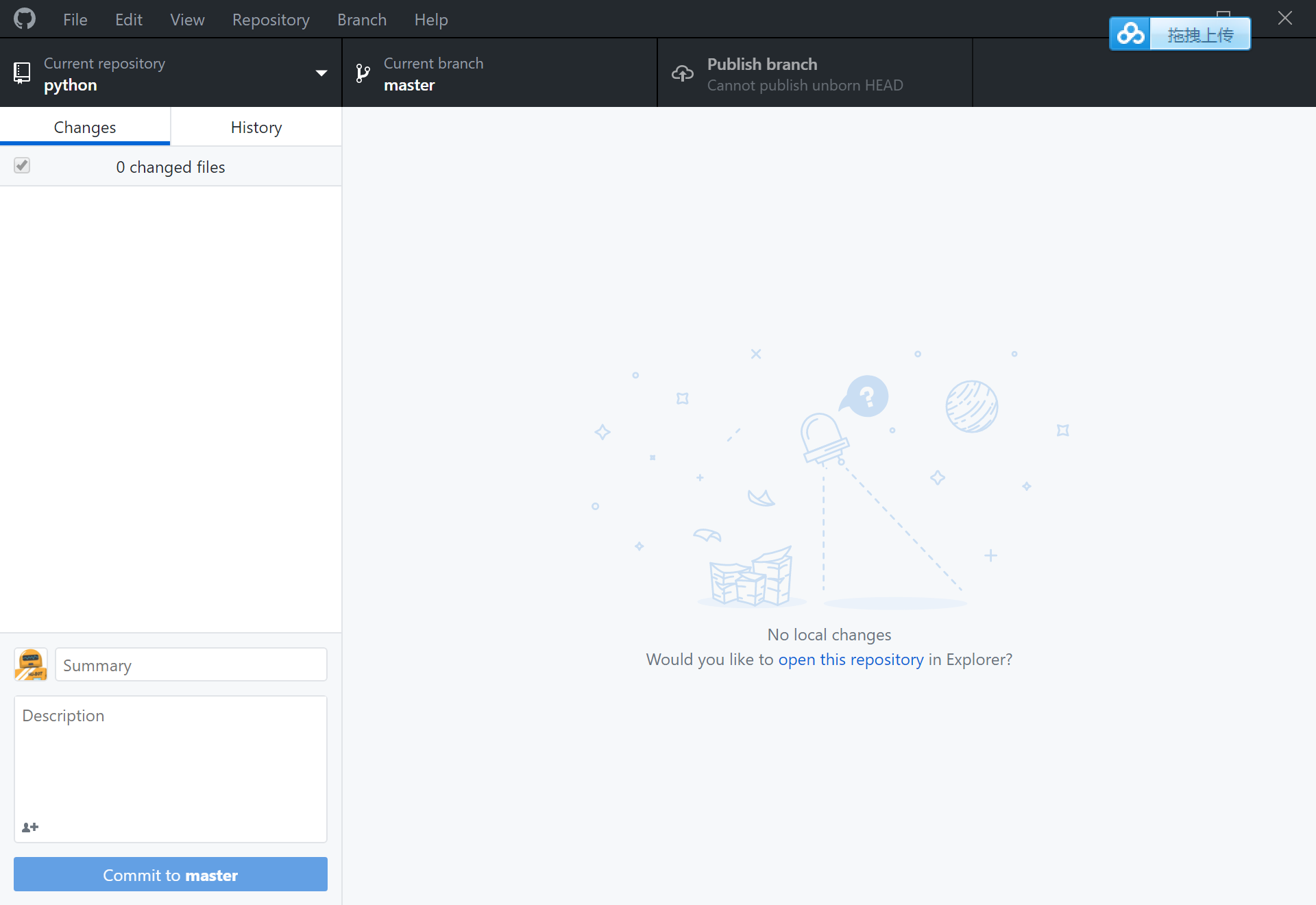
GitHub 是一個很好的工具，可以讓我們共同維護程式碼，因此我使用 GitHub作為教案以及程式碼分享的平台，如下圖：



在我的 andrewcywang 帳號中，我開了一個 python 的 Repository，所有的分享都會放在其中。

### GitHub Desktop

有了存放空間之後，也在桌面安裝了 GitHub Desktop，可以讓所有資料同步。如下圖：



## Python 簡介

Python（英國發音：/ˈpaɪθən/ 美國發音：/ˈpaɪθɑːn/），是一種廣泛使用的高階程式語言，屬於通用型程式語言，由吉多·范羅蘇姆 創造，第一版釋出於 1991 年。可以視之為一種改良的 LISP。作為一種直譯語言，Python 的設計哲學強調代碼的可讀性和簡潔的語法。相比於 C++ 或 Java，Python 讓開發者能夠用更少的代碼表達想法。不管是小型還是大型程式，該語言都試圖讓程式的結構清晰明了。

與 Scheme、Ruby、Perl、Tcl 等動態型別程式語言一樣，Python 擁有動態型別系統和垃圾回收功能，能夠自動管理記憶體使用，並且支援多種編程範式，包括物件導向、命令式、函數式和程序式編程。其本身擁有一個巨大而廣泛的標準庫。

### Python應用範圍

#### Web程式

Python經常被用於Web開發。比如，通過mod\_wsgi模組，Apache可以運行用Python編寫的Web程式。使用Python語言編寫的Gunicorn作為Web伺服器，也能夠執行Python語言編寫的Web程式。Python定義了WSGI標準應用介面來協調Http伺服器與基於Python的Web程式之間的溝通。一些Web框架，如Django、Pyramid、TurboGears、Tornado、web2py、Zope、Flask等，可以讓程式設計師輕鬆地開發和管理複雜的Web程式。

Python對於各種網路協定的支援很完善，因此經常被用於編寫伺服器軟體、網路爬蟲。第三方函式庫Twisted支援非同步線上編寫程式和多數標準的網路協定（包含用戶端和伺服器），並且提供了多種工具，被廣泛用於編寫高效能的伺服器軟體。另有gevent這個流行的第三方庫，同樣能夠支援高效能高並行的網路開發。

#### GUI開發

Python本身包含的Tkinter庫能夠支援簡單的GUI開發。但是越來越多的Python程式設計師選擇wxPython或者PyQt等GUI套件來開發跨平台的桌面軟體。使用它們開發的桌面軟體執行速度快，與用戶的桌面環境相契合。通過PyInstaller還能將程式釋出為獨立的安裝程式包。

#### 作業系統

在很多作業系統裡，Python是標準的系統元件。大多數Linux發行版和Mac OS X都整合了Python，可以在終端機下直接執行Python。有一些Linux發行版的安裝器使用Python語言編寫，比如Ubuntu的Ubiquity安裝器、Red Hat Linux和Fedora的Anaconda安裝器。在RPM系列Linux發行版中，有一些系統元件就是用Python編寫的。Gentoo Linux使用Python來編寫它的Portage軟體包管理系統。Python標準庫包含了多個調用作業系統功能的函式庫。通過pywin32這個第三方軟體包，Python能夠存取Windows的COM服務及其它Windows API。使用IronPython，Python程式能夠直接調用.Net Framework。

#### 科學、遊戲、與其他應用

NumPy、SciPy、Matplotlib可以讓Python程式設計師編寫科學計算程式。有些公司會使用Scons代替make構建C++程式。

很多遊戲使用C++編寫圖形顯示等高效能模組，而使用Python或者Lua編寫遊戲的邏輯、伺服器。相較於Python，Lua的功能更簡單、體積更小；而Python則支援更多的特性和資料類型。很多遊戲，如EVE Online使用Python來處理遊戲中繁多的邏輯。

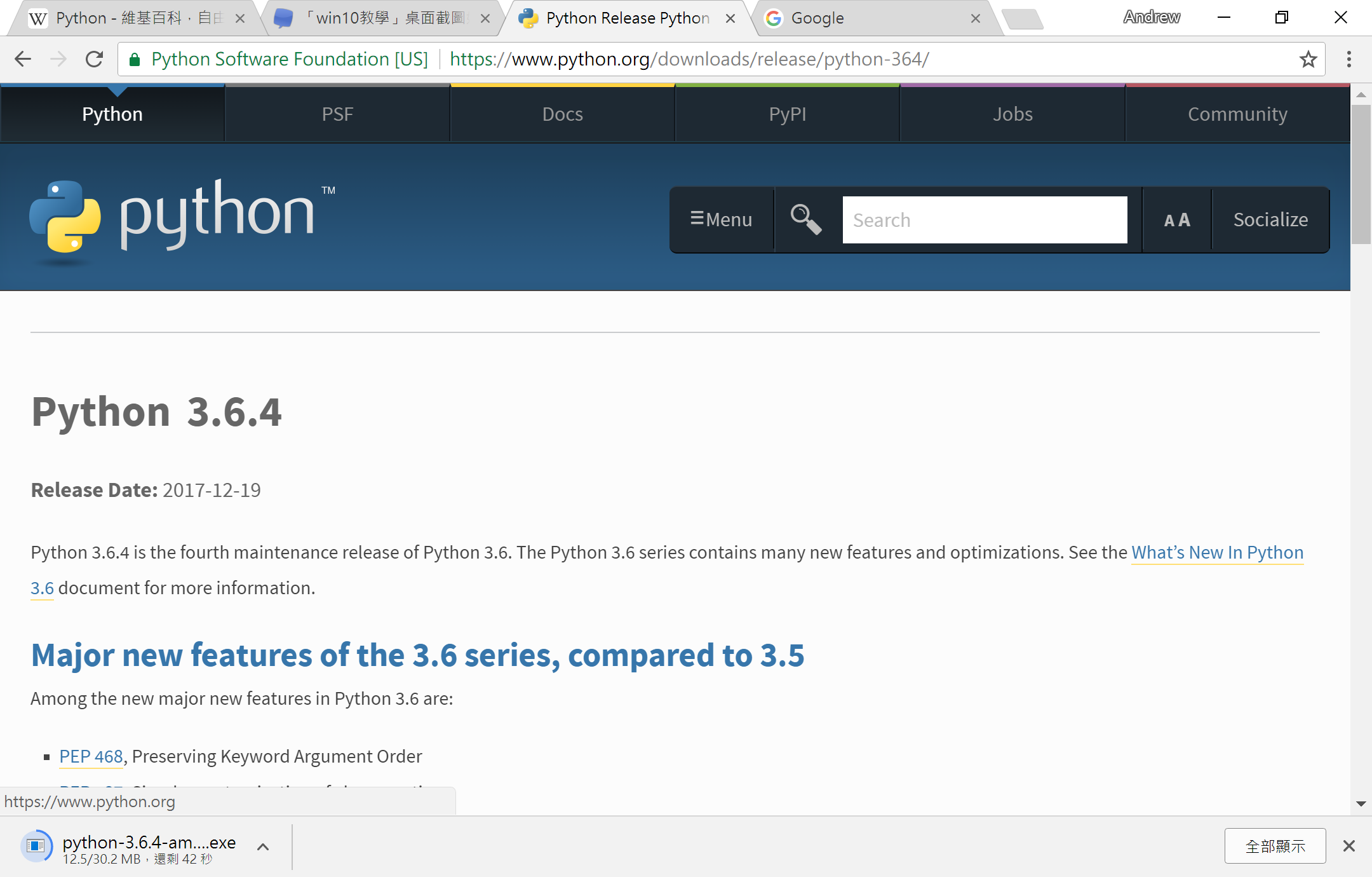
YouTube、Google、Yahoo!、NASA都在內部大量地使用Python。OLPC的作業系統Sugar項目的大多數軟體都是使用Python編寫。

## Python 開發環境

工欲善其事，必先利其器，程式設計也是這樣，熟悉開發環境是應該學習的第一步。

### IDLE

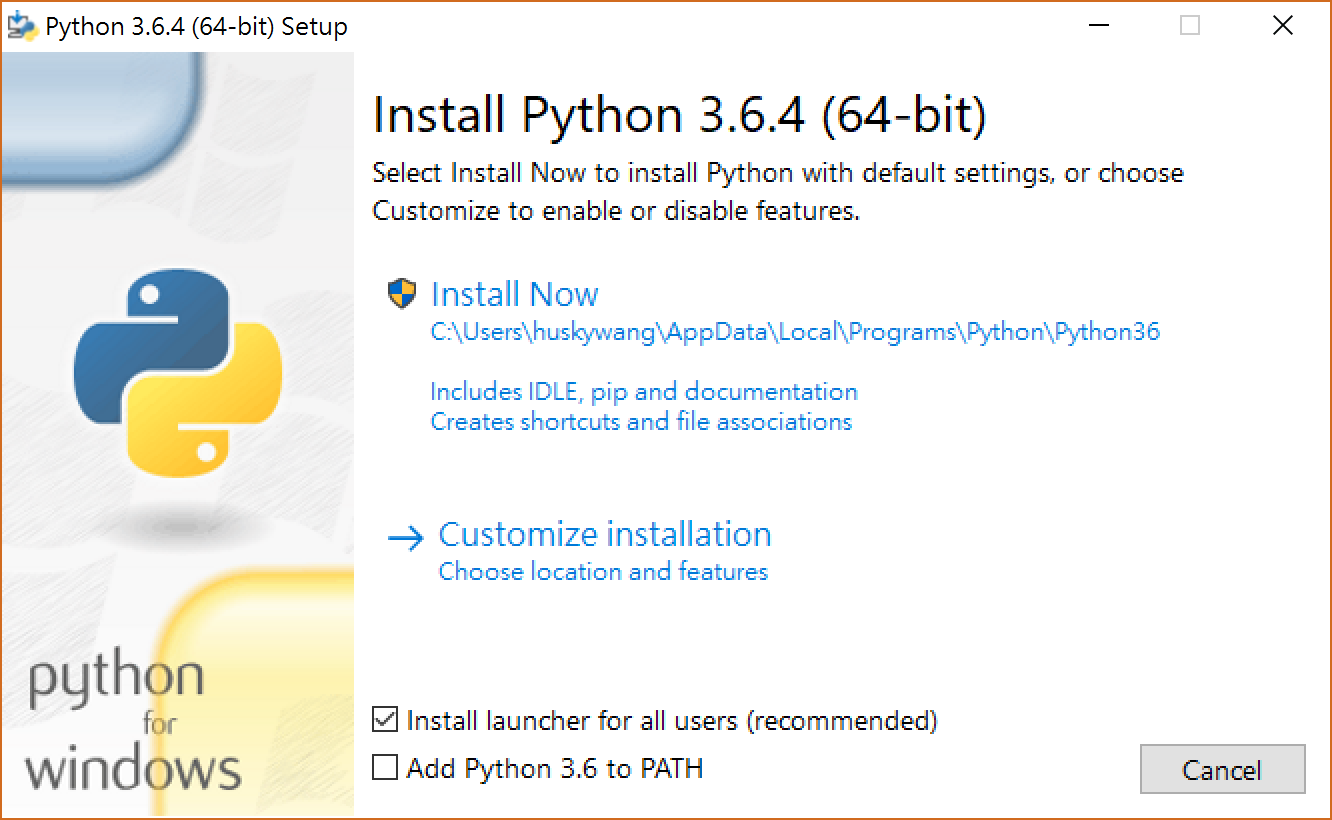
IDLE是Python的官方標準開發環境，可以從官方網站www.python.org下載。



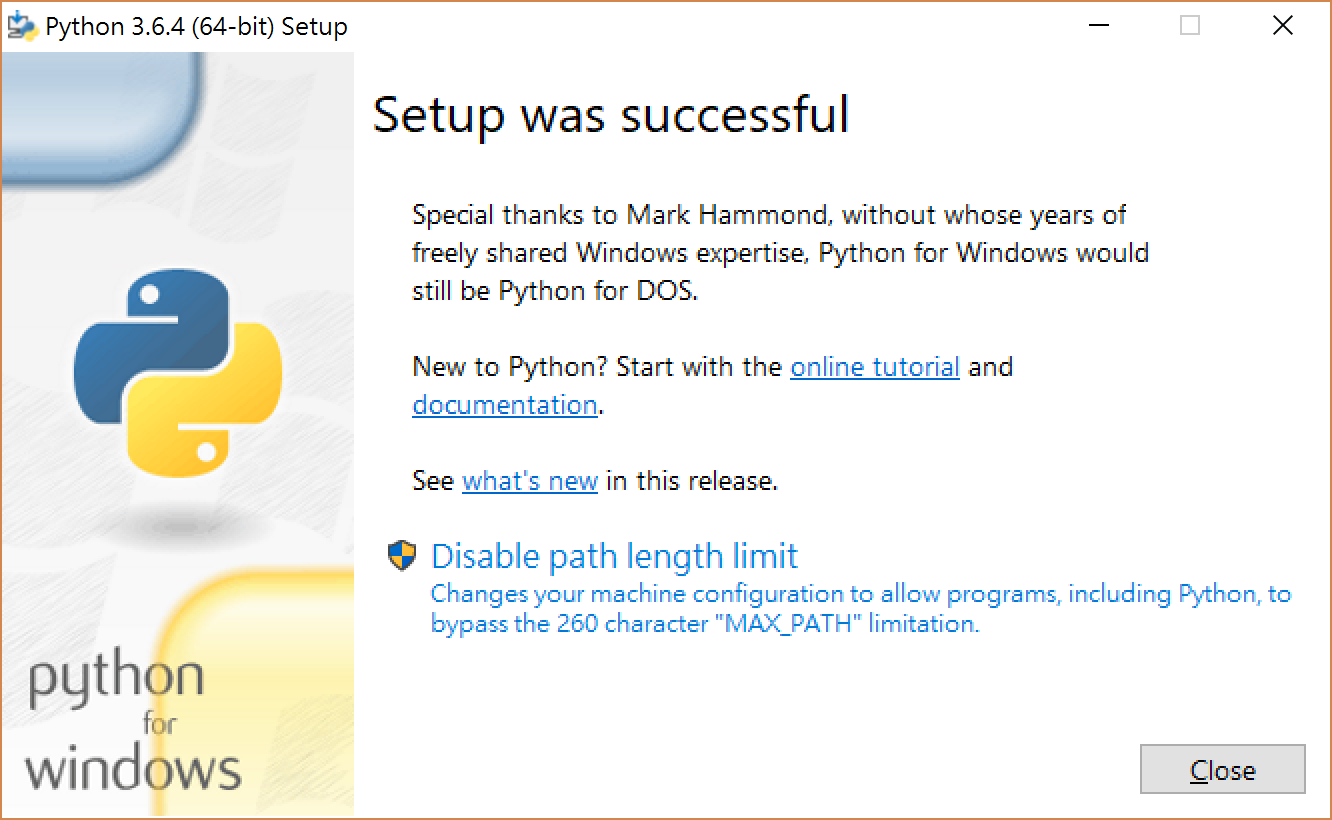
目前最新的版本會顯示在官方網頁，要下載 Windows 的安裝版本。有了最基本的工具，就可以開始寫程式了。為了打好基礎，建議從最基本的開始。

### 安裝 Python 3.6.4

直接執行 python-3.6.4-amd64.exe，進入安裝畫面：



安裝的第一步，建議將 Add Python 3.6 to PATH打勾，這樣在DOS視窗執行的時候，就不必一直切換目錄，按下 Install Now安裝。安裝完畢之後會出現下圖：

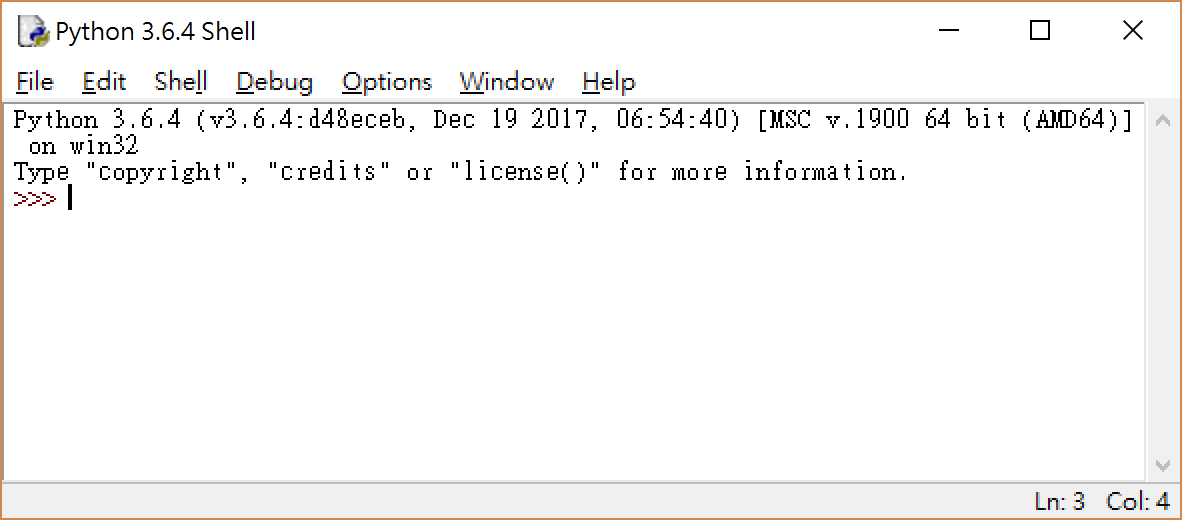


我也建議按下 Disable path length limit，突破過去DOS的限制。

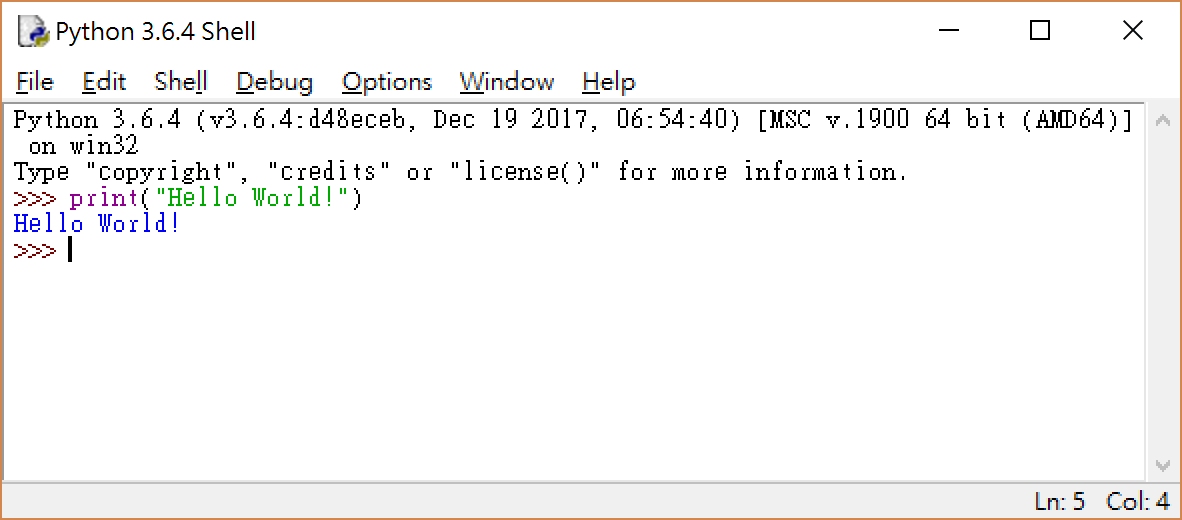
安裝完成之後，我們就可以從[開始]功能表執行[IDLE Python 3.6]，正式進入 Python 的世界。

### 撰寫第一個Python程式

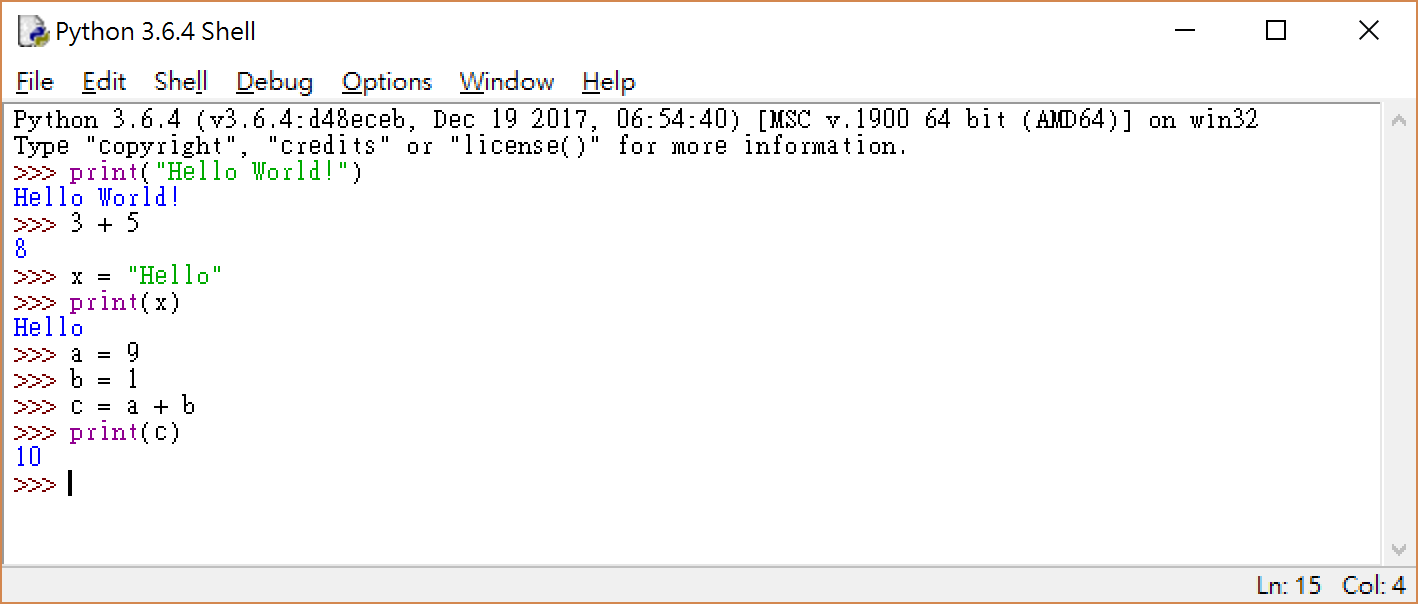
讓我們撰寫一個 Hello World! 程式，這是學習任何一種程式語言的慣例。開啟 Python IDLE之後，會看到下面的畫面：



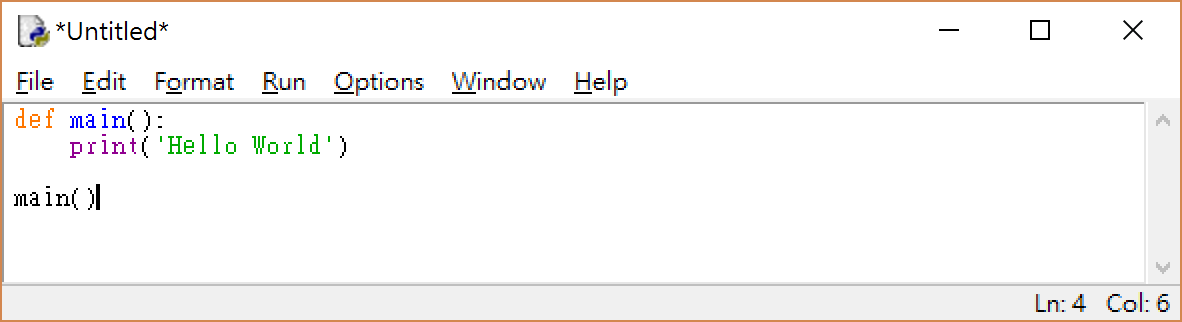
感覺上，我們又好像回到了DOS時代，這畫面就是一個簡單的執行殼，我們直接在>>>後面輸入程式指令，按下ENTER之後執行。Hello World! 程式就是讓 Python 輸出一個 Hello World!字串，我們使用 print 函數來達成：



我們現在看到的這個模式，稱之為交互模式，直接在Python提示字元 “>>>” 輸入命令並按下[ENTER]按鈕即可，如果命令正確的話，馬上就可以看到結果：

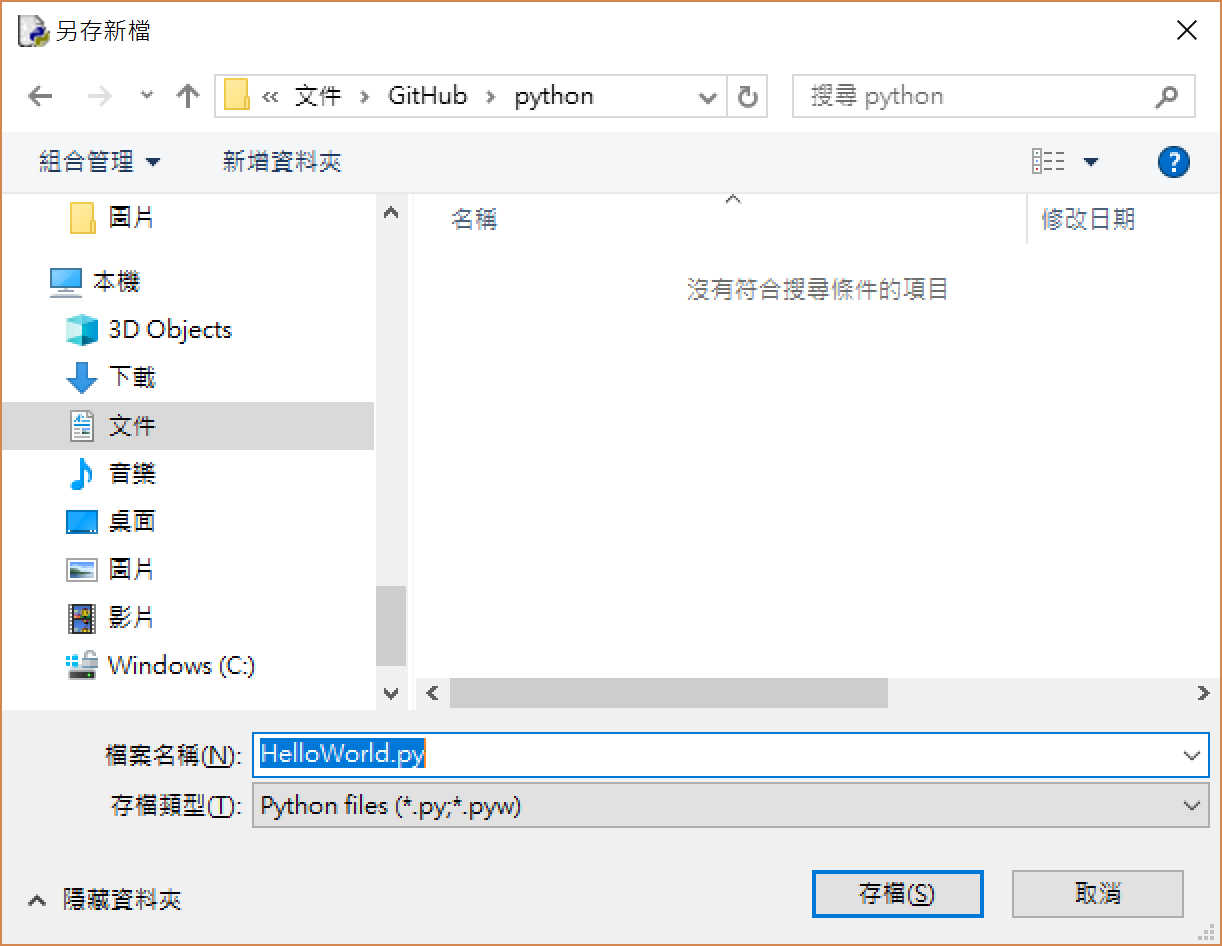


在這裡執行命令就好像跟Python對話一樣，但是都是一行一行執行的，並不像執行程式，程式都是由多個命令所組成的，然後儲存在一個檔案之中，如果我們要把 Hello World 寫成一個程式該如何做呢？第一步當然是建立一個新的程式檔案。在IDLE介面使用功能表[FILE] [NEW FILE]新建一個檔案，這時候會出現一個新的文字編輯器，按照 Python 的語法，程式碼是寫在一個叫做 main() 的函數裡面，這和C語言很像，定義好之後再執行 main() 即可，寫法如下：

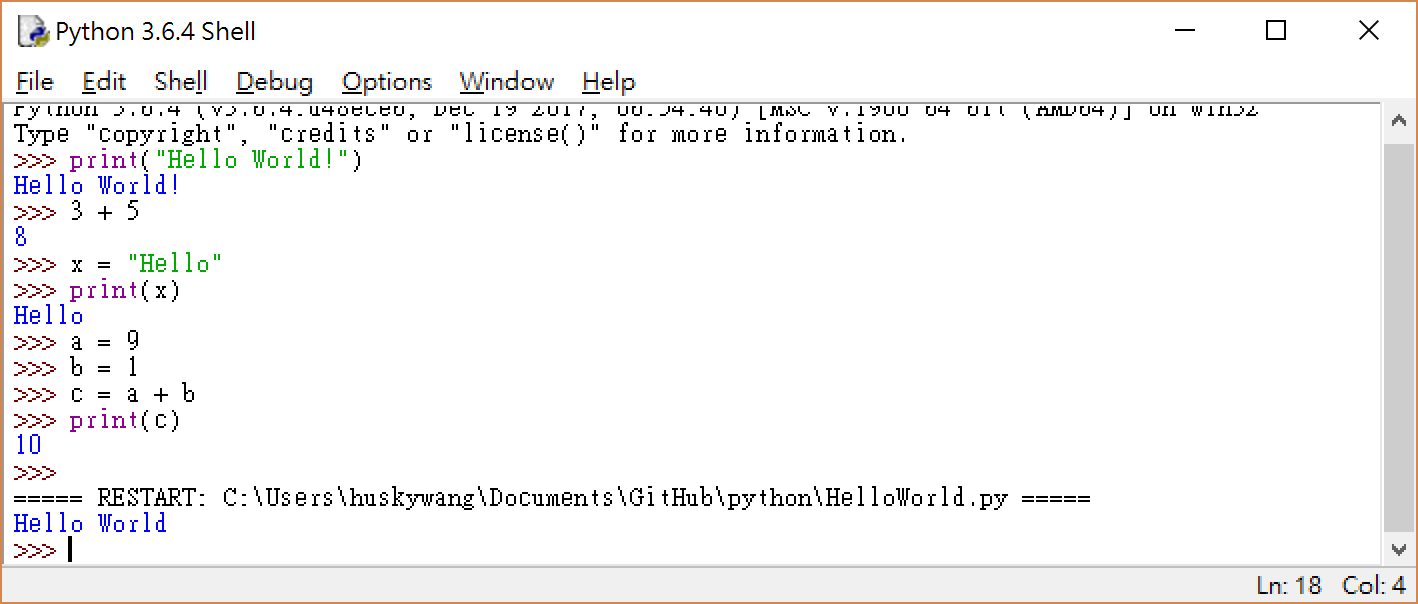


前兩行我們使用關鍵字 def 定義了 main()，要做的事情則利用縮排寫在裡面，就是 print('Hello World') ，然後第三行就是讓 Python 執行 main()。

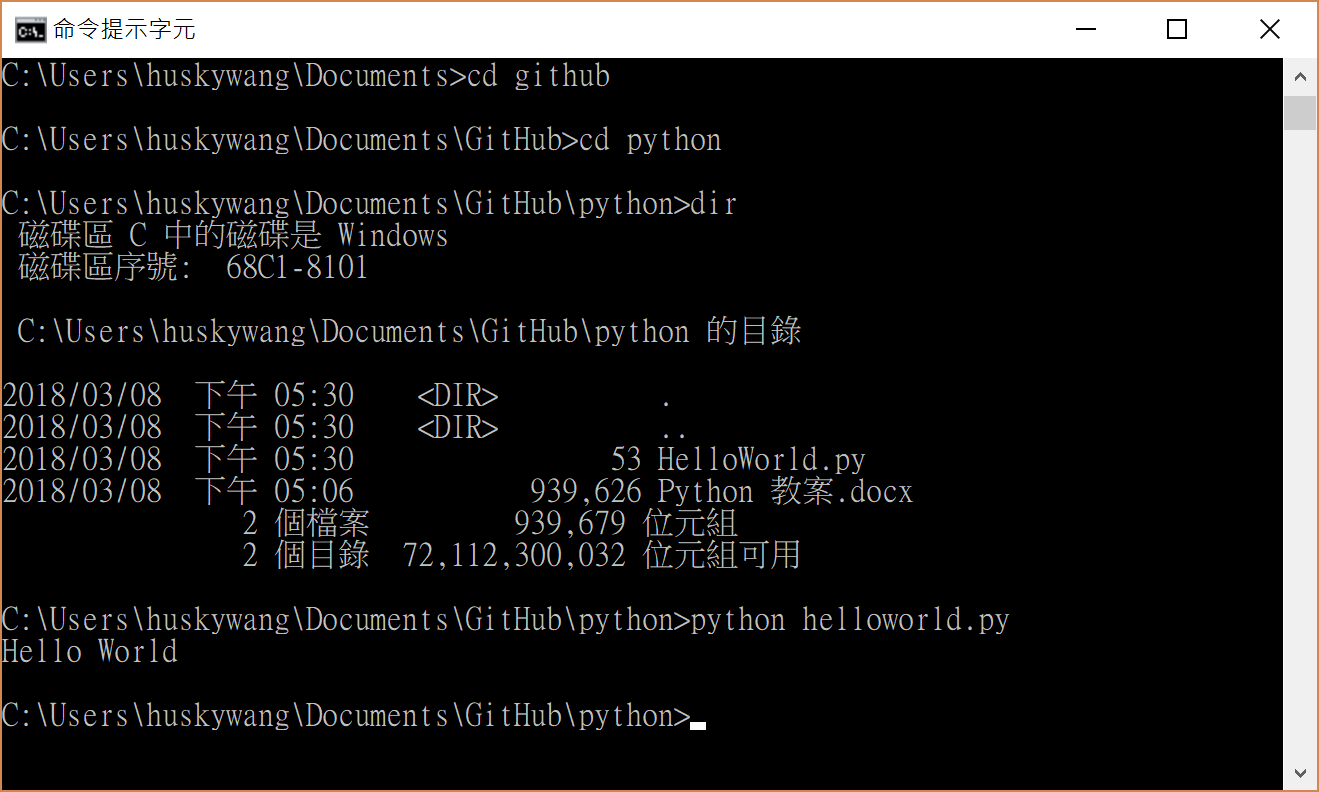
程式寫好之後一定要存檔，Python程式檔是以 .py 結尾的，請點選[FILE] [SAVE AS...]，然後選擇自己放程式的目錄，輸入檔案名稱之後即可：



如何執行這個程式呢？直接開啟剛才的 HelloWorld.py，在編輯器裡面選取[Run] [Run Module]，結果就會出現在IDLE介面之中，如下圖的最後兩行：



這種方式，是直接在 Python 的環境之下執行，如何我們要在電腦作業系統的環境下執行該如何做呢？就必須在DOS視窗中下指令 “python helloworld.py” 來執行，結果如下：



要退出DOS畫面，輸入 exit 即可。

## Python 語言基礎

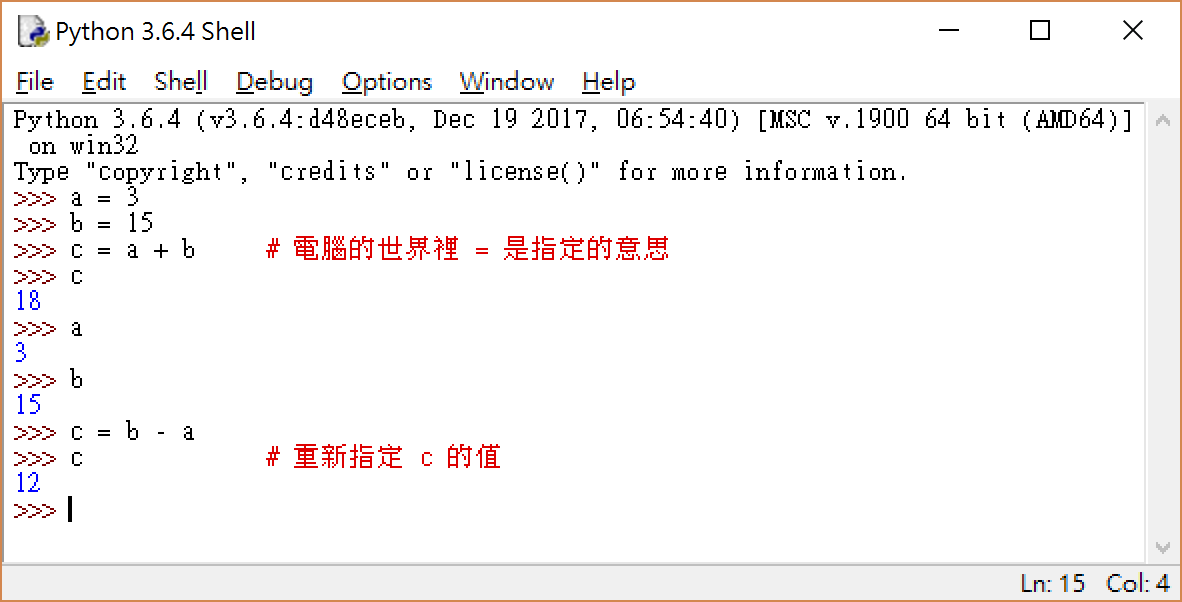
物件是 Python 語言最基本的概念之一，在 Python 中的一切都是物件，常用的內建物件如數字、字串、列表、字典、元組、檔案、檔案、集合等等，連函數都是物件的一種。除此之外，還有大量的標準庫物件和擴展庫物件。Python 預設會安裝標準庫，但需要匯入之後才能使用其中的物件；擴展庫物件則需要先安裝擴展庫，然後再匯入來使用其中的物件。

### 變數、運算子與運算式

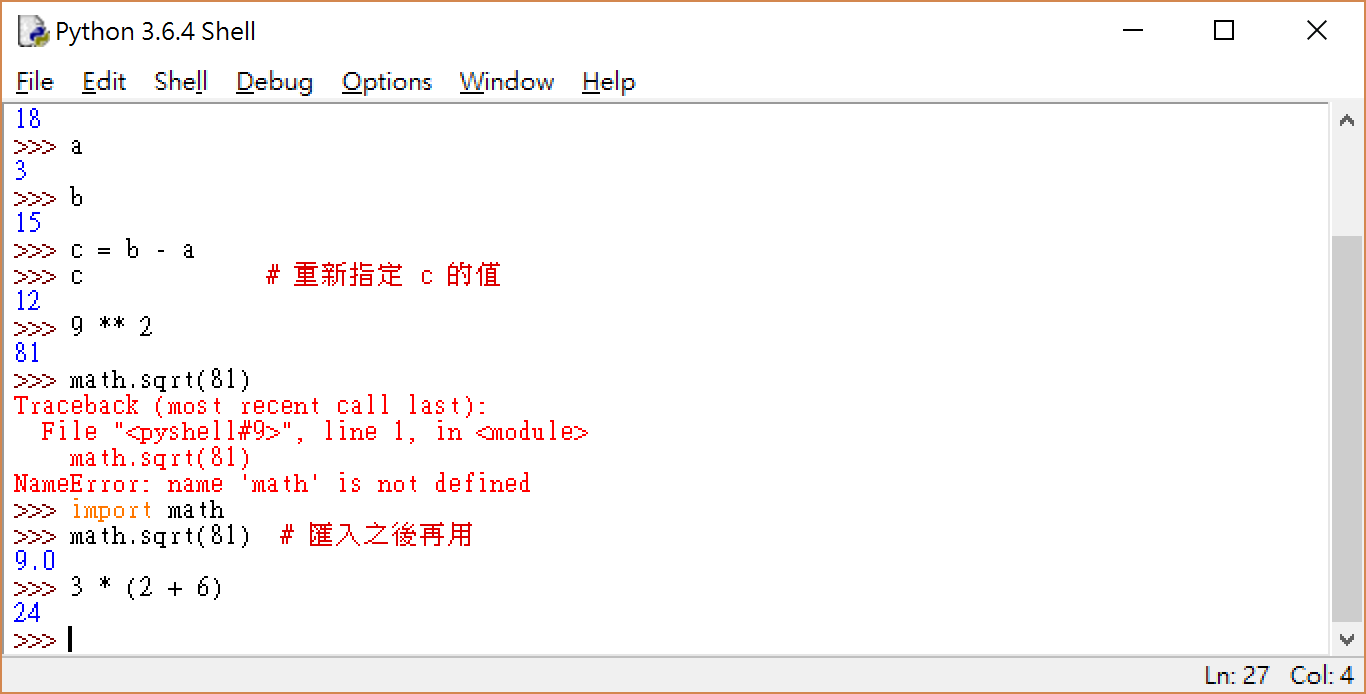
電腦語言是由數學算式演化過來的，所以一切都可以從數學來看待，如果有一個數學式子是這樣寫的：

A + B = C

這整個式子我們就稱為運算式，其中A、B、C是變數，用來存放Python的物件(例如數字)， “+” 與 “=” 就是運算;子，了解這個之後，我們就可以寫程式了：



這裡面出現的加減乘除運算，是最基本的，Python 已經將最基本的物件都匯入了，所以我們無須再做匯入的動作，但是如果是要做平方根的運算，它則是放在 math 這個標準庫裡面，我們必須使用 import 將它匯入才能使用：



Sqrt() 是 math 庫的函數之一，如果我們常常會使用到 sqrt() 函數，在匯入 math 之後，每次都要寫成 math.sqrt ，其實有點麻煩。

>>> import math

>>> math.sqrt(81) # 匯入之後再用

9.0

>>> 3 \* (2 + 6)

24

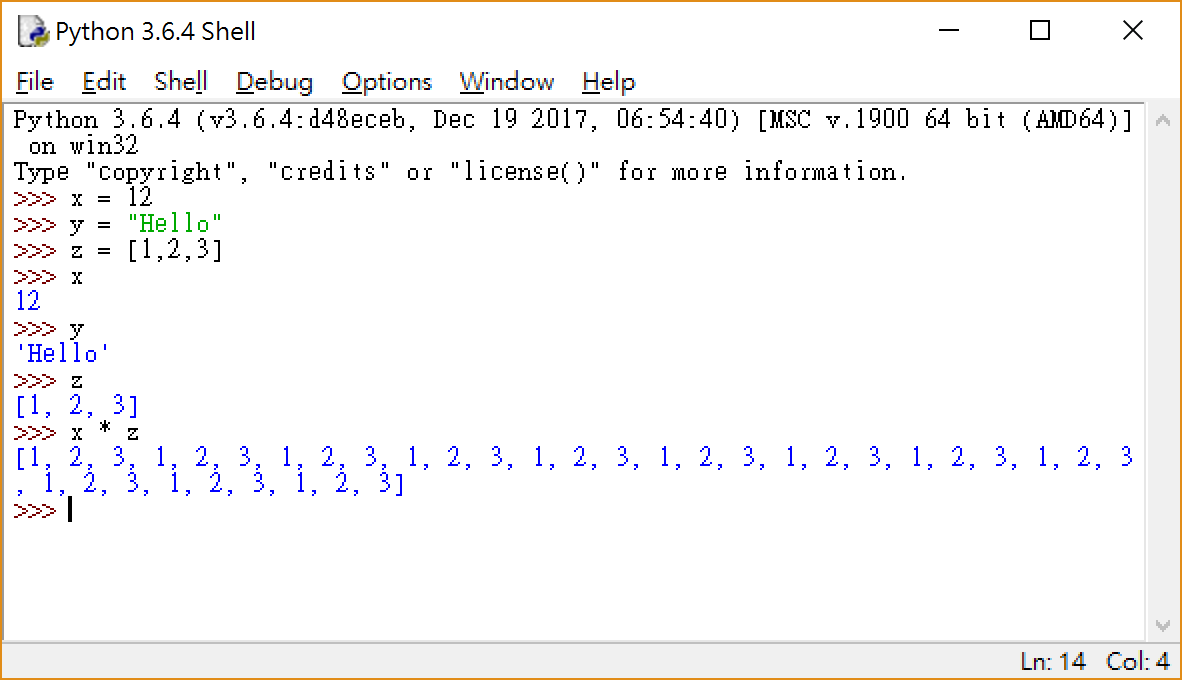
>>> from math import sqrt

>>> sqrt(81)

9.0

如果我們用 from math import sqrt 指定把 math 中的 sqrt() 匯入，那麼在使用的時候就比較省事了。

前面所提的所有變數的值都是數字以及字串(例如 “Hello World”)，我們再介紹Python 第三種很有用的物件叫做列表，而列表是用 [ … ] 來包裝。



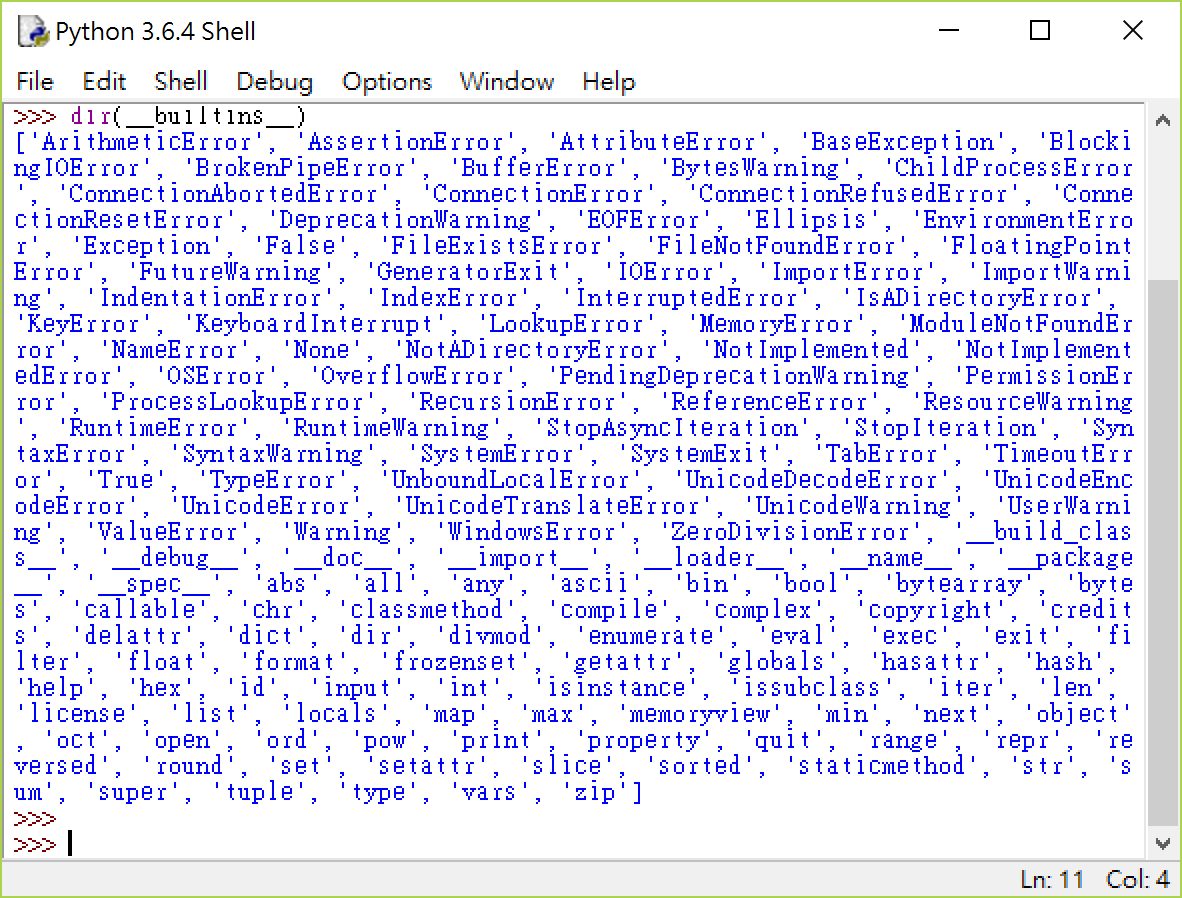
PS: 數字跟列表可以相乘，結果會跑出 12 個 [1,2,3]。但是不同型態不能相加。

### 常用的內建函數

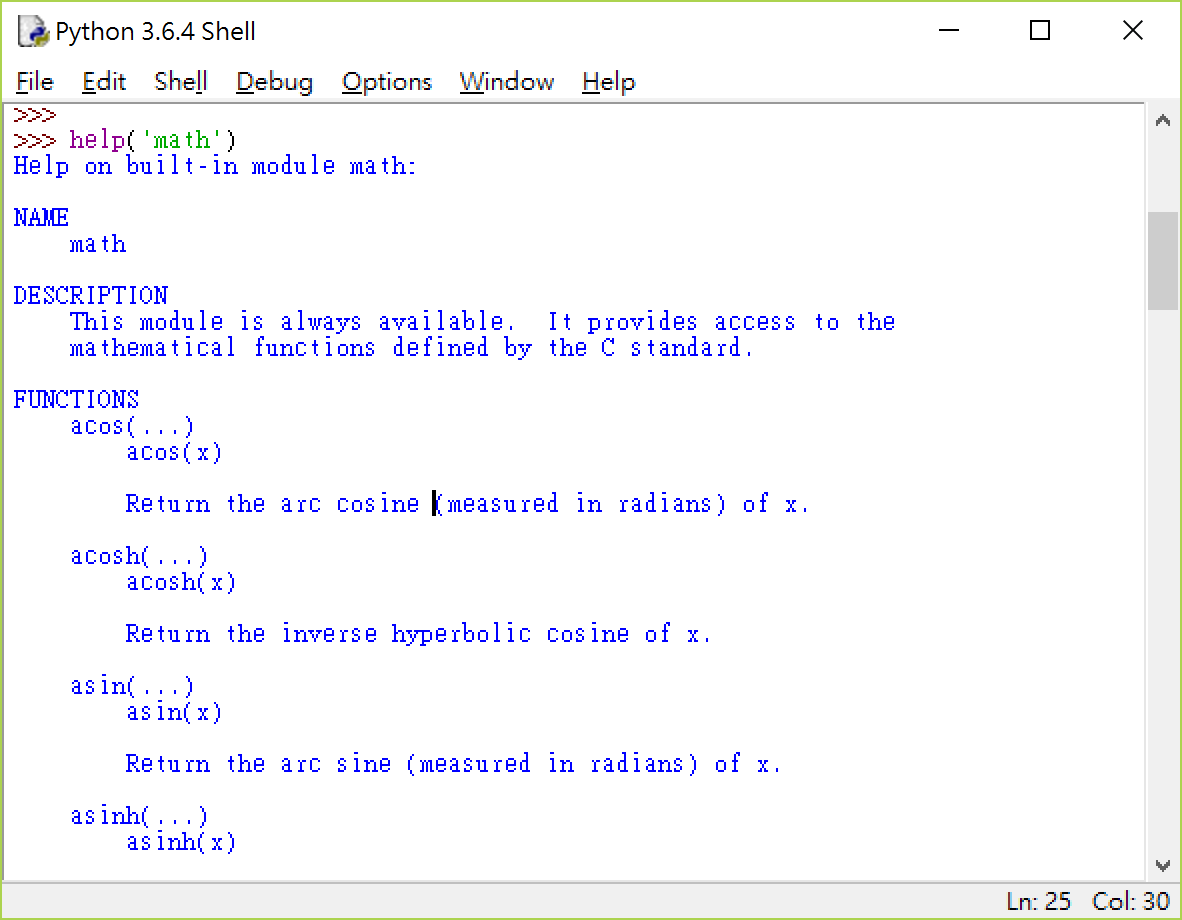
數學函數：y = f(x)

電腦語言函數：y = function\_name(x) 🡺 有傳回值，proc(x) 🡺 無傳回值

可以透過 dir() 函數，列出所有 Python 的內建函數：



而 help() 函數，可以查看某個模組或是函數的用法，不需要匯入函數就可以查看該模組的說明文件，例如：help(‘math’)，參考下圖：



我們就來使用 dir() 與 help() 函數來舉個例子，如果使用者想知道 math 模組裡面有哪些函數可以用，然後進一步想知道其中的 sin 函數的用法：

>>> import math

>>> dir(math)

['\_\_doc\_\_', '\_\_loader\_\_', '\_\_name\_\_', '\_\_package\_\_', '\_\_spec\_\_', 'acos', 'acosh', 'asin', 'asinh', 'atan', 'atan2', 'atanh', 'ceil', 'copysign', 'cos', 'cosh', 'degrees', 'e', 'erf', 'erfc', 'exp', 'expm1', 'fabs', 'factorial', 'floor', 'fmod', 'frexp', 'fsum', 'gamma', 'gcd', 'hypot', 'inf', 'isclose', 'isfinite', 'isinf', 'isnan', 'ldexp', 'lgamma', 'log', 'log10', 'log1p', 'log2', 'modf', 'nan', 'pi', 'pow', 'radians', 'sin', 'sinh', 'sqrt', 'tan', 'tanh', 'tau', 'trunc']

>>> help(math.sin)

Help on built-in function sin in module math:

sin(...)

sin(x)

Return the sine of x (measured in radians).

>>>

有三個常用的內建函數，很適合用來寫遊戲，就是 max()、min()、與 sum()，如果有 a b c 三顆骰子，分別擲出 2 4 6 三個點數，Python 要如何找出最大值與最小值呢？又要如何算出三顆骰子的總和？

**隨堂操作練習**

在互動環境中，查詢 math 標準庫中有哪些與三角函數相關的函數，並比較 cos() 與 acos() 的差異

答：

用Python 表示會是這樣：

>>> a = 2

>>> b = 4

>>> c = 6

>>> dice = [a,b,c]

>>> dice

[2, 4, 6]

>>> min(dice)

2

>>> max(dice)

6

>>> sum(dice)

12

>>>

上面的例子有一個缺點，就是骰子a永遠是2，骰子b永遠是4…，事實上骰子應該是隨機的，所以再介紹一個更好用的函數，就是產生隨機數字的函數：randint()，這個函數放在 random 標準庫裡面。如果要取 1 到 6 之間的亂數，可以用 randint(1,6) 來達成，重寫上面的例子：

>>> from random import randint

>>> a = randint(1,6)

>>> b = randint(1,6)

>>> c = randint(1,6)

>>> dice = [a,b,c]

>>> dice

[4, 2, 3]

>>> print(min(dice),max(dice),sum(dice))

2 4 9

>>>

如果同樣的例子，不是擲三顆，而是擲10顆骰子呢？有一個簡單的方法，就是直接在列表裡面放十個骰子的值，就可以省去很多變數：

>>> dice = [randint(1,6), randint(1,6), randint(1,6), randint(1,6), randint(1,6), randint(1,6), randint(1,6), randint(1,6), randint(1,6), randint(1,6)]

>>> dice

[2, 4, 6, 1, 5, 3, 2, 1, 6, 3]

>>> print(min(dice),max(dice),sum(dice))

1 6 33

>>>

這種寫法很難看，dice = [ … ] 太長，當然，沒有人會這樣寫，這違背了 Python 簡潔的特性。我們可以在列表裡面使用 for 迴圈來達成同樣的目的：

>>> dice = [randint(1,6) for i in range(10)]

>>> dice

[6, 4, 2, 2, 4, 2, 5, 4, 3, 1]

>>> print(min(dice),max(dice),sum(dice))

1 6 33

>>>

這種寫法可以先記起來，後面會解釋這種語法。有了這樣的語法，就算要擲100顆骰子也沒有問題。如下：

>>> dice = [randint(1,6) for i in range(100)]

>>> dice

[5, 5, 1, 1, 4, 2, 5, 1, 5, 4, 1, 1, 2, 1, 1, 3, 4, 2, 3, 3, 5, 6, 5, 5, 4, 3, 6, 4, 3, 2, 4, 4, 6, 2, 5, 4, 3, 1, 5, 1, 2, 5, 5, 5, 2, 5, 3, 1, 6, 2, 2, 5, 3, 2, 6, 4, 2, 6, 6, 2, 2, 4, 5, 3, 3, 6, 2, 4, 1, 6, 1, 2, 1, 1, 4, 4, 4, 1, 1, 2, 2, 4, 2, 1, 1, 1, 2, 5, 2, 6, 5, 2, 3, 1, 4, 5, 6, 3, 5, 2]

>>> print(min(dice),max(dice),sum(dice))

1 6 325

>>>

隨堂測驗

題目：一副撲克牌有52張牌，有四種花色，每種花色有13張牌，從每種花色中隨機抽出一張牌，找出這四張牌的最大數、最小數、與總和。

答：>>> from \_\_\_\_\_\_\_ import \_\_\_\_\_\_\_\_\_

>>>

>>>

>>>

### 常用運算子與運算式

Python 是純物件導向語言，熟悉物件導向的人應該知道，物件由資料和行為兩個部分組成。對於初學者而言，可以先跳過這個部分，只要知道運算子也屬於表現物件行為的一種形式。

一般我們常用的運算子包含算術運算子(大家都學過數學)、關係運算子(大家都很愛比較)、邏輯運算子(大家都學過中文)以及位元運算子(比較難，因為大家不見得都學過電腦)。當然，Python 還提供了其他類別的運算子，先不提，我們就從最基礎的前三個開始介紹。

#### 算數運算子

直接看例子：

>>> x = 12

>>> y = 4

>>> x + y

16

>>> x - y

8

>>> x \* y

48

>>> x / y

3.0

>>> z = 5

>>> x / z

2.4

>>> x // z #直接求整商

2

>>> x % z #直接求餘數

2

>>>2 \*\* 3 # n 次方

8

想想我們學過的三種物件的資料類別，數字、字串、以及列表，數字做上面的那些運算絕對沒有問題，但是文字和列表呢？有些就不行了，但是至少相加應該沒有問題：

>>> a = 'abcd'

>>> b = '1234'

>>> a + b

'abcd1234'

>>> c = ['x','y','z']

>>> d = [5,6,7]

>>> c + d

['x', 'y', 'z', 5, 6, 7]

>>>

除了數字、字串、列表之外，現在我再多介紹一種資料類別，這也是我們在國中數學裡面學過的，就是集合。同樣的，集合的表示方法與數學相同，用 { … } 表示，而集合的運算有聯集(|)、交集(&)、差集(-)、與對稱差集(^)：

你和朋友一起玩了三場槍戰遊戲，第一場出賽的有 a b c d e 五個人，第二場出賽的有 a c e f 四個人，第三場出賽的有 a b g 三個人，我們可以用Python 的集合來表示：

>>> game1 = {'a','b','c','d','e'}

>>> game2 = {'a','c','e','f'}

>>> game3 = {'a','b','g'}

請問這三場遊戲總共有哪些人出賽？有哪些人一直都有參賽？

>>> game1 | game2 | game3 #聯集,三場遊戲總共有哪些人出賽

{'b', 'c', 'd', 'g', 'f', 'a', 'e'}

>>> game1 & game2 & game3 #交集,有哪些人一直都有參賽

{'a'}

第一場有出賽，但是第三場沒有出賽的有那些人？第一場與第三場遊戲，只參加一場的有那些人？

>>> game1 - game3 #差集,第一場有出賽，但是第三場沒有出賽的有那些人

{'c', 'd', 'e'}

>>> game1 ^ game3 #對稱差集,第一場與第三場遊戲，只參加一場的有那些人

{'c', 'd', 'g', 'e'}

不用害怕數學跟電腦語言，其實只不過是把我們平常講的中文用另一種符號表示而已。

#### 關係運算子

妳比我大三歲，我薪水比你高，大家都會比較，用 Python 來表示如下：

>>> yourage = 19

>>> myage = 16

>>> yourage > myage

True

>>> yourage <= myage

False

>>> yourage == myage

False

>>> yourage != myage

True

>>>

電腦的回答只有兩種，真(True)或假(False)。

#### 邏輯運算子

And、or、not 這三種邏輯運算子通常會和關係運算子合併使用，你今年19歲，我今年16歲，他今年20歲。妳比我大而且我比他大嗎？這句中文裡面的 “比” 就是關係運算子，出現的 “而且” 就是邏輯運算子中的 “and”。妳比我大或者我比他大嗎？其中 “或者” 就是 “or”。我不(沒有)比他大嗎？其中的 “不(沒有)” 就是 “not”。

>>> yourage = 19

>>> myage = 16

>>> hisage = 20

>>> (yourage > myage) and (myage > hisage)

False

>>> (yourage > myage) or (myage > hisage)

True

>>> not (myage > hisage)

True

>>>

### 人機對話基本介面

到目前為止，在 Python 的互動環境中，都是我們對 Python 說話而 Python 回答，那是否可以反過來？ Python 問我們問題我們回答。當然可以，內建函數 input() 就可以做到。我們就讓 Python 問我們年紀好了。

>>> input('你今年幾歲:')

你今年幾歲:16

'16'

>>> x = 16

>>> x

16

>>>

Python 的 input() 很單純，它問的問題就當成一個字串寫在()裡面，我們回答的答案，也會被當成字串，對 input() 而言，無論我們輸入的是數字還是文字，都會被當成字串，上面的例子很清楚，我們回答 16 原本是個數字，但是後面的輸出則是 ‘16’，跟x 比較一下就知道不同。

因此，我們需要另一個函數幫我們轉換，就是eval()，順便可以用 type() 函數查看一下資料類別。請看下面的例子：

>>> age = input('How old are you? ')

How old are you? 20

>>> type(age)

<class 'str'>

>>> eval(age) # 把age 轉成數字之後輸出，並不影響age本身

20

>>> age

'20'

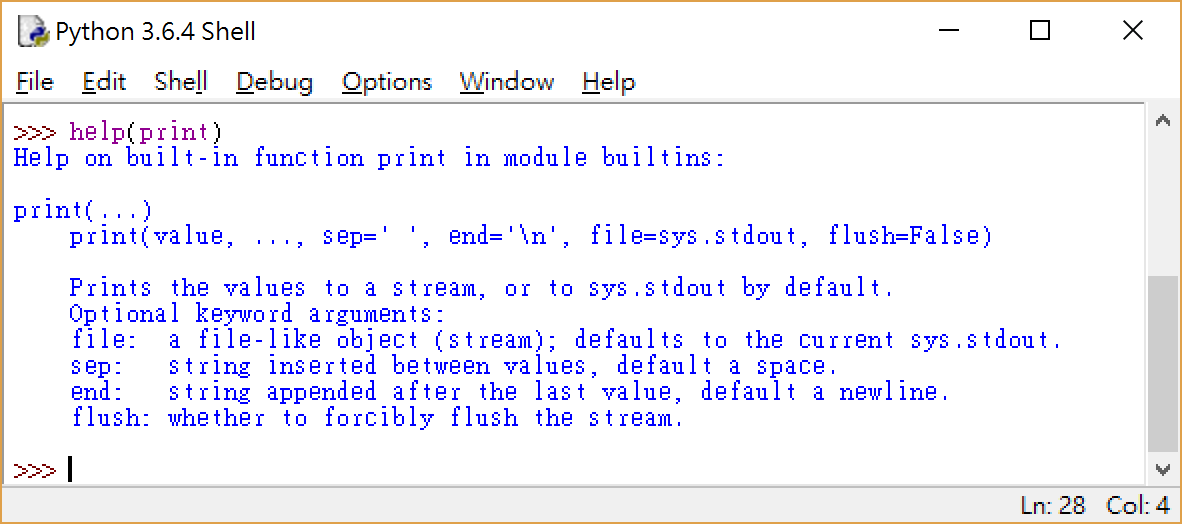
>>> age = eval(age) # 轉成數字之後再指派給 age

>>> age

20

>>>

跟 input() 相對應的函數，就是 print()，利用 help() 來看語法：



順便提一下，只要是內建的函數在我們查詢的時候都會看到 in module builtins 這樣的說明。Print() 是把 values 輸出在螢幕上，當然也可以輸出到檔案裏面。其中 sep, end 是指每個 value 之間的隔離符號以及最後的換行符號，請看範例：

>>> age = input("What's your age? ")

What's your age? 21

>>> print("Your age is: ",age)

Your age is: 21

>>> print(1,2,3,7,11)

1 2 3 7 11

>>> print(1,2,3,7,11,sep='\t')

1 2 3 7 11

>>>

補充說明，字串可以用 ‘ … ‘ 和 “ … “ 兩種引號包裹，有時候字串裡面包含 ‘ 的時候，就可以用雙引號包裹。

也可以利用for 迴圈來產生列印結果，for迴圈的寫法，後面出現冒號的時候，後面的縮排部分都會被認為是 for 迴圈裡面要執行的東西，這點和其他語言都不一樣，像是 C 、Java 都是使用{ … } 來表示：

C:

for () {

……

……

}

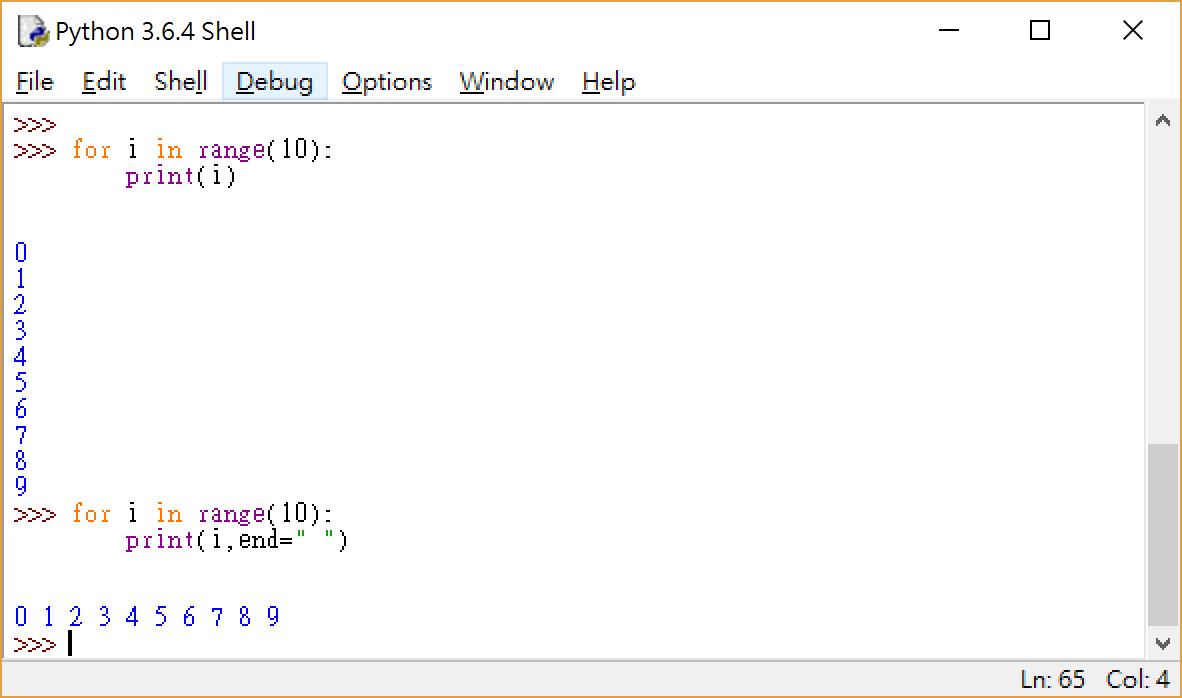
Python:

for … :

……

……

試著自己在互動式介面輸入一下便知：



輸入時，在冒號之後按下 ENTER 時，游標會跳到下一行並且自動縮排，print 輸入完畢按下第二次 ENTER 時，縮排結束，並執行顯示結果。想一下我為何要用 end=’ ‘？

如果想要輸出到檔案，就要使用 open() 函數，並在 print() 裡面加入 file=，指定要打開的檔案：

fp=open('C:\\Users\\Documents\\GitHub\\python\\test.txt','a+')

>>> for i in range(10):

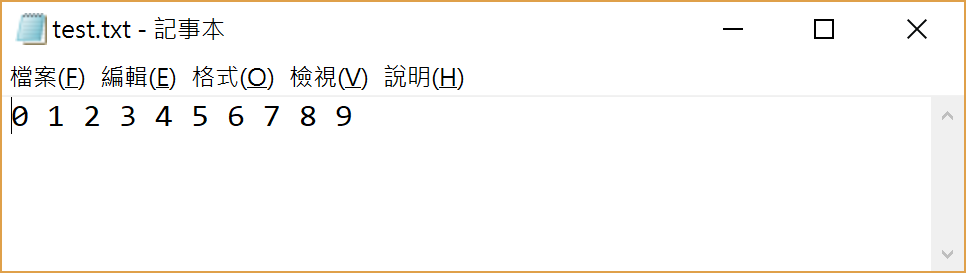
print(i, end=' ', file=fp)

>>> fp.close()

>>>

執行完畢之後，檔案就會建立在我們指定的目錄中了，檔案 test.txt 的內容如下，請自行比較一下是否和輸出在螢幕上的結果一樣：

PS: 檔案在初期比較少用，先記下語法就好。重要原則一定要知道，有open()，就有 close()。



課後作業

寫一個程式，需要存成 practice.py 檔案

參考 def main(): 的寫作方式。

使用 input() print() 等函數，執行之後，會詢問年齡

回答你的年齡(例如 16)之後，Python 會印出：

你明年的年齡是: 17

## 模組的安裝與使用

Python 之所以在各行各業都被青睞，是因為它有適合於各行各業的擴展庫。列出前15名好用的擴展庫參考一下。

### 核心庫

#### 1. NumPy (提交數: 15980, 貢獻者數: 522)

當開始處理Python中的科學任務，Python的SciPy Stack肯定可以提供幫助，它是專門為Python中科學計算而設計的軟體集合（不要混淆SciPy庫，它是SciPy Stack的一部分，和SciPy Stack的社區）這樣我們開始來看一下吧。然而，SciPy Stack相當龐大，其中有十幾個庫，我們把焦點放在核心包上（特別是最重要的）。

關於建立科學計算棧，最基本的包是Numpy（全稱為Numerical Python）。它為Python中的n維陣列和矩陣的操作提供了大量有用的功能。該庫提供了NumPy陣列類型的數學運算向量化，可以改善性能，從而加快執行速度。

#### 2. SciPy (提交數: 17213, 貢獻者數: 489)

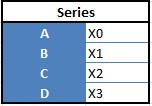
SciPy是一個工程和科學軟體庫。包含線性代數，優化，集成和統計的模組。SciPy庫的主要功能是建立在NumPy上，從而它的陣列大量的使用了NumPy的。它通過其特定子模組提供有效的數值常式，並作為數位積分、優化和其他常式。

#### 3. Pandas (提交數: 15089, 貢獻者數：762)

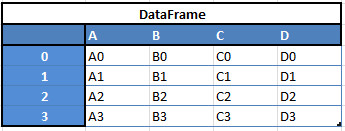
Pandas是一個Python包，旨在通過“標記”和“關係”資料進行工作，簡單直觀。Pandas是資料整理的完美工具。它設計用於快速簡單的資料操作，聚合和視覺化。

庫中有兩個主要的資料結構：

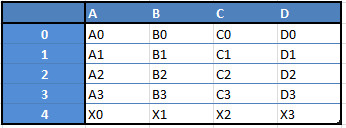
“系列”（Series），一維



“數據幀”（Data Frames），二維



例如，當您要從這兩種類型的結構中接收到一個新的Dataframe時，通過傳遞一個Series，您將收到一個單獨的行到DataFrame的DF：



這裡稍微列出了你可以用Pandas做的事情：

* 輕鬆刪除並添加資料幀（DataFrame）中的列
* 將資料結構轉換為資料幀（DataFrame）物件
* 處理丟失的資料，表示為NaN
* 功能強大的分組
* 視覺化

#### 4.Matplotlib (提交數: 21754, 貢獻者數: 588)

又一個SciPy Stack核心套裝軟體以及 Python庫，Matplotlib為輕鬆生成簡單而強大的視覺化而量身定制。它是一個頂尖的軟體（在NumPy，SciPy和Pandas的幫助下），它使Python成為像MatLab或Mathematica這樣的科學工具的競爭對手。

然而，這個庫是低層級的，這意味著你需要編寫更多的代碼才能達到高級的視覺化效果，而且通常會比使用更多的高級工具付出更多的努力，但總體上這些努力是值得的。

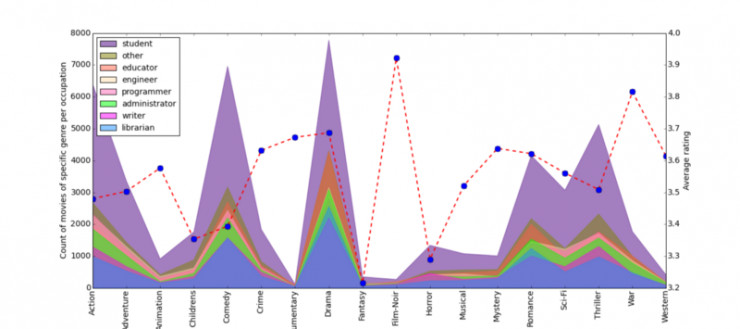
只要付出一點你就可以做任何視覺化：

* 線圖
* 散點圖
* 橫條圖和長條圖
* 餅狀圖;
* 莖圖
* 輪廓圖
* 場圖
* 頻譜圖

還有使用Matplotlib創建標籤，網格，圖例和許多其他格式化實體的功能。基本上，一切都是可定制的。

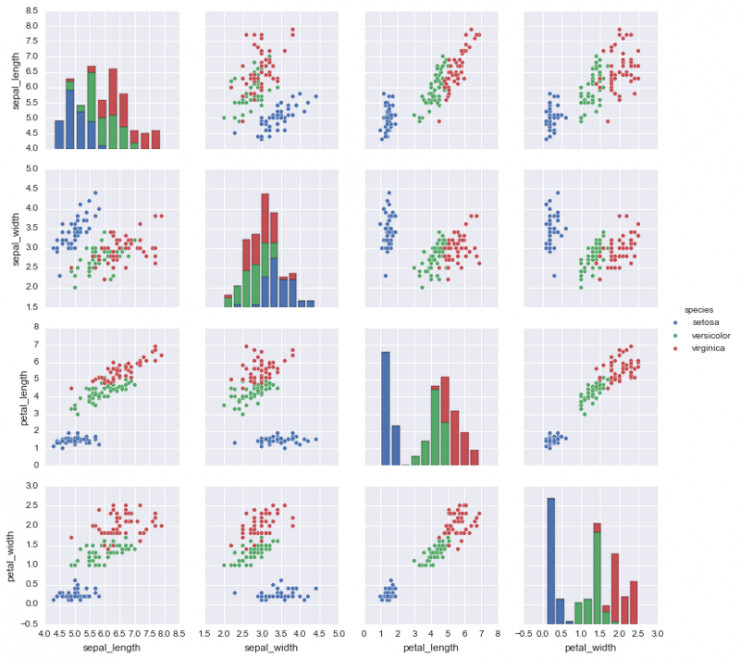
該庫由不同的平臺支援，並使用不同的GUI套件來描述所得到的視覺化。不同的IDE（如IPython）都支援Matplotlib的功能。

還有一些額外的庫可以使視覺化變得更加容易。



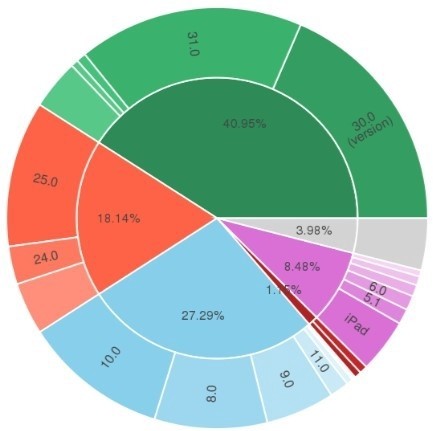
#### 5. Seaborn (提交數: 1699, 貢獻者數: 71)

Seaborn主要關注統計模型的視覺化；這種視覺化包括熱圖，這些熱圖（heat map）總結資料但仍描繪整體分佈。Seaborn基於Matplotlib，並高度依賴於此。



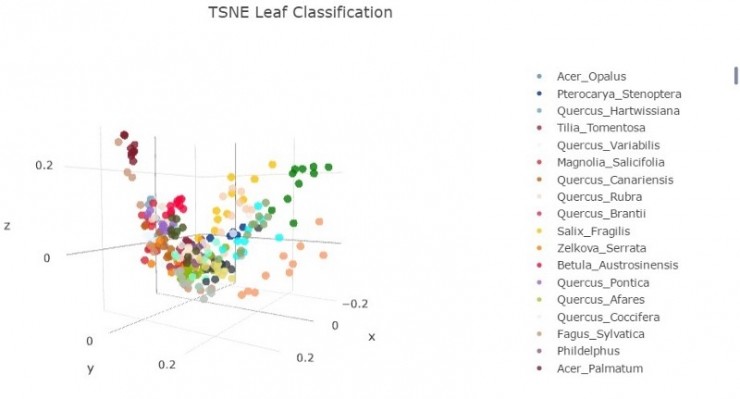
#### 6. Bokeh (提交數: 15724, 貢獻者數: 223)

另一個很不錯的視覺化庫是Bokeh，它針對互動式視覺化。與以前的庫相比，它獨立於Matplotlib。正如我們提到的，Bokeh的主要焦點是交互性，它通過現代流覽器以資料驅動文檔（d3.js）的風格呈現。



#### 7. Plotly (提交數: 2486, 貢獻者數: 33)

它是一個基於Web用於構建視覺化的工具箱，提供API給一些程式設計語言（Python在內）。在plot.ly網站上有一些強大的、上手即用的圖形。為了使用Plotly，你將需要設置API金鑰。圖形將在伺服器端處理，並發佈到互聯網，但有一種方法可以避免。



### 機器學習

#### 8. SciKit-Learn (提交數：21793, 貢獻者數：842)

Scikits是Scikits Stack額外的套裝軟體，專為像影像處理和機器學習輔助等特定功能而設計。對於機器學習輔助，scikit-learn是所有套裝軟體裡最突出的一個。它建立在SciPy之上，並大量利用它的數學運算。

scikit-learn給常見的機器學習演算法公開了一個簡潔、一致的介面，可簡單地將機器學習帶入生產系統中。該庫中集成了有品質的代碼和良好的文檔、簡單易用並且十分高效，是使用Python進行機器學習的實際行業標準。

#### 9.Theano. (提交數：25870, 貢獻者數：300)

Theano是一個Python套裝軟體，它定義了與NumPy類似的多維陣列，以及數學運算和運算式。此庫是被編譯的，可實現在所有架構上的高效運行。最初由蒙特利爾大學機器學習組開發，它主要用於滿足機器學習的需求。

值得注意的是，Theano緊密結合了NumPy在低層次上的運算 。另外，該庫還優化了GPU和CPU的使用，使資料密集型的計算平臺性能更佳。

效率和穩定性微調保證了即使在數值很小的情況下，仍有更精確的結果，例如，即使只給出x的最小值，log（1 + x）仍能計算出合理的結果。

#### 10. TensorFlow. (提交數: 16785,貢獻者數: 795)

TensorFlow來自Google的開發人員，它是資料流程圖計算的開源庫，為機器學習不斷打磨。它旨在滿足穀歌對訓練神經網路的高需求，並且是基於神經網路的機器學習系統DistBelief的繼任者。然而，TensorFlow並不限制于穀歌的科學應用範圍 – 它可以通用于多種多樣的現實應用中。

TensorFlow的關鍵特徵是它的多層節點系統，可以在大型資料集上快速訓練神經網路。這為谷歌的語音辨識和圖像物件識別提供了支援。

#### 11. Keras. (提交數: 3519,貢獻者數: 428)

最後我們來看看Keras。它是一個用Python編寫的開源的庫，用於在高層的介面上構建神經網路。它簡單易懂，具有高級可擴展性。Keras使用Theano或TensorFlow作為後端，但微軟現在正努力整合CNTK（微軟的認知工具包）作為新的後端。

設計中的簡約方法旨在通過建立緊湊型系統進行快速、簡便的實驗。

Keras真的容易上手，並在持續完善它的快速原型能力。它完全用Python編寫，可被高度模組化和擴展。儘管它以易上手、簡單和以高層次為導向，但是Keras足夠有深度並且足夠強大，去支援複雜的模型。

### 自然語言處理

#### 12. NLTK (提交數: 12449,貢獻者數: 196)

這個庫的名稱“Natural Language Toolkit”，代表自然語言工具包，顧名思義，它用於符號學和統計學自然語言處理（NLP) 的常見任務。 NLTK旨在促進NLP及相關領域（語言學，認知科學人工智慧等）的教學和研究，目前受到重點關注。

NLTK的功能允許很多操作，例如文本標記，分類和標記，實體名稱識別，建立語料庫，可以顯示語言內部和各句子間的依賴性、詞根、語義推理等。所有的構建模組都可以為不同的任務構建複雜的研究系統，例如情緒分析，自動總結。

#### 13. Gensim (提交數: 2878,貢獻者數: 179)

它是一個用於Python的開源庫，為有向量空間模型和主題模型的工作提供了使用工具。這個庫是為了高效處理大量文本而設計的，所以不僅可以進行記憶體處理，還可以通過廣泛使用NumPy資料結構和SciPy操作來獲得更高的效率。Gensim高效也易於使用。

Gensim旨在與原始和非結構化的數位文本一起使用。 它實現了諸如hierarchical Dirichlet processes（HDP），潛在語義分析（LSA）和潛在Dirichlet分配（LDA）之類的演算法，以及tf-idf，隨機預測，word2vec和document2vec，便於檢查一組文檔中有重複模式的文本 （通常稱為語料庫）。所有的演算法均是無監督的，意味著不需要任何參數，唯一的輸入只有語料庫。

### 資料採擷，統計學

#### 14. Scrapy (提交數: 6325,貢獻者數: 243)

Scrapy庫是用於從網路結構化檢索資料（如連絡人資訊或URL）,可以用來設計crawling程式（也稱為蜘蛛bots）。

它是開源的，使用用Python編寫的。最開始只是如它的名字暗示的一樣，只用來做scraping，但是它現在已經在完整的框架中發展，能夠從API採集資料並作為通用的crawlers了。

該庫在介面設計中標榜著“不要重複自己” 它推薦用戶們編寫泛化得到、可被重複使用的通用代碼，從而構建和擴展大型的crawlers。

Scrapy的架構圍繞著Spider class構建，這其中包含了crawler追從的一套指令。

#### 15. Statsmodels (提交數: 8960,貢獻者數: 119)

你可能從名字就猜出大概了，statsmodels使用戶能夠通過使用各種統計模型的估算方法進行資料採擷，並執行統計判斷和分析。

許多有用的特徵是可被描述的，並通過使用線性回歸模型，廣義線性模型，離散選擇模型，魯棒線性模型，時間序列分析模型，各種估計方法得出統計結果。

這個庫還提供了廣泛的標定功能，专门用于大数据统计中的性能优化工作。

### 安裝 Python 擴展庫

目前 pip 已經成為管理 Python 擴展庫的主流方式。Pip不僅可以即時查看本機已經安裝的擴展庫列表，還支援 Python 擴展庫的安裝、升級、和卸載等操作。

#### 常用pip指令的使用方法

* pip install Package 安裝 Package 模組
* pip list 列出目前已安裝的所有模組
* pip install --upgrade Package 升級Package 模組
* pip uninstall Package 卸載Package 模組

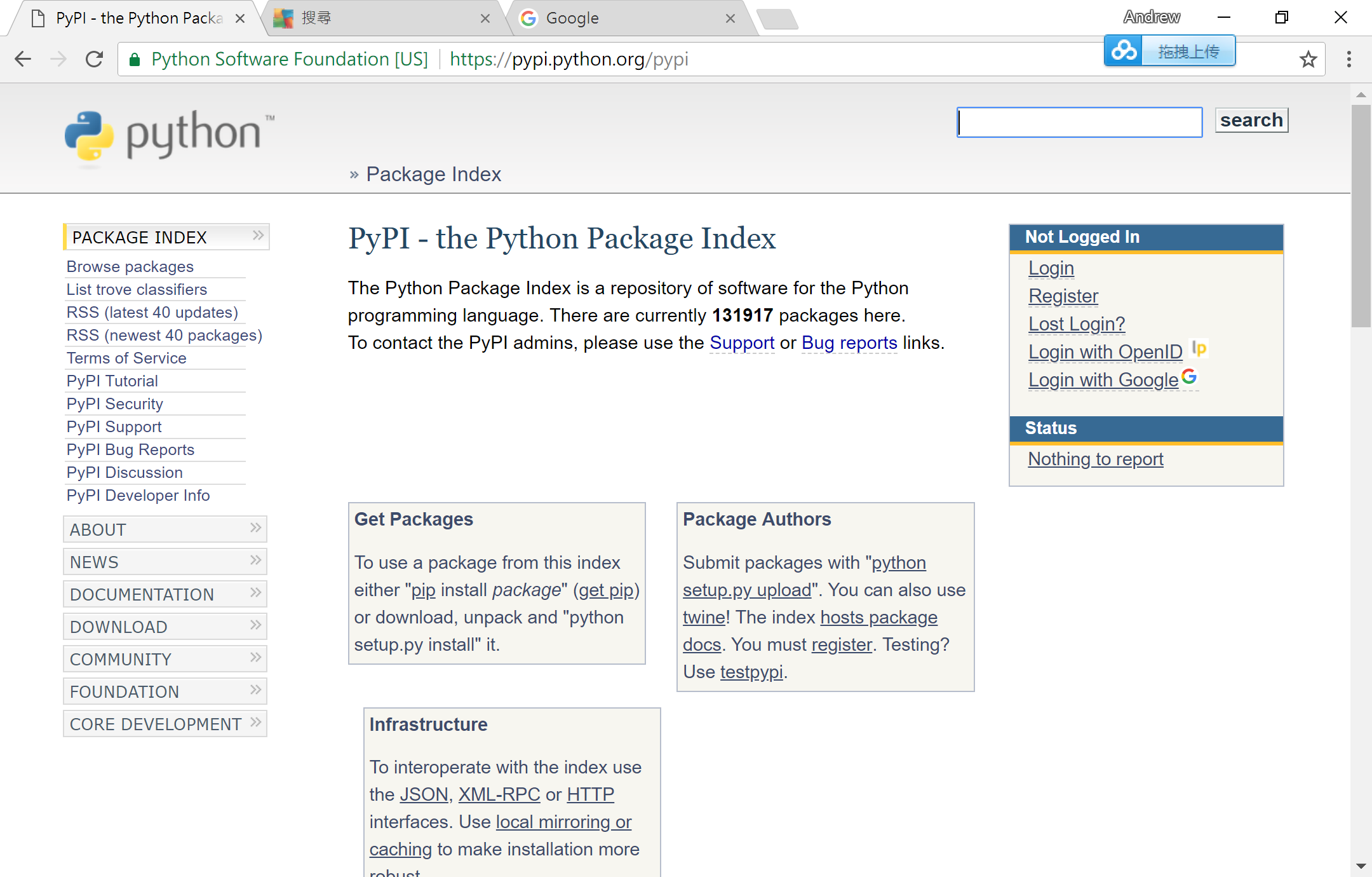
從 <https://pypi.python.org/pypi> 可以獲得一份 Python 擴展庫的綜合列表，到今天為止，已經有 131917 個涉及各領域的擴展庫，並且每天都在以幾十個的速度增加。

有些擴展庫安中的時候要求本機已安裝對應的 C++ 編譯器，或者暫時還沒有與本機 Python 版本對應的官方版本，這時可從 <http://www.lfd.uci.edu/~gohlke/pythonlibs/> 下載對應的 whl 檔，然後在命令列以 pip 命令進行安裝，例如：

Pip install pygame-1.9.2a0-cp35-none-win\_amd64.whl

小技巧：以命令安裝 Python 擴展庫時，需要在 cmd 環境中進行，並且切換至 pip 命令所在的目錄，我們可以用檔案總管尋找 pip.exe ，開啟該目錄，然後執行 cmd，把檔案總管中的目錄複製下來，再貼到 cd 指令後面。

目錄為：C:\Users\huskywang\AppData\Local\Programs\Python\Python36\Scripts



為何 Python 的目錄如此複雜，是因為 Python 支援多個虛擬環境的建立，每個虛擬環境都是包含 Python 和對應擴展庫的一個目錄，多個虛擬環境之間互不干擾。

### 模組匯入與使用

Python 預設安裝僅包含部分機本或核心模組，啟動時也只載入基本模組，當有需要時再明確的載入其他模組。這樣可以減少程式執行的壓力，而且具備極強的可擴展性。Python 匯入模組的方法主要有：

#### import 模組名稱 [as 別名]

採用這種方式，使用時需要在物件前面加上模組名稱或是別名：

>>> import math

>>> math.sin(0.5)

0.479425538604203

>>> import random as rd

>>> x = rd.random()

>>> n = rd.randint(1,100)

>>> n

70

>>> x

0.9477336091251174

>>>

#### from 模組名稱 import 物件名稱 [as 別名]

這種方法無須以模組名稱作為首碼，可以減少程式人員輸入的程式碼：

>>> from math import sin

>>> sin(3)

0.1411200080598672

>>> from math import sin as f

>>> f(3)

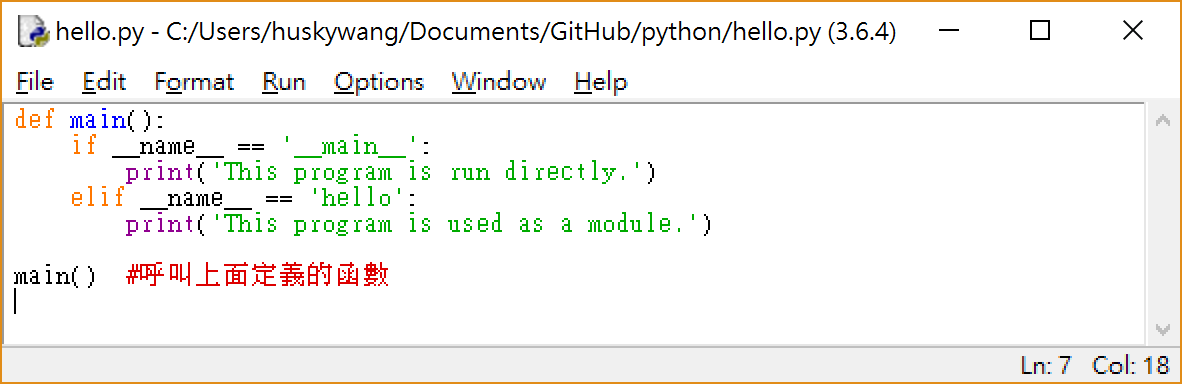
0.1411200080598672

>>>

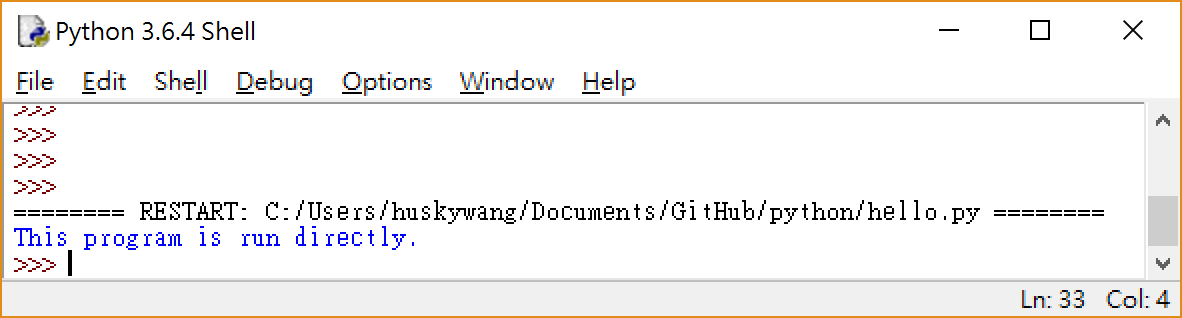
以 from import 的方式明確的匯入指定的物件，而非匯入整個模組，可以適當的提高程式載入和執行速度。

#### 撰寫自己的模組和套件

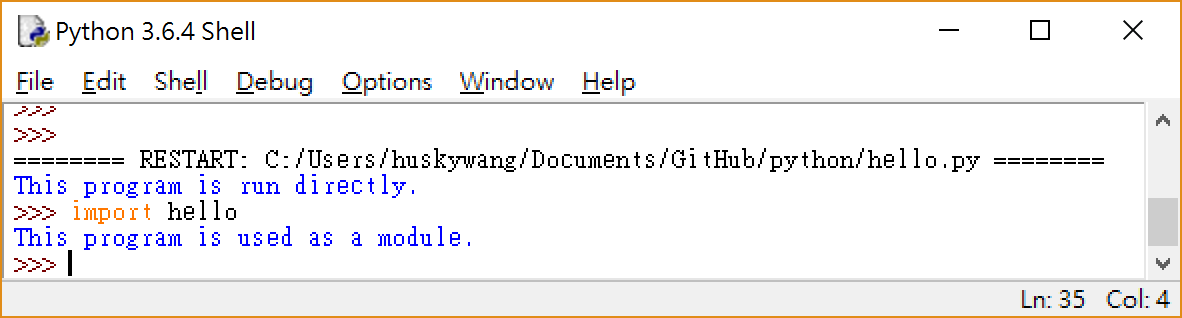
自己寫的 Python 程式除了可以直接執行之外，還能夠作為模組匯入並使用其中的物件。\_\_name\_\_ 屬性便可識別程式的使用方法。每個 Python 腳本在執行的時候都會有一個 \_\_name\_\_ 屬性，如果腳本作為模組匯入，則其 \_\_name\_\_ 屬性值將自動設為模組名稱；如果獨立執行腳本，則其 \_\_name\_\_ 屬性值將自動設為 “\_main\_\_”，例如，hello.py 的程式碼如下：



在IDLE 中直接執行該程式時，得到的結果如下：



而在 IDLE 交互模式中以 import hello 匯入該模組時，得到下面的結果：



PS: hello.py 程式建議自己動手做一次，有助於了結程式的寫法。Python 與一般結構化程式語言不同，不會使用 { … } 來定義 Block，而是用縮排的方式。

課後測驗

請寫出 Python 的程式架構，並儲存為 program.py

def \_\_\_\_\_\_ :

if \_\_\_\_\_\_\_\_\_\_\_\_\_ :

print(\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_)

elif \_\_\_\_\_\_\_\_\_\_\_\_ :

print(\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_)

\_\_\_\_\_\_\_\_

## Python 序列

Python 的序列類似 C 或 Basic 語言的一維與多維陣列，但功能要強大許多，使用上也更加靈活。

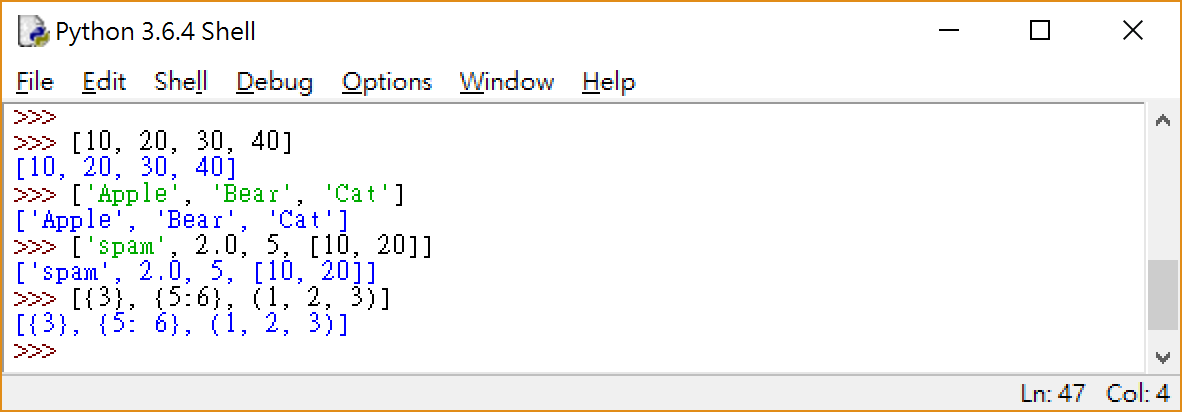
Python 常用的序列結構有列表、元組、字典、字串、集合等等，大致可以分為有序與無序兩類：其中列表、元組、字串屬於有序序列(有順序的)，字典、集合屬於無序序列。前一個章節有討論過列表、字串、集合三個，但是在這裡會更深入的討論。

對於有序序列而言，都會支援雙向索引，第一個元素的索引為 0，第二個元素的索引為 1，依此類推。反向的話，倒數第一個元素的索引為 -1，倒數底二個元素的索引為 -2，依此類推。使用負整數作為索引是 Python 序列的一大特色，熟練之後，可以大幅提升開發效率。

### 列表(List)與列表推導式

列表在上一章節有簡單介紹過，是 Python 內建重要的可變序列之一，它是包含若干元素的有序連續記憶體空間。形式上，列表的所有元素放在一對中括號裡面，相鄰的元素之間以逗點分開。

在 Python 中，同一個列表中元素的資料類型可以不一樣，例如可以分別為整數、實數、字串等基本資料類型，或者是列表、元組、字典、集合以及其他自訂類型的物件。下面幾種都是合法的列表物件：



隨堂練習

玩擲骰子遊戲，一次值四個骰子，共擲三次

請問如何用列表表示？

>>>

### 列表(List)的建立與刪除

以 = 直接將列表賦予值給變數，即可建立列表物件，例如：

>>> a\_list = [1, 2, 3, 4, 5]

>>> a\_list

[1, 2, 3, 4, 5]

>>> b\_list = [] # 空值

>>> b\_list

[]

>>>

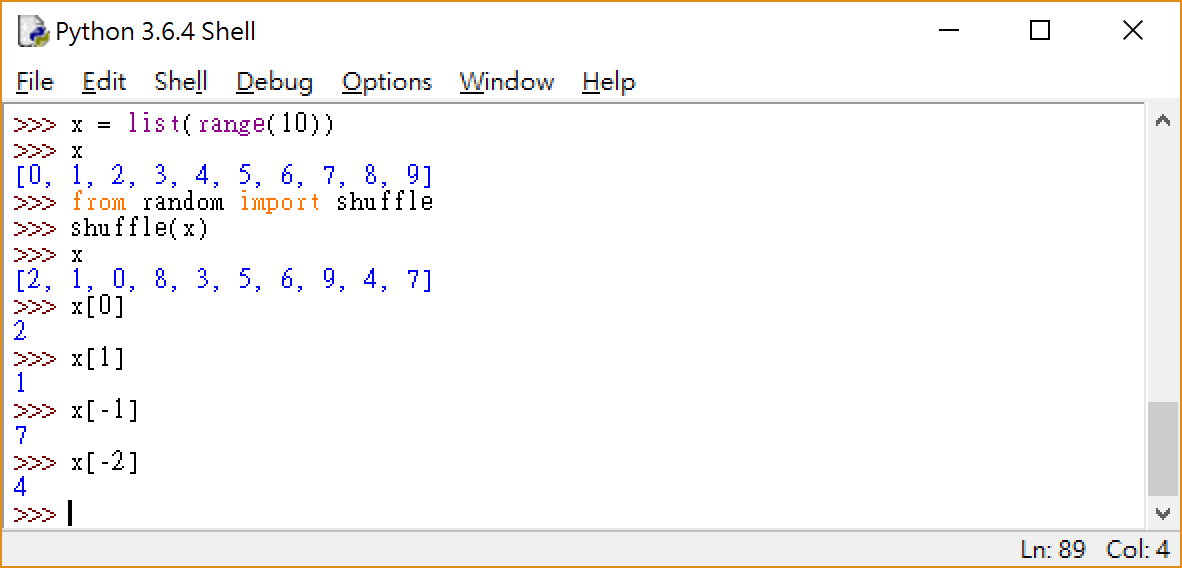
也可以使用 list() 函數將元組、range 物件、字串、字典、集合等資料類別轉換為列表，請先看下面的例子：



字典比較特殊，如果直接用 list() 轉換，只會轉換鍵，如果要同時轉換鍵:值，則需使用 items()。

在 Python 社群中，習慣將 list() 還有後面很快就會學到的 tuple()、set()、dict() 等函數稱為 “工廠函數”，因為這些函數可以生產新的資料型態。

建立列表之後，就可以使用整數作為索引存取其中的元素，其中0代表第一個，前面有說明過：



當不再使用列表時，可使用 del 刪除。

隨堂實作

填空題：

>>> x = [1, 2, 3]

>>> x

\_\_\_\_\_\_\_\_\_

>>> del x[1]

>>> x

\_\_\_\_\_\_\_\_\_

>>> del x

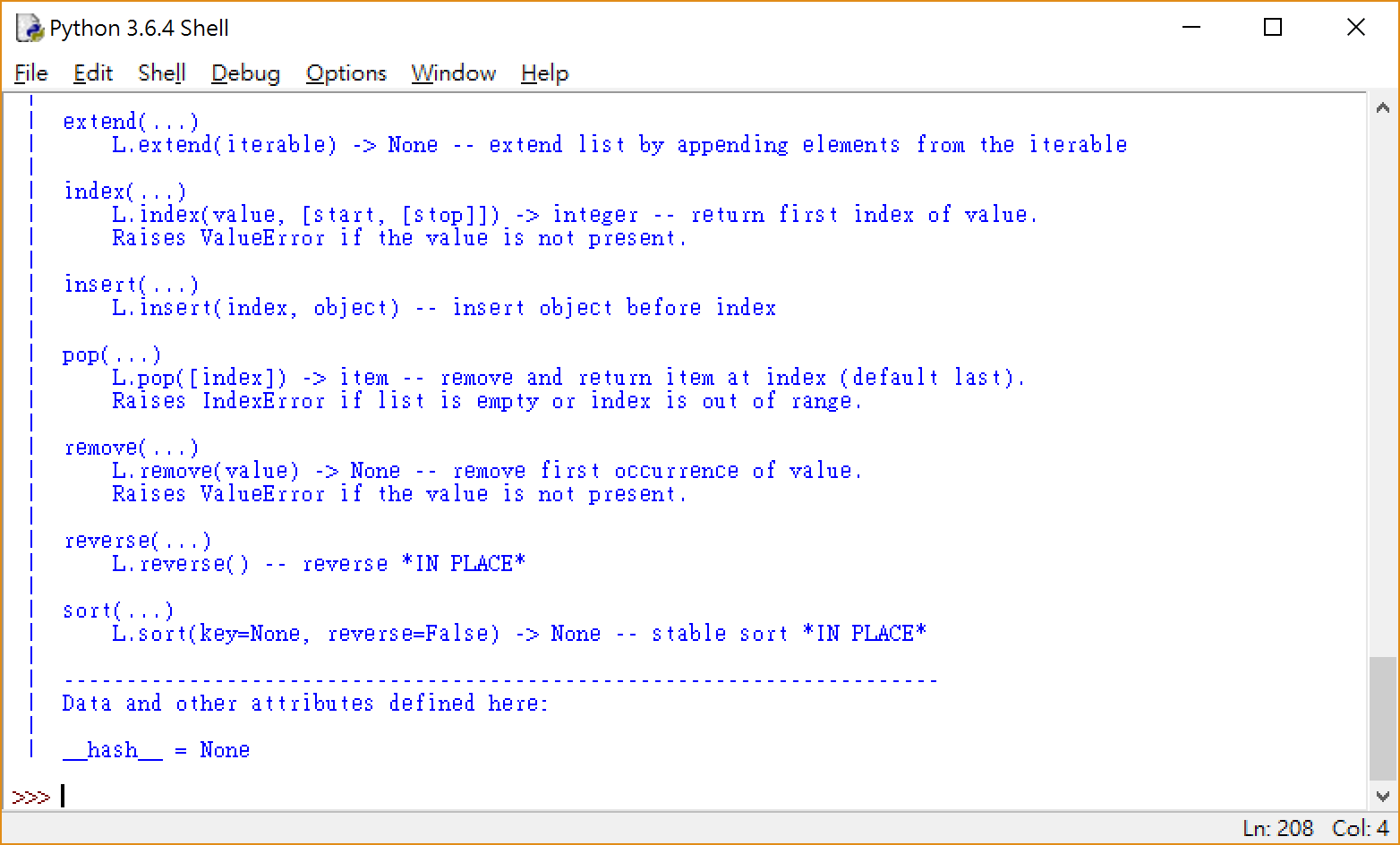
>>> x

\_\_\_\_\_\_\_\_\_

### 列表常用方法

首先說明一下函數與方法的不同，雖然他們看起來很像，但是呼叫方式卻不同，函數是以 function() 的方式呼叫，方法則是包裝在物件之內的函數，所以是以 object.method() 的方式呼叫。熟悉物件導向程式設計(OOP)的人，應該駕輕就熟。物件裡面會包含兩種東西，一是變數(variable)，一是函數(function)，物件內的變數稱之為屬性(attribute)，物件內的函數稱之為方法(method)。

還記得如何使用 help() 來求助嗎? help() 括號裡面的參數可以是一個物件，輸入 help([])，就可以得到所有關於 list 的方法，如下圖:



複製貼上再整理一下，就可以當成我們的教材，刪除不常用的，留下常用的，如下表，這裡出現的都是方法，所以可以看到(以reverse() 為例)其呼叫方式為 L.reverse()：

>>> help([])

Help on list object:

class list(object)

| list() -> new empty list

| list(iterable) -> new list initialized from iterable's items

| Methods defined here:

| append(...)

| L.append(object) -> None -- append object to end

| clear(...)

| L.clear() -> None -- remove all items from L

| copy(...)

| L.copy() -> list -- a shallow copy of L

| count(...)

| L.count(value)->integer--return number of occurrences of value

| extend(...)

| L.extend(iterable) -> None -- extend list by appending elements

| from the iterable

| index(...)

| L.index(value, [start, [stop]]) -> integer -- return first

| index of value.

| Raises ValueError if the value is not present.

| insert(...)

| L.insert(index, object) -- insert object before index

| pop(...)

| L.pop([index]) -> item -- remove and return item at index

| (default last).

| Raises IndexError if list is empty or index is out of range.

| remove(...)

| L.remove(value) -> None -- remove first occurrence of value.

| Raises ValueError if the value is not present.

| reverse(...)

| L.reverse() -- reverse \*IN PLACE\*

| sort(...)

| L.sort(key=None, reverse=False) -> None--stable sort \*IN PLACE\*

| -------------------------------------------------------------------

一定要練習看英文的說明，這樣可以快速地得到協助，又可以加強英文能力，一舉兩得。

#### append()、insert()、extend()

這三種方法都能增加元素列表物件，請自行參考上面的 help 說明。這三種方法都屬於原地操作，也就是說不會改變列表的位址。下面直接用例子說明用法：

>>> x = [1, 2, 3]

>>> id(x)

1856880275848

>>> x.append(4)

>>> x

[1, 2, 3, 4]

>>> id(x)

1856880275848

>>> x.insert(0, 0) # 在第一個位置插入 0

>>> x

[0, 1, 2, 3, 4]

>>> id(x)

1856880275848

>>> x.extend([5, 6, 7, 8])

>>> x

[0, 1, 2, 3, 4, 5, 6, 7, 8]

>>> id(x)

1856880275848

>>>

我們也可以使用運算子 + 和 \* 來達到增加列表元素的目的，但這兩個運算子不屬於原地操作，而是返回新的列表，如下：

>>> x = [1, 2, 3]

>>> id(x)

1856880279496

>>> x = x + [4]

>>> x

[1, 2, 3, 4]

>>> id(x)

1856870756488

>>> x = x \* 2

>>> x

[1, 2, 3, 4, 1, 2, 3, 4]

>>> id(x)

1856880017544

>>>

#### pop()、remove()、clear()

這三個方法則是用來刪除列表中的元素，其中pop()會刪除並返回指定位置(預設是最後一個)的元素，remove()則刪除列表中第一個與指定值相等的元素，clear()是用來清空列表，這三種方法也屬於原地操作。

實機操作

>>> x = [1, 2, 3, 4, 5, 6, 7]

>>> x.pop()

\_\_\_\_\_\_\_\_

>>> x.pop(0)

\_\_\_\_\_\_\_\_

>>> x.clear()

>>> x

\_\_\_\_\_\_\_\_

>>> x = [1, 2, 1, 1, 2]

>>> x.remove(2)

>>> x

\_\_\_\_\_\_\_\_

>>> del x[3] # 並不是 list 的方法，為一函數

>>> x

\_\_\_\_\_\_\_\_

#### count()、index()

列表方法 count() 返回列表中指定元素出現的次數，index() 則返回指定元素在列表中首次出現的位置，如果元素不存在則拋出異常。

實機操作

>>> x = [1, 2, 2, 3, 3, 3, 4, 4, 4, 4]

>>> x.count(3)

\_\_\_\_\_\_\_\_

>>> x.count(5)

\_\_\_\_\_\_\_\_

>>> x.index(2)

\_\_\_\_\_\_\_\_

>>> x.index(4)

\_\_\_\_\_\_\_\_

>>> x.index(5)

\_\_\_\_\_\_\_\_

>>> 5 in x

\_\_\_\_\_\_\_\_

>>> 3 in x

\_\_\_\_\_\_\_\_

#### sort()、reverse()

這兩個方法，我們會跟另外兩個函數 sorted()、reversed() 做比較，區別方法與函數的不同，以及in-place 與非 in-place 的不同。

實機操作

>>> x = list(range(11)

>>> import random

>>> \_\_\_\_\_\_\_.shuffle(x)

>>> x

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

>>> x.reverse()

>>> x

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

>>> x.sort()

>>> x

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

>>> x.sort(key=str)

>>> x

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

sorted()與reversed() 就完全不同了，首先它們是函數而不是list的方法，所以呼叫方式不同，其次它們在排序或逆序之後，會傳回新的序列，不影響原本序列。其中 reversed()更特別，它會傳回一個逆向排序的反覆運算物件，由於傳回的是一個物件，所以我們不會直接看到序列的元素，需要額外處理才看的見：

實機操作

>>> x = list(range(11)

>>> import random

>>> \_\_\_\_\_\_\_.shuffle(x)

>>> x

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

>>> reversed(x)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

>>> list(reversed(x))

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

>>> sorted(x)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

>>> sorted(x, key=str)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

>>> x

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

排序方法中的 key 參數，可以實現更複雜的排序動作，我們用個實際的劇本來練習。假設有兩個隊伍參加線上遊戲，分別是A隊與B對，每隊有三個人，以及每個人的得分：

A 隊：

Bob, 86分

Andrew, 95分

Mandy, 83分

B 隊：

Ruby, 89分

Elvis, 91分

Peter, 77分

我們嘗試使用二維列表來記錄：

實機操作

>>> gameresult = [['Bob', 86, 'A'],

['Andrew', 95, 'A'],

['Ruby', 89, 'B'],

['Elvis', 91, 'B'],

['Mandy', 83, 'A'],

['Peter', 77, 'B']]

>>> gameresult

[['Bob', 86, 'A'], ['\_\_\_\_\_\_', 95, 'A'], ['Ruby', \_\_\_\_, '\_\_'], ['\_\_\_\_\_\_', 91, 'B'], ['Mandy', 83, 'A'], ['\_\_\_\_\_\_', 77, 'B']]

>>> from operator import itemgetter

>>> sorted(gameresult, key=itemgetter(2)) #按照子列表第三個元素排序

[['Bob', 86, 'A'], ['Andrew', 95, 'A'], ['\_\_\_\_\_', \_\_\_, '\_\_'], ['Ruby', 89, 'B'], ['Elvis', 91, 'B'], ['\_\_\_\_\_\_', \_\_\_, '\_\_']]

>>> sorted(gameresult, key=itemgetter(2,0))#先第三個元素再第一個元素

[['\_\_\_\_\_\_', \_\_\_, '\_\_'], ['Bob', 86, 'A'], ['Mandy', 83, 'A'], ['\_\_\_\_\_\_\_', \_\_\_, '\_\_'], ['Peter', 77, 'B'], ['Ruby', 89, 'B']]

問題：先按照隊伍排序(順序不拘)，再按照分數高低(由高到低)排序

(提示 reverse=True 降冪排序)

>>> \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

當然，Python 還有更複雜的排序方式，我們先練習到這。

### 內建函數對列表的操作

除了列表物件本身的方法之外，很多 Python 的內建函數也可以操作列表，例如 max()、min()、sum()等函數，這三個之前說明過。而len()則是傳回列表元樹的個數。這四個比較簡單，請自行練習。

接下來說明兩個與列表轉換為元組的相關函數，zip() 和 enumerate()，zip()函數重新組合多個列表的元素成為元組，並返回包含這些元組的 zip物件；enumerate() 函數返回包含若干索引和值的反覆運算物件。看說明想必會昏倒，我們直接看例子：

>>> from random import shuffle

>>> x = list(range(11))

>>> y = list(range(11))

>>> shuffle(x)

>>> x

[5, 8, 7, 4, 10, 6, 1, 9, 3, 2, 0]

>>> y

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

>>> zip(x, y)

<zip object at 0x0000024155383C88>

>>> list(zip(x, y))

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

>>> enumerate(x)

<enumerate object at 0x0000024155370B40>

>>> list(enumerate(x))

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

>>>

zip() 把兩個列表中的元素打包成元組物件，enumerate() 則把一個列表的元素與自行加在前面的index 數字打包成元組物件。因為物件無法被看到，所以我們利用 list() 將其轉換成列表之後呈現。至於什麼是元組(tuple)，下一個章節就會說明。

#### map()

接下來要談的是 Python 最強悍的地方，我們會漸漸發現 Python 與其它程式語言的不同之處。在 Python 中，map()、reduce()、filter() 是函數式程式設計的重要呈現形式。

內建函數 map() 能將一個函數依序作用到序列或反覆運算器物見的每個元素，然後返回 map 物件作為結果，其中每個元素是元序列元素經過該函數處理後的結果，不對原序列作任何修改。

光看敘述我敢保證一定看不懂，不用擔心，我們透過實機操作來了解其意義。

實機操作

>>> x = list(range(5))

>>> x

[0, 1, 2, 3, 4]

>>> map(str, x)

<map object at 0x0000024155382D30>

>>> list(map(str, x))

[\_\_\_\_\_, \_\_\_\_\_, \_\_\_\_\_, \_\_\_\_\_, \_\_\_\_\_]

>>> def add5(v):

return v+5

>>> list(map(add5, x))

[\_\_\_, \_\_\_, \_\_\_, \_\_\_, \_\_\_]

>>> x

[0, 1, 2, 3, 4]

>>> y = list(range(5, 10))

>>> y

[5, 6, 7, 8, 9]

>>> def add(a, b):

return a+b

>>> list(map(add, x, y))

[\_\_\_, \_\_\_, \_\_\_, \_\_\_, \_\_\_]

>>> list(map(lambda a,b:a+b, x, y))

[\_\_\_, \_\_\_, \_\_\_, \_\_\_, \_\_\_]

>>> [add(a, b) for a, b in zip(x, y)]

[\_\_\_, \_\_\_, \_\_\_, \_\_\_, \_\_\_]

>>>

請注意最後兩行指令，一個使用 lambda 語法，一個使用 for 語法，這就是 Python 沒有最簡、只有更簡的特性。

Lambda 語法直接把 “結果=f(參數)” 這樣的函數寫法寫成 “lambda 參數:結果”，for 語法在這裡則稱為列表推導式，後面會進一步說明。

試著使用學習英文的語法分析方法，分析這兩個指令的執行步驟。

#### reduce()

標準庫 functools 中的函數 reduce() 會將一個接收2個參數的函數，以遞減的方式從左到右依序作用到一個數列中的所有元素。用白話文說，就是先以第一個和第二個元素作為參數丟給函數執行，執行的結果再和第三個元素當成兩個新的參數丟給函數執行，依此類推，一直到所有元素執行完畢為止。元素會越來越遞減，最後剩下一個結果。

接續上面說明 map() 的例子，繼續往下實作，因為我們會用到前面自定義的 add()函數：

實機操作

>>> from functools import reduce

>>> seq = [1, 2, 3, 4, 5]

>>> reduce(add, seq)

\_\_\_\_\_

>>> reduce(lambda a, b: a+b, seq)

\_\_\_\_\_

>>>

其實就是在計算 1 加到 5 的總和。

#### filter()

內建函數 filter() 將一個單參數函數作用到序列，並返回該序列中使得該函數的返回值為 True 的所有元素組成的 filter 物件。就像是一個過濾器。

我們在玩線上遊戲的時候通常要取一個別名，如果系統對別名的命名方式規定必須是文字或者是數字的時候，就可以用過濾器把合格的別名挑出來：

實機操作

>>> seq = ['foo', 'xbox360', '!!!', '$\_$', 'husky']

>>> def isok(x):

return x.isalnum()

>>> list(filter(isok, seq))

[\_\_\_\_\_\_\_\_\_\_, \_\_\_\_\_\_\_\_\_\_, \_\_\_\_\_\_\_\_\_\_]

>>> [x for x in seq if x.isalnum()]

['foo', 'xbox360', 'husky']

>>> list(filter(lambda x: x.isalnum(), seq))

['foo', 'xbox360', 'husky']

>>>

請進一步說明 lambda 與 for 的執行步驟：

Lambda:

For:

### 列表推導式

前面出現過很多列表推導式的例子，其實在邏輯上它相當於一個迴圈，只是形式更加簡潔，例如：

>>> a\_list = [x\*x for x in range(10)]

>>> a\_list

[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]

>>> b\_list = []

>>> for x in range(10):

b\_list.append(x\*x)

>>> b\_list

[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]

>>>

隨堂測驗 設計黑白棋棋盤

大部分的人應該有玩過黑白棋，沒有玩過應該也有看過

黑白棋的棋盤是一個 8x8 大小的方格盤

通常我們會使用 二維矩陣 來表示

在 Python 中可以用 二維序列 表示成：

>>> board = [[0,0,0,0,0,0,0,0],

[0,0,0,0,0,0,0,0],

[0,0,0,0,0,0,0,0],

[0,0,0,0,0,0,0,0],

[0,0,0,0,0,0,0,0],

[0,0,0,0,0,0,0,0],

[0,0,0,0,0,0,0,0],

[0,0,0,0,0,0,0,0]]

>>> board

[[0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0]]

>>>

請用 for 完成(提示如下):

>>> [[ 0 for …

>>>

### 切片

切片也是 Python 序列的重要操作之一，可以這樣來解釋，如果有學過 Basic 語言的人一定寫過 Basic 的 for 迴圈，FOR I = 1 TO 10 STEP 1，這個迴圈裏面包含了三個東西，起始值，結束值(Python則不包含)，與間隔。

Python 的切片則寫成這樣：[起始值:結束值(但不包含):間隔]

回想一下我們常常用到的 range(10)，我們用 [i for i in range(10)] 來建立一個 0 到 9 的列表: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]。其實就等同於 [i for i in range(0,10,1)]，在 range() 括號裡面同樣也接受三個數字：

起始值(預設0) : 結束值(但不包含) : 間隔(預設1)

這樣所有的謎題應該就解開了吧！

隨堂測驗 建立一個 [1, 3, 5, 7, 9]列表

>>> [ i for \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ ]

>>>

#### 使用切片取的列表部分元素

直接使用範例教學：

>>> mylist = [i for i in range(3,18,2)]

>>> mylist

[3, 5, 7, 9, 11, 13, 15, 17]

>>> mylist[::]

[3, 5, 7, 9, 11, 13, 15, 17]

>>> mylist[::-1]

[17, 15, 13, 11, 9, 7, 5, 3]

>>> mylist[::2]

[3, 7, 11, 15]

>>> mylist[1::2]

[5, 9, 13, 17]

>>> mylist[3:6]

[9, 11, 13]

>>> mylist[0:100]

[3, 5, 7, 9, 11, 13, 15, 17]

>>> mylist[100:]

[]

>>> mylist[100]

Traceback (most recent call last):

File "<pyshell#101>", line 1, in <module>

mylist[100]

IndexError: list index out of range

>>>

#### 使用切片對列表元素進行增、刪、改

利用切片能夠快速地實現很多目的，直接看下面的執行結果：

>>> aList = [3,5,7]

>>> aList[len(aList):]

[]

>>> aList

[3, 5, 7]

>>> aList[len(aList):] = [9]

>>> aList

[3, 5, 7, 9]

>>> aList[:3] = [1,2,3]

>>> aList

[1, 2, 3, 9]

>>> aList[:3] = []

>>> aList

[9]

>>> aList = list(range(10))

>>> aList

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

>>> aList[::2] = [0] \* (len(aList)//2)

>>> aList

[0, 1, 0, 3, 0, 5, 0, 7, 0, 9]

>>> aList[3:3] = [4,5,6]

>>> aList

[0, 1, 0, 4, 5, 6, 3, 0, 5, 0, 7, 0, 9]

>>> aList[20:30] = [3] \* 2

>>> aList

[0, 1, 0, 4, 5, 6, 3, 0, 5, 0, 7, 0, 9, 3, 3]

>>>

上面的範例如果有任何看不懂的地方，請分段執行。請務必確認全部都弄清楚了之後，再繼續往下。

一般人初學程式語言，通常會從 BASIC開始，當從 BASIC語言進入C語言的世界的時候，最令人頭疼的觀念，就是指標，然後就開始傳值、傳址，搞得初學者一頭霧水。接下來我們要導入 Python 的指標概念，在C語言指標叫做 pointer，在Python 就叫做id。

從最簡單的概念開始，當我們把變數a設定為 5 這個數值的時候，用Python 會寫成 a = 5，這點完全沒有問題，對人類而言，我們看到的是a裡面存著5這個數字，但是對 Python而言，則是5 存在 a 所指的記憶體位址上面。

當我們接著指定 b = a 的時候，我們會認為 a 的值是 5 以及 b 的值是 5，焦點會在值上面，但是Python是把 a, b 指向同一個記憶體位置上面，焦點在位址上面。從例子來看：

>>> a = 5

>>> id(a)

1779396192

>>> b = a

>>> id(b)

1779396192

>>> b = 5

>>> id(b)

1779396192

>>> b = 6

>>> id(b)

1779396224

>>> c=5

>>> d=5

>>> id(c)

1779396192

>>> id(d)

1779396192

>>> d=7

>>> id(d)

1779396256

>>>

不要小看這範例，它詳細說明了 Python 是如何處理記憶體的，不同的變數名稱，即使我們在不同地方指定同一個數值，它們還是指向同一個記憶體位址，直到另一個變數指定了不同數值之後，位址才會改變。

為何要說明這些？因為稍後我們就要討論切片返回的是列表元素的淺複製，與列表物件的直接賦予值是不一樣的：

>>> aList = [3,5,7]

>>> bList = aList

>>> id(aList)

2479625896072

>>> id(bList)

2479625896072

>>> aList == bList # 值相同

True

>>> aList is bList # 同一個物件

True

>>> bList[1] = 8

>>> aList

[3, 8, 7]

>>> cList = aList[::] # 切片，淺複製

>>> aList == cList

True

>>> aList is cList # 不同物件

False

>>> cList[1] = 9

>>> cList

[3, 9, 7]

>>> aList

[3, 8, 7]

>>>

不過數字跟列表有一點不同，如下：

>>> a = 5

>>> b = 5

>>> a is b

True

>>> a = [1,2,3]

>>> b = [1,2,3]

>>> a is b

False

>>>

原因應該是列表太過於龐大，在分開指定值的時候，沒有必要一一比對是否每個元素都相同，反而沒有效率。

### 元組與生成器推導式