

Python入門

Yi-ting Chiang

中原大學應用數學系

February 21, 2023

使用Python進行資料分析



關於Python

- 一種script language
 - 不同平台只須提供python interpreter, 即可輕易執行相同程式碼
- 程式碼精簡
- 提供許多可應用於資料分析以及機器學習的工具

Python安裝與使用

要在自己的主機上安裝Python，可以到<https://www.python.org/>，根據自己的作業系統來選擇要下載的檔案。以微軟Windows為例¹，連上網站之後網頁往下拉，在下面Download的地方選Latest²：

 Get Started Whether you're new to programming or an experienced developer, it's easy to learn and use Python. Start with our Beginner's Guide	 Download Python source code and installers are available for download for all versions! Latest: Python 3.11.2	 Docs Documentation for Python's standard library, along with tutorials and guides, are available online. docs.python.org	 Jobs Looking for work or have a Python related position that you're trying to hire for? Our relaunched community-run job board is the place to go. jobs.python.org
--	--	---	--

¹Linux，FreeBSD等unix-like作業系統可用各自的套件管理程式來直接安裝

²目前(2023年2月)是Python 3.11.2

Python安裝與使用

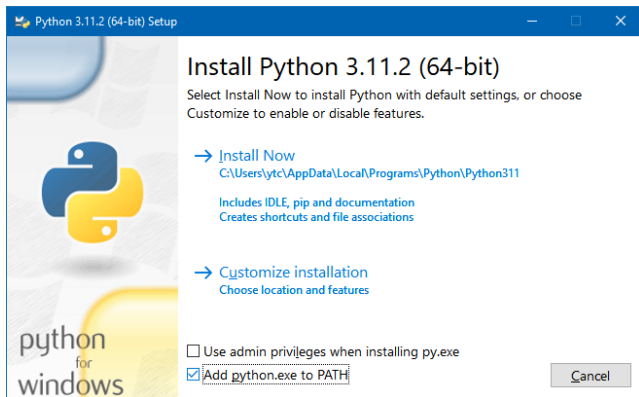
點選上頁圖中的Lastest Python的版本之後，連到的網頁往下拉，會看到如下圖的列表。根據自己的作業系統(32或64位元Windows)，下載“Windows installer”來安裝。

Files

Version	Operating System	Description	MD5 Sum	File Size	GPG	Sigstore
Gzipped source tarball	Source release		f6b5226ccb5ae1ca9376aaba0b0f673	26437858	SIG	CRT SIG
XZ compressed source tarball	Source release		a957cffb58a89303b62124896881950b	19893284	SIG	CRT SIG
macOS 64-bit universal2 installer	macOS	for macOS 10.9 and later	e038c3d5cee8c5210735a764d3f36f5a	42835777	SIG	CRT SIG
Windows embeddable package (32-bit)	Windows		64853e569d7cb0d154779300ff9c9b6	9574852	SIG	CRT SIG
Windows embeddable package (64-bit)	Windows		ae7de44ecbe2d3a37dbde3ce669d31b3	10560465	SIG	CRT SIG
Windows embeddable package (ARM64)	Windows		747090b80a52e8bbcb5cb65f78fee575	9780864	SIG	CRT SIG
Windows installer (32-bit)	Windows		2123016702bbb45688baedc3695852f4	24155760	SIG	CRT SIG
Windows installer (64-bit)	Windows	Recommended	4331ca54d9eacdbe6e97d6ea63526e57	25325400	SIG	CRT SIG
Windows installer (ARM64)	Windows	Experimental	040ab03501a65cc26bd340323bb1972e	24451768	SIG	CRT SIG

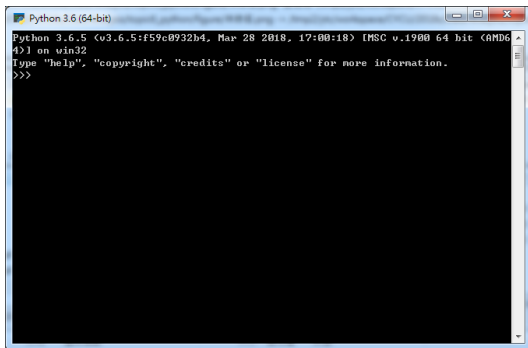
Python安裝與使用

執行安裝用的執行檔之後，在下面畫面建議把Add Python x.x to PATH勾選起來：



Python安裝與使用

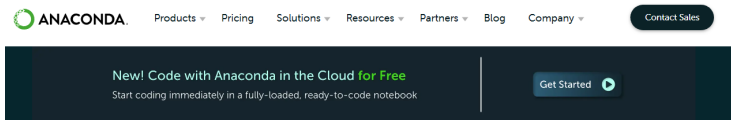
安裝完之後，就可以在“開始”那邊選擇Python來執行，看到下面的畫面，可以開始使用了：

A screenshot of a Windows command prompt window titled "Python 3.6 (64-bit)". The window has a black background with white text. The text inside the window reads: "Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 17:00:18) [MSC v.1900 64 bit (AMD64)] on win32", "Type 'help', 'copyright', 'credits' or 'license' for more information.", and ">>>". The window has standard Windows window controls (minimize, maximize, close) in the top right corner.

```
Python 3.6 (64-bit)
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 17:00:18) [MSC v.1900 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Python安裝與使用

python目前最大的優勢是有許多現有套件可以使用，因此除上述方式之外，可以使用Anaconda來一起安裝python以及常用套件



Data science technology for a better world.

Anaconda offers the easiest way to perform Python/R data science and machine learning on a single machine. Start working with thousands of open-source packages and libraries today.

Download 

For Windows

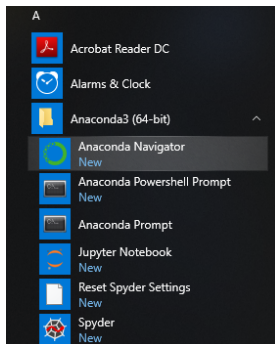
Python 3.9 • 64-Bit Graphical Installer • 621 MB

[Get Additional Installers](#)

在上面網頁中點選下載(Download)，網頁會自動跳出下載視窗。下載後安裝即可。

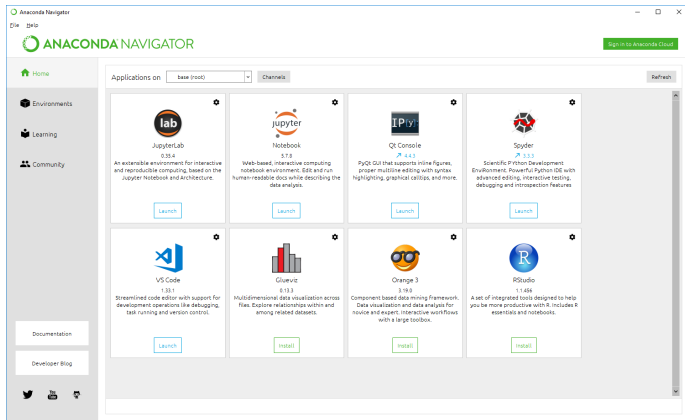
Python安裝與使用

點選32或64-bit安裝檔後安裝，一切照預設步驟，安裝成功後，在Windows的開始選單的Anaconda3資料夾下會看到有下面的幾個程式可以使用



可點選Anaconda prompt，會進入如前面安裝基本python環境的命令列畫面。也可以點選Anaconda Navigator來選擇Anaconda

Python安裝與使用



上圖為Anaconda Navigator的畫面。可以點選左方的Environment來管理(安裝/移除/升級)目前所安裝的套件。這個功能也可以在Anaconda prompt下用指令“conda”來執行。指令“conda install <package_name>”可根據所指定的套件名稱(package name)來安裝特定python套件。如“conda install numpy”，會安裝numpy套件。

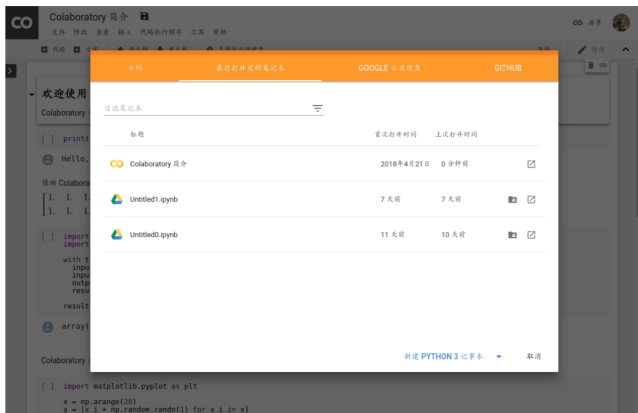
Python安裝與使用

安裝好python，或利用anaconda安裝好python以及其他常用套件之後，目前常見的編輯/執行python的方式如下：

- 使用一般文字編輯器編輯python原始程式碼, 在命令列下執行
- 使用Anaconda提供的spyder開發環境
- 使用Microsoft提供的汎用程式開發環境Visual Studio Code

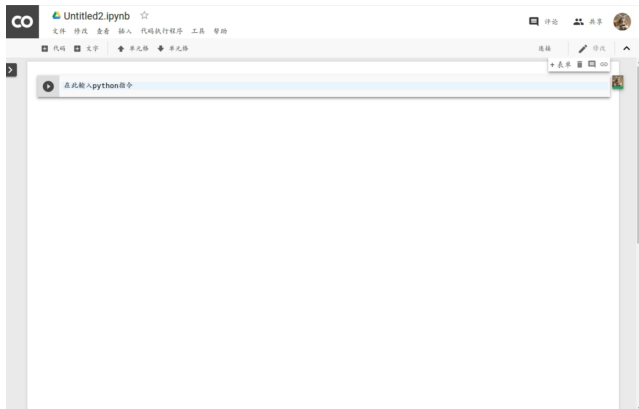
Python安裝與使用

若是無法在自己的電腦下建立python環境，也可以使用google提供的python線上環境Google Colaboratory。需要google帳號。開啓之後可以看到類似這樣的畫面：



Python安裝與使用

可以選擇一個之前存檔的ipynb檔，或是在選單上選擇文件⇒新建python3記事本(若要用python2就選python2記事本)，之後出現下圖的環境：



在上圖左方三角形的右邊“在此輸入python指令”的地方輸入python程式碼，按下三角形來執行

檢查python環境及重要套件版本

python允許在一個作業系統的一個使用者帳號中安裝多個不同版本。各版本之間套件不會共用。此外，python中有些套件還在發展中。有時會需要檢查所安裝的版本。可以在windows命令列之下執行下面指令，顯示你的python版本：

```
python --version
```

如前面投影片第6頁，可以看到進入python命令列時，會列出目前的python版本。而若是在python命令列下，可以用以下兩種方式：

```
>>> import platform
>>> platform.python_version()
'3.10.6'
>>> import sys
>>> sys.version
'3.10.6 (main, Nov 14 2022, 16:10:14) [GCC 11.3.0]'
```

上述的“import”指令是指定載入特定模組(可提供一些其他功能的工具)之意。

確認python程式執行時的工作目錄

所謂的工作目錄(working directory)是指目前你要執行程式，讀取資料，寫出資料時的來源或目標資料夾。在python中，可以用下面的指令來取得目前的工作目錄：

```
import os  
print(os.getcwd())
```

使用不同開發環境，則各有其他方式可以確認。

數值型態資料運算

下面先介紹Python的幾種資料型別。首先是數值型態資料。也就是整數以及浮點數。在Python中可以直接輸入數字得到數值資料，並使用下面幾種運算子(operator)對其進行運算：³

```
>>> 10+3 # 加法
13
>>> 10-3 # 減法
7
>>> 10*3 # 乘法
30
>>> 10/3 # 除法
3.3333333333333335
>>> 10//3 # 整數除法
3
>>> 10**3 # 次方
1000
>>> 10 % 3 # 取餘數
```

注意計算10/3的結果，它是因為浮點數的精確度問題所造成的

³在Python3下所得結果

數值型態資料運算

運算符號會有優先順序。常見的狀況就是一般所知的“先乘除後加減”。另外，次方運算優先權會高於乘法。如下例：

```
>>> 1+2*3
7
>>> 2*3+1
7
>>> 2+3*1
5
>>> 24/6*3
12.0
>>> 3*2**3    # 3*(2^3)
24
>>> 3**2*3    # (3^2)*3
27
```

python除這些運算子之外，還有其他運算子，各有其優先順序。詳情可以查閱python文件：<https://docs.python.org/3/reference/expressions.html#operator-precedence>

輸出

前述的範例都是在python的命令列環境下執行. 輸入一個四則運算後按下enter, 會把結果直接顯示在螢幕上. 但如果想指定印出某些資料, 就會需要用到print(). print()會把小括號內的東西處理之後, 印出結果. 如下例:

```
>>> a = 3
>>> b = 6
>>> print(a+b)
9
```

符號“=”功用是將左邊的結果設定給右邊的變數. 上面程式設定變數a的數值為3, b的數值為6. 然後要求print指令印a+b. 電腦會先計算a+b, 然後把結果9印出. 甚至你可以同時指定多個變數, 寫出如下的程式碼:

```
>>> a=2
>>> b=5
>>> a,b=b,a
>>> print(a,b)
5 2
```

輸出

前面提到過python是一種script language(描述語言). 與C語言等不同, 編寫完的程式可以直接在命令列下執行. 可以把前面的python範例, 用純文字編輯器輸入, 存為一個純文字檔後,



```
print.py - Notepad
File Edit Format View Help
a=3
b=6
print(a+b)
```

在命令列下呼叫python來執行此檔案

```
ytq@I-WLS:/mnt/e/CYCU/2020-2/Data Analysis/topic6_python/codes$ python print.py
9
ytq@I-WLS:/mnt/e/CYCU/2020-2/Data Analysis/topic6_python/codes$
```

數值型態資料

和C/C++/Java等程式語言不同，Python在使用變數之前不需要經過宣告，可以直接使用。只需要將其數值設定給該變數即可。例如下面的例子；

```
height = 20.5 # 把變數height設為20.5
width = 30    # 把變數width設為30
area= height * width # 把height和width相乘，結果給變數area
print(area)    # 印出area
```

上述程式碼會印出615.0

```
ytic@I-WLS:/mnt/e/CYCU/2020-2/Data Analysis/topic6_python/codes$ python print_area.py
615.0
ytic@I-WLS:/mnt/e/CYCU/2020-2/Data Analysis/topic6_python/codes$
```

注意上例井字號#在python中是標示出單行的註解。該行井字號後面的內容不會被視為程式碼的一部分。

數值型態資料

下面例子給定梯形的上，下底以及高，可以利用下面的python程式來計算面積：

```
height = 8.5 # 高為20.5
up_b = 10    # 上底10
down_b = 16  # 下底16
area= (up_b + down_b)*height/2
print(area)  # 印出area
```

括號的部份會先執行，因此結果會印出110.5。

```
yt@l-WLS:/mnt/e/CYCU/2020-2/Data Analysis/topic6_python/codes$ python print_area2.py
110.5
yt@l-WLS:/mnt/e/CYCU/2020-2/Data Analysis/topic6_python/codes$
```

如果寫成`up_b + down_b*height/2`，結果先乘除後加減，會變成78.0

數值型態資料

想知道資料是屬於哪種型態的型別，可以用“type()”來判定。如下例：⁴

```
>>> a=1
>>> b=0.7
>>> type(1)
<class 'int'>
>>> type(a)
<class 'int'>
>>> type(0.7)
<class 'float'>
>>> type(b)
<class 'float'>
>>> type(a+b)
<class 'float'>
>>> a+b
1.7
```

⁴此範例為python 3.7.4版下執行結果

數值型態資料

如果想強制轉換資料型別，可以用下面的語法，這運算稱為`cast`:

```
>>> int(1.5)
1
>>> float(3)
3.0
>>> float('7')
7.0
>>> float('7.6')
7.6
>>> float("8.1")
8.1
>>> int("2")
2
>>> int("2.1")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: invalid literal for int() with base 10: '2.1'
```

用單或雙引號標示起來的是“字串”。注意1.5轉換成整數(`int`)會無條件捨去小數部份。另外，帶有小數的字串(如上例的"2.1")不能轉換成十進位整數，會出現錯誤訊息。

數值型態資料

除了加減乘除等上面的運算之外，數值資料之間可以做比較關係的運算：

```
>>> a,b,c=3,10,3 #指定a=3,b=10,c=3
>>> a<b
True
>>> a>b
False
>>> a<=c
True
>>> a>=b
False
>>> a==c
True
>>> b==c
False
>>> a!=b
True
>>> a!=c
False
```

```
>>> b%a > c
False
>>> a<=c<b
True
>>> b>c>=a
True
>>> a < b and c < b
True
```

$a <= c < b$ 在Python中和 $a <= c$ and $c < b$ 同義。
另外，python推薦以類英語的語法撰寫程式，故你也可以用下面的語法：

```
>>> a is c
True
>>> a is b
False
>>> a is not b
True
>>> a is c and a is not b
True
```

is或/is not和==/!=在上面看來一樣，但其實有不同之處。在後面再對此進行解說。

字串型態資料

Python的字串可以用單引號或雙引號來標記。例如：

```
>>> course='Data Analysis'
>>> Today="Friday"
>>> course[0]
'D'
>>> course[1]
'a'
>>> course[0:2]
'Da'
>>> Today[1:4]
'rid'
```

注意上面的[a:b]區間指定法表示從位置a開始到位置b之前，不包括位置b。

字串型態資料

如果要字串包括單或雙引號，可以用下面語法：

```
>>> a="this is a test"
>>> b='"this is a test"'
>>> a
'this is a test'
>>> b
'"this is a test"'
>>> b[0]
','
```

或用“逃脫字元”\。如下例：

```
>>> b="\\"this is a test\\"
>>> b
'"this is a test"'
>>> b[0]
','
```

字串型態資料

另外在Python中可以用負號來指定“從後面開始的”第幾個元素。區間指定可以只指定開始位置，或是結束位置部份

```
>>> course='Data Analysis'
>>> Today="Friday"
>>> course[-1]
's'
>>> course[-2]
'i'
>>> course[:5]
'Data '
>>> Today[:3]
'Fri'
>>> Today[3:]
'day'
>>> course[:5]+Today[3:]
'Data day'
```

字串型態資料

如果指定的範圍超過界限會發生什麼事？在C/C++當中這種情形常常會造成程式被系統強制中止，或是出現不可預期的結果。Python中則是：

```
>>> Today[0:20]
'Friday'
>>> course[3:15]
'a Analysis'
>>> Today[15:20]
''
```

不過如果不是指定區間，就會出現錯誤：

```
>>> Today[20]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: string index out of range
```

字串型態資料

要計算字串長度的話，可以用`len()`：

```
>>> Today="Friday"
>>> course='Data Analysis'
>>> len(Today)
6
>>> len(course)
13
```

字串裡面可以包括一些特殊字元，例如

```
>>> course='Data\nAnalysis'
>>> Today="is\tFriday"
```

上面的`\n`是換行，`\t`是tab。嘗試印出上面的結果會得到：

```
>>> print(Today)
is      Friday
>>> print(course)
Data
Analysis
```

上面的`print()`是印出資料用的函數。

字串型態資料

前面示範過用`int()`和`float()`把字串轉數值。反過來要把數值轉字串，可以利用`str()`。如下例：

```
>>> num=1.234
>>> str(num)
'1.234'
>>> print("num="+str(num))
num=1.234
```

上面的`+`運算可以將兩個字串連成一個新的字串。如果沒有先將數值資料轉成字串就用`+`，會出現下面的錯誤訊息：

```
>>> print("num="+num)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: must be str, not float
```

另一個方式是指定“格式化字串”來傳給`print()`印出資料。如下例：

```
>>> str="abc"
>>> z=f"str={str}. Length of str={len(str)}"
>>> print(z)
str=abc. Length of str=3
```

字串前方的符號“`f`”表示後面的是“格式化”字串(**f-strings**)。格式化字串裡面的大括號可以放入變數或運算，大括號內的變數或運算會被自動轉換成字串形式，加入整個字串。

上例其實也可以直接把格式化字串扔給`print()`，不須經過變數`z`。

字串型態資料

Python的f-strings有一些方便功能。例如指定同時印出資料與變數名稱，保留的空間，多餘空間填0或其他符號，對齊方式，與輸出小數點後位數：

```
>>> Money=12345.6789
>>> print(f"{Money}")
12345.6789
>>> print(f"{Money=}")
Money=12345.6789
>>> print(f"{Money=:,}") #加上千位分隔符號
Money=12,345.6789
>>> print(f"{Money=:11}")
Money= 12345.6789
>>> print(f"{Money=:011}")
Money=012345.6789
>>> print(f"{Money=:<011}") #往左對齊
Money=12345.67890
>>> print(f"{Money=:>011}") #往右對齊
Money=012345.6789
>>> print(f"{Money=:011.2f}") #輸出2位小數
Money=00012345.68
>>> print(f"{Money=:<011.2f}") #往左對齊，輸出2位小數
Money=12345.68000
>>> print(f"{Money=:>011.2f}") #往右對齊，輸出2位小數
Money=00012345.68
>>> print(f"{Money=:#<11.2f}") #往左對齊填井字符號
Money=12345.68###
>>> print(f"{Money=:#>11.2f}") #往右對齊填井字符號
Money=###12345.68
```

注意

可以指定印出字串中的某一個或某範圍的字元，但不能指定修改字串裡面某一個或某範圍的字元。例如：

```
>>> course="Data Analysis"
>>> course[4]='#'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'str' object does not support item assignment
>>> course[5:13]="Generate"
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'str' object does not support item assignment
>>> course="Data Generate"
>>> print(course)
Data Generate
```

最後重新設定字串的部份，會讓舊的字串被整個刪除，以新的字串取代。

List型態資料

Python的list裡面可以存放多個不同的資料，例如：

```
>>> even=[2,4,6,8,10]
>>> even
[2, 4, 6, 8, 10]
>>> len(even) # 取得list內的元素數量
5
>>> even[3]
8
>>> even[-1]
10
>>> even[1:3]
[4, 6]
>>> even[1:10]
[4, 6, 8, 10]
>>> even[5]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: list index out of range
```

從上面的例子可以看到，list也可以用指定範圍以及指定特定位置的方式來存取元素。

超出範圍時其表現跟字串一樣。不過不像字串，list可以指定修改裡面的某個元素。

List型態資料

List內可以同時放進多種不同型別的資料。例如：

```
lst=[5, 'pi', 3.1415926]
>>> len(lst)
3
>>> lst[:]
[3, 'pi', 3.1415926]
>>> lst[1]
'pi'
>>> lst[-1]
3.1415926
```

另外，和python的字串不同，list可以直接指定修改裡面的某個元素：

```
>>> lst[1]='pa'
>>> lst
[5, 'pa', 3.1415926]
```

List型態資料

List裡面也可以放入一個list. 此手法可用來形成一個多維陣列:

```
>>> M=[[1,2,3],[4,5,6],[7,8,9]]
>>> M
[[1, 2, 3], [4, 5, 6], [7, 8, 9]]
>>> M[0]
[1, 2, 3]
>>> M[-1]
[7, 8, 9]
>>> M[1]
[4, 5, 6]
>>> M[1][2]
6
>>> M2=[M,'Matrix!']
>>> M2
[[[1, 2, 3], [4, 5, 6], [7, 8, 9]], 'Matrix!']
```

對list做+運算的結果是合併兩個list:

```
>>> [2,4,6,8,10]+[1,3,5,7,9]
[2, 4, 6, 8, 10, 1, 3, 5, 7, 9]
>>> even=[2,4,6,8,10]
>>> odd=[1,3,5,7,9]
>>> nums=even+odd
>>> nums
[2, 4, 6, 8, 10, 1, 3, 5, 7, 9]
```

List型態資料

python中的list提供許多方便的功能，例如可以用count()計算某個list中的某元素有多少個：

```
>>> M=[1,3,5,7,9,2,3,4,5,5]
>>> M.count(1)
1
>>> M.count(3)
2
>>> M.count(5)
3
```

用append()在一個list後面加入元素：

```
>>> M=[1,3,5,7,9]
>>> M
[1, 3, 5, 7, 9]
>>> M.append("add")
>>> M
[1, 3, 5, 7, 9, 'add']
```

用del指令指定刪除list中的特定元素：

```
>>> M=[1,3,5,7,9]
>>> M[3]
7
>>> del M[3]
>>> M
[1, 3, 5, 9]
```

List型態資料

還可以用`remove`, `pop`從list中移除資料，或用`clear`清除整個list:

```
>>> M=[1,3,5,7,9,2,3,4,5,5]
>>> M.remove(5)
>>> M
[1, 3, 7, 9, 2, 3, 4, 5, 5]
>>> c=M.pop()
>>> c
5
>>> M
[1, 3, 7, 9, 2, 3, 4, 5]
>>> M.pop()
5
>>> M.pop()
4
>>> M
[1, 3, 7, 9, 2, 3]
>>> M.pop(2)
7
>>> M
[1, 3, 9, 2, 3]
>>> M.clear()
>>> M
[]
```

`remove(e)`會移除list中資料等於e的元素。如果有
多個，會移除索引值最小(最左邊的)。
`pop()`不指定索引會傳回並移除list最尾端的元
素，也可指定索引值來移除特定元素。

List型態資料

可以在list型態資料上做許多運算：

```
>>> M=[1,3,5,7,9,2,3,4,5,5]
>>> max(M) # 找出最大值
9
>>> min(M) # 找出最小值
1
>>> sum(M) # 計算總和
44
>>> M.sort() # 排序
>>> M
[1, 2, 3, 3, 4, 5, 5, 5, 7, 9]
>>> M.sort(reverse=True)
>>> M
[9, 7, 5, 5, 5, 4, 3, 3, 2, 1]
```

```
>>> MS=["AxC", "AXD", "ABD", "Fgh", "FghA"]
>>> max(MS)
'FghA'
>>> min(MS)
'ABD'
>>> MS.sort()
>>> MS
['ABD', 'AXD', 'AxC', 'Fgh', 'FghA']
>>> MS.sort(reverse=True)
>>> MS
['FghA', 'Fgh', 'AxC', 'AXD', 'ABD']
```

上例左方示範對內容全為數值的list做各種運算。右方則是對字串型態的來做。從右方可看出python對於字串大小比較，順序是“字典序”，且大寫字母比較“小”。

Dictionary型態資料

Python中的Dictionary型別的資料是由多組(key:value)的配對所形成的。可以定義這樣的資料型別，根據key來存取value：

```
>>> food={'name': 'pizza', 'quantity': 1, 'prize': 400}
>>> food
{'quantity': 1, 'name': 'pizza', 'prize': 400}
>>> food['name']
'pizza'
>>> food['quantity']
1
>>> food['quantity']+=3 # 可以直接修改某項的內容
>>> food
{'quantity': 4, 'name': 'pizza', 'prize': 400}
>>> food['shape']='circle' # 可以增加新的key:value資料
>>> food
{'shape': 'circle', 'quantity': 4, 'name': 'pizza', 'prize': 400}
```

注意上例中印出food內部元素的順序和定義food時不同。

(續上頁)

```
>>> food.items()
dict_items([('shape', 'circle'), ('quantity', 4), ('name', 'pizza'),
('prize', 400)])
>>> food.keys()
dict_keys(['shape', 'quantity', 'name', 'prize'])
>>> food.values()
dict_values(['circle', 4, 'pizza', 400])
>>> list(food.keys())
['shape', 'quantity', 'name', 'prize'] # 將key轉到一個list當中
```


下面是在Dictionary中紀錄更複雜的value欄位資料：

```
character = {'name': {'first': 'John', 'last': 'Smith'},
... 'class': 'Hero',
... 'spell': ['Blaze', 'Heal'],
... 'level': 5}
>>> print(character)
{'class': 'Hero', 'spell': ['Blaze', 'Heal'],
'name': {'first': 'John', 'last': 'Smith'}, 'level': 5}
>>> character['level']
5
>>> character['level']+=1
>>> character['spell'].append('Expel') # 在key 'spell'對應的list value中增加一項
>>> print(character)
{'class': 'Hero', 'spell': ['Blaze', 'Heal', 'Expel'],
'name': {'first': 'John', 'last': 'Smith'}, 'level': 6}
```

內建文件

操作某些資料型別的時候，如果有疑問，除了可以用網路來查之外，也可以用`help()`指令來查詢特定資料型別支援的內建方法，例如對一個浮點數`f`下`help(f)`指令，會拿到下面的訊息：

```
Help on float object:
class float(object)
|   float(x) -> floating point number
|
|   Convert a string or number to a floating point number, if possible.
|
|   Methods defined here:
|
|   __abs__(self, /)
|       abs(self)
|   .....(中略)
|   -----
|   Data descriptors defined here:
|
|   imag
|       the imaginary part of a complex number
|
|   real
|       the real part of a complex number
```

流程控制

程式語言最常用的流程控制語法if在Python裡面是下面的形式：

```
if <condition 1>:
    statement 11
    statement 12
    ...
elif <condition 2>:
    statement 21
    statement 22
    ...
elif <condition 3>:
    statement 31
    statement 32
    ...
else:
    statement 41
    statement 42
    ...
```

注意各if/elif/else後面的冒號“:”。此外，不像C/C++或Java等程式語言不會強制在程式碼前縮排，寫Python時要注意縮排程式碼來區分各程式碼屬於哪個區段。

流程控制

以下面的程式為例：

```
a=5
b=3
if a%b is 0:
    print("b divides a!")
else:
    print("b cannot divide a!")
```

輸出會是“b cannot divide a!”。但如果寫成

```
a=5
b=3
if a%b is 0:
    print("b divides a!")
else:
    print("b cannot divide a!")
```

會出現下面訊息：

```
File "./cond_err1.py", line 4
    print("b cannot divide a!")
    ^
IndentationError: expected an indented block
```

流程控制

如果是多層的if-elif敘述，每個statement要根據所屬的if或elif等敘述來縮排：

```
if <condition 1>:
    print("in condition1")
    if <condition2>:
        print("in condition2 under condition1")
    elif:
        print("still in condition2")
    print("in condition1")
print('not in any if conditions')
```

另外，也有將檢查式放在後面的語法：

```
>>> a=5
>>> b=3
>>> c = "b divides a!" if a%b==0 else "b cannot divide a!"
>>> print(c)
b cannot divides a!
>>> a=6
>>> c = "b divides a!" if a%b==0 else "b cannot divide a!"
>>> print(c)
b divides a!
```

流程控制

下面例子以if-else檢查某個輸入年份是否為閏年:

```
year = 2100

if year % 400 is 0:
    print(f"{year} is leap year")
elif year % 4 is 0:
    if year % 100 is 0:
        print("{} is not leap year".format(year))
    else:
        print("%d is leap year"%year)
else:
    print(str(year)+" is not leap year")
```

由於2100不可被400整除, 而可被4整除, 但也可被100整除, 故結果會印出它不是閏年.

注意此範例的幾種不同印出字串格式的語法. 對此部份有興趣的同學可自行參照本課程參考書籍.

多個條件可以用邏輯運算and, or, not來連在一起. 如下例:

```
year = 2100

if year % 400 is 0:
    print(f"{year} is leap year")
elif year % 4 is 0 and year % 100 is not 0:
    print("{} is leap year".format(year))
else:
    print(str(year)+" is not leap year")
```

注意elif後面的部份. 此例以and串接原本分開檢查的兩個部份. 讓程式比較簡潔易懂.

流程控制

串接多個邏輯運算時要注意計算優先順序, 下面兩例結果一模一樣:

```
year = 2100

if year % 400 is 0 or year % 4 is 0 and year % 100 is not 0:
    print("{} is leap year".format(year))
else:
    print(str(year)+" is not leap year")
```

```
year = 2100

if year % 400 is not 0 and (year % 4 is not 0 or year % 100 is 0):
    print("{} is not leap year".format(year))
else:
    print(str(year)+" is leap year")
```

要注意第二例後面的括號不可省略. 因為`and`的優先順序比`or`高. 如果省略, 條件就會變成“`year`必須不為400的倍數且不是4的倍數, 或者`year`是100的變數”. 結果2000年因為是100的變數, 滿足後段, 會變成非閏年.

迴圈

Python裡面有while和for兩種迴圈。while的語法如下：

```
while <condition>:  
    statement 1  
    statement 2  
    ...
```

在while裡面，當condition為真時，會持續反覆執行statements，直到condition為假才會跳出迴圈。

可以利用break敘述不經while的條件檢查就跳出迴圈。範例如下：

```
while <condition 1>:  
    statement 1  
    if <condition 2>:  
        break;  
    ...
```

上例中，若是condition 2為真，if裡面的break就會被執行，迴圈就會停止。

迴圈

可以進一步在while外面加上else。這時如果是因condition 1為假而離開迴圈，而非經由condition 2到達break離開的話，就會執行else下面的敘述：

```
while <condition 1>:  
    statement 1  
    if <condition 2>:  
        break;  
    ...  
else:  
    statements
```

舉例來說，下面的程式碼檢查y是否為質數，如果不是質數就印出最大的因數，否則印出0：

```
y=95  
x=y//2 # 整數除法, 先從y的一半開始  
while x>1:  
    if y%x == 0:  
        print(x)  
        break  
    x=x-1 # 測試下一個x  
else:  
    print(0)
```

上例會印出最大的因數19。如果把y改成某個質數，就會執行else部份的程式，印出0。

迴圈

Python裡面的for迴圈常用在反覆地從例如list或字串等序列資料取出元素來處理。語法是：

```
for <target> in <object>:  
    statements
```

例如：

```
data=[121,35,58,34,87,60,47,281]  
data_sum=0  
for x in data:  
    data_sum+=x  
print(data_sum/len(data))
```

上面範例會將內部都是數值資料的list “data”中所有的數值一個一個取出，並加到變數data_sum中。迴圈結束時data_sum的數值就會是整個list中的所有數值總和。最後印出數值總和除以這個list的長度。結果就是這個list內所有數值的平均。此範例結果會輸出90.375

迴圈

```
S="God's in his heaven;"
L=[]
for w in S:
    L.append(w)
print(L)
```

上例會將字串S拆成一個一個字元放入list L當中，因此輸出是：

```
['G', 'o', 'd', "'", 's', ' ', 'i', 'n', ' ', 'h', 'e', 'a', 'v', 'e', 'n', ';']
```

如果是每個單位元素是一個字串的list，如下例

```
S=[" All 's", " right", " with", " the", " world."]
L=[]
for w in S:
    L.append(w)
print(L)
```

則輸出是

```
["All's", 'right', 'with', 'the', 'world.']
```

迴圈

下面是兩層for迴圈的範例。此程式在P中尋找是否有和Q一樣的字串，根據結果印出不同訊息：

```
P=["Jenson","Lia","Smith","Kalafina"]
Q=["Smith","ClariS"]
for x in Q:
    for y in P:
        if y is x:
            print(x+" is found in P!")
            break
    else:
        print(x+" is not found in P")
```

結果會印出下面兩行：

```
Smith is found in P!
ClariS is not found in P
```

迴圈

可以用for迴圈反覆提取list內資料，如下例：

```
>>> M=[1,10,100,1000]
>>> for i in M:
...     print(i)
...
1
10
100
1000
```

結果會依序把list M當中的元素一個一個印出來. 相同方法也可用在dictionary上：

```
>>> food={'name': 'pizza', 'quantity': 1, 'prize': 400}
>>> for i in food:
...     print(i,food[i])
...
quantity 1
name pizza
prize 400
>>> for key, item in food.items():
...     print(key,item)
...
quantity 1
name pizza
prize 400
```

迴圈

可以在list裡面內包for迴圈，來產生新的list：

```
>>> M
[[1, 2, 3], [4, 5, 6], [7, 8, 9]]
>>> col0=[r[0] for r in M]
>>> col0
[1, 4, 7]
>>> col_plus=[r[1]+1 for r in M]
>>> col_plus
[3, 6, 9]
>>> M_even=[M[i][j] for i in [0,1,2] for j in [0,1,2] if M[i][j]%2 == 0]
>>> M_even
[2, 4, 6, 8]
>>> M_trans=[[r[i] for r in M] for i in [0,1,2]]
>>> M_trans
[[1, 4, 7], [2, 5, 8], [3, 6, 9]]
```

也可以利用類似語法來建立Dictionary:

```
>>> v=[2,3,5,7,11,13]
>>> x_square={x: x**2 for x in v}
>>> x_square
{2: 4, 3: 9, 5: 25, 7: 49, 11: 121, 13: 169}
```

迴圈

利用內包迴圈的語法，可以用來“過濾”資料。如下例：

```
data=[124,35,58,34,87,60,47,281]
new_data=[x for x in data if x < 100]
print(new_data)
```

結果會從data中挑出小於100的放入新的list中。輸出：

```
[35, 58, 34, 87, 60, 47]
```

如果是多維度的list，可根據同一個row其他的column來做類似的事情。如下例：

```
data=[[20,"a"],[15,"b"],[5,"b"],[13,"b"],[87,"a"]]
new_data=[x[0] for x in data if x[1] is "b"]
print(new_data)
```

此例的list內的元素都是一個長度為2的list。程式只挑出每個元素中第1個資料是“b”的list中的第0個元素，放入新的list中。故輸出是

```
[15, 5, 13]
```


另外，Python有一個函數`range()`可以用來產生一個區間的整數，其語法如下：

```
range(x)
```

會產生 $0, 1, \dots, x-1$ 的整數。如果寫

```
range(x, y)
```

就會產生 $x, x+1, \dots, y-1$ 的整數。如果是寫

```
range(x, y, step)
```

產生的是 $x, x+step, x+2*step, \dots, x+n*step$ ，其中 $x+n*step < y$

可以用`range()`來產生一串等差數列放入`list`中：

```
>>> L=list(range(3))  
>>> print(L)  
[0, 1, 2]
```

迴圈

下面範例程式展示利用for來反覆地取出函數range()產生的數值的方法：

```
for x in range(5):  
    print(x)  
print("")
```

```
for x in range(2,5):  
    print(x)  
print("")
```

```
for x in range(0,5,2):  
    print(x)
```

左方程式輸出為：

0
1
2
3
4

2
3
4

0
2
4

關於比較運算

在流程控制最後，我們說明一下比較運算。流程控制經常需要比較兩變數相同或不同。這會用到前面介紹過`is`和`is not`以及`==`和`!=`運算。它們在許多地方可以通用，但事實上這兩種運算有所不同。見下例：

```
>>> a = 24
>>> b = 24
>>> if a is b:
...     print(1)
... else:
...     print(0)
...
1
>>> if a == b:
...     print(1)
... else:
...     print(0)
...
1
```

```
>>> a = [1,2]
>>> b = [1,2]
>>> if a is b:
...     print(1)
... else:
...     print(0)
...
0
>>> if a == b:
...     print(1)
... else:
...     print(0)
...
1
```

`is`或`is not`是在比較兩個變數是否為相同物件，`==`和`!=`是比較兩個變數是否相等。`python`為了加速，在`python`命令列環境下，在設定變數為某些相同整數⁵時，會讓這些變數指向記憶體中相同位置的整數物件。如此做比較時速度較快。故會有上面的差別出現。

⁵-5到256之間的整數

自訂函數

目前為止看到的範例都是從頭作到尾的一條龍程式碼。不過和大多數的程式語言一樣，Python可以自己定義函數，把一些會重複呼叫使用的小功能寫成函數，方便程式維護與除錯。其語法如下：

```
def function_name(arg1,arg2,...,argK):  
    statement 1  
    statement 2  
    ...  
    statement N  
    return <data>
```

上面的`arg1,...,argK`是傳入的參數。參數數量也可以是0個，也就是沒有任何輸入參數；下面的`return`後面跟著此函數的傳回資料。

要注意的細節和前面流程控制的`if`及`while`等一樣：函式名稱後面要有冒號，而且`statements`前面要縮排。

自訂函數

下面的程式會印出費式數列：

```
def fib(n):      # 函數定義
    a, b = 1, 1
    for i in range(n):
        print(a, end=" ") # 印出之後結尾不要換行，而是加上空白
        a, b = b, a+b
```

如果呼叫`fib(12)`，執行結果是

```
1 1 2 3 5 8 13 21 34 55 89 144
```

自訂函數

如果只要費氏數列的第 n 項，程式可以改寫成下面的形式：

```
def fib2(n):    # 函數定義
    a, b = 1, 1
    for i in range(n-1): # 1,2,...n-1. 共n次
        a, b = b, a+b
    return a

a=100
b=fib2(12)      # 在此呼叫上面定義的fib函數
print("b="+str(b),"a="+str(a))
```

上面程式碼，函數fib2()執行完之後會把結果傳回來。在程式外面用一個變數b來記住函數傳回值，並印出來。

注意

函數fib2()裡面的a,b，和函數外的a,b無關。也就是說，在裡面修改的a,b和外面的a,b是不同的，裡面的a的數值改變並不會影響到外面的a。因此上面程式的輸出會是

```
b=144 a=100
```

自訂函數

下面是用list回傳整個費式數列 F_1, F_2, \dots, F_n 的方法：

```
def fib3(n):      # 函數定義
    a, b = 1, 1
    L=[]
    for i in range(n-1): # 1,2,...n-1. 共n次
        L.append(a) # 將 $F_i$ 放到List尾端
        a, b = b, a+b
    return L

L=fib3(12)
print(L)
```

呼叫fib3(12)並印出的話，結果會拿到：

```
[1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144]
```

自訂函數

函數內可以設定預設的變數數值。如下例：

```
def fib4(n, ErrMsg="ERROR!"):      # 函數定義
    a, b = 1, 1
    L=[]
    if n < 0:
        return ErrMsg
    for i in range(n): # 0,1,2,...n-1. 共n次
        L.append(a) # 將 $F_i$ 放到List尾端
        a, b = b, a+b
    return L

fiblist=fib4(-12)      # 在此呼叫上面定義的fib函數
print(fiblist)
```

如果沒有指定該變數數值，就會用預設數值，因此上面的程式會輸出“ERROR!”。

注意，雖然上面的fib_msg函數有兩個輸入參數，但呼叫時可以只給一個參數。

自訂函數

呼叫的同時指定某變數名稱並設值，若是該變數有預設值，就會被改變。如下例：

```
def fib_msg(n, ErrMsg="ERROR!"):    # 函數定義
    a, b = 1, 1
    L=[]
    if n < 0:
        return ErrMsg
    for i in range(n): # 0,1,2,...n-1. 共n次
        L.append(a) # 將 $F_i$ 放到List尾端
        a, b = b, a+b
    return L

print(fib_msg(ErrMsg="No Negative Values!",n=10))
print(fib_msg(ErrMsg="No Negative Values!",n=-10))
```

上面第一個print會輸出[1, 1, 2, 3, 5, 8, 13, 21, 34, 55]，然後第二個print輸出“No Negative Values!”

注意，如果有指定各變數名稱，輸入的順序可以不按照函式定義所給的順序來排列。

自訂函數

注意

某些Python的資料型別，例如list或Dictionary，如果在函數中設定預設值，之後重複呼叫的話，其數值會維持上一次被呼叫的時候的數值。

如下例：

```
def f(x, s=[]):  
    s.append(x)  
    print(s)
```

```
f(1)  
f(2)  
f(4)
```

上面程式輸出是：

```
[1]  
[1, 2]  
[1, 2, 4]
```

這種型態的資料在Python中稱為mutable object。這種資料設定預設值的話，其預設值只會被設定一次，第二次呼叫時不會重新設定，因此會留下之前的資料。

自訂函數

避免上述情況發生的方式是，避免讓參數的預設值設為mutable object. 如下例:

```
def f(x, s=None):  
    if s is None:  
        s=[]  
    s.append(x)  
    print(s)
```

```
f(1)  
f(2)  
f(4)
```

上面程式輸出就會變成：

```
[1]  
[2]  
[4]
```

自訂函數

數值資料，字串等屬於immutable object，因此結果會完全不同：

```
def f(x, s=""):  
    s+=x  
    print(s)  
  
f("1")  
f("2")  
f("4")
```

上面程式輸出是：

```
1  
2  
4
```

自訂函數

另一個要注意的地方是，python規定函式的定義中，若混合有給預設值和沒給預設值的參數，沒預設值的參數必須列在有預設值的參數前面。否則會出現語法錯誤：

```
>>> def fun(a="123",b):  
...     print(a)  
...     print(b)  
...  
File "<stdin>", line 1  
SyntaxError: non-default argument follows default argument
```

導入模組

在Python中可以將一些已經寫好的函數存檔，並且在之後導入。如之前的費式數列：

```
def fib_print(n):      # 函數定義
    a, b = 1, 1
    for i in range(n): # i=0,1,2,...n-1. 共n個
        print(a, end=" ") # 印出之後結尾不要換行，而是加上空白
        a, b = b, a+b

def fib_list(n):       # 函數定義
    a, b = 1, 1
    L=[]
    for i in range(n): # i=0,1,2,...n-1. 共n個
        L.append(a) # 將 $F_i$ 放到List尾端
        a, b = b, a+b
    return L
```

將其存檔為“call_fib.py”之後，可以用下面的語法導入並呼叫，得到結果如下：

```
>>> import call_fib
>>> call_fib.fib_print(10)
1 1 2 3 5 8 13 21 34 55 >>>
```

因為fib_print印出時沒有換行，因此會產生上面>>>在行尾的結果。

導入模組

```
def fib_print(n):      # 函數定義
    a, b = 1, 1
    for i in range(n): # i=0,1,2,...n-1. 共n個
        print(a, end=" ") # 印出之後結尾不要換行，而是加上空白
        a, b = b, a+b

def fib_list(n):       # 函數定義
    a, b = 1, 1
    L=[]
    for i in range(n): # i=0,1,2,...n-1. 共n個
        L.append(a) # 將 $F_i$ 放到List尾端
        a, b = b, a+b
    return L
```

可以用下面指令在導入時指定另一個名字給模組名稱：

```
>>> import call_fib as cf
```

這樣呼叫時就可以用指定的名稱取代原模組名稱：

```
>>> cf.fib_print(10)
1 1 2 3 5 8 13 21 34 55 >>>
>>> print(cf.fib_list(10))
[1, 1, 2, 3, 5, 8, 13, 21, 34, 55]
```

導入模組

除了整個模組都導入，也可以指定只導入某些函數。這樣可以不需要指定模組名稱：

```
def fib_print(n):      # 函數定義
    a, b = 1, 1
    for i in range(n): # i=0,1,2,...n-1. 共n個
        print(a, end=" ") # 印出之後結尾不要換行，而是加上空白
        a, b = b, a+b

def fib_list(n):       # 函數定義
    a, b = 1, 1
    L=[]
    for i in range(n): # i=0,1,2,...n-1. 共n個
        L.append(a) # 將Fi放到List尾端
        a, b = b, a+b
    return L
```

```
>>> from call_fib import fib_list
>>> print(fib_list(10))
[1, 1, 2, 3, 5, 8, 13, 21, 34, 55]
>>> fib_print(10)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'fib_print' is not defined
```

因為只有指定導入fib_list，因此呼叫fib_print會出現錯誤訊息。

導入模組

因此，也可以用上面的`as`語法把導入的函數改名：

```
def fib_print(n):      # 函數定義
    a, b = 1, 1
    for i in range(n): # i=0,1,2,...n-1. 共n個
        print(a, end=" ") # 印出之後結尾不要換行，而是加上空白
        a, b = b, a+b

def fib_list(n):       # 函數定義
    a, b = 1, 1
    L=[]
    for i in range(n): # i=0,1,2,...n-1. 共n個
        L.append(a) # 將 $F_i$ 放到List尾端
        a, b = b, a+b
    return L
```

如下例。將導入的模組名稱，可以呼叫

```
>>> from call_fib import fib_list as fl
>>> print(fl(10))
[1, 1, 2, 3, 5, 8, 13, 21, 34, 55]
```

檔案I/O

在Python中進行檔案輸出入時，必須先開啓檔案：

```
f=open(<filename>,<open_mode>)
```

上面參數**filename**是檔案名稱。**open_mode**是開啓檔案模式，是一個字串。常見的開啓模式有：

- 'r': 讀取檔案(預設為此模式)
- 'w': 寫入檔案。若檔案已存在，舊的檔案內容會先被刪掉
- 'a': 寫入檔案。若檔案已存在，會接在舊的檔案內容後面開始寫
- 't': 要讀寫的檔案是文字格式的檔案(預設為此模式)
- 'b': 要讀寫入的檔案為二進位格式

例如

```
f=open("data.txt",'w')
```

表示要寫入一個名為data.txt，以文字模式寫入。而

```
f=open("data.txt",'r+b')
```

表示要以二進位檔案的形式讀一個檔名為data.txt的檔案

而結束檔案操作時，要記得呼叫close()方法。語法如下：

```
f.close()
```

這個f就是上述紀錄open()所傳回的資料的變數。

```
>>> f=open("data.txt","w")
>>> f.write("Live long ")
10
>>> f.write("and prosper\n")
12
>>> f.close()
```

上述程式碼會產生一個名為data.txt的檔案，內容為“Live long and prosper”，並在最後換行。

上述程式碼寫入兩次資料，分別傳回的10和12，意思是第一次寫入了10個字元，第二次是寫入了12個字元

檔案I/O

```
>>> f=open("data.txt","r")
>>> txt=f.read()
>>> print(txt)
Live long and prosper
```

```
>>> txt
'Live long and prosper\n'
```

```
>>> f=open("data.txt","w")
>>> f.write("Live long ")
10
>>> f.write("and prosper\n")
12
>>> f.close()
```

`read()`會把檔案內容一次讀進來。因此，上述程式碼會讀入剛才產生的`data.txt`的所有內容，“Live long and prosper\n”。可以看到讀進來的`txt`在最後面有一個換行

```
fp=open("PippaPasses.txt","w")
fp.write("God's in his heaven\nAll's right with the world!")
fp.close()

fp=open("PippaPasses.txt","r")
lyric=[]
for line in fp:
    row=line.split()
    lyric.append(row)
fp.close()
print(lyric)
```

上述程式碼先在檔案內寫入兩行句子，然後將寫入的內容讀出。每個句子分別放入一個不同的list中，並把這兩個list都放入一個list裡面。程式輸出：

```
[["God's", 'in', 'his', 'heaven'], ["All's", 'right', 'with', 'the', 'world!']]
```

注意上述範例不是使用read()，而是用for迴圈來一行一行從fpr讀取檔案內容。對於大檔案來說，這樣讀取會是比较好的作法。

以下範例改使用with敘述來開啓檔案：

```
with open("PippaPasses.txt", "r") as fp:
    lyric = []
    for line in fp:
        row = line.split()
        lyric.append(row)

print(lyric)
```

使用with語法來開啓檔案有個好處：不需要自己呼叫close()來關閉檔案。with的block結束後，檔案會自己關閉。

另外也可以用`readline()`來一行一行讀取。範例如下：

```
fp=open("PippaPasses.txt","r")
lyric=[]
line=fp.readline()
while line != "":
    row=line.split()
    lyric.append(row)
    line=fp.readline()
fp.close()
print(lyric)
```

以下則是結合`with`和`readline`的作法：

```
with open("PippaPasses.txt","r") as fp:
    lyric=[]
    line=fp.readline()
    while line != "":
        row=line.split()
        lyric.append(row)
        line=fp.readline()
    print(lyric)
```

讀取csv檔

一般的csv檔案是以逗號分隔資料欄位的純文字檔案。可以用內建的csv套件來讀取：

```
import csv

f=open("iris.csv","r")
rd=csv.reader(f,delimiter=",")
head=next(rd) # 如果第一個row有欄位名稱，要跳過它需要這個敘述

data=[]
for row in rd:
    data.append(row)

x=[row[0:-1] for row in data] # get all columns except for the last one
x=[[float(v) for v in x_row] for x_row in x] # convert str to float
y=[row[-1] for row in data] # get only the last column
```

上面的範例是讀取一個第一個row有欄位名稱的csv檔鳶尾花資料集。最後一個column是label，其餘的是feature。因為feature內容都是數值，故轉換為float形式的資料。

讀取csv檔

csv套件還可以將資料讀取進來之後，根據欄位名稱形成dictionary。如下例：

```
import csv

with open("iris.csv","r") as f:
    rd=csv.DictReader(f,delimiter=",")
    data=[ row for row in rd ]

for i in range(2):
    print(f"row {i}:")
    for key,value in data[i].items():
        print(f"{key} is {value}")
    print()
```

左例輸出是：

```
row 0:
sepal length (cm) is 5.1
sepal width (cm) is 3.5
petal length (cm) is 1.4
petal width (cm) is 0.2
class is Iris-setosa

row 1:
sepal length (cm) is 4.9
sepal width (cm) is 3.0
petal length (cm) is 1.4
petal width (cm) is 0.2
class is Iris-setosa
```