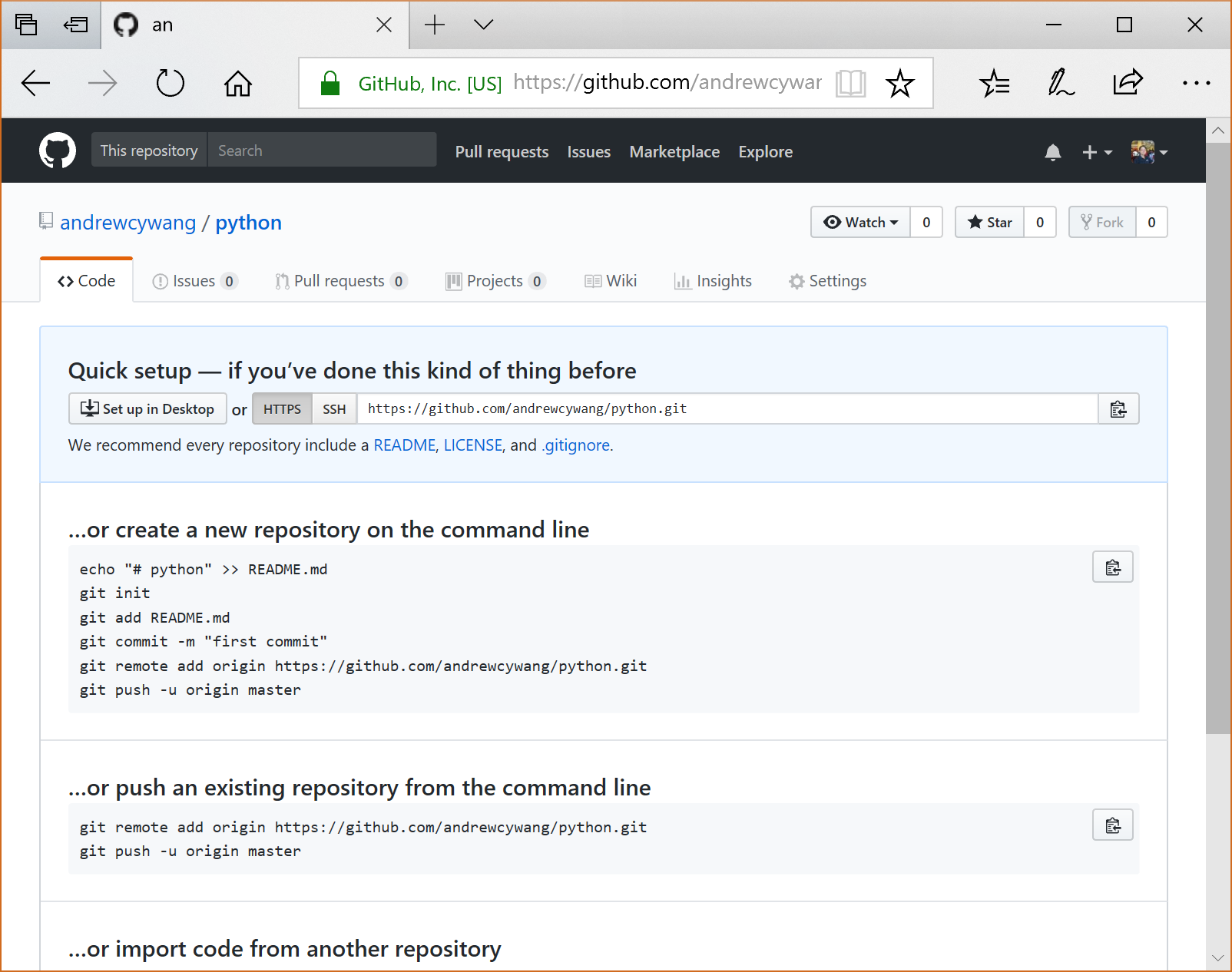
# . Python原來可以這樣玩 .

作者：王昌元

## 課前準備

### GitHub Repository

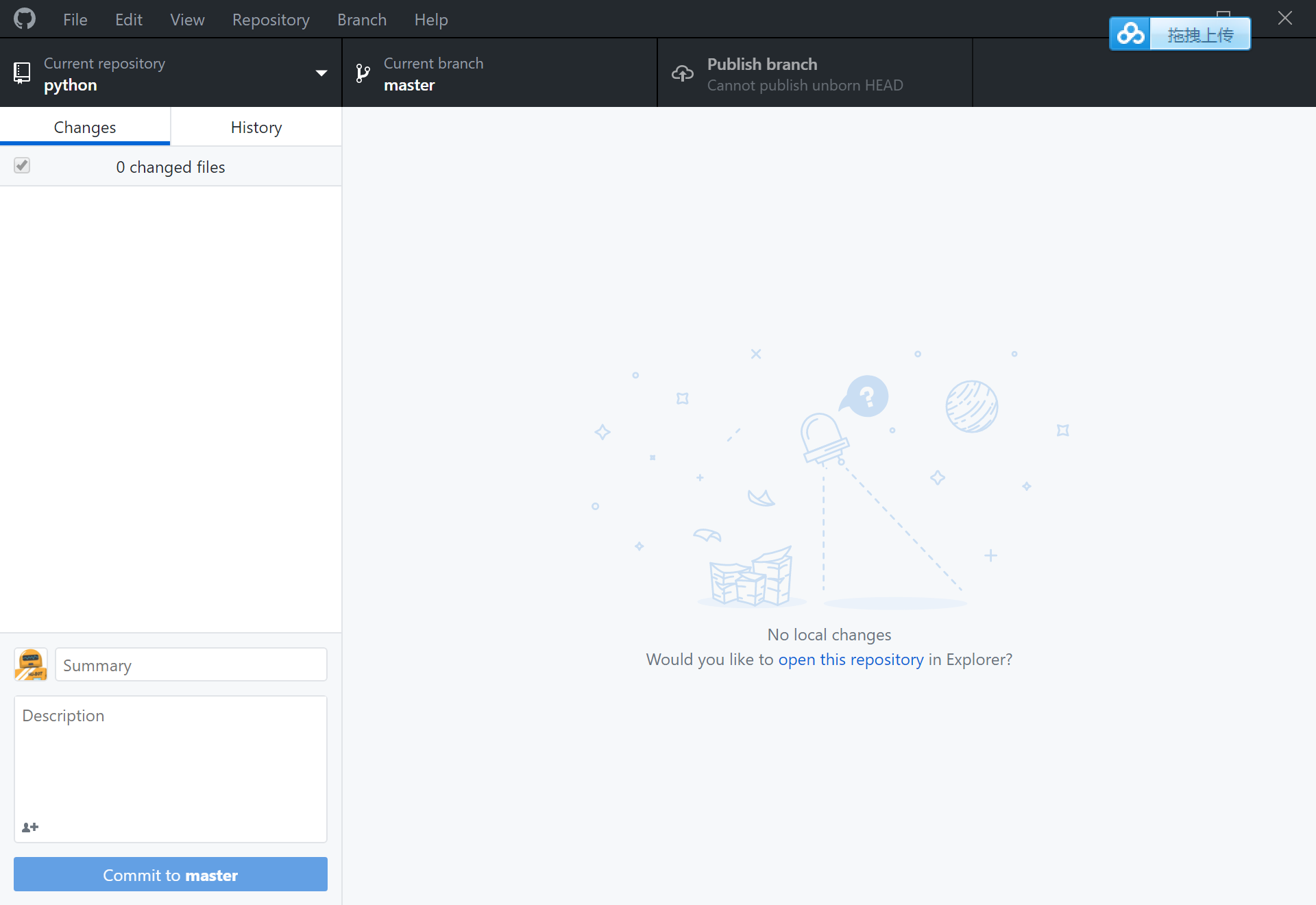
GitHub 是一個很好的工具，可以讓我們共同維護程式碼，因此我使用 GitHub作為教案以及程式碼分享的平台，如下圖：



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。

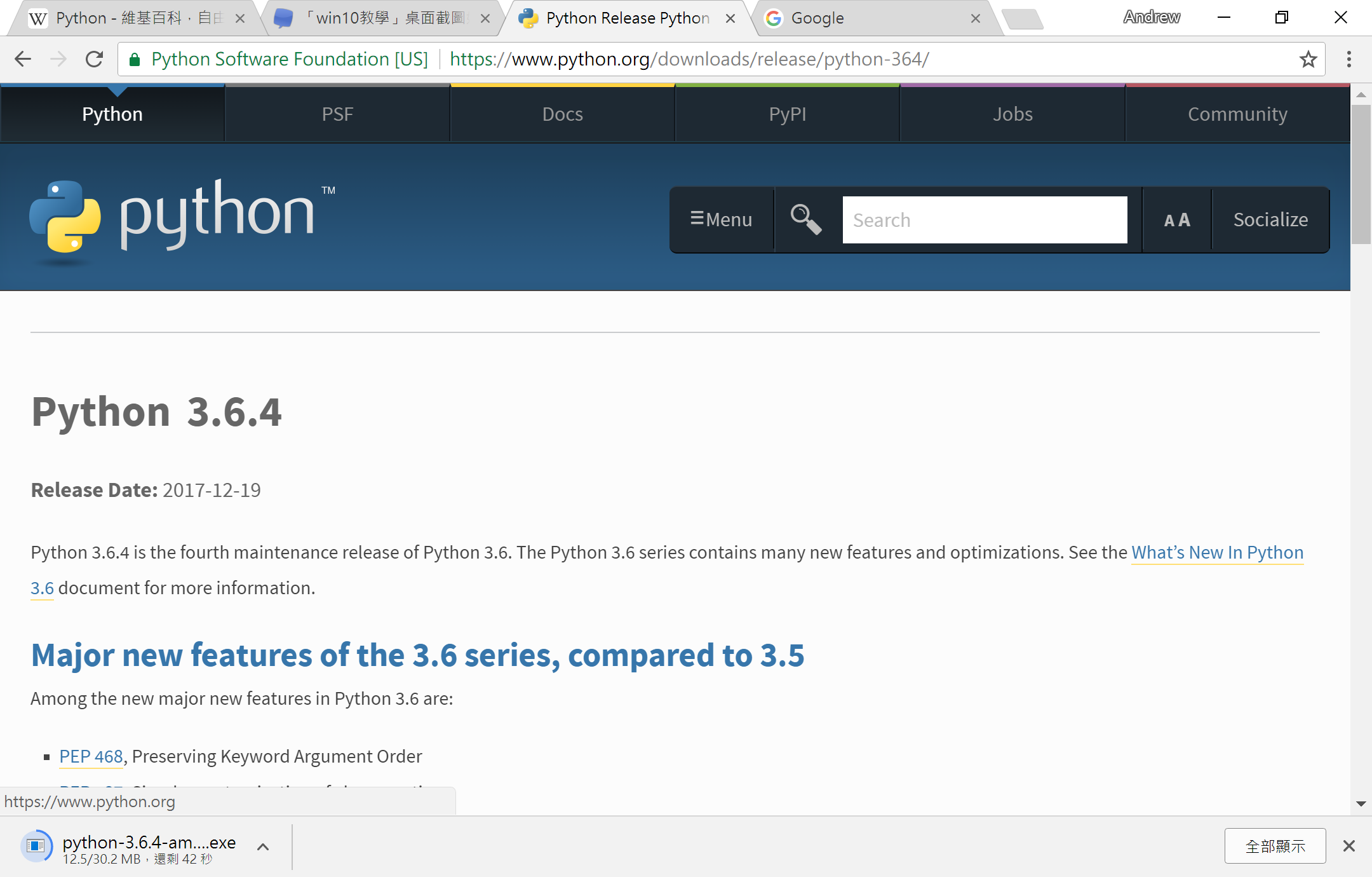
YouTube、Google、Yahoo!、NASA都在內部大量地使用Python。OLPC的作業系統Sugar項目的大多數軟體都是使用Python編寫。

## 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

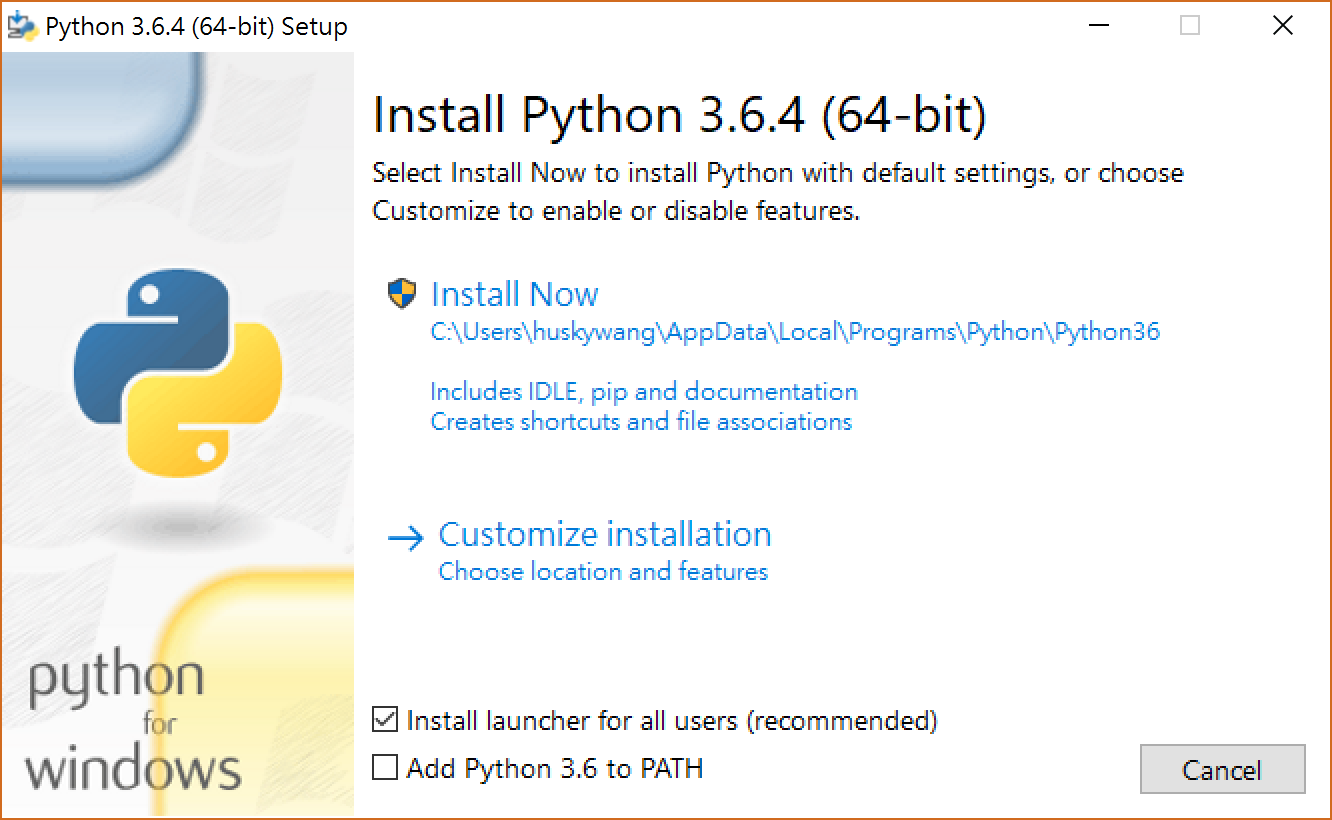
IDLE是Python的官方標準開發環境，可以從官方網站www.python.org下載。



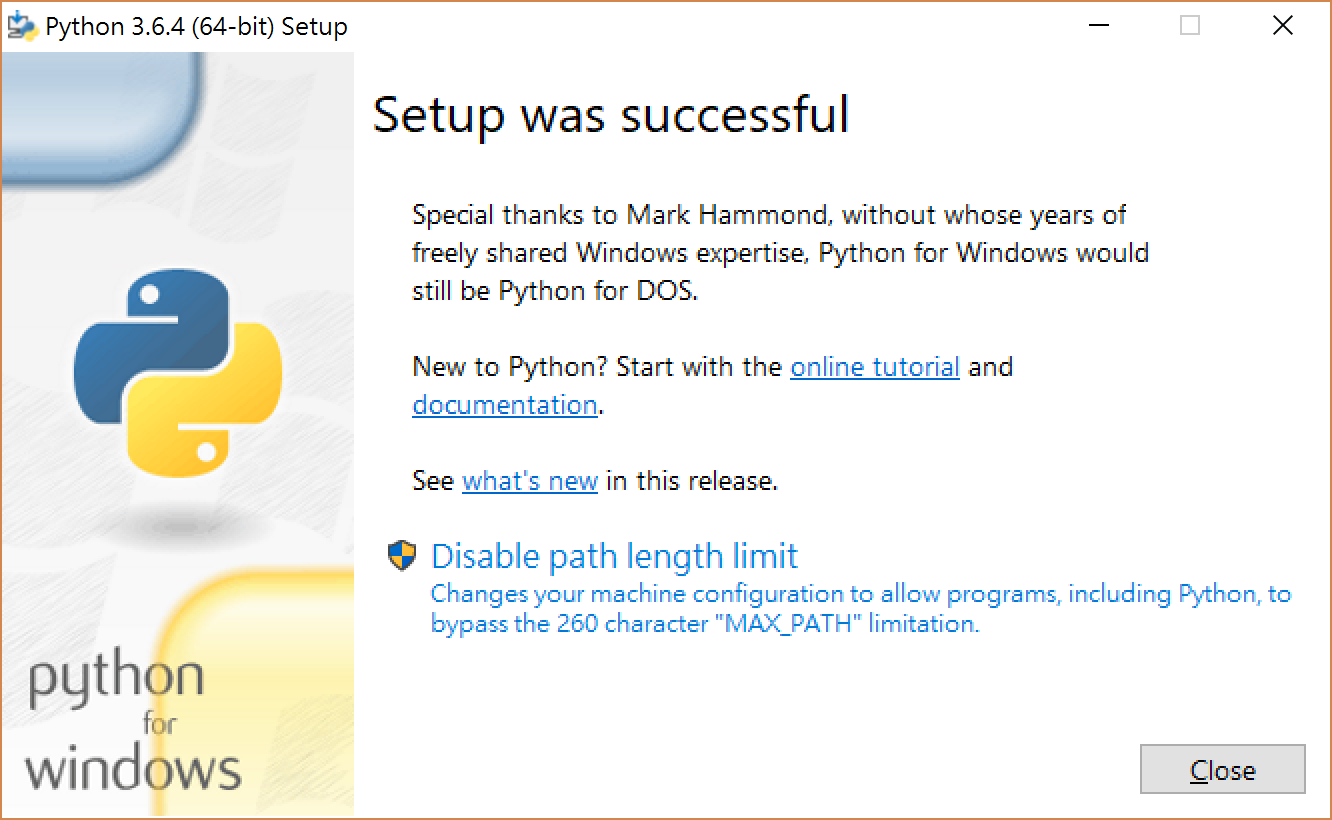
目前最新的版本會顯示在官方網頁，要下載 Windows 的安裝版本。有了最基本的工具，就可以開始寫程式了。為了打好基礎，建議從最基本的開始。

### 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：

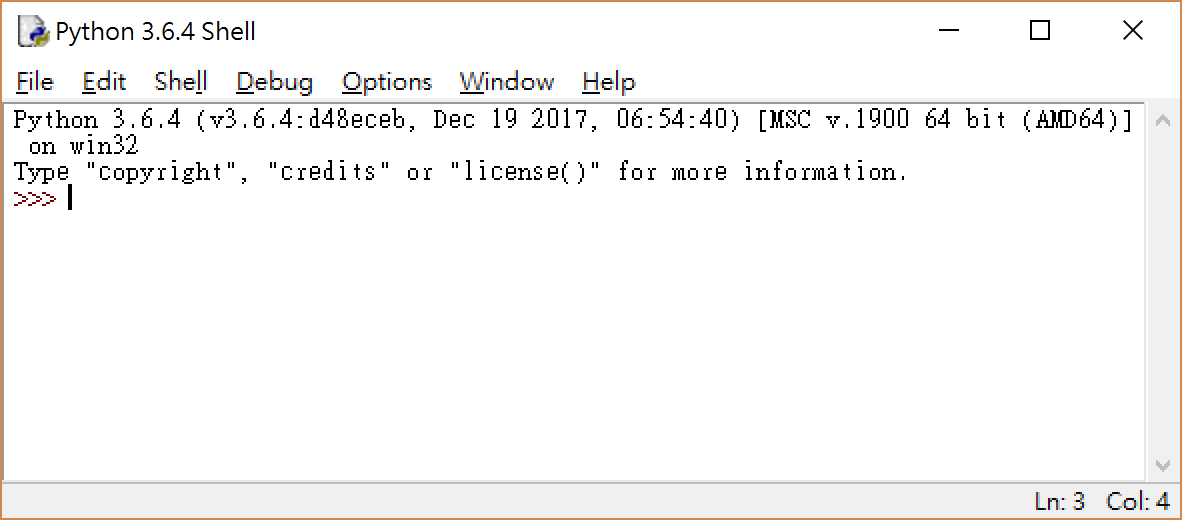


我也建議按下 Disable path length limit，突破過去DOS的限制。

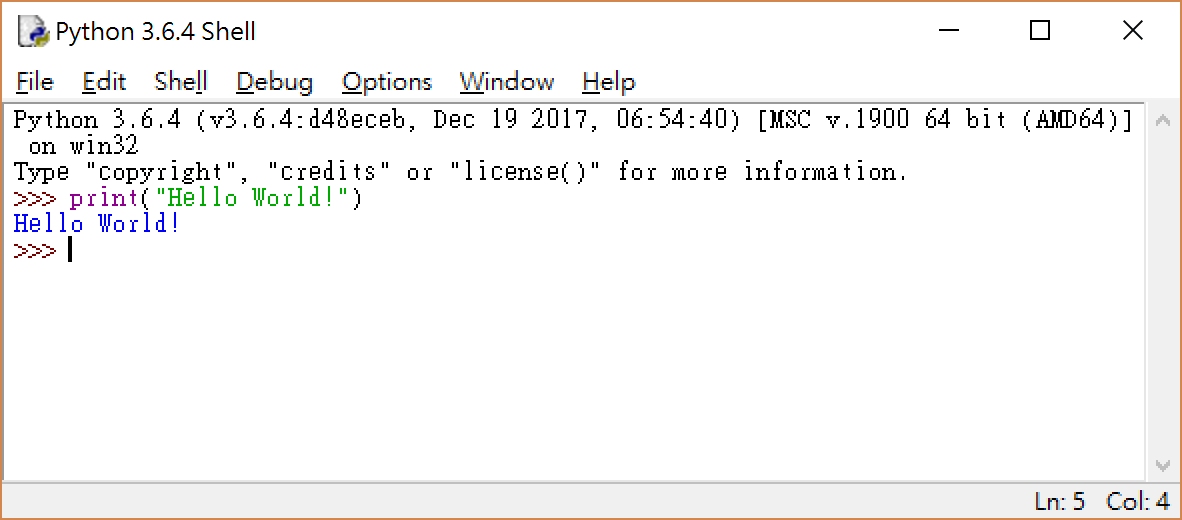
安裝完成之後，我們就可以從[開始]功能表執行[IDLE Python 3.6]，正式進入 Python 的世界。

### 撰寫第一個Python程式

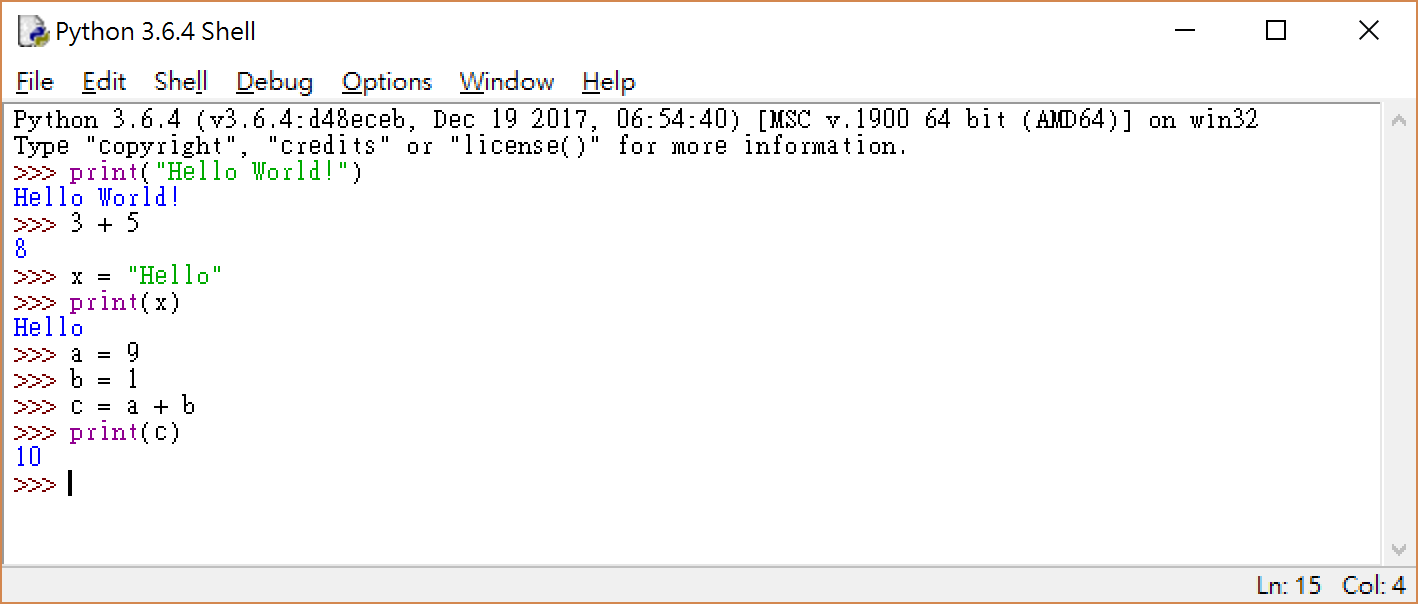
讓我們撰寫一個 Hello World! 程式，這是學習任何一種程式語言的慣例。開啟 Python IDLE之後，會看到下面的畫面：



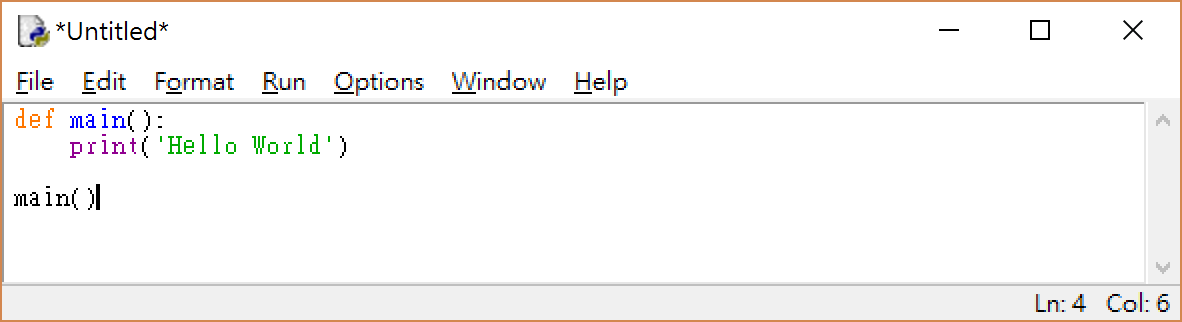
感覺上，我們又好像回到了DOS時代，這畫面就是一個簡單的執行殼，我們直接在>>>後面輸入程式指令，按下ENTER之後執行。Hello World! 程式就是讓 Python 輸出一個 Hello World!字串，我們使用 print 函數來達成：



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：

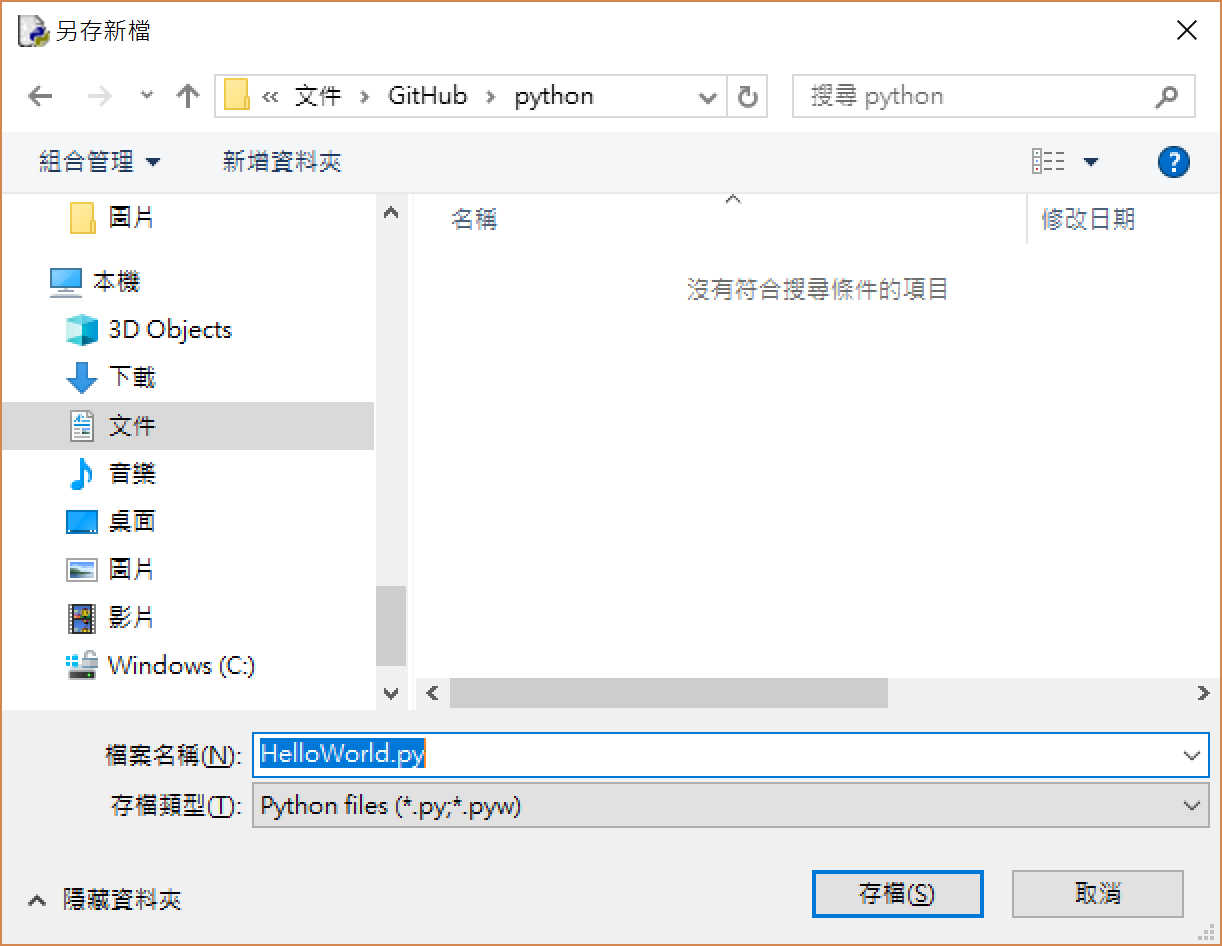


在這裡執行命令就好像跟Python對話一樣，但是都是一行一行執行的，並不像執行程式，程式都是由多個命令所組成的，然後儲存在一個檔案之中，如果我們要把 Hello World 寫成一個程式該如何做呢？第一步當然是建立一個新的程式檔案。在IDLE介面使用功能表[FILE] [NEW FILE]新建一個檔案，這時候會出現一個新的文字編輯器，按照 Python 的語法，程式碼是寫在一個叫做 main() 的函數裡面，這和C語言很像，定義好之後再執行 main() 即可，寫法如下：

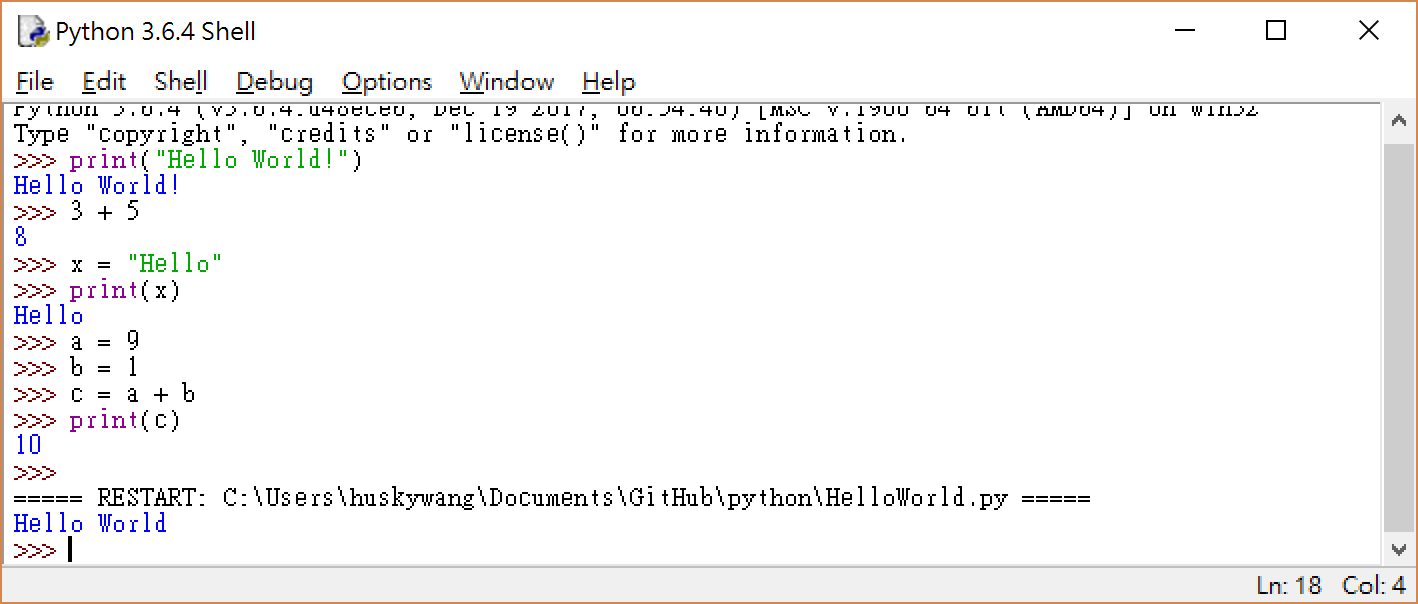


前兩行我們使用關鍵字 def 定義了 main()，要做的事情則利用縮排寫在裡面，就是 print('Hello World') ，然後第三行就是讓 Python 執行 main()。

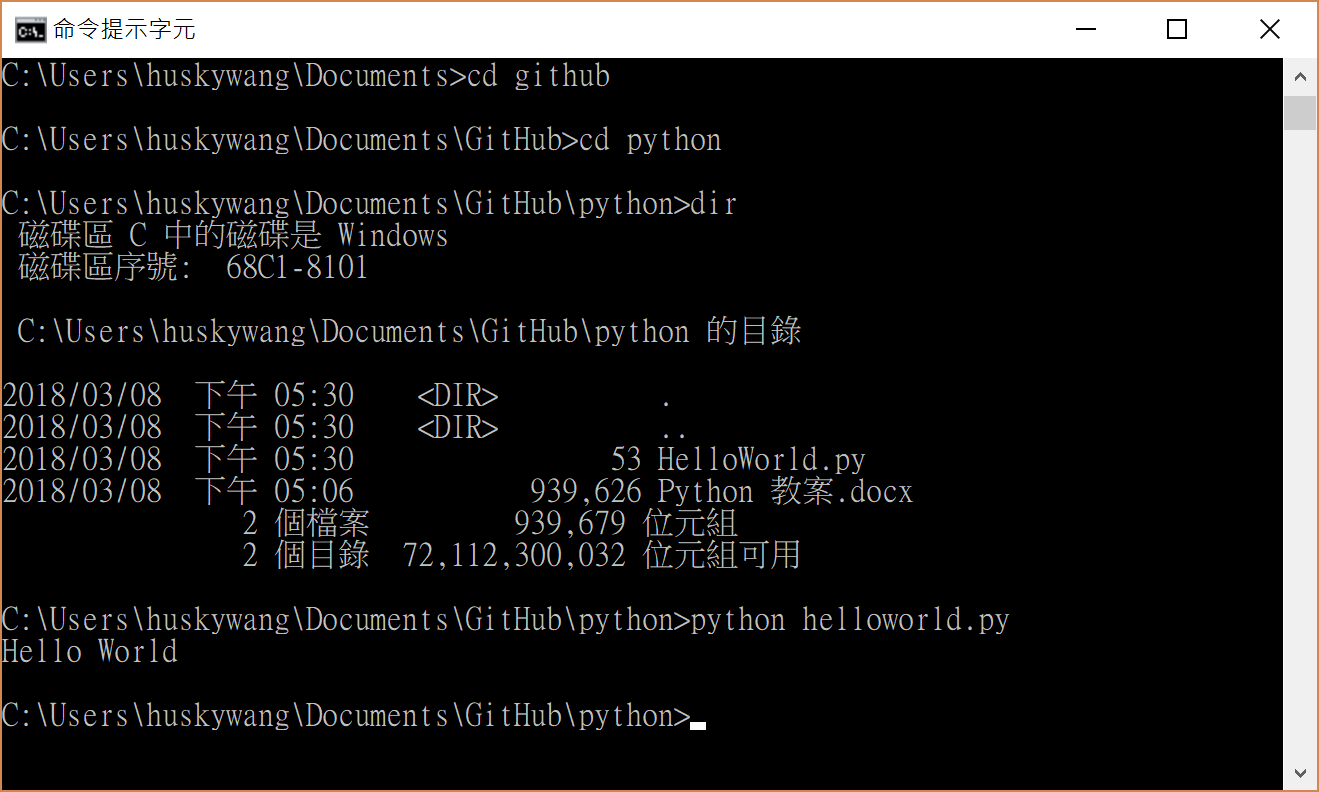
程式寫好之後一定要存檔，Python程式檔是以 .py 結尾的，請點選[FILE] [SAVE AS...]，然後選擇自己放程式的目錄，輸入檔案名稱之後即可：



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

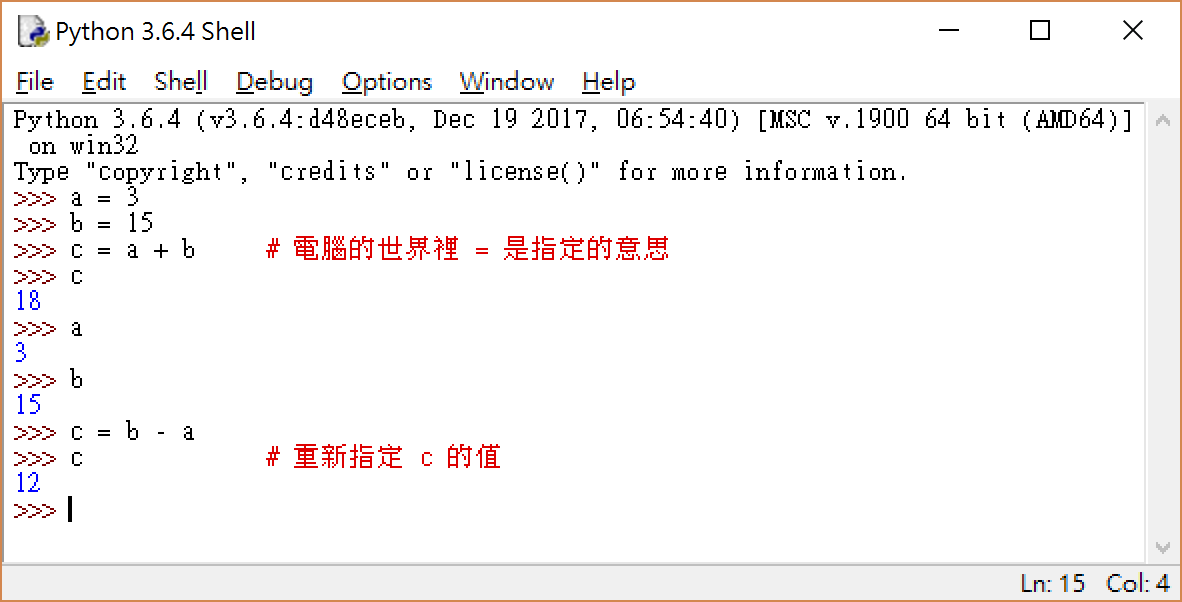
物件是 Python 語言最基本的概念之一，在 Python 中的一切都是物件，常用的內建物件如數字、字串、列表、字典、元組、檔案、檔案、集合等等，連函數都是物件的一種。除此之外，還有大量的標準庫物件和擴展庫物件。Python 預設會安裝標準庫，但需要匯入之後才能使用其中的物件；擴展庫物件則需要先安裝擴展庫，然後再匯入來使用其中的物件。

### 變數、運算子與運算式

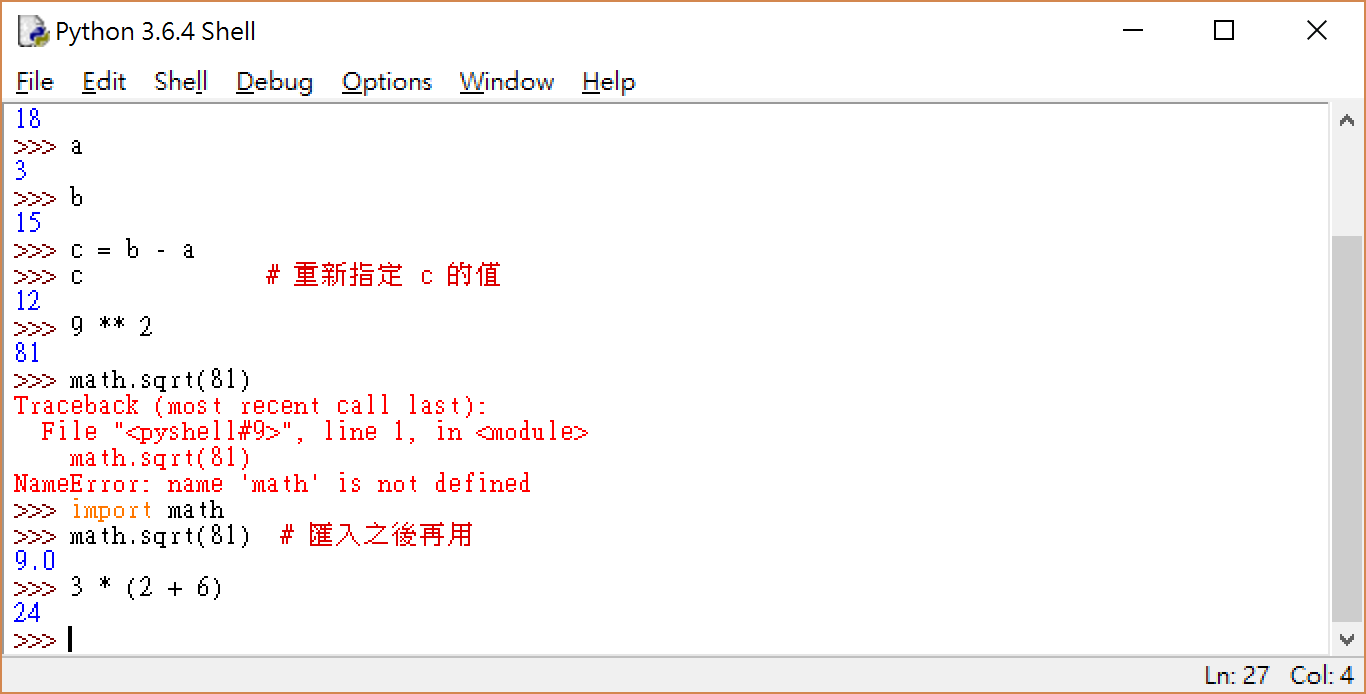
電腦語言是由數學算式演化過來的，所以一切都可以從數學來看待，如果有一個數學式子是這樣寫的：

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

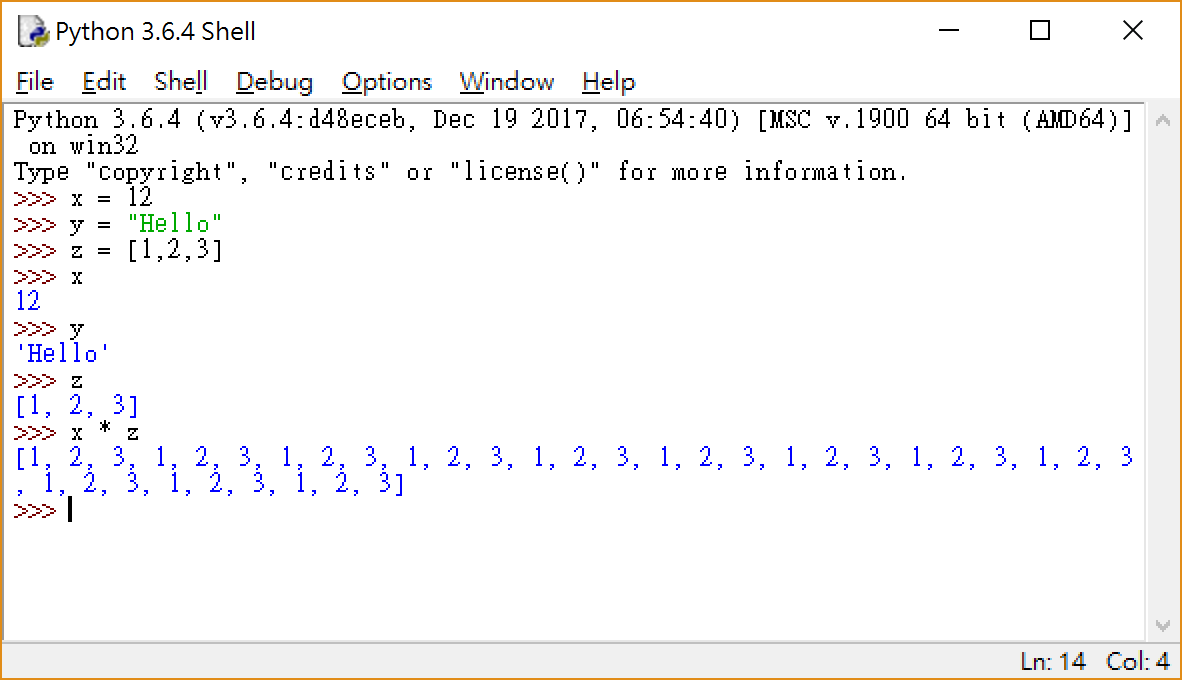
>>> from math import sqrt

>>> 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。



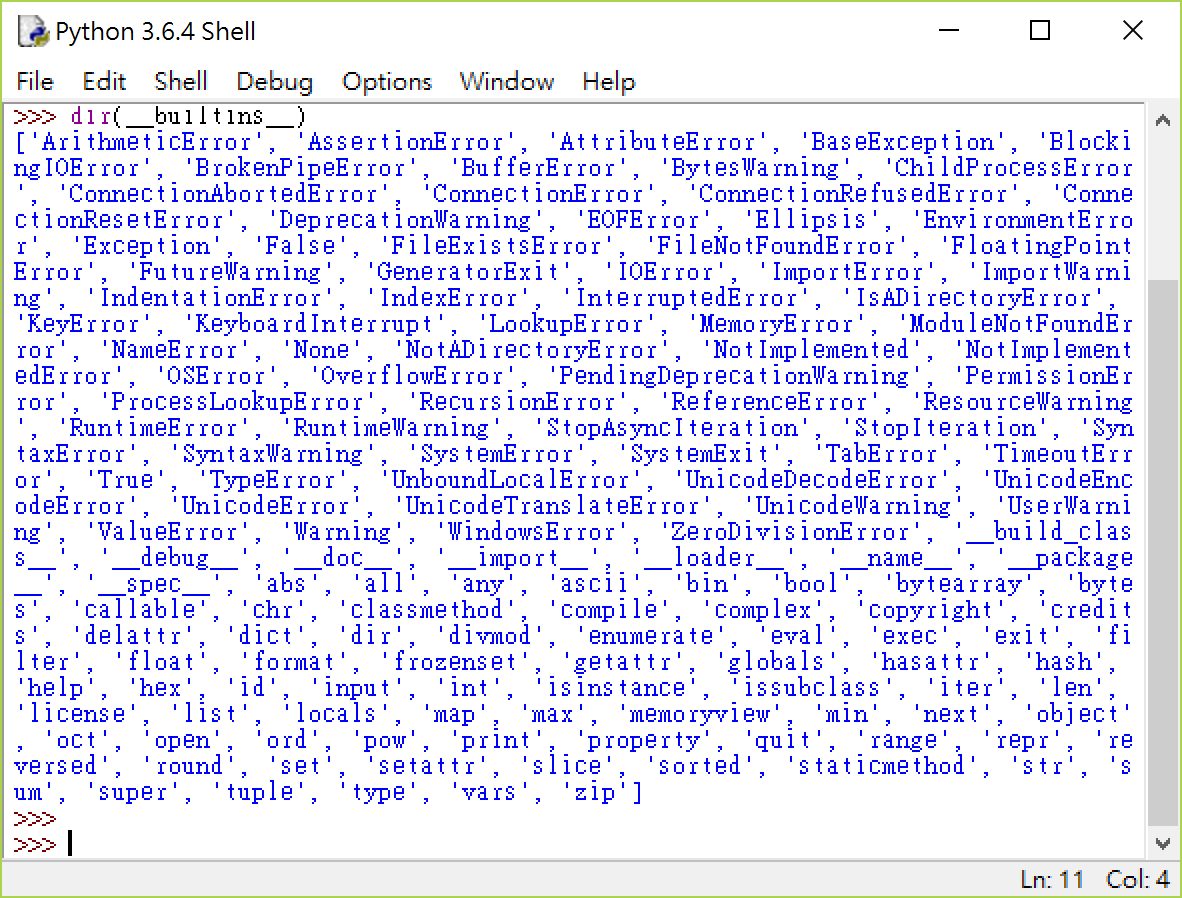
PS: 數字跟列表可以相乘，結果會跑出 12 個 [1,2,3]。但是不同型態不能相加。

### 常用的內建函數

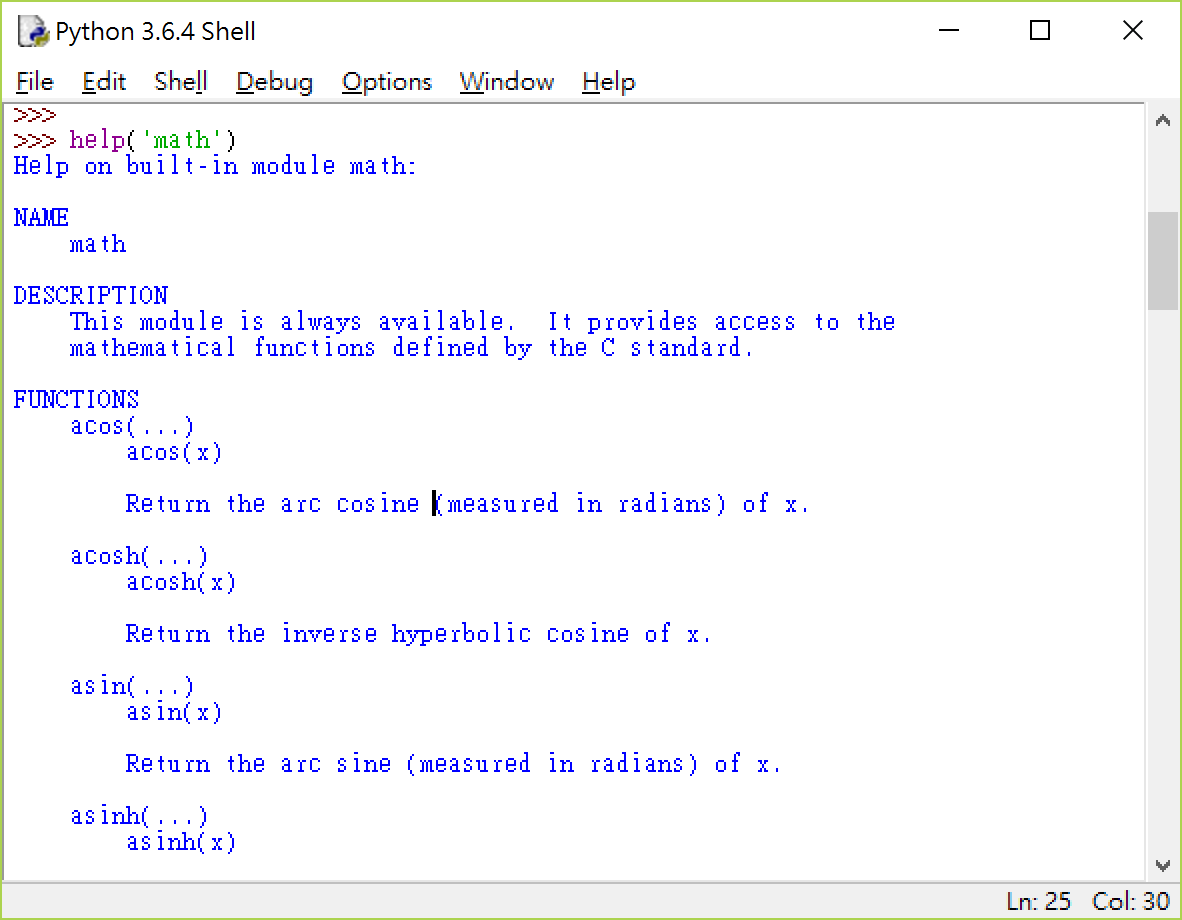
數學函數：y = f(x)

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'

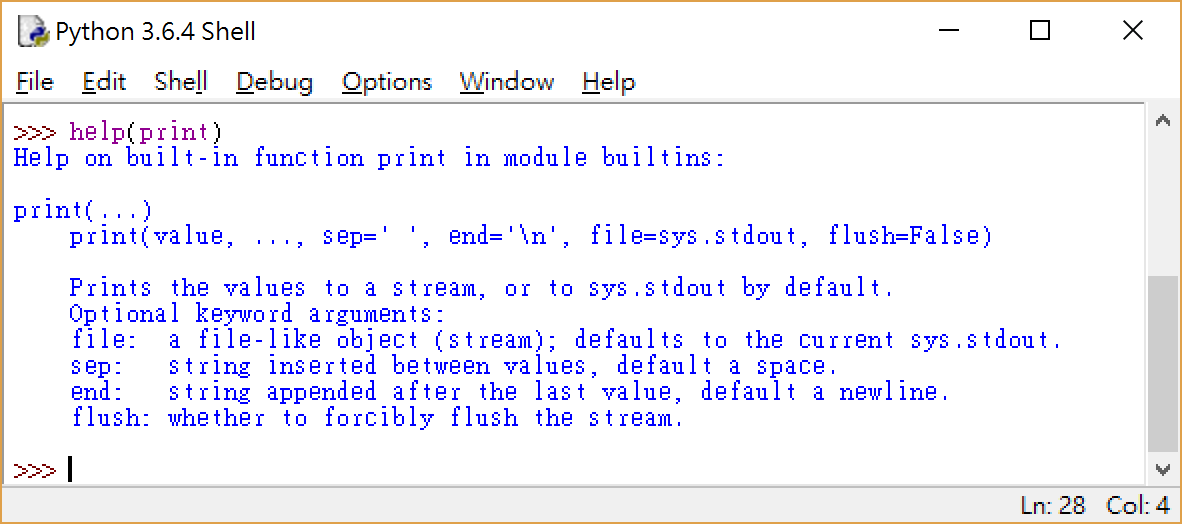
>>> age = eval(age) # 轉成數字之後再指派給 age

>>> 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 () {

……

……

}

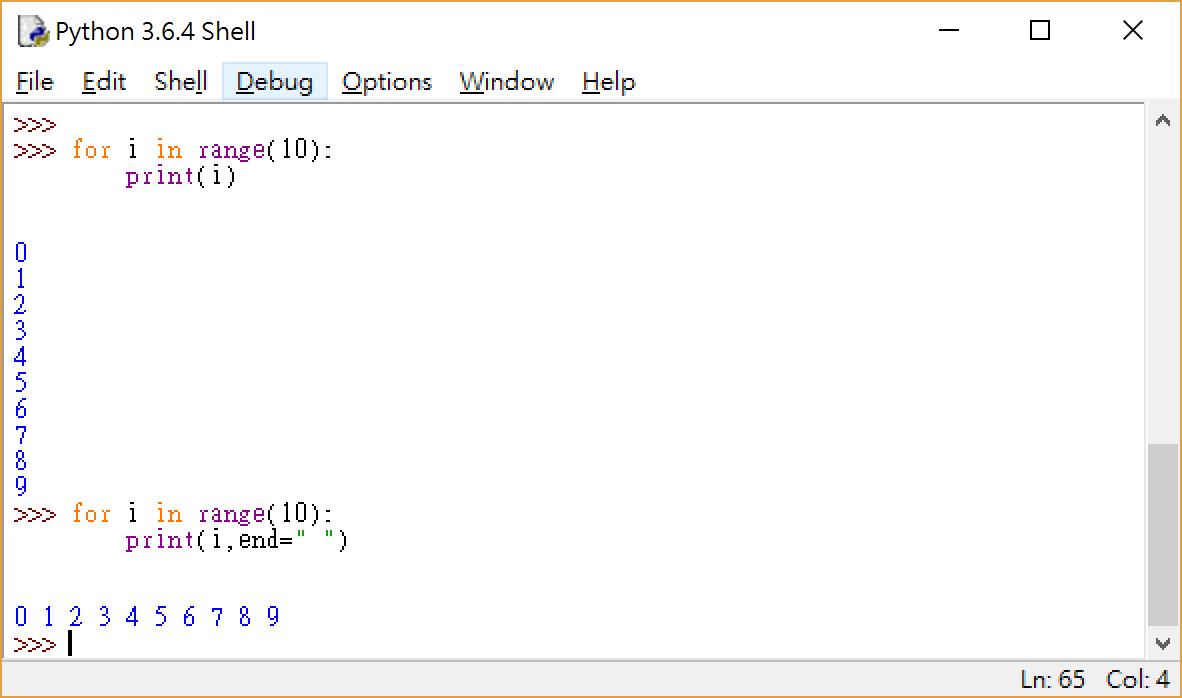
Python:

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):

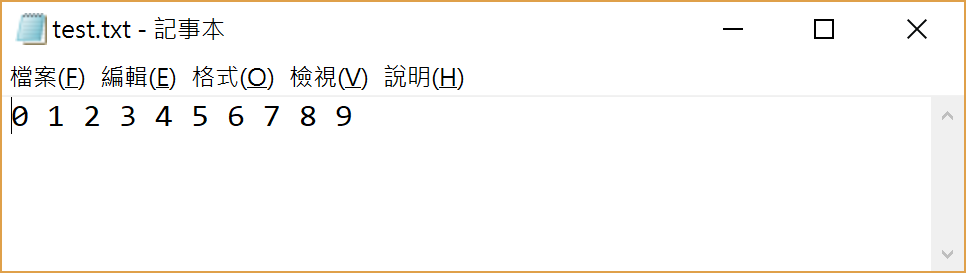
print(i, end=' ', file=fp)

>>> 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)

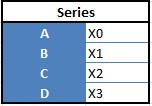
SciPy是一個工程和科學軟體庫。包含線性代數，優化，集成和統計的模組。SciPy庫的主要功能是建立在NumPy上，從而它的陣列大量的使用了NumPy的。它通過其特定子模組提供有效的數值常式，並作為數位積分、優化和其他常式。

#### 3. Pandas (提交數: 15089, 貢獻者數：762)

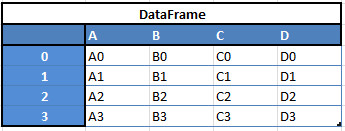
Pandas是一個Python包，旨在通過“標記”和“關係”資料進行工作，簡單直觀。Pandas是資料整理的完美工具。它設計用於快速簡單的資料操作，聚合和視覺化。

庫中有兩個主要的資料結構：

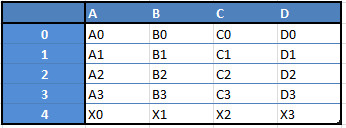
“系列”（Series），一維



“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。

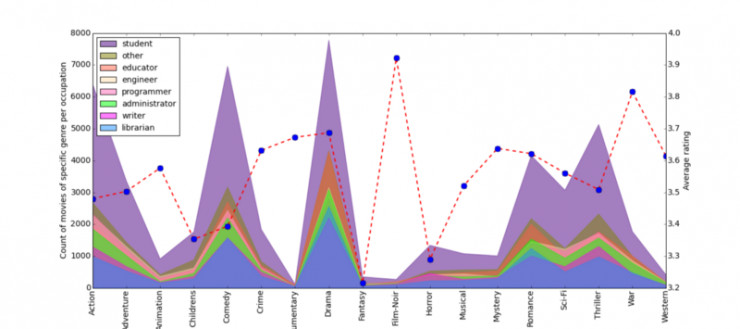
只要付出一點你就可以做任何視覺化：

* 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。

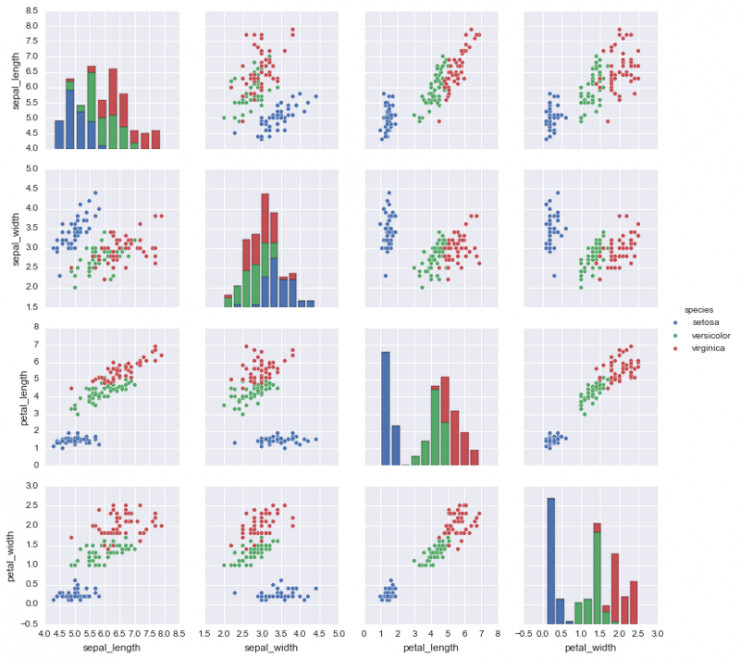
該庫由不同的平臺支援，並使用不同的GUI套件來描述所得到的視覺化。不同的IDE（如IPython）都支援Matplotlib的功能。

還有一些額外的庫可以使視覺化變得更加容易。



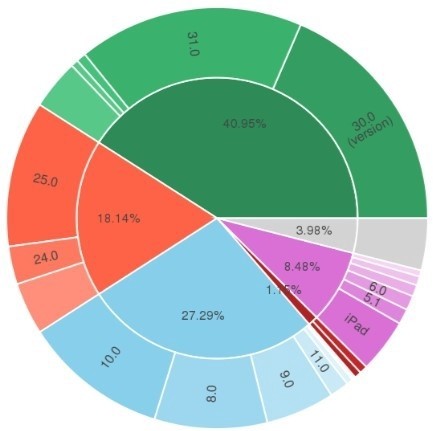
#### 5. Seaborn (提交數: 1699, 貢獻者數: 71)

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。



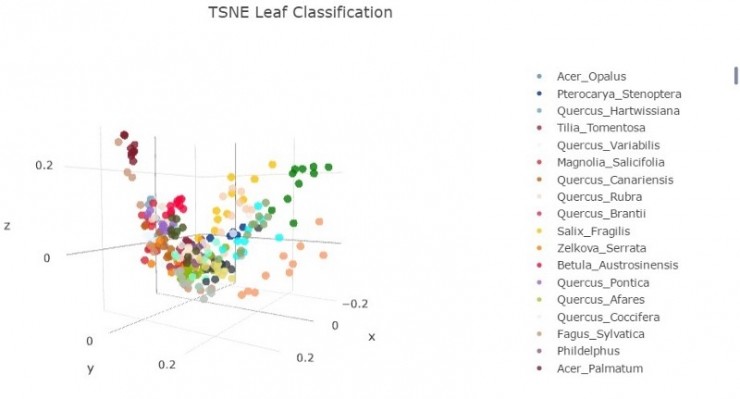
#### 6. Bokeh (提交數: 15724, 貢獻者數: 223)

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

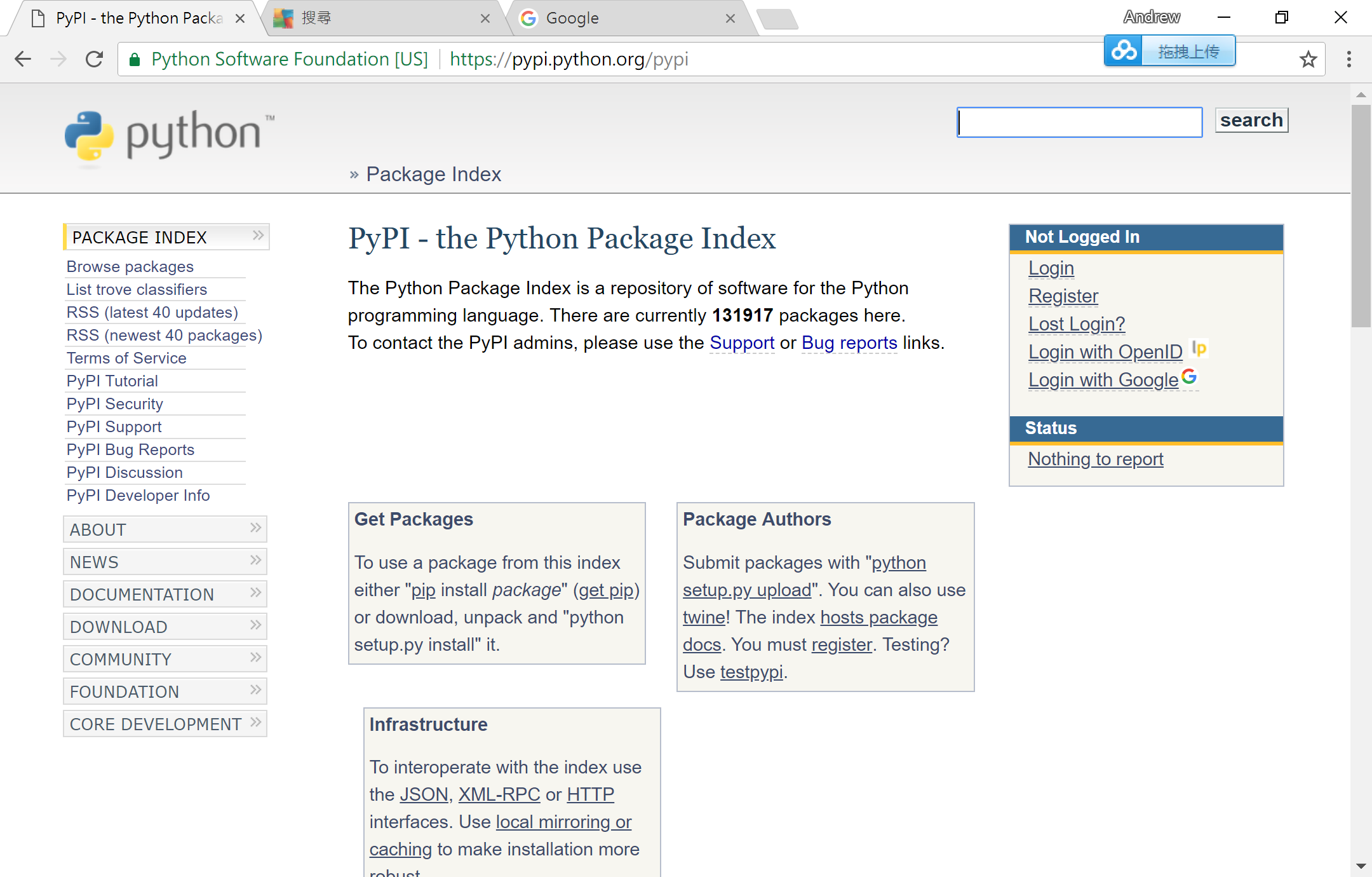
從 <https://pypi.python.org/pypi> 可以獲得一份 Python 擴展庫的綜合列表，到今天為止，已經有 131917 個涉及各領域的擴展庫，並且每天都在以幾十個的速度增加。

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)

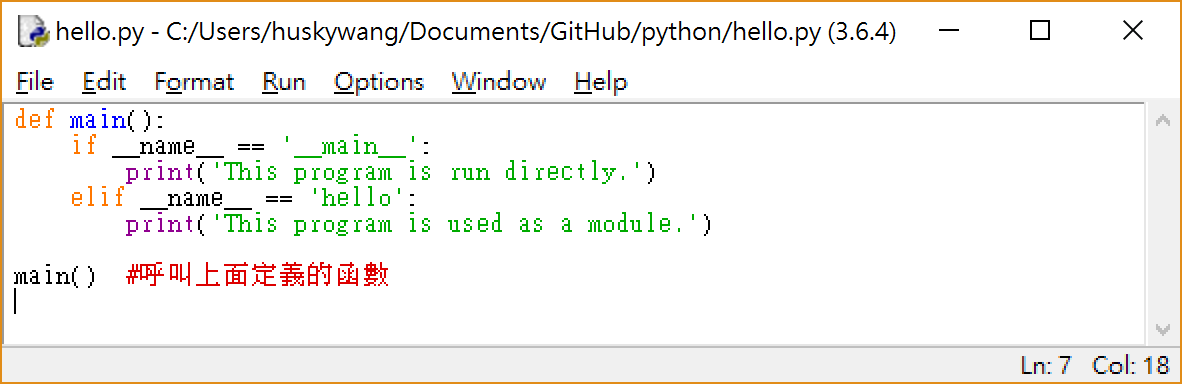
0.1411200080598672

>>>

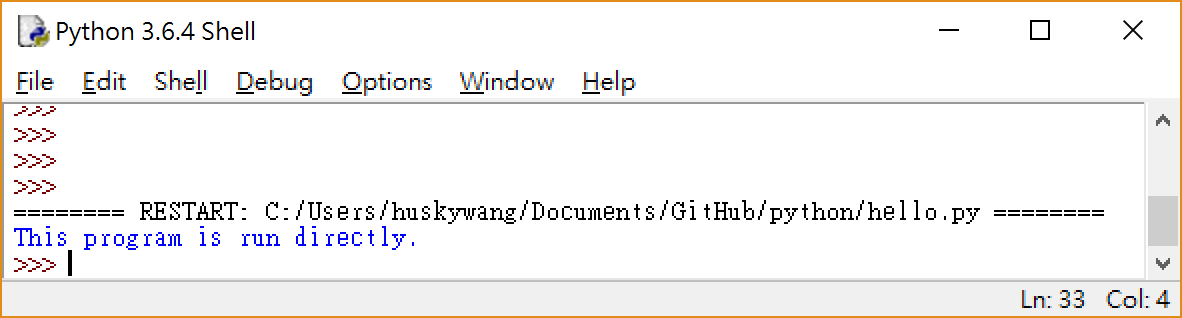
以 from import 的方式明確的匯入指定的物件，而非匯入整個模組，可以適當的提高程式載入和執行速度。

#### 撰寫自己的模組和套件

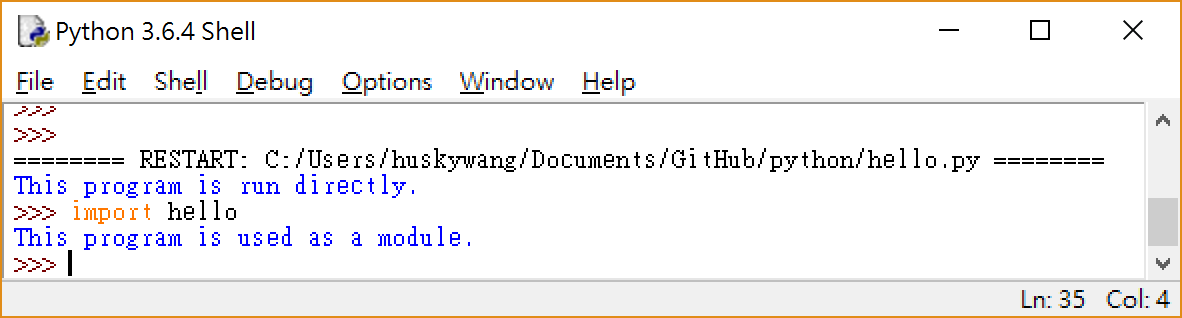
自己寫的 Python 程式除了可以直接執行之外，還能夠作為模組匯入並使用其中的物件。\_\_name\_\_ 屬性便可識別程式的使用方法。每個 Python 腳本在執行的時候都會有一個 \_\_name\_\_ 屬性，如果腳本作為模組匯入，則其 \_\_name\_\_ 屬性值將自動設為模組名稱；如果獨立執行腳本，則其 \_\_name\_\_ 屬性值將自動設為 “\_main\_\_”，例如，hello.py 的程式碼如下：



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。

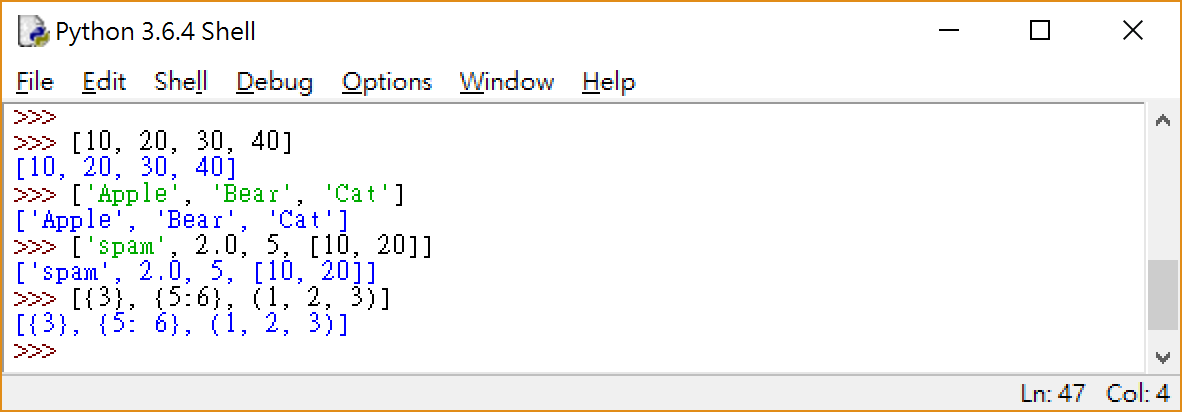
Python 常用的序列結構有列表、元組、字典、字串、集合等等，大致可以分為有序與無序兩類：其中列表、元組、字串屬於有序序列(有順序的)，字典、集合屬於無序序列。前一個章節有討論過列表、字串、集合三個，但是在這裡會更深入的討論。

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

[]

>>>

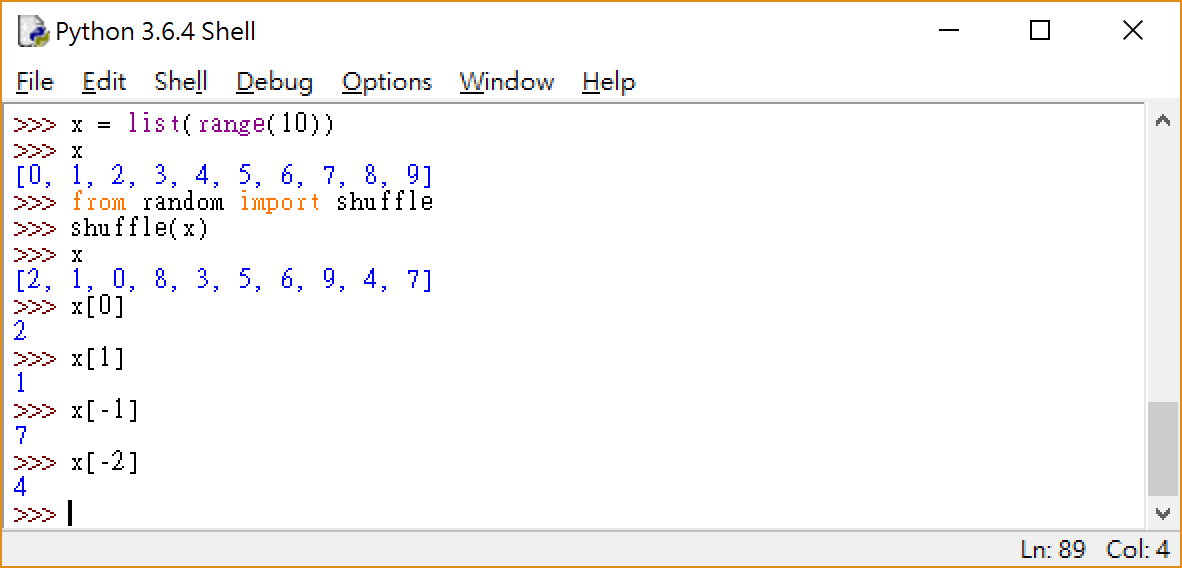
也可以使用 list() 函數將元組、range 物件、字串、字典、集合等資料類別轉換為列表，請先看下面的例子：



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

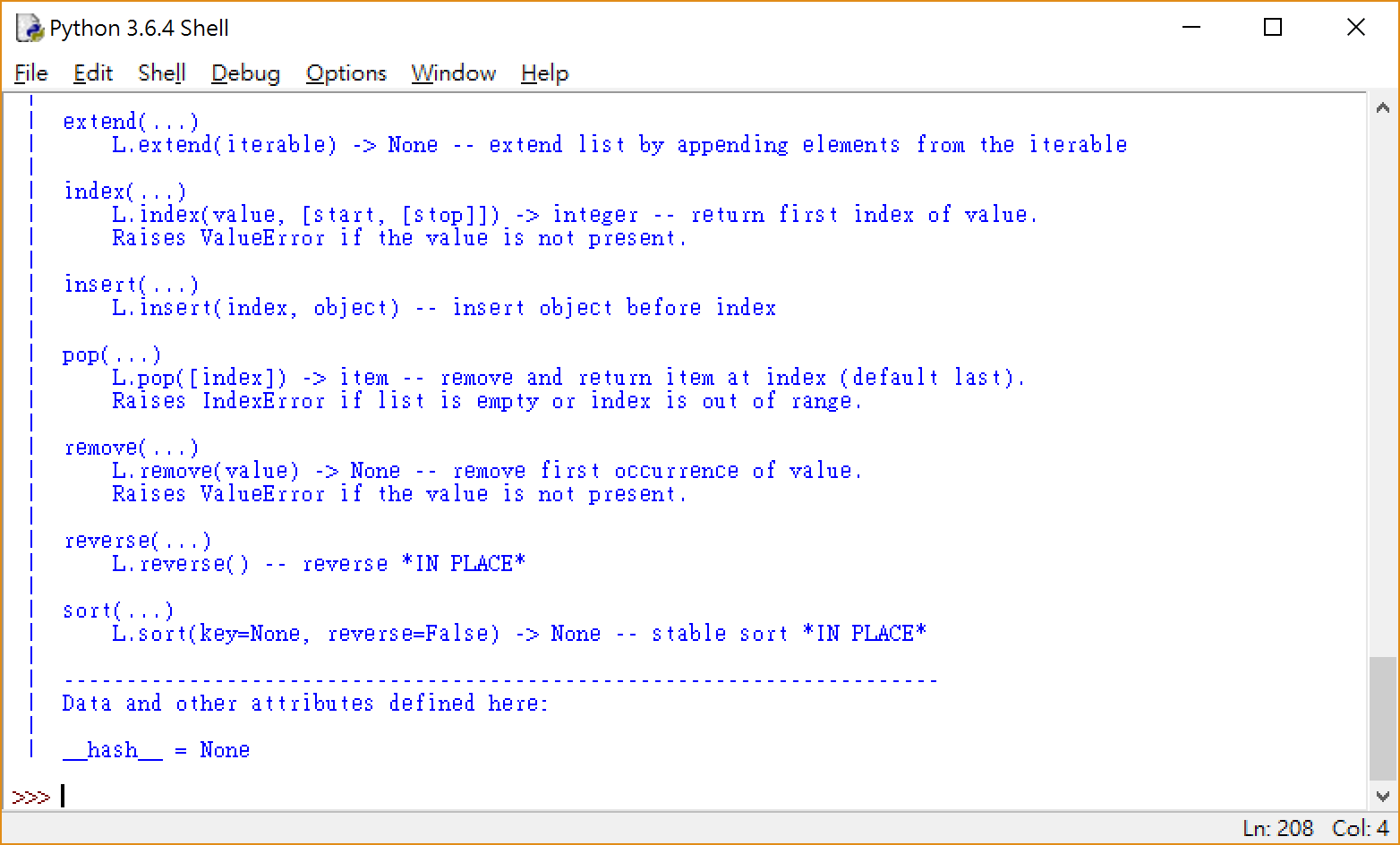
>>> x

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

### 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。Python 針對整數、字元、字串資料類別的變數會指向同一個位址，但像是浮點數、序列則不會。

### 元組與生成器推導式

元組也是 Python 一種重要的序列結構，從形式而言，元組的所有元素放在一對小括號中，元素之間以逗號分隔。如下：

>>> x = (1,2,3)

>>> x

(1, 2, 3)

>>> type(x)

<class 'tuple'>

>>> x = (3)

>>> x

3

>>> type(x)

<class 'int'>

>>> x = (3,)

>>> x

(3,)

>>> type(x)

<class 'tuple'>

>>> x = ()

>>> x

()

>>> tuple(range(5))

(0, 1, 2, 3, 4)

>>>

元組屬於不可變序列，與列表不同。而Python 對原組做了大量最佳化，因此，存取和處理的速度比列表更快。

另外，作為不可變序列，元組與整數、字串一樣，可以當成字典的鍵，有關字典的部分，我們後面說明。

雖然元組是不可變序列，但是如果其內部包含了可變序列，情況就複雜了：

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

>>> x[0][0] = 5

>>> x

([5, 2], 3)

>>>

元組的生成器推導式與列表推導式非常類似，主要的差異在於，生成器推導式的結果是一個生成器物件，而非元組。不像列表推導式生成的就是列表。生成器物件可以依照我們的需求產生元組或是列表，還有一個特性就是生成器推導式會用 \_\_next\_\_() 方法來巡訪元素，而且當巡訪結束之後，就巡訪不到元素了。請看下面的範例：

>>> g = ((i + 2)\*\*2 for i in range(10))

>>> g

<generator object <genexpr> at 0x00000223B390E200>

>>> tuple(g)

(4, 9, 16, 25, 36, 49, 64, 81, 100, 121)

>>> list(g)

[]

>>> g = ((i + 2)\*\*2 for i in range(10))

>>> list(g)

[4, 9, 16, 25, 36, 49, 64, 81, 100, 121]

>>> tuple(g)

()

>>> g = ((i + 2)\*\*2 for i in range(10))

>>> g.\_\_next\_\_()

4

>>> g.\_\_next\_\_()

9

>>>

### 字典

字典(dictionary) 是包含若干 “鍵:值” 元素的無序可變序列，為何叫做字典，顧名思義就是可以透過鍵來查值，如同查字典，下面的例子一看應該就明白：

>>> mydict = {'apple':'蘋果', 'bear':'熊', 'cat':'貓'}

>>> mydict

{'apple': '蘋果', 'bear': '熊', 'cat': '貓'}

>>> mydict['apple']

'蘋果'

>>> mydict['蘋果']

Traceback (most recent call last):

File "<pyshell#50>", line 1, in <module>

mydict['蘋果']

KeyError: '蘋果'

>>>

#### 字典的建立，元素的增加、修改與刪除

除了上面的方法，我們可以使用 dict() 與 zip()來建立字典：

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

>>> value = ['a','b','c','d']

>>> mydict = dict(zip(keys,value))

>>> mydict

{1: 'a', 2: 'b', 3: 'c', 4: 'd'}

>>> mylist = list(zip(keys,value))

>>> mylist

[(1, 'a'), (2, 'b'), (3, 'c'), (4, 'd')]

>>> print(mydict)

{1: 'a', 2: 'b', 3: 'c', 4: 'd'}

>>> d = dict(name='Wang', age=18)

>>> d

{'name': 'Wang', 'age': 18}

>>> d = dict.fromkeys(['name','age','sex']) #建立空字典

>>> d

{'name': None, 'age': None, 'sex': None}

>>>

如同列表一樣，字典可以透過 mydict[‘鍵’]=’值’ 的方式來改變字典元素的值，我們還可以用 update() 方法，將另一個字典的 “鍵:值” 一次性的加到目前的字典物件，我們接續上面的例子：

>>> d

{'name': None, 'age': None, 'sex': None}

>>> d['name']='Wang'

>>> e = {'name':'Andrew','age':20, 'phone':'0922332333'}

>>> d

{'name': 'Wang', 'age': None, 'sex': None}

>>> e

{'name': 'Andrew', 'age': 20, 'phone': '0922332333'}

>>> d.update(e)

>>> d

{'name': 'Andrew', 'age': 20, 'sex': None, 'phone': '0922332333'}

>>>

也可以新增和刪除字典元素，並在字典裏面加入列表元素：

>>> d

{'name': 'Elvis', 'age': 20, 'sex': None, 'phone': '0922332333'}

>>> d['score'] = [ 92, 94, 86 ]

>>> d

{'name': 'Elvis', 'age': 20, 'sex': None, 'phone': '0922332333', 'score': [92, 94, 86]}

>>> del d['sex']

>>> d

{'name': 'Elvis', 'age': 20, 'phone': '0922332333', 'score': [92, 94, 86]}

>>>

隨堂測驗 建立一個字典的列表

看了上面的例子是否感覺可以利用在線上遊戲隊員基本資料以及分數上面

把 d 字典當成一個隊員，裡面有姓名、年齡、電話、以及三次比賽成績

如果團隊裡面有五個隊員該如何做呢？

>>> d

{'name': 'Elvis', 'age': 20, 'phone': '0922332333', 'score': [92, 94, 86]}

>>> d

{'name': 'Elvis', 'age': 20, 'phone': '0922332333', 'score': [92, 94, 86]}

>>> team = [d]

>>> team

[{'name': 'Elvis', 'age': 20, 'phone': '0922332333', 'score': [92, 94, 86]}]

>>> e = {'name': 'Zoe', 'age': 30, 'phone': '0911331331', 'score': [91, 74, 76]}

>>> team.append(e)

>>> team

[{'name': 'Elvis', 'age': 20, 'phone': '0922332333', 'score': [92, 94, 86]}, {'name': 'Zoe', 'age': 30, 'phone': '0911331331', 'score': [91, 74, 76]}]

>>>

>>>

>>>

>>>

>>>

>>>

>>>

我們可以透過 d[‘name’]的方式讀取元素的值，但是如果該鍵不存在，則會拋出錯誤：

>>> d

{'name': 'Elvis', 'age': 20, 'phone': '0922332333', 'score': [92, 94, 86]}

>>> d['name']

'Elvis'

>>> d['address']

Traceback (most recent call last):

File "<pyshell#121>", line 1, in <module>

d['address']

KeyError: 'address'

>>>

當有錯誤拋出的時候，就代表程式被中斷了，所以如果我們是在寫程式，當然不希望程式因為錯誤而跳出，這時候我們就必須用下面的技巧來處理：

>>> if 'Address' in d:

print(d['Address'])

else:

print('不存在')

不存在

>>> try:

print(d['Address'])

except:

print('不存在')

不存在

>>>

雖然上面的方法可以滿足要求，但是程式碼很長，我們可以利用 get() 方法來達成同樣的結果：

>>> d

{'name': 'Elvis', 'age': 20, 'phone': '0922332333', 'score': [92, 94, 86]}

>>> d.get('name','Not Exist')

'Elvis'

>>> d.get('address','Not Exist')

'Not Exist'

>>>

另外還有一種方式，就是如果該鍵不存在的時候，就新增該鍵，並加入一個預設值：

>>> d.setdefault('name','XXX')

'Elvis'

>>> d.setdefault('address','XXX')

'XXX'

>>> d

{'name': 'Elvis', 'age': 20, 'phone': '0922332333', 'score': [92, 94, 86], 'address': 'XXX'}

>>>

最後，對字典做反覆運算時，預設是巡訪字典的鍵，當然，我們可以使用 item()方法同時巡訪 鍵:值；keys() 方法返回所有鍵，value() 方法返回所有值。

>>> e

{'name': 'Zoe', 'age': 30, 'phone': '0911331331', 'score': [91, 74, 76]}

>>> for item in e:

print(item)

name

age

phone

score

>>> for item in e.items():

print(item)

('name', 'Zoe')

('age', 30)

('phone', '0911331331')

('score', [91, 74, 76])

>>> e.keys()

dict\_keys(['name', 'age', 'phone', 'score'])

>>> e.values()

dict\_values(['Zoe', 30, '0911331331', [91, 74, 76]])

>>>

### 集合

集合是無序可變序列，使用一對大括號作為符號，元素之間以逗點分隔，同一個集合內的每個元素不可以重複，跟數學上的集合是相同的。

前面的章節已經介紹過集合的基本運算：聯集、交集、差集、對稱差集。

#### 集合的操作和運算

使用 add() 方法來增加新元素，使用 update() 方法來合併另一個集合到目前集合中：

>>> s = {1,2,3}

>>> s.add(3)

>>> s

{1, 2, 3}

>>> s.add(4)

>>> s

{1, 2, 3, 4}

>>> s.update({4,5})

>>> s

{1, 2, 3, 4, 5}

>>>

集合物件 pop() 方法會隨機刪除一個元素並返回該元素，如果集合為空集合則拋出異常；remove() 方法是刪除集合中的元素，如果元素不存在會拋出異常；discard() 會從集合中刪除一特定元素，如果不存在則忽略此操作；clear() 方法刪除所有元素：

>>> s

{1, 2, 3, 4, 5}

>>> s.discard(6)

>>> s

{1, 2, 3, 4, 5}

>>> s.remove(6)

Traceback (most recent call last):

File "<pyshell#175>", line 1, in <module>

s.remove(6)

KeyError: 6

>>> s.remove(5)

>>> s

{1, 2, 3, 4}

>>> s.pop()

1

>>> s

{2, 3, 4}

>>> s.clear()

>>> s

set()

>>> s.pop()

Traceback (most recent call last):

File "<pyshell#182>", line 1, in <module>

s.pop()

KeyError: 'pop from an empty set'

>>>

### 序列拆解

這功能其實跟 Python 的語言特性有關，一般程式語言的 = 只能一對一，但是 Python 可以多對多：

>>> x = 1

>>> x , y = 1, 2

>>> x

1

>>> y

2

>>> x, y, z = range(3)

>>> x

0

>>> y

1

>>> z

2

>>> a = [4,5,6]

>>> b, c, d = a

>>> b

4

>>> c

5

>>> d

6

>>>

## 程式控制結構與函數設計

前面的章節，我們都在互動介面環境底下玩指令，其實還沒有開始寫程式，要撰寫一個完整的程式，就必須要了解程式控制的語法，學會這些語法，我們就可以讓電腦按照我們的指示做事。

### 條件運算式

別著急，一開始我們還是在互動式環境下面練習。我們用中文來詮釋條件運算式，如果這件事是真的，就執行甲方案，若不是，就執行乙方案，用電腦就會寫成(請務必學會如何輸入，輸入:自動縮排，del鍵結束縮排，代表 block 結束)：

>>> a = True

>>> if a == True:

print('甲方案')

else:

print('乙方案')

甲方案

>>>

重點來了，Python 是如何定義 True 跟 False 呢? 條件運算式的值只要不是 False、0、0.0、空值None、空列表、空元組、空集合、空字典、空字串等，Python 均認為是 True 的等價。所以，下面的例子都被當成 True (最後的例子是 False)。

>>> if 3:

print('True')

True

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

>>> if a:

print(a)

[1, 2, 3]

>>> a = []

>>> if a:

print(a)

else:

print('Empty')

Empty

>>>

搭配前面講過的關係運算子與邏輯運算子，我們就可以寫有判斷能力的程式了。

條件運算式有幾種形式：

#### 單分支選擇結構

if 運算式:

語句區塊

當運算式為 True 的時候，就執行語句區塊。範例如下：

>>> x = input('Input two numbers: ')

Input two numbers: 12 31

>>> a, b = map(int, x.split())

>>> if a < b:

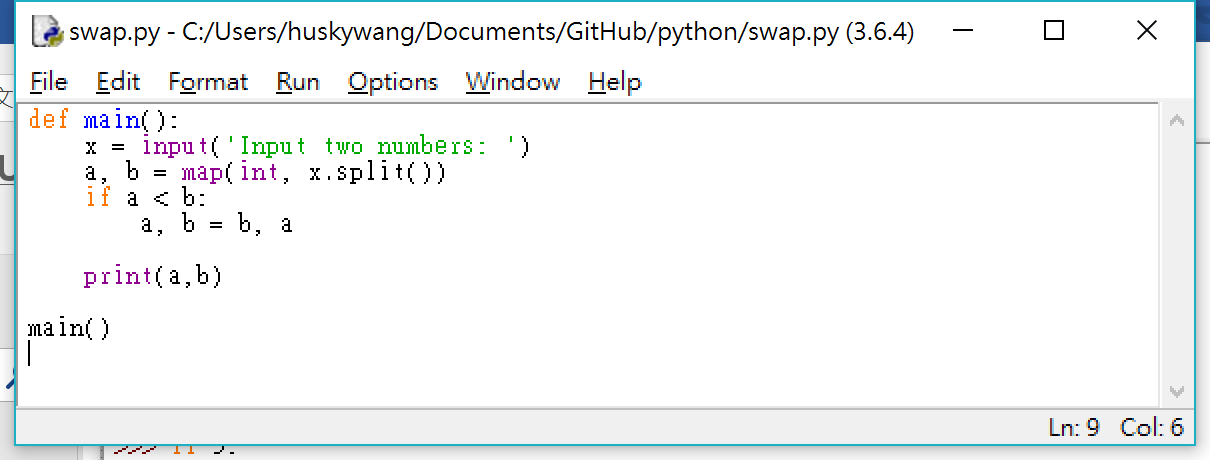
a, b = b, a

>>> print(a,b)

31 12

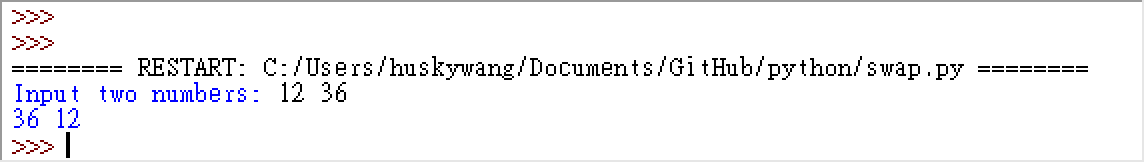
>>>

我們現在練習把上面在互動環境寫的程式碼，存成真正的程式檔，這樣我們才能清楚地學會程式的架構。



主程式名稱是 main()，冒號下面是整個 def main(): 的 block，注意縮排，縮排結束代表 def main(): 定義結束，然後最後面的 main() 是執行此程式。

同樣的 if a < b: 的下一行也要縮排，代表如果為 True 的話要執行的 block，最然後執行 print(a, b)。



隨堂練習

請完成上面的程式，並存成 swap.py 檔，且可以正確執行

在 Python 中，縮排很重要，同一個程式碼區塊(block)必須保證相同的內縮量，如果程式碼很短，也可以選擇在同一排，如下：

>>> if 3>2: print('ok')

ok

>>> if True: print(3); print(5)

3

5

>>>

#### 雙分支選擇結構

if 運算式:

語句區塊1

else:

語句區塊2

實機操作

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

>>> ch = list(\_\_\_(\_\_\_, num))

>>> ch

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

>>> if ch:

print(ch)

else:

print('empty')

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

>>>

Python 還提供三元運算子，格式為：

Value1 if condition else Value2

來幾個練習：

實機操作

>>> 'true' if False else 'false'

'\_\_\_\_\_\_'

>>> 'true' if True else 'false'

'\_\_\_\_\_\_'

>>> a = 5

>>> print(6) if a>3 else print(5)

\_\_\_

>>> print(6 if a>5 else 5)

\_\_\_

>>> b = 6 if a>13 else 9

>>> b

\_\_\_

>>>

#### 多分支選擇結構

if 運算式1:

語句區塊1

elif 運算式2:

語句區塊2

elif 運算式3:

語句區塊3

……

else:

語句區塊n

實機操作

>>> def func(score):

if score > 100:

return 'wrong score. must <= 100'

elif score >= 90:

return 'A'

elif score >= 80:

return 'B'

elif score >= 70:

return 'C'

elif score >= 60:

return 'D'

elif score >= 0:

return 'E'

else:

return 'wrong score. must > 0'

>>> func(120)

'\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_'

>>> func(99)

'\_\_\_\_\_\_\_\_\_\_\_\_'

>>> func(62)

'\_\_\_\_\_\_\_\_\_\_\_\_'

>>> func(49)

'\_\_\_\_\_\_\_\_\_\_\_\_'

>>>

#### 巢狀選擇結構

結構中還有結構，只要記得 Python 的縮排原則即可，有待後面實際寫程式的時候再說明。

作業

一、

撰寫程式 SayHello.py：

執行之後會詢問性別(male or female)，然後電腦回答：

先生您好，或是 小姐您好

二、

撰寫程式 Interview.py

這是一個工作面試程式，執行之後電腦會詢問：

1. 年紀
2. 畢業成績

然後按照年齡判斷，如果 <18 則輸出：抱歉，您未滿18歲

如果 > 30，則輸出：抱歉，您超出年齡限制

年齡沒問題之後再判斷畢業成績

如果 < 80 則輸出：抱歉，您的成績未達標

最後通過則輸出：恭喜您錄取了

### 迴圈結構

現在開始，我們進入程式語言最有趣的部份，專門用來處理重複性的動作。Python 主要有兩種形式的迴圈結構：for loop and while loop。

while 條件運算式:

循環區塊

[else:

else子句程式區塊]

和

for 取值 in 序列或反覆運算物件:

循環區塊

[else:

else子句程式區塊]

首先，我們來計算 1+2+3+…+100 的結果：

>>> s = 0

>>> for i in range(1,101):

s += i

else:

print(s)

5050  
>>>

相同的，我們可以使用 while 迴圈達成同樣的事情：

>>> s = i = 0

>>> while i <= 100:

s += i

i += 1

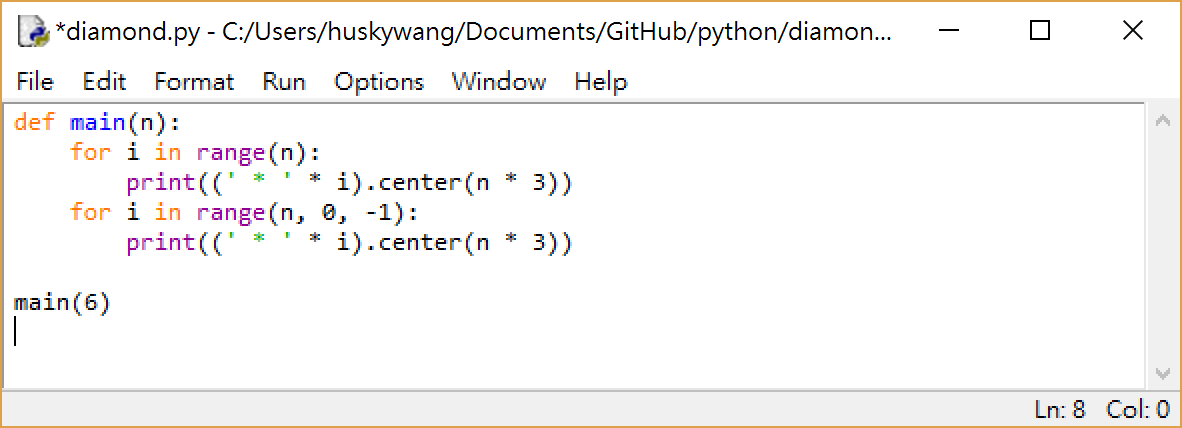
else:

print(s)

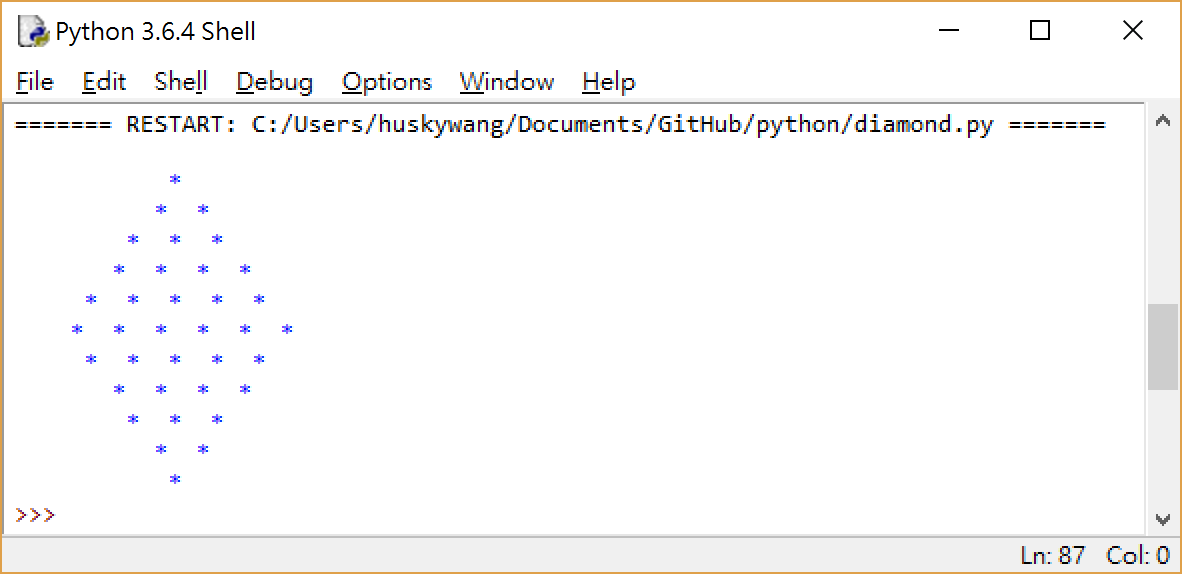
5050

>>>

程式設計時，一般優先考慮 for loop，接下來的程式巧妙的運用 range() 函數控制迴圈的次數，並輸出由 \* 組成的菱形圖案(請自行輸入並存為 diamond.py)：



結果如下：



作業

一、

使用 for loop 在螢幕上輸出：

1 2 3 4 5 6 7 8 9

>>>

二、

使用兩層 for loop 在螢幕上輸出九九乘法表的值：

1 2 3 4 5 6 7 8 9

2 4 6 8 10 12 14 16 18

3 6 9 12 15 18 21 24 27

…

…

…

9 18 27 36 45 54 63 72 81

>>>

三、

請完成一九九乘法表的程式 multiple.py

1x1=1 1x2=2 1x3=3 … 9x9=81

四、

請完成1加到 100 的程式 add100.py

輸出結果為：

1+2+3+4+…(全部要顯示出來)…+100=5050

#### break 和 continue 語句

while 和 for 迴圈都可以使用 break 和 continue 語句，一旦執行 break 語句，將使的迴圈提前結束。Continue 語句的作用是提前結束本次迴圈，進入下一次迴圈。

下面程式用來計算小於100 的最大質數：

>>> for n in range(100,1,-1):

for i in range(2,n):

if n%i == 0:

break

else:

print(n)

break

97

>>>

刪除最後一個 break 並略加修改之後，便能輸出100以內的所有值數：

>>> for n in range(100,1,-1):

for i in range(2,n):

if n%i == 0:

break

else:

print(n, end=' ')

97 89 83 79 73 71 67 61 59 53 47 43 41 37 31 29 23 19 17 13 11 7 5 3 2

>>>

隋棠練習

將上面的程式碼稍加修改成為程式，PrimeNumber.py

執行之後先問一個數目字，然後列出這個數字內的所有質數

>>>

下面的例子可以輸出列表中的元素：

>>> a\_list = ['a','b','cat','x','example']

>>> for i, v in enumerate(a\_list):

print('列表的第', i+1, '個元素是:', v)

列表的第 1 個元素是: a

列表的第 2 個元素是: b

列表的第 3 個元素是: cat

列表的第 4 個元素是: x

列表的第 5 個元素是: example

>>>

習題

完成程式: NarcissisticNumber.py

找出所有三位數的水仙花數

說明：在數論中，水仙花數（Narcissistic number），也被稱為超完全數字不變數（pluperfect digital invariant, PPDI）、自戀數、自冪數、阿姆斯壯數或阿姆斯特朗數（Armstrong number），用來描述一個N位非負整數，其各位數字的N次方和等於該數本身。

以三位數為例，153就是一個三位超完全數字不變數，其各個數之立方和等於該數：

153 = 13 + 53 + 33。

很多數學題目，也都可以用 for loop 來解題，人類在解題的時候，會使用代數，但是電腦的運算速度很快，可以用嘗試錯誤法逐個檢查。例如雞兔同籠的問題，假設共有雞、兔30隻，腳有90隻、求雞兔各多少隻?

>>> for chi in range(0,31):

if 2 \* chi + (30-chi) \* 4 == 90:

print('雞:',chi,' 兔:',30-chi)

雞: 15 兔: 15

>>>

### 函數的設計與使用

輸入 🡪 函數 🡪 輸出

這是函數的基本架構，對於輸入和輸出，是有彈性的，可以同時有輸入有輸出，可以只有輸入或只有輸出，也可以沒有輸入也沒有輸出，當然，這種函數很奇怪。更厲害的是，不但可以有多個輸入，還可以有多個輸出。

y = f(x)

y = f()

f(x)

y = f(a, b, c)

x, y = f(a, b, c)

#### 基本語法

當然，程式語言的寫法與數學不同，Python 的函數語法如下：

def 函數名稱([參數列表]):

函數本體

[return 變數列表]

我們用一個例子說明，如果我們要算兩場遊戲的平均分數，可以直接在主程式裡面寫 (game1 + game2) / 2，但是，如果這個算是會出現很多次，就可以考慮也成函數 y = average(a, b)，如下：

>>> def average(a, b):

avg = (a + b) / 2

return avg

>>> game1 = 80

>>> game2 = 90

>>> game\_avg = average(game1, game2)

>>> game\_avg

85.0

>>>

Python 跟其他程式語言如 C++、Java 最不同的地方，就是可以傳回兩個以上的值，延續上個例子，如果我們想一次傳回兩次遊戲的總分跟平均，只要改寫程式如下：

>>> def sum\_avg(a, b):

sm = a + b

ag = sm / 2

return sm, ag

>>> game\_sum, game\_avg = sum\_avg(game1, game2)

>>> game\_sum

170

>>> game\_avg

85.0

>>>

#### 預設值參數

“重要的事情說三遍”，如果我們要寫一個說n遍函數，可以設計如下：

>>> def say(message, times):

for i in range(times):

print(message, end = ' ')

print()

>>> say('Hello', 3)

Hello Hello Hello

>>>

不過，一般來說，我們通常只說一遍，說很多遍屬於特例，因此，我們可以把 times 的預設值設定成1，當只要說一遍的時候，就可以省去 times 這個參數，這樣做更有效率，程式如下：

>>> def say(message, times = 1):

for i in range(times):

print(message, end = ' ')

print()

>>> say('特別重要', 5)

特別重要 特別重要 特別重要 特別重要 特別重要

>>> say('您好')

您好

>>>

#### 可變長度參數

可變長度參數在定義的時候，主要有兩種方式， “\*parameter” 與 “\*\*parameter” ，前者用來定義多個參數，放在元組之中，後者是放在字典之中，我們這裡只談第一種。

回顧之前算兩次分數的總和與平均的函數 sum\_avg()，其實功能上有很大的限制，當我們玩了三場以上的遊戲的時候，就沒有辦法使用了，因為參數寫死的就只有兩個，如果我們希望不管輸入多少個參數，都可以算出總合與平均的話，就必須利用 \*parameter 來傳遞元組，再用 for I in parameter 來個別取出元組中的值，這樣就沒有個數的限制了，範例如下：

def sum\_avg(\*p):

s , n = 0, 0

for i in p:

s += i

n += 1

return s, s/n

>>> x, y= sum\_avg(1, 2, 3)

>>> x

6

>>> y

2.0

>>> x, y = sum\_avg(2, 4, 6, 8, 10)

>>> x

30

>>> y

6.0

>>> x, y = sum\_avg(80, 95, 74, 93)

>>> x

342

>>> y

85.5

>>>

#### Lambda 運算式

Python 走的是極簡風，Lambda 就是極簡風的代表，它用來宣告沒有函數名稱的小函數，看下面計算平均的例子便知：

>>> lambda x, y : (x+y)/2

<function <lambda> at 0x0000018E52BE51E0>

>>> g = lambda x, y : (x+y)/2

>>> g(10,20)

15.0

如果是初學程式語言的人，建議使用一般的函數，畢竟 Lambda 的可讀性不高。

習題

一、建立一個函數，計算圓面積

>>> from math import pi

>>> pi

3.141592653589793

>>> \_\_\_\_\_ CircleArea(r):

\_\_\_\_\_\_\_\_\_\_ pi \* r \* r

>>> CircleArea(5)

78.53981633974483

>>>

二、建立一個函數 circle2()，傳回圓的周長與面積

>>>

>>>

>>>

>>>

>>>

三、建立一函數 swap(a, b)，將 a, b 交換

>>>

>>>

>>>

>>>

>>>

四、建立一函數 my\_sort(x, y, .....) 可以接收 n 個參數，將參數按照大小順序排列，用氣泡排序法

>>>

>>>

>>>

>>>

>>>

>>>

>>>

>>>

>>>

>>>

>>>

五、將以上的函數包含在主程式 MyFunction.py 之中。

自行存檔

## Windows GUI 程式設計

在此之前，我們所寫的程式都是命令列的程式，透過 input()、print() 一行一行跟程式互動，這是一種古老的方式，現在已經進入 Windows 時代了，如果還用以前DOS的方式寫程式，有很多事情會無法達成，例如，你要寫一個計算機程式，請問如何用DOS模式來寫呢?

好消息是，Python 提供了一個好用的 GUI模組 tkinter，Python 的 IDLE 就是用 tkinter 製作的。

### Hello World! For Windows

按照慣例，我們又要寫一個 Hello World 程式，創建我們第一個 Windows 視窗：

開啟一個新的檔案，輸入下面的程式碼：

from tkinter import \*

my\_window = Tk()

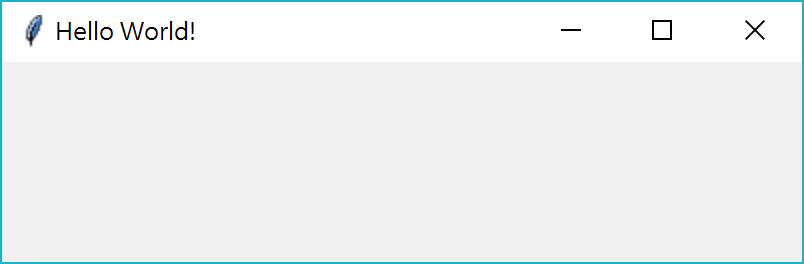
my\_window.title('Hello World!')

my\_window.geometry('400x100')

my\_window.mainloop()

這程式分成三個部分，首先一定要 import tkinter 的所有方法，第二步透過 Tk() 函數創建一個視窗物件 my\_window，然後定義其中的 title 與 geometry (寬 x 高)。第三步就是進入視窗程式的主迴圈。

空白視窗的執行結果如下(那根羽毛其實很像台灣)：



接下來，我們要將 tkinter 物件加入我們的空白視窗，tkinter 提供非常多的視窗物件，例如： Button、Label、Canvas、Menu、Entry 等等等，我們就從最基本的開始。

#### Label 標籤

Label 是用來顯示單列文字的物件，請接著修改前面的程式碼：

from tkinter import \*

my\_window = Tk()

my\_window.title('Hello World!')

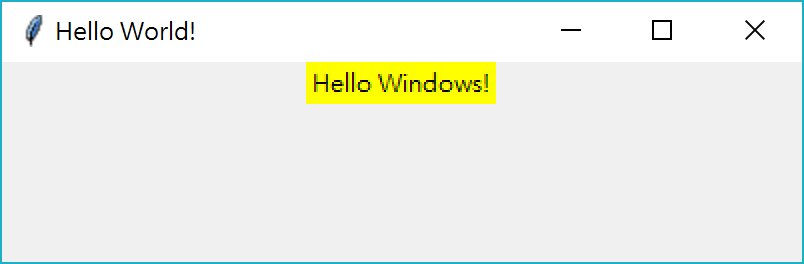
my\_window.geometry('400x100')

my\_label = Label(my\_window, text = 'Hello Windows!', bg = 'yellow')

my\_label.pack()

my\_window.mainloop()

要在視窗上面建置物件有兩個步驟，首先透過 Label() 創建物件並傳入必要的參數，然後使用 pack() 布局方式將此物件放置在視窗裡。pack() 是最基本的布局方式，就是由上到下依序放置。結果如下：



#### Entry 文字方塊

前面的 Label 可以用來輸出文字，但是如何讓使用者輸入文字呢？就必須使用 Entry，修改程式如下：

from tkinter import \*

my\_window = Tk()

my\_window.title('Hello World!')

my\_window.geometry('400x100')

my\_label = Label(my\_window, text = 'Hello Windows!', bg = 'yellow')

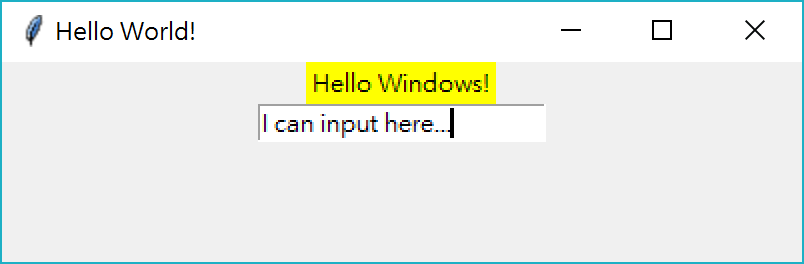
my\_label.pack()

my\_enter = Entry(my\_window, width = 20)

my\_enter.pack()

my\_window.mainloop()

結果如下：



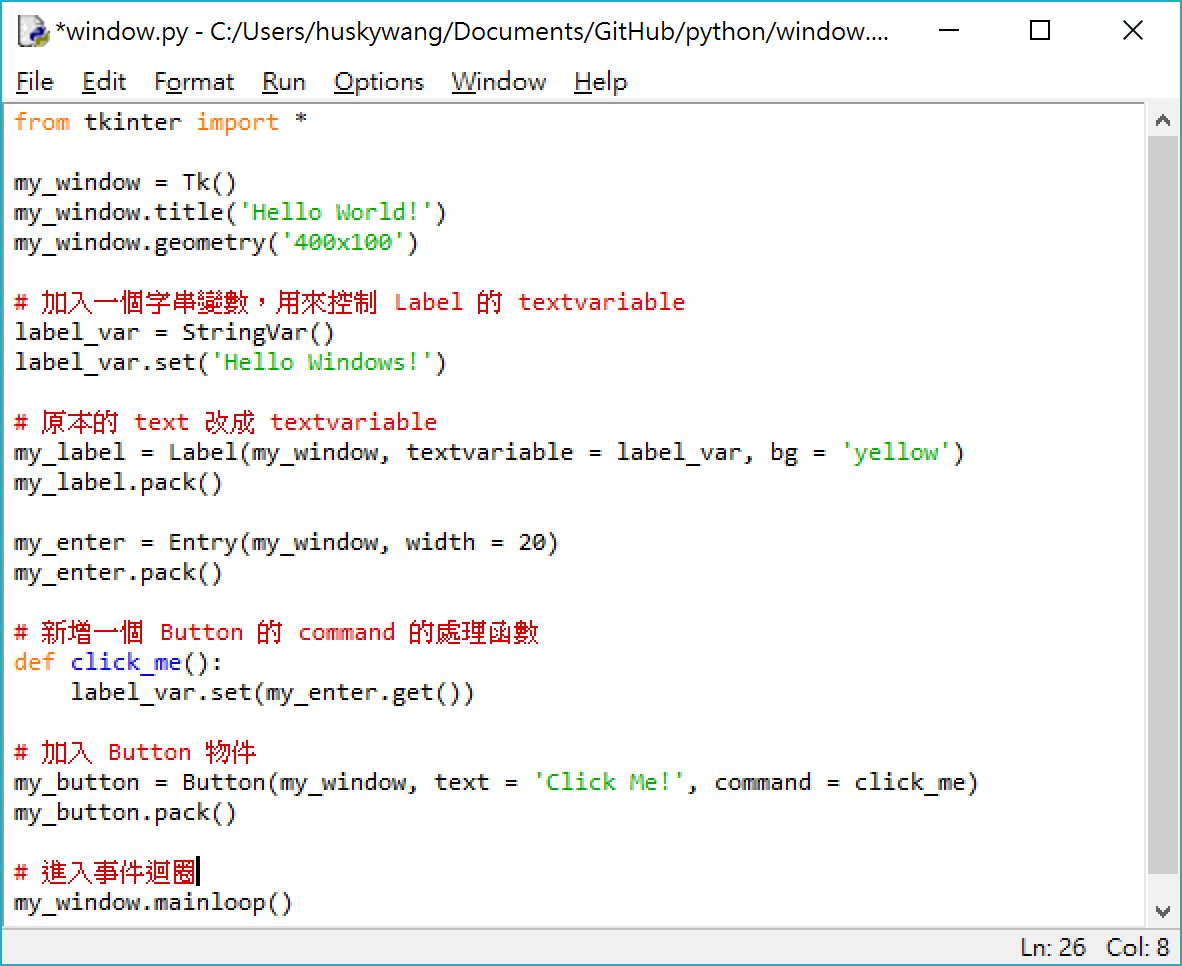
到目前為止，我們已經可以顯示字串，以及輸入字串，但是什麼事情都不能做，因為我們還沒有處理輸入完畢之後要執行的命令。這個命令通常會在 Button 裡面處理。所以接下來，我們來介紹最重要的部分，Button。

### 事件驅動程式設計

設計視窗應用程式，屬於事件驅動程式設計。還記得在前面的主程式裡面，有一個 mainloop()，這個函數就是讓我們的視窗進入一個等待事件的迴圈，當我們在視窗上面的某個物件上，例如按鈕，用滑鼠點了一下的時候，這個迴圈就會偵測到一個 mouse click 的事件，然後再按照我們替這個事件事先設計好的 command 來執行。

#### Button 按鈕

按鈕是最常用來接受事件的物件，我們現在將前面的視窗程式，設計成讓使用者在 Entry 裡面輸入文字，按下按鈕之後，就在 Label 中顯示所輸入的文字。程式碼如下：



裡面所有修改的部分，我都做了備註，方便閱讀，執行的結果如下：

