

目次

Module 1. Python 技術及開發環境介紹	1
認識 Python 應用	1
建置 Python 開發環境	1
安裝 Python 單一環境	2
安裝 Anaconda 開發軟體	6
如何執行 Python 程式	16
使用 IDE	16
建立開發專案的資料夾	22
讓專案資料夾使用 Conda 環境	24
建立與執行 .py 檔案	26
參考資料	27
Module 2. 資料型態與變數	28
資料型態	28
變數	29
變數命名規則、關鍵字與縮排	30
變數命名規則	30
關鍵字	30
縮排	31
運算子	31
算術運算子	32
賦值運算子	32
比較運算子	33
邏輯運算子	34
成員運算子	34
身分運算子	35
位元運算子	35
運算子優先順序	36
註解	36
參考資料	38
Module 3. 控制結構	38
循序	38
選擇	39
重複	40
Module 4. 字串、串列、元組、字典、集合	43
字串	43
補充：字串格式化	44
補充：Input	47

串列.....	48
補充：List Comprehension	51
元組.....	52
字典.....	53
集合.....	55
參考資料	57
Module 5.函式	57
基本用法	58
變數傳遞	58
區域變數與全域變數	59
匿名函式 Lambda	61
參考資料	63
Module 6.物件導向程式設計	63
基本認識物件導向	63
類別.....	64
方法.....	64
屬性.....	65
繼承.....	66
參考資料	67
Module 7.例外處理.....	68
認識例外處理	68
try-except 陳述.....	68
例外處理 raise 陳述.....	70
參考資料	70
Module 8. 檔案處理.....	70
open() 函式.....	70
讀取檔案	71
寫入檔案	71
刪除檔案	72
參考資料	72
Module 9. 模組	72
定義模組	72
使用模組	73
__name__ 內建變數的用法.....	74
參考資料	76

● GitHub 專案連結

https://github.com/telunyang/python_basics

- YouTube 頻道

<https://www.youtube.com/@darreninfo-boatman>

Module 1. Python 技術及開發環境介紹

認識 Python 應用

Python 是一種跨平台 (Cross-platform)、開放原始碼 (Open Source)、擁有大量社群 (Communities)、物件導向 (Object-Oriented Language) 的程式語言，由 Guido van Rossum (Python 最初設計者兼程式架構師) 在 1989 年底發明。Python 名稱的發想，是來自 Rossum 喜愛的電視喜劇《Monty Python's Flying Circus》。

Python 的語法簡潔、明確，可以整合許多工具來開發各式各樣的功能與服務，例如網站建置、資料分析與視覺化、機器學習、深度學習、視窗表單設計、自動化、遊戲開發、資料庫操作等，都是 Python 的強項。

Python 目前主流是 3.x 版本，過去的 2.x 版本已經不再被官方維護，強烈建議大家安裝 3.x 版來進程式開發與學習。

建置 Python 開發環境

Python 的環境建置來源，通常分為官方網站 (<https://www.python.org/>) 與專門用於科學計算的 Anaconda (<https://www.anaconda.com/>) 或 MiniConda (<https://docs.conda.io/en/latest/miniconda.html>)，個人會建議使用 Anaconda，原因如下：

1. 可透過圖形化介面操作，開發用的套件 (Packages) 清單、環境版本控制等，管理上較為容易。
2. 常用工具與科技計算用的會幫你事先安裝完成，對初學者較為友善。當同學慢慢熟悉開發流程的時候，也可以使用終端機 (Terminal) 來安裝適合自己的工具或套件。

Anaconda 也會有它的缺點，例如：

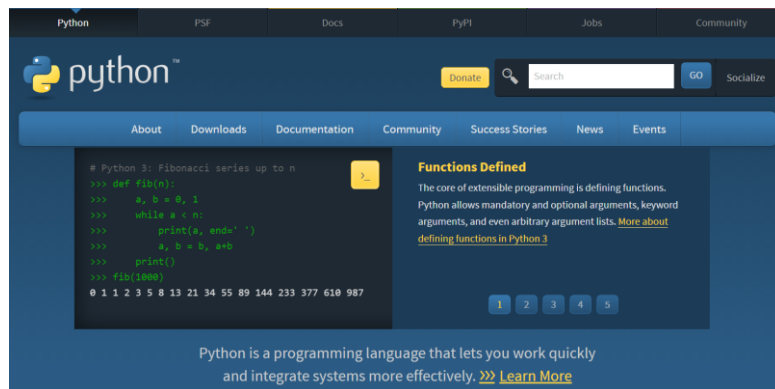
1. 下載安裝的檔案很大、時間很久，對進階的開發者而言，預先下載的工具不見得是開發者要的，換句話說，一開始會有很多短時間用不著，或是要學習到一個階段的時候，才有機會用到的工具。對於初學者來說，透過 Anaconda 建立第一個開發環境是很重要的，繁雜的流程，容易讓人望而卻步。
2. 安裝工具 conda & pip 容易讓人搞不清楚。conda 與 pip 都可以安裝開發環境當中的套件：
 - conda 主要是安裝開發的「環境」(Environment)，例如透過 conda 選定一個 python 版本，然後給予自訂名稱作為開發環境

的識別，以後可以任意切換不同 python 版本的環境。

- **pip** 純粹是套件安裝與管理工具，在不同的 conda 環境中，針對環境使用的 python 版本，來決定下載、安裝哪個 python 版本適用的套件。

安裝 Python 單一環境

官方網站 (<https://www.python.org/>)



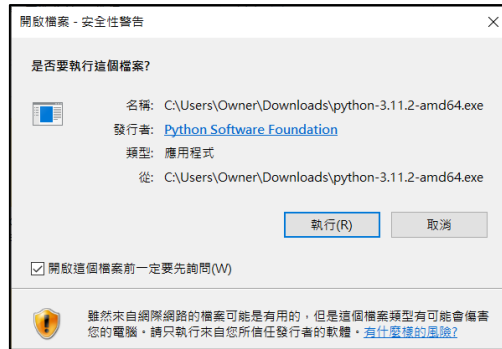
圖：python.org 首頁



圖：游標移至 Download，看到 Python 下載按鈕，直接點它下載安裝程式



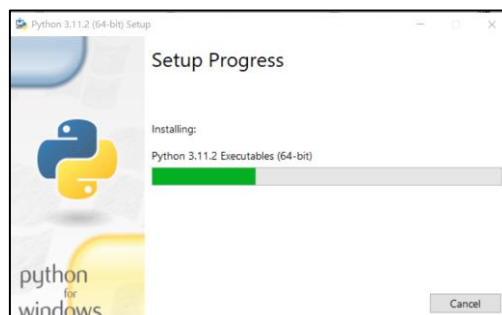
圖：快速點兩下，執行安裝程式



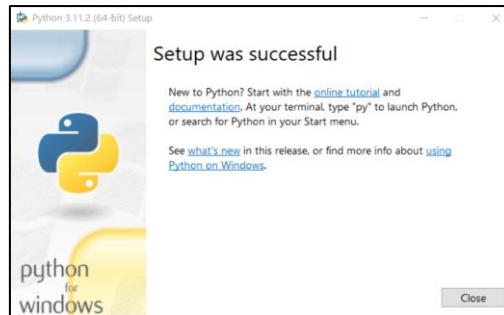
圖：若有安全性警告，直接按下「執行」



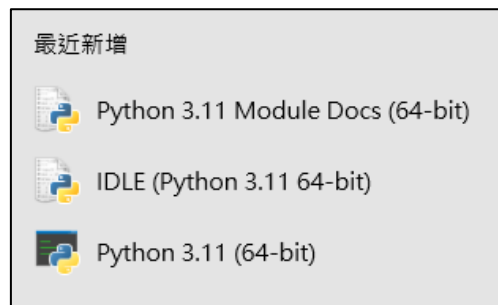
圖：打勾「Add python.exe to PATH」，按下上方的 Install Now
(若有安全性提示，可以按下同意)



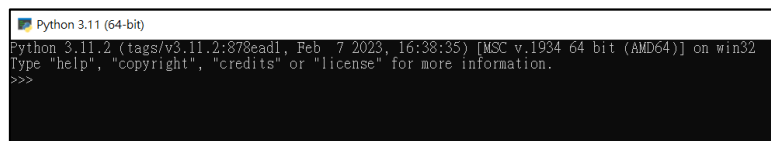
圖：等待安裝過程



圖：安裝成功畫面



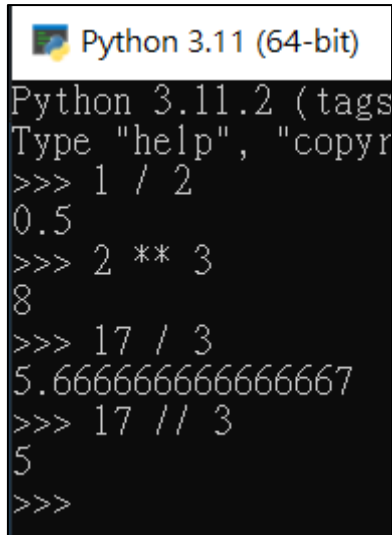
圖：到 Windows 選單，尋找最近新增，選擇「Python 3.xx (64-bit)」



圖：python 指令操作介面

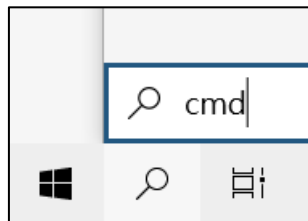
範例
<pre>>>> 1 / 2 (輸入完，按下 Enter) 0.5 >>> 2 ** 3 (輸入完，按下 Enter) 8 >>> 17 / 3 (輸入完，按下 Enter) 5.666666666666667 >>> 17 // 3 (輸入完，按下 Enter) 5</pre>

表：小試一下

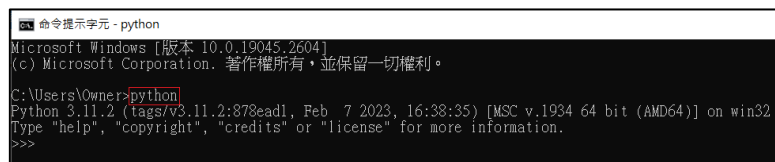


```
Python 3.11 (64-bit)
Python 3.11.2 (tags
Type "help", "copyr
>>> 1 / 2
0.5
>>> 2 ** 3
8
>>> 17 / 3
5.666666666666667
>>> 17 // 3
5
>>>
```

圖：執行結果。可輸入 `quit()` 或 `exit()`，再按 Enter 離開

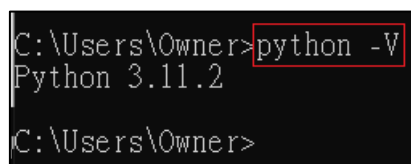


圖：搜尋 `cmd`，按下 Enter，進入命令提示字元



```
命令提示字元 - python
Microsoft Windows [版本 10.0.19045.2604]
(c) Microsoft Corporation. 著作權所有，並保留一切權利。
C:\Users\Owner>python
Python 3.11.2 (tags/v3.11.2:878ead1, Feb. 7 2023, 16:38:35) [MSC v.1934 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
```

圖：在命令提示字元輸入「`python`」後，按下 Enter，可使用 `python`



```
C:\Users\Owner>python -V
Python 3.11.2
C:\Users\Owner>
```

圖：可以在命令提示字元輸入「`python -V`」，按下 Enter，
得知當前 Python 的版本
若是安裝過程中，沒有打勾「`python.exe to PATH`」，
無法在命令提示字元中使用 `python` 指令

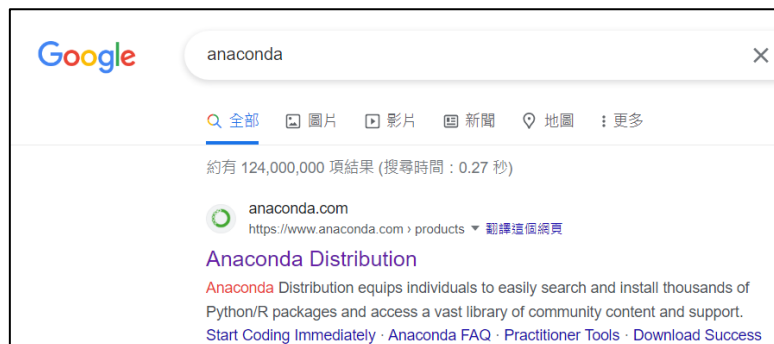
安裝 Anaconda 開發軟體

安裝 Anaconda

參考連結

[1] Anaconda - Individual Edition

<https://www.anaconda.com/products/individual>



圖：可以先至 google 檢索「anaconda」，找到 Anaconda Distribution 的連結

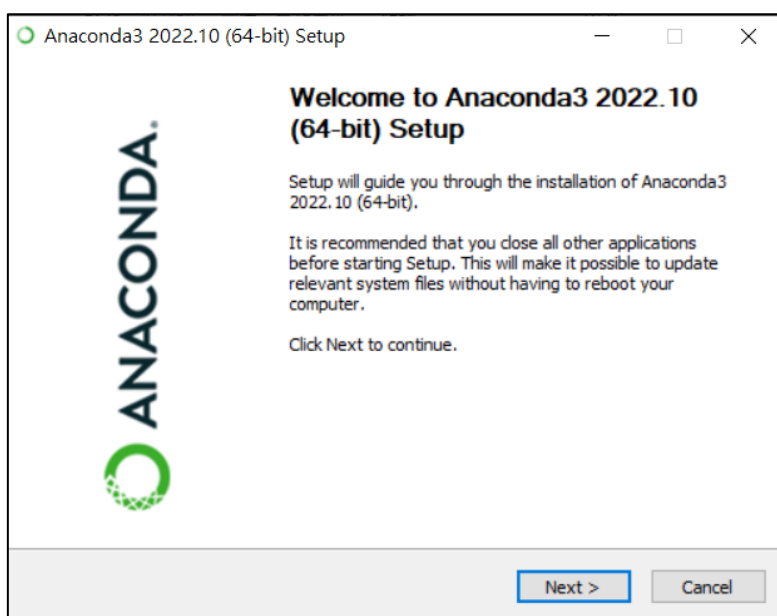


圖：在 Windows 選項下，選擇 64-Bit Graphical Installer，並下載下來

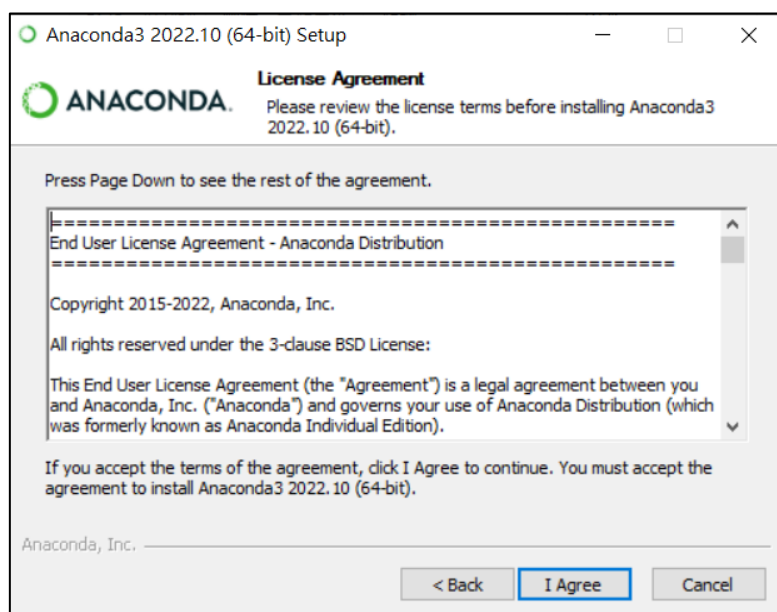


Anaconda3-202
2.10-Windows-x
86_64.exe

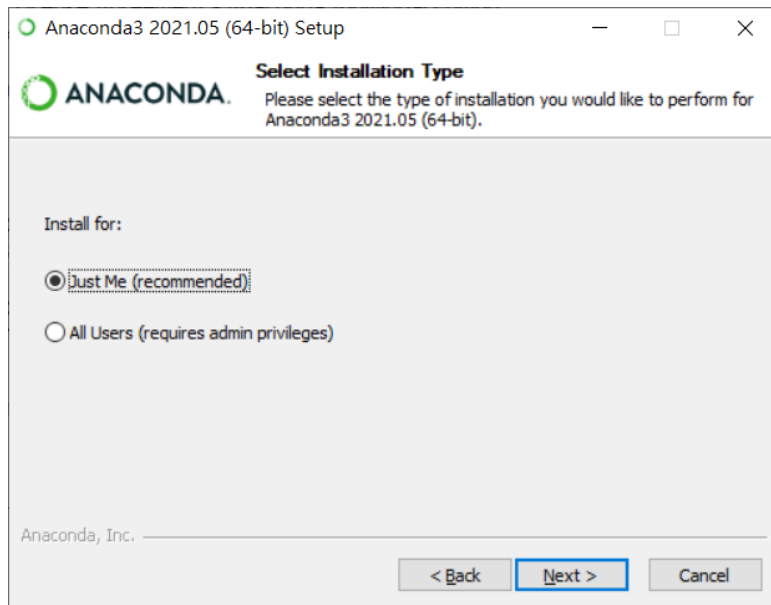
圖：下載後，快速點兩下進行安裝
有安全性警告，可按下執行或同意



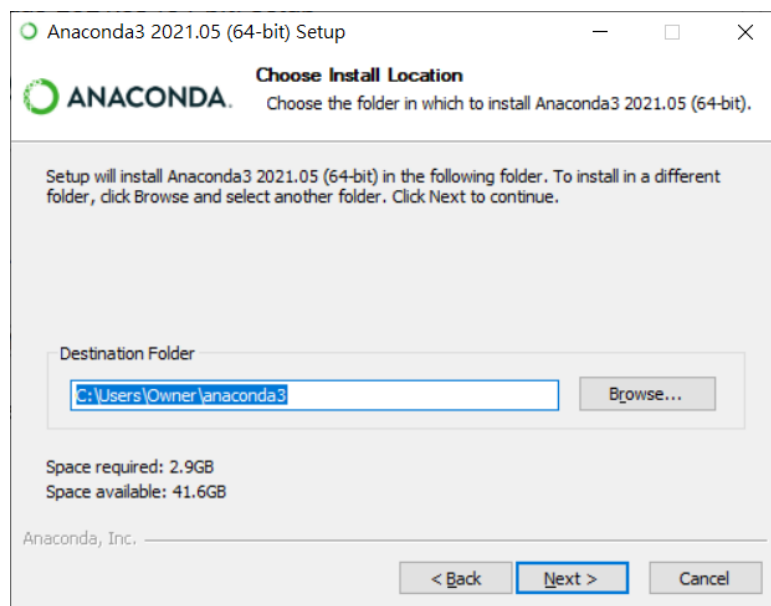
圖：按下「Next」



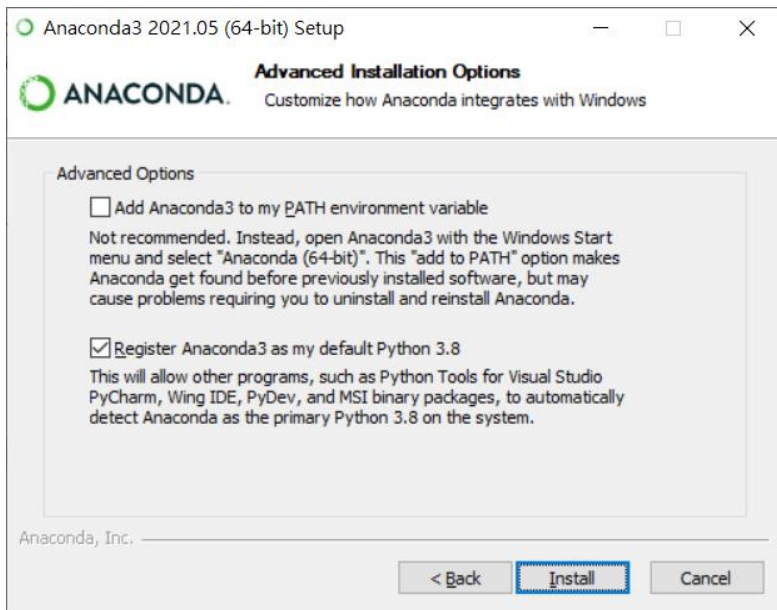
圖：按下「I Agree」



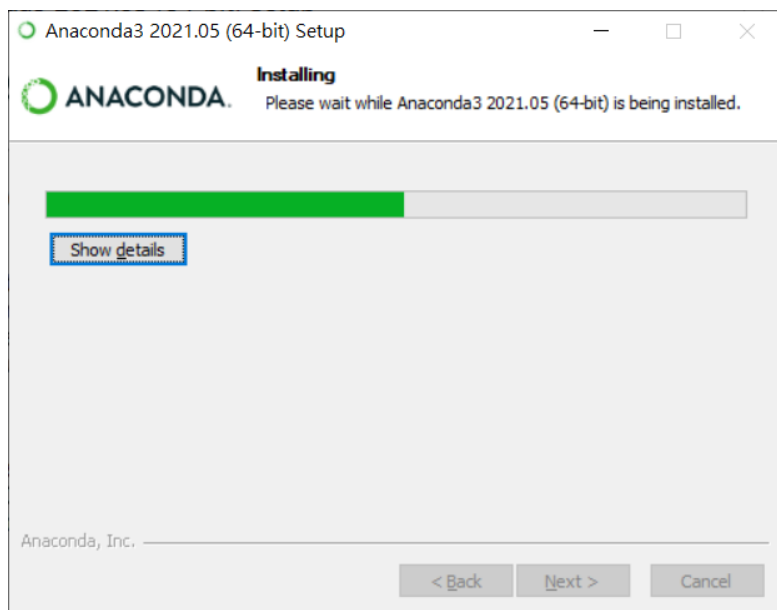
圖：依需求選擇後，按下「Next」



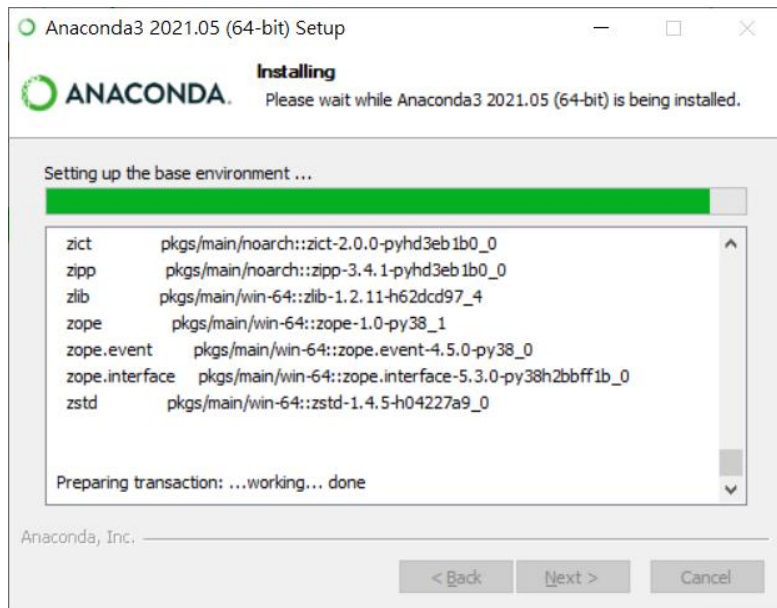
圖：安裝位置會在使用者資料夾中的 anaconda3，依需求設定，按下「Next」



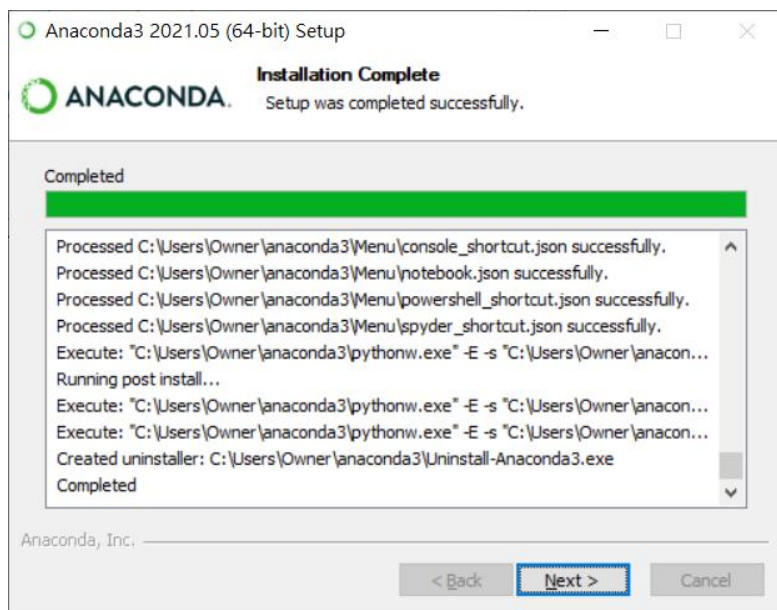
圖：依需求選擇，按下「Install」，進行安裝



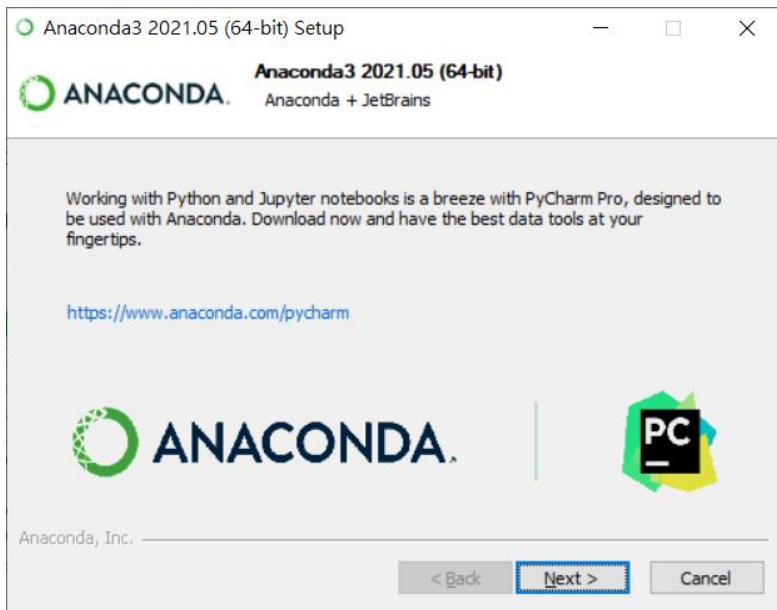
圖：安裝過程，需要一段時間



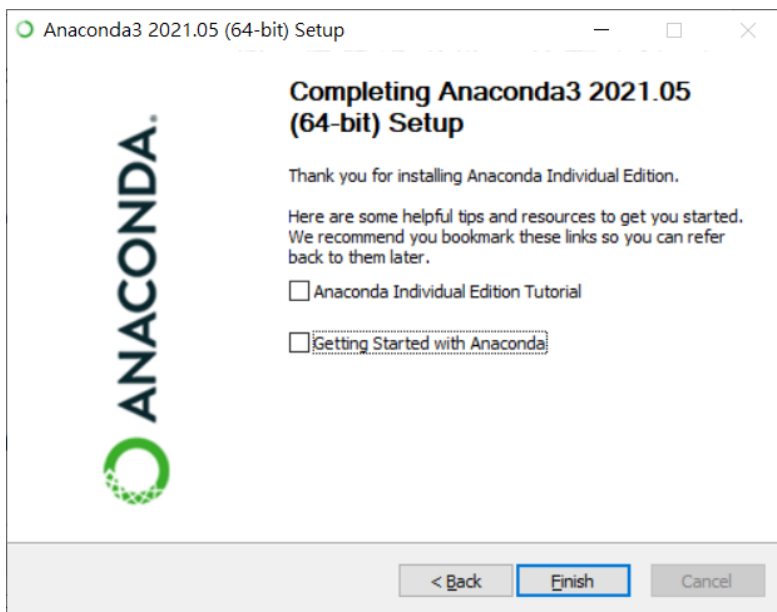
圖：按下「Show details」，會看到安裝過程



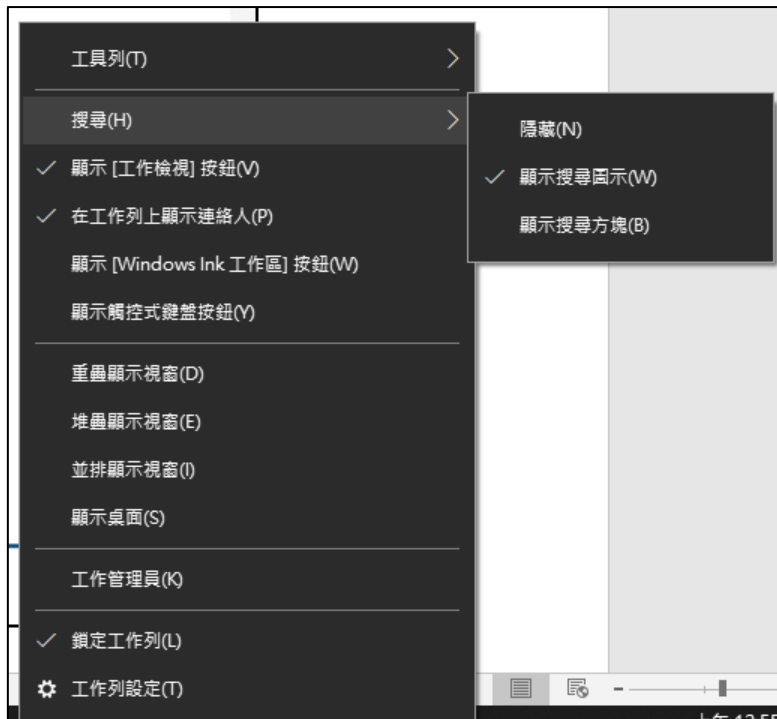
圖：安裝完成後，按下「Next」



圖：按下「Next」



圖：取消勾選圖片中的兩個選項後，按下「Finish」



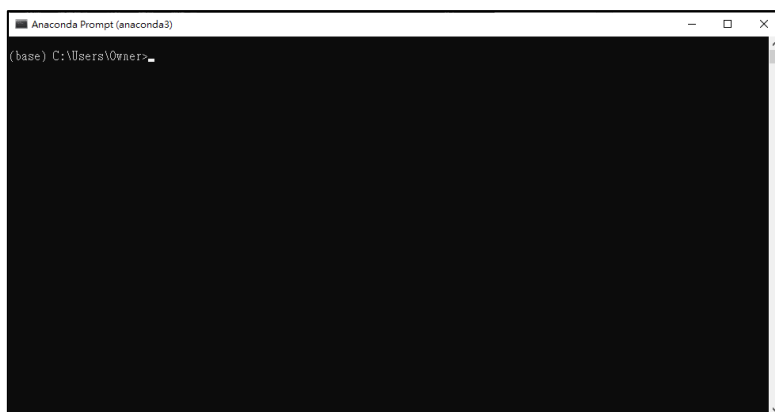
圖：顯示搜尋圖示



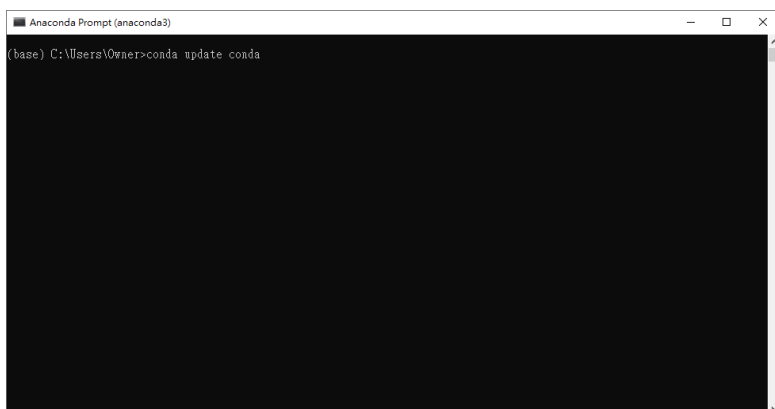
圖：搜尋圖示類似放大鏡，按下搜尋圖示



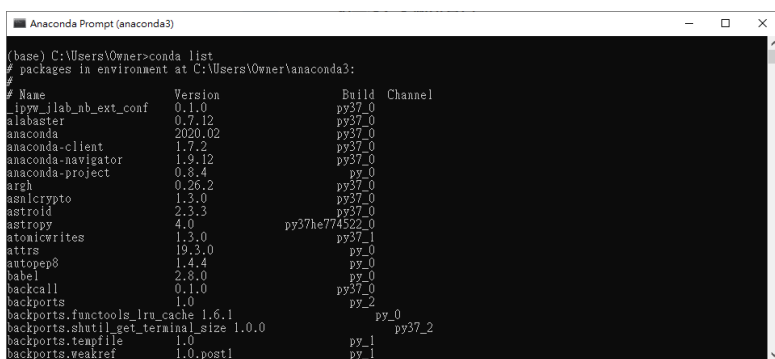
圖：搜尋「anaconda prompt」，按下「Anaconda Prompt (anaconda3)」



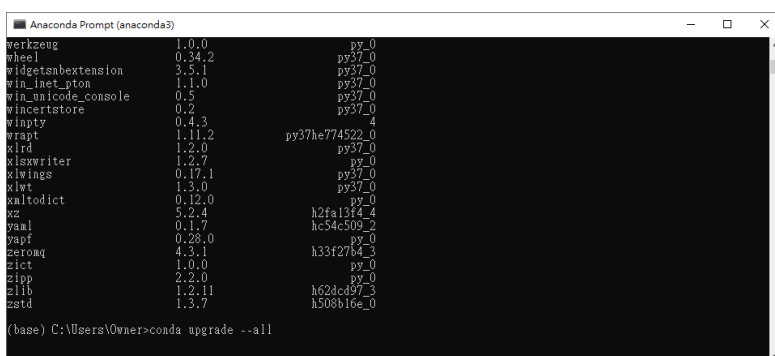
圖：出現 Anaconda Prompt，類似 Windows 的命令提示字元



圖：輸入「conda update conda」，更新 conda 本身



圖：輸入「conda list」，會顯示目前預設安裝的套件（在虛擬環境 base 下）



圖：輸入「conda upgrade --all」，更新當前所有套件

```
Anaconda Prompt (anaconda3) - conda upgrade --all
sphinxcontrib-qth- 1.0.2-py_0 --> 1.0.3-py_0
sphinxcontrib-ser- 1.1.3-py_0 --> 1.1.4-py_0
sphinxcontrib-web- 1.2.0-py_0 --> 1.2.1-py_0
spyder 4.0.1-py37_0 --> 4.1.3-py37_0
spyder-kernels 1.8.1-py37_0 --> 1.9.1-py37_0
sqlalchemy 1.3.13-py37he774522_0 --> 1.3.16-py37he774522_0
sqlite 3.31.1-he774522_0 --> 3.31.1-h2a8f88b_1
toronado 6.0.3-py37he774522_3 --> 6.0.4-py37he774522_1
tqdm 4.42.1-py_0 --> 4.46.0-py_0
wcwidth 0.1.8-py_0 --> 0.1.9-py_0
werkzeug 1.0.0-py_0 --> 1.0.1-py_0
xlswriter 1.2.7-py_0 --> 1.2.8-py_0
xlwings 0.17.1-py37_0 --> 0.19.0-py37_0
xz 5.2.4-h2fa13f4_4 --> 5.2.5-h62cd497_0
zict 1.0.0-py_0 --> 2.0.0-py_0
zipp 2.2.0-py_0 --> 3.1.0-py_0
zlib 1.2.11-h62cd497_3 --> 1.2.11-h62cd497_4

The following packages will be DOWNGRADED:
anaconda 2020.02-py37_0 --> custom-py37_1
lzo 2.10-h6df0209_2 --> 2.10-he774522_2

Proceed ([y]/n)?
```

圖：按下「y」後，再按鍵盤「Enter」

```
Anaconda Prompt (anaconda3) - conda upgrade --all
lzo 2.10-h6df0209_2 --> 2.10-he774522_2

Proceed ([y]/n)? y

Downloading and Extracting Packages
kiwisolver-1.2.0 | 55 KB | ##### 100%
seaborn-0.10.1 | 163 KB | ##### 100%
dask-2.16.0 | 14 KB | ##### 100%
jupyter-core-4.6.3 | 85 KB | ##### 100%
conda-4.8.3 | 2.8 MB | ##### 100%
cython-0.29.17 | 1.8 MB | ##### 100%
prompt_toolkit-3.0.4 | 11 KB | ##### 100%
fsspec-0.7.1 | 56 KB | ##### 100%
requests-2.23.0 | 93 KB | ##### 100%
sphinxcontrib-apple- 27 KB | ##### 100%
typed-ast-1.4.1 | 141 KB | ##### 100%
curl-7.69.1 | 126 KB | ##### 100%
python-libarchive-c- 46 KB | ##### 100%
anaconda_depends-20 | 6 KB | ##### 100%
python-language-serv- 94 KB | ##### 100%
qtawesome-0.7.0 | 726 KB | ##### 100%
spyder-kernels-1.9.1 | 96 KB | ##### 100%
sqlalchemy-1.3.16 | 1.5 MB | 0%
```

圖：更新套件中

```
Anaconda Prompt (anaconda3)
- DEBUG menuinst_win32: __init__(199): Menu; name: 'Anaconda$($PY_VER) $(PLATFORM)', prefix: 'C:\Users\Owner\anaconda3'
- env_name: 'None', mode: 'user', used_mode: 'user'
DEBUG menuinst_win32:create(323): Shortcut cmd is C:\Users\Owner\anaconda3\pythonw.exe, args are ['C:\Users\Owner\anaconda3\vcvp.py', 'C:\Users\Owner\anaconda3', 'C:\Users\Owner\anaconda3\python.exe', 'C:\Users\Owner\anaconda3\Scripts\spyder-script.py']
- DEBUG menuinst_win32:create(323): Shortcut cmd is C:\Users\Owner\anaconda3\python.exe, args are ['C:\Users\Owner\anaconda3\vcvp.py', 'C:\Users\Owner\anaconda3', 'C:\Users\Owner\anaconda3\python.exe', 'C:\Users\Owner\anaconda3\Scripts\spyder-script.py', '--reset']
done
(base) C:\Users\Owner>
```

圖：套件更新完成

```
Anaconda Prompt (anaconda3) - python
- DEBUG menuinst_win32: __init__(199): Menu; name: 'Anaconda$($PY_VER) $(PLATFORM)', prefix: 'C:\Users\Owner\anaconda3'
- env_name: 'None', mode: 'user', used_mode: 'user'
DEBUG menuinst_win32:create(323): Shortcut cmd is C:\Users\Owner\anaconda3\pythonw.exe, args are ['C:\Users\Owner\anaconda3\vcvp.py', 'C:\Users\Owner\anaconda3', 'C:\Users\Owner\anaconda3\python.exe', 'C:\Users\Owner\anaconda3\Scripts\spyder-script.py']
- DEBUG menuinst_win32:create(323): Shortcut cmd is C:\Users\Owner\anaconda3\python.exe, args are ['C:\Users\Owner\anaconda3\vcvp.py', 'C:\Users\Owner\anaconda3', 'C:\Users\Owner\anaconda3\python.exe', 'C:\Users\Owner\anaconda3\Scripts\spyder-script.py', '--reset']
done
(base) C:\Users\Owner>python
Python 3.7.7 (default, May 6 2020, 11:45:54) [MSC v.1916 64 bit (AMD64)] :: Anaconda, Inc. on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
```

圖：輸入「python」，進入 python 執行環境

```

Anaconda Prompt (anaconda3) - python
- DEBUG maininst_win32: __init__(199): Name: name: 'Anaconda$(PY_VER) $(PLATFORM)', prefix: 'C:\Users\Owner\anaconda3'
, env_name: 'None', mode: 'user', used_mode: 'user'
DEBUG maininst_win32:create(323): Shortcut cmd is C:\Users\Owner\anaconda3\pythonw.exe, args are ['C:\Users\Owner\anaconda3\Scripts\pythonw.exe', 'C:\Users\Owner\anaconda3\pythonw.exe', 'C:\Users\Owner\anaconda3\Scripts\ipython-script.py']
- DEBUG maininst_win32:create(323): Shortcut cmd is C:\Users\Owner\anaconda3\python.exe, args are ['C:\Users\Owner\anaconda3\Scripts\python.exe', 'C:\Users\Owner\anaconda3\python.exe', 'C:\Users\Owner\anaconda3\Scripts\ipython-script.py', '--reset']
done

(base) C:\Users\Owner>python
Python 3.7.7 (default, May 6 2020, 11:45:54) [MSC v.1916 64 bit (AMD64)] :: Anaconda, Inc. on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> print("Hello, world")
Hello, world
>>>
```

圖：輸入「`print("Hello, world")`」，輸出「Hello, world」，python 安裝成功

```

Anaconda Prompt (anaconda3)
- DEBUG maininst_win32: __init__(199): Name: name: 'Anaconda$(PY_VER) $(PLATFORM)', prefix: 'C:\Users\Owner\anaconda3'
, env_name: 'None', mode: 'user', used_mode: 'user'
DEBUG maininst_win32:create(323): Shortcut cmd is C:\Users\Owner\anaconda3\pythonw.exe, args are ['C:\Users\Owner\anaconda3\Scripts\pythonw.exe', 'C:\Users\Owner\anaconda3\pythonw.exe', 'C:\Users\Owner\anaconda3\Scripts\ipython-script.py']
- DEBUG maininst_win32:create(323): Shortcut cmd is C:\Users\Owner\anaconda3\python.exe, args are ['C:\Users\Owner\anaconda3\Scripts\python.exe', 'C:\Users\Owner\anaconda3\python.exe', 'C:\Users\Owner\anaconda3\Scripts\ipython-script.py', '--reset']
done

(base) C:\Users\Owner>python
Python 3.7.7 (default, May 6 2020, 11:45:54) [MSC v.1916 64 bit (AMD64)] :: Anaconda, Inc. on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> print("Hello, world")
Hello, world
>>> exit()

(base) C:\Users\Owner>
```

圖：按下「`exit()`」回到指令輸入的環境

安裝、切換與刪除 Conda 環境 (Environment)

預設是 (base)，如果有切換環境的需求，例如手上處理著不同 Python 版本和其它相關套件的專案，需要不時切換版本來開發，此時可以建立一到多個 Conda 環境，需要的時候可以切換，不需要的時候可以刪除。

在執行以下的指令前，需要確認目前是否在 Anaconda Prompt 當中，或是支援 Conda 的 Terminal 環境。終端機顯示預設路徑時，最前面會有 (base)，代表目前正在預設的 Conda 環境當中。

安裝 Conda 環境

範例

```
conda create --name 自訂環境名稱 python=版本號，例如 3.9
```

```
conda create --name web_scraping python=3.9
```

列出所有 Conda 環境

```
conda env list
```

離開當前的 Conda 環境

<code>conda deactivate</code>

切換 / 啟動 Conda 環境

<code>conda activate web_scraping</code>
--

刪除 Conda 環境

<code>conda env remove -n web_scraping</code>

如何執行 Python 程式

使用 IDE

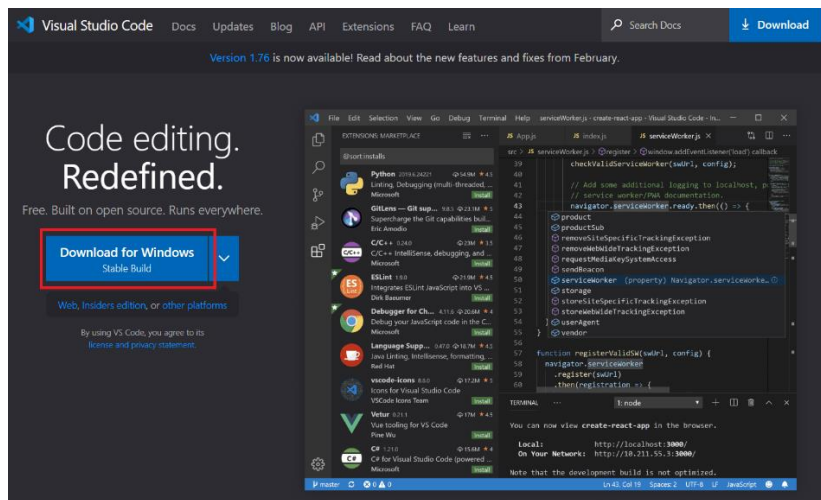
整合式開發環境 (Integrated Development Environment, IDE) 可以協助開發者方便使用類似記事本的程式碼輸入介面，側欄又有類似檔案總管的管理功能。常見的 Python IDE 有：

1. Visual Studio Code
2. Eclipse with PyDev
3. Sublime Text
4. Atom
5. Spyder
6. PyCharm
7. Vim
8. Emacs
9. Thonny
10. Wing

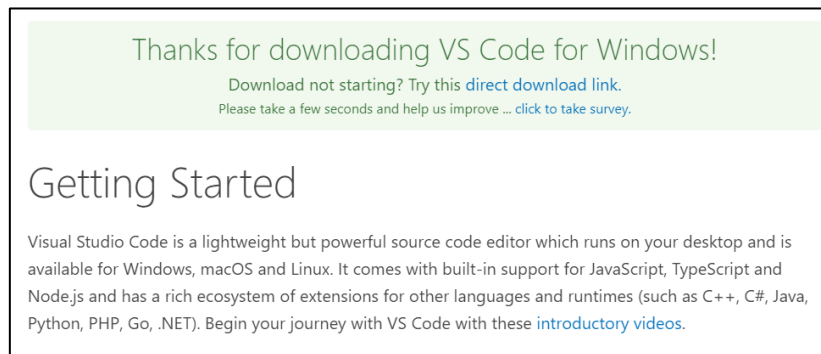
在課程中，我們選用「Visual Studio Code」（簡稱為 VS code）來作為課程使用的 IDE。VS code 是微軟開發的跨平台 (Cross-platform) 程式開發工具，除了開發程式之外，還可以使用豐富的擴充功能 (Extensions) 來提升開發的效率。



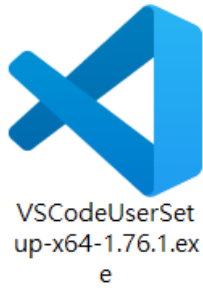
圖：在 google 搜尋「vs code」，進入 vs code 首頁



圖：網站會偵測使用者的作業系統，依需求選擇後，按下 Download for XXX



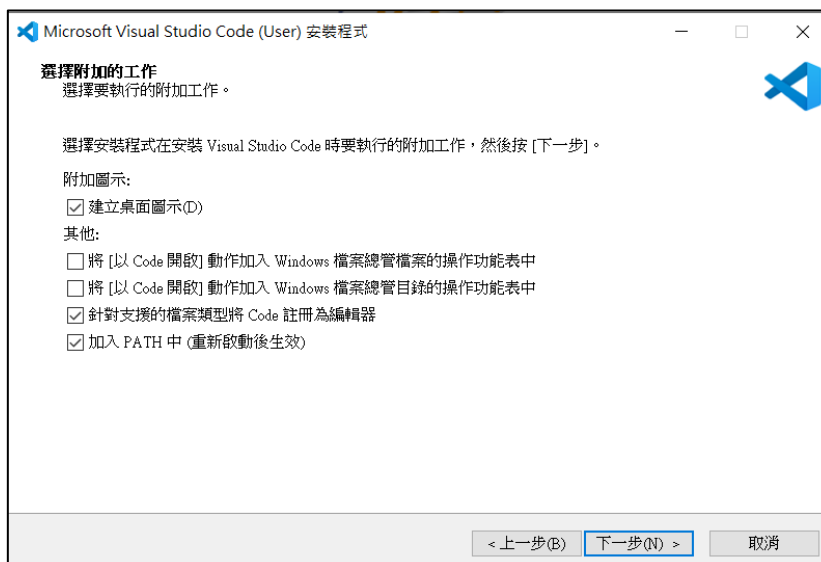
圖：看到這個畫面，安裝檔案會自動下載，
沒有下載的話，可以按下「direct download link」



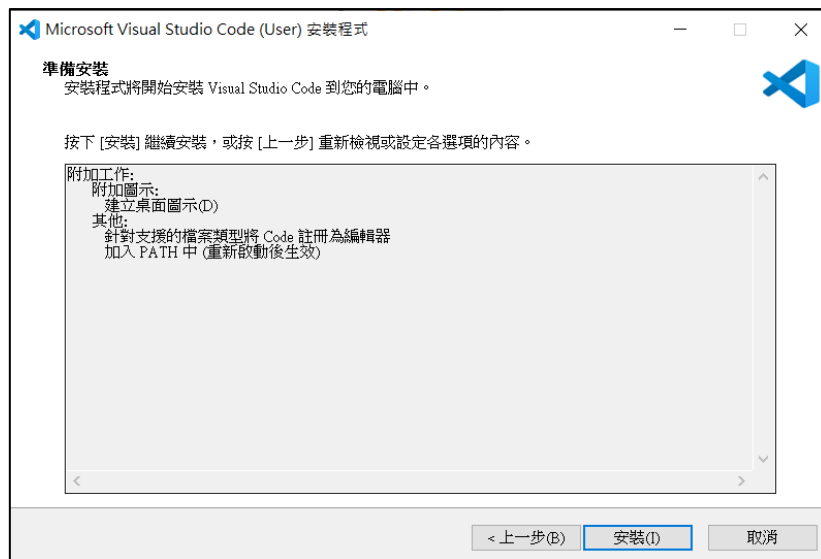
圖：快速連點兩下安裝，有安全性警告就按執行或同意



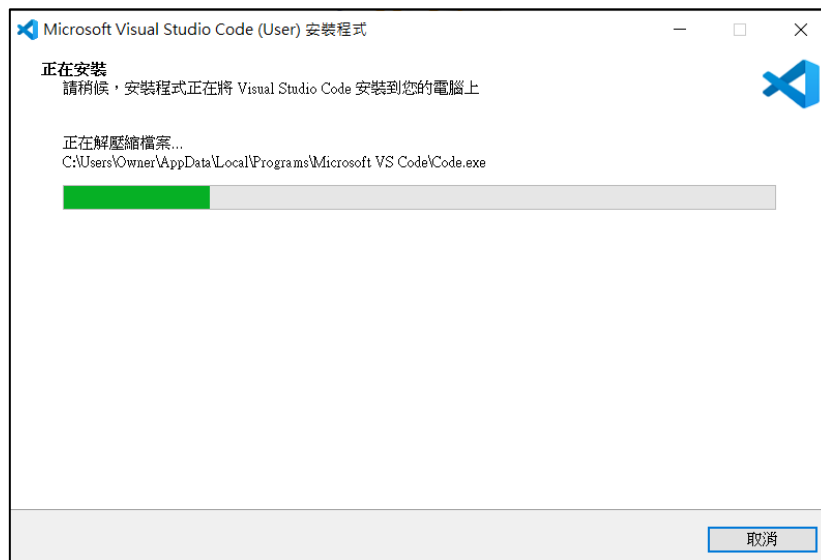
圖：勾選「我同意」，按「下一步」



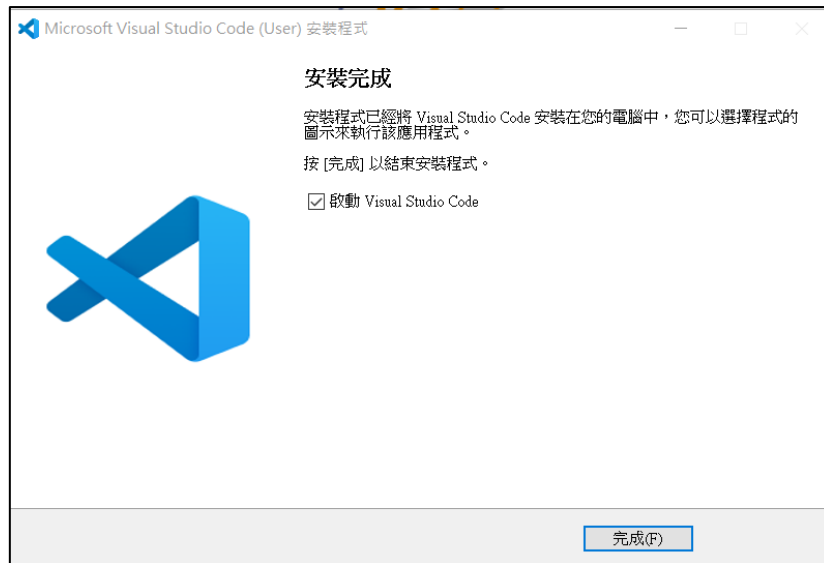
圖：勾選「建立桌面圖示」，按「下一步」



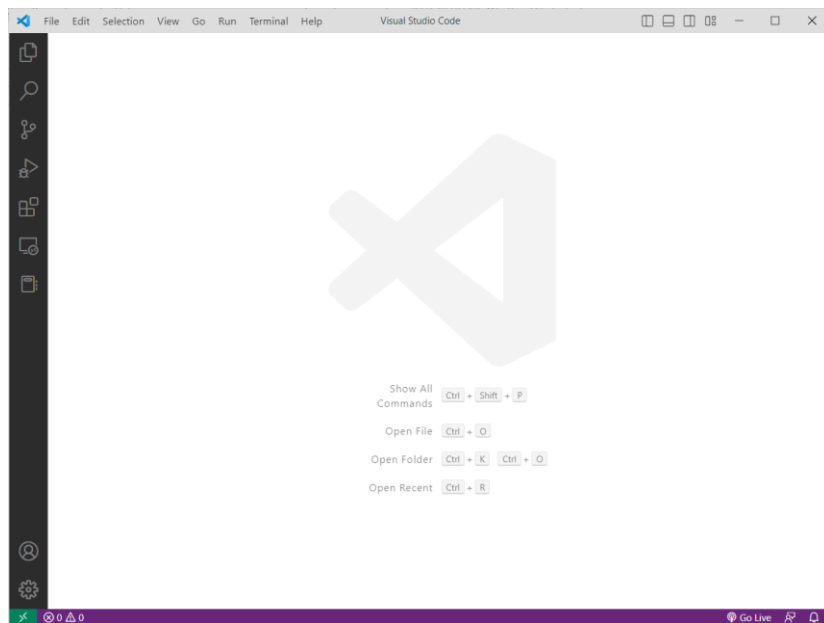
圖：按下「安裝」



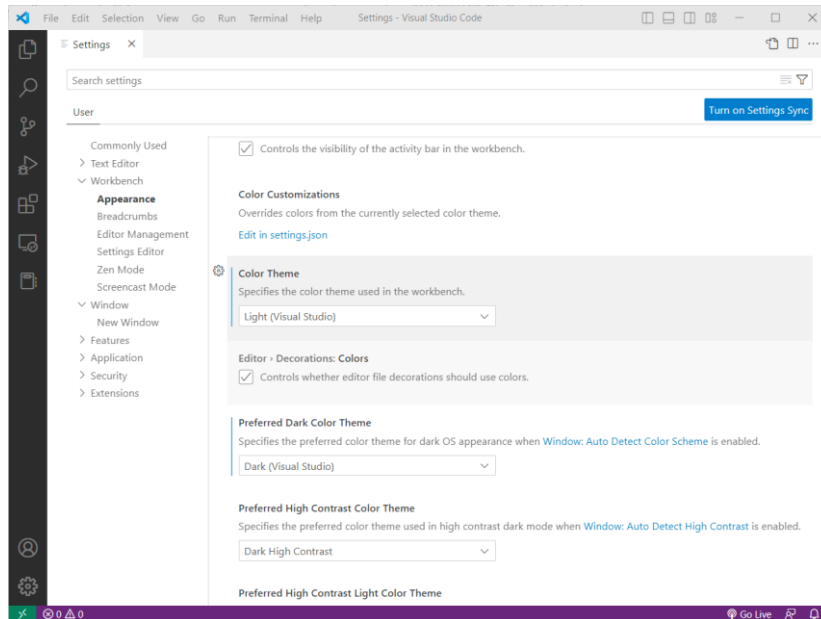
圖：安裝過程



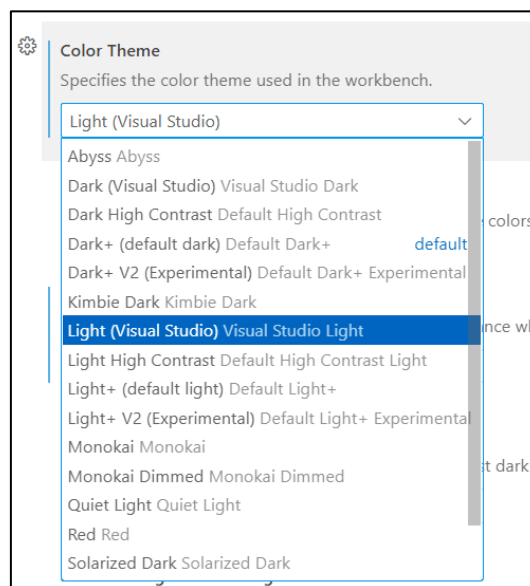
圖：勾選「啟動 Visual Studio Code」，按下「完成」



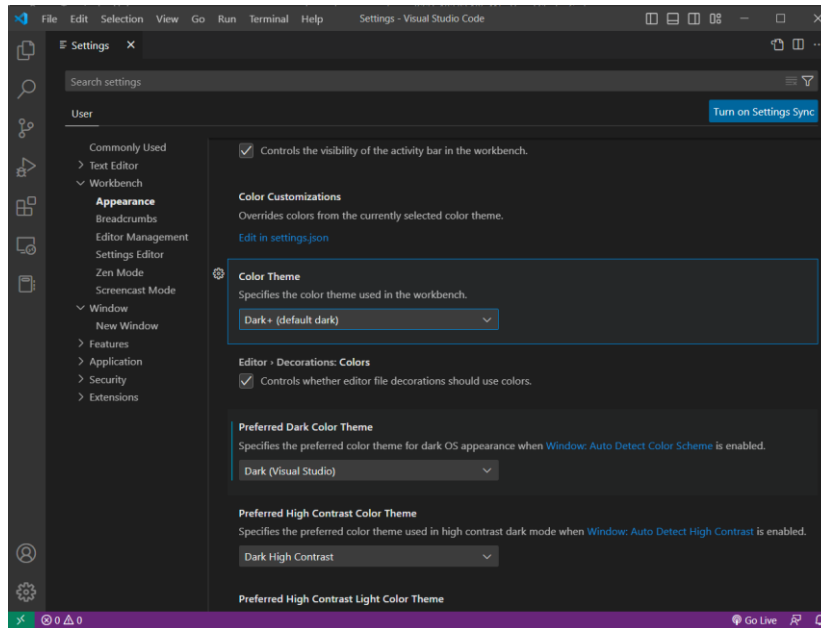
圖：VS code 的畫面



圖：如果想要換佈景顏色，按下「Ctrl + ,」來開啟設定

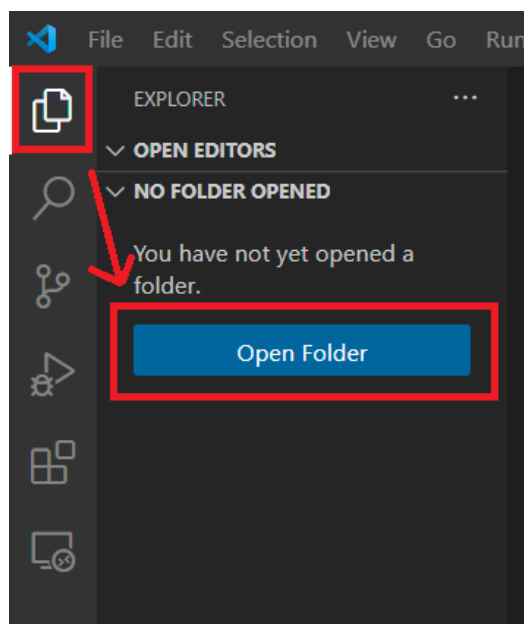


圖：在 Color Theme 下方的選單中，選擇偏好的佈景，例如選擇「Dark+」

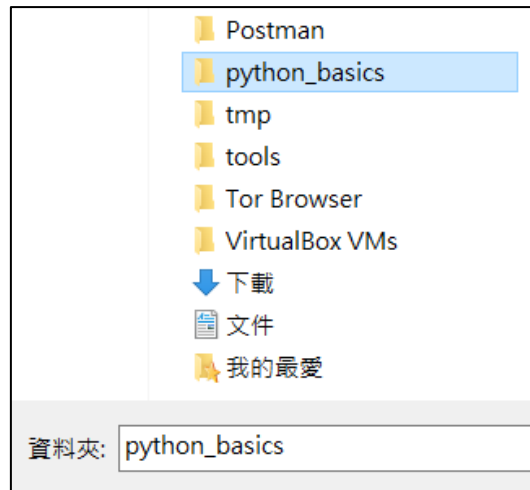


圖：選擇「Dark+」，可以立即套用選擇的佈景，
可以把左上方的頁籤 Settings 關掉（透過按下 Settings 右邊的 x 圖示）

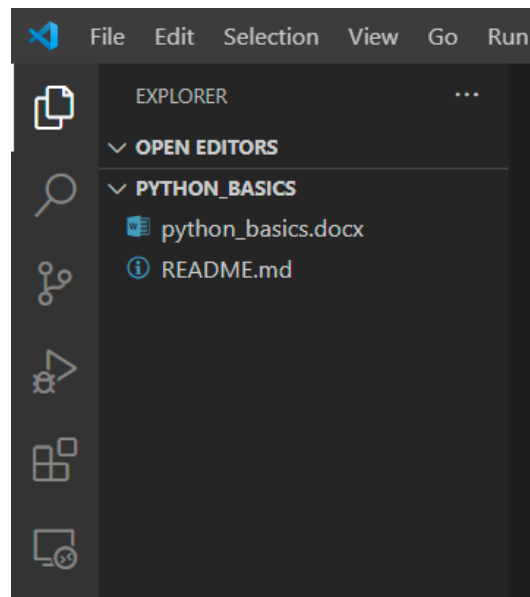
建立開發專案的資料夾



圖：按下左上角檔案總管 (Explorer)，再按下開啟資料夾 (Open Folder)，

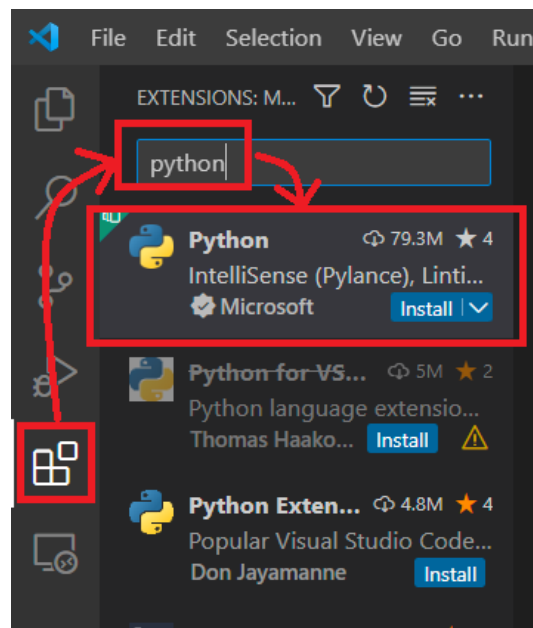


圖：選擇一個已存在的資料夾，或是手動新增，再選擇新資料夾作為專案資料夾，按下「選擇資料夾」



圖：選擇資料夾後的呈現方式

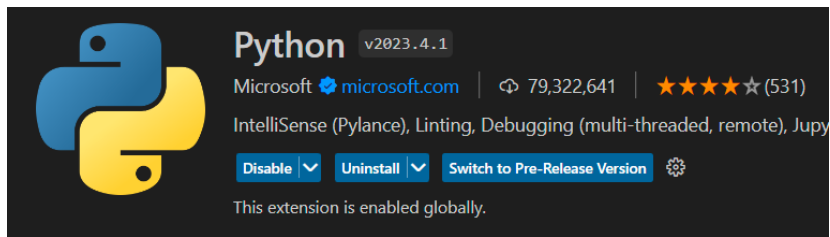
讓專案資料夾使用 Conda 環境



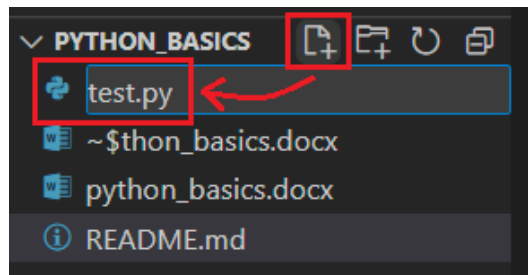
圖：按下擴充功能 (Extensions)，在上方輸入「python」，點選「Python」



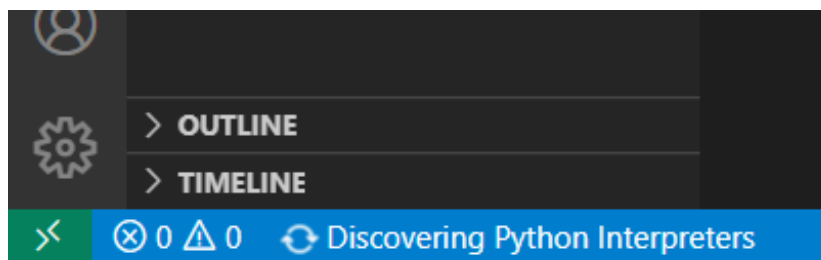
圖：按下 Install



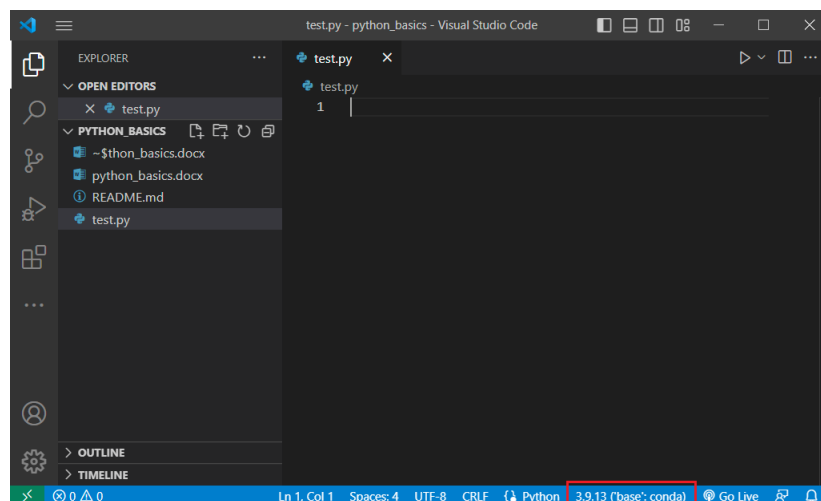
圖：出現 Uninstall 字眼，代表安裝完成，關閉頁籤，回到專案資料夾



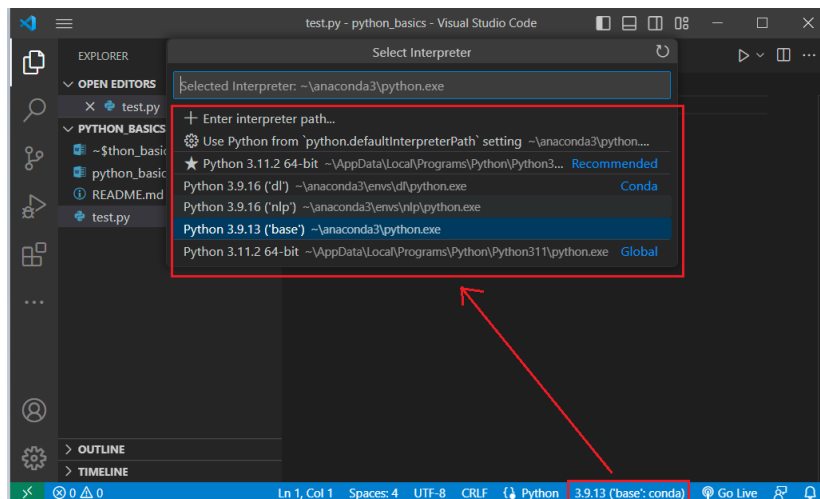
圖：新增檔案，輸入檔案為 test.py，按下 Enter



圖：開啟 .py 的檔案後，VS code 自動尋找直譯器，需要一點時間



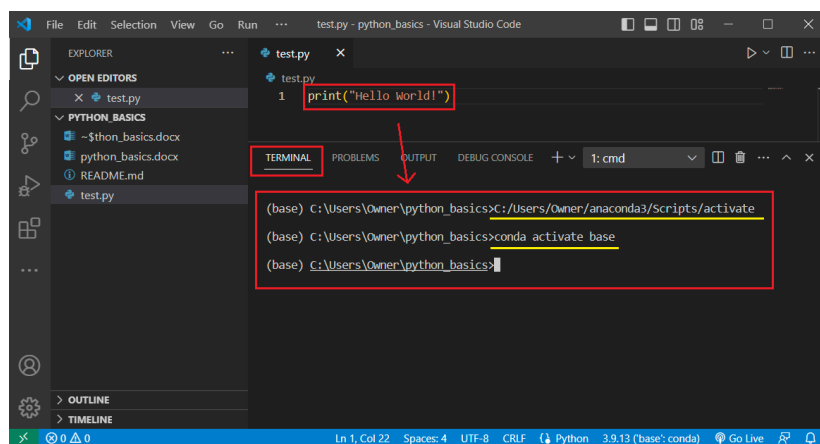
圖：在右下角看到安裝 Anaconda 的預設環境 (base)



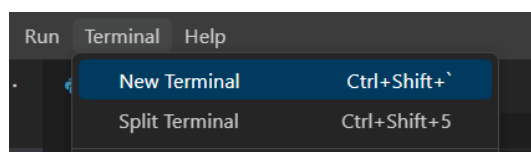
圖：按下右下角的 conda 版本圖示，可以看到安裝過的環境列表，可自由切換

建立與執行 .py 檔案

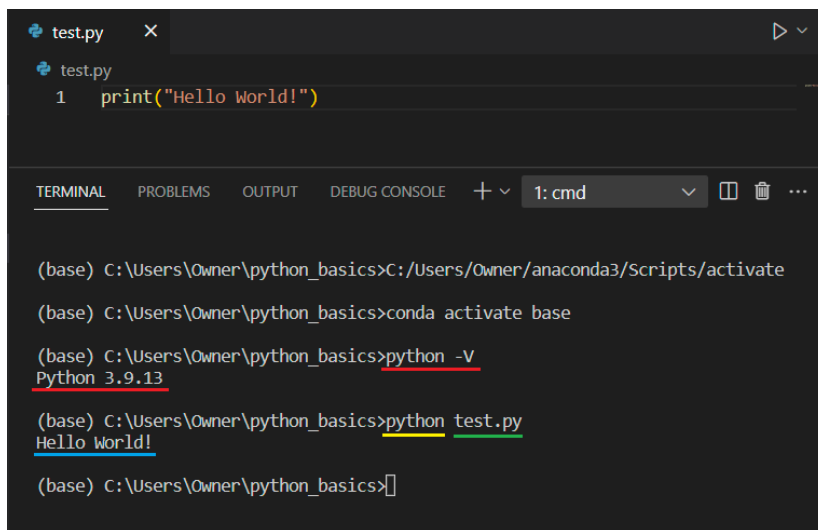
先在剛才的 test.py 檔案中，輸入「`print("Hello World!")`」，按下快速鍵「`Ctrl + s`」來儲存文字內容（程式碼）。



圖：按下快速鍵「`Ctrl + ~`」，顯示終端機（Terminal），並啟動 Conda 環境



圖：或是在功能列，選擇「Terminal」→「New Terminal」，效果亦同



```
test.py x
test.py
1 print("Hello World!")

TERMINAL PROBLEMS OUTPUT DEBUG CONSOLE + v 1: cmd v
(base) C:\Users\Owner\python_basics>C:/Users/Owner/anaconda3/Scripts/activate
(base) C:\Users\Owner\python_basics>conda activate base
(base) C:\Users\Owner\python_basics>python -V
Python 3.9.13
(base) C:\Users\Owner\python_basics>python test.py
Hello World!
(base) C:\Users\Owner\python_basics>
```

圖：先確認 base 環境的 python 版本，然後執行 test.py 程式，出現「Hello World!」

參考資料

[1] 維基百科：Python

<https://zh.wikipedia.org/zh-tw/Python>

[2] How To Install Python 3 on Windows 10

<https://phoenixnap.com/kb/how-to-install-python-3-windows>

[3] Windwos&Mac Python 初學者為什麼選擇 Anaconda 為開發環境呢？-入門系列

<https://medium.com/誤闖數據叢林的商管人 zino/windwos-mac 初學者為什麼選擇 aanconda-為>

[開發環境呢-入門系列-f892697f1a53](https://medium.com/誤闖數據叢林的商管人 zino/windwos-mac 初學者為什麼選擇 aanconda-為)

[4] Conda create environment and everything you need to know to manage conda virtual environment

<https://www.machinelearningplus.com/deployment/conda-create-environment-and-everything-you-need-to-know-to-manage-conda-virtual-environment/>

[5] 【程式】Python 初學者的開發環境推薦

<https://liah.pixnet.net/blog/post/227668100-python-初學者的開發環境推薦>

[6] VS Code Python 擴充套件改進環境設定流程，並強化 IntelliSense 對單元測試的支援

<https://www.ithome.com.tw/news/155413>

Module 2. 資料型態與變數

資料型態

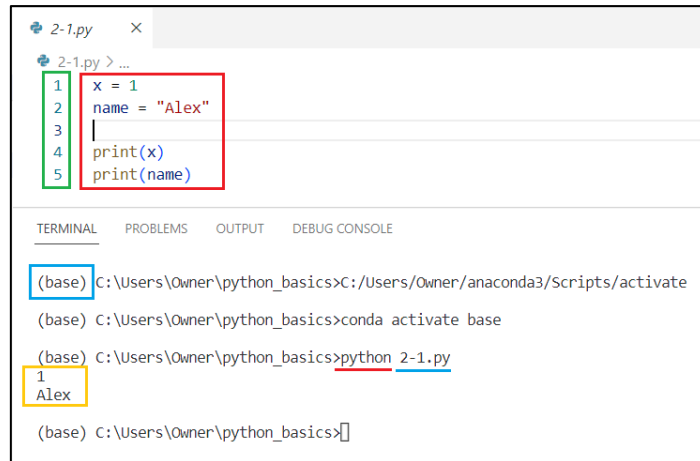
資料型態	語法	說明
文字 Text	<code>str</code>	<ul style="list-style-type: none">● 用於建立字串(string)或字元(character)。● 通常會使用雙引號("")或單引號('')將其前後包括起來。● 例如 "Hello World!"或是 'Hello World!'。● 一個字以上，可以視為「序列」(Sequece)
數值 Numeric	<code>int</code> , <code>float</code>	<ul style="list-style-type: none">● 常見的如整數(1234)或浮點數(3.1415926)。
序列 Sequence	<code>list</code> , <code>tuple</code> , <code>range</code>	<ul style="list-style-type: none">● <code>list</code> 可以一次儲存多筆資料在單一變數(變數可以想像成資料暫存的地方或是替身)，使用中括號「[]」將資料整理在一起，例如 ["人", "帥", "真", "不賴"]，或是 [9, 4, 8, 7]等。● <code>tuple</code> 跟 <code>list</code> 很像，只是裡面的值不能改變，使用中括號「()」將資料整理在一起，例如 ("人", "帥", "真", "好")，或是 (, 4, 8, 7)。● <code>range</code> 可以建立一個數值的列表，通常會搭配 <code>for</code> 迴圈語法來使用，將數值一個一個拿出來用，例如 <code>range(10)</code>，它會等同於 [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]。
字典 Dictionary	<code>dict</code>	<ul style="list-style-type: none">● 以 <code>key:value</code> 的格式儲存資料，同時可以儲存很多 <code>key:value</code> 組合的資料。● 使用大括號「{}」將資料整理在一起。● 例如

		<pre>{ "name": "Darren", "age": 18, }</pre>
集合 Set	set	<ul style="list-style-type: none"> ● 類似 list 和 tuple，但它儲存的資料「不能重複」。 ● 使用大括號「{ }」將資料整理在一起。 ● 有別於 dict，它不是使用 key:value 的方式來儲存資料。 ● 例如 { "蘋果", "橘子", "水梨" }。
布林 Boolean	bool	<ul style="list-style-type: none"> ● 用來表示真(True)或假(False)。 ● 通常用於條件判斷的結果為真或是假。
空值 None	None	<ul style="list-style-type: none"> ● 通常用於變數初次建立的時候，一時之間不確定要給什麼值，可以先給 None。

變數

變數 (variable) 就是一個暫存資料 (值) 的地方，會將資料儲存在電腦記憶體當中，原則上什麼值都給可以賦予、分配 (assign) 給它。

2-1.py
<pre>x = 1 name = "Alex" print(x) print(name)</pre>



The image shows a code editor window with a file named '2-1.py'. The code is as follows:

```
1 x = 1
2 name = "Alex"
3 |
4 print(x)
5 print(name)
```

Below the code editor is a terminal window. The terminal shows the following commands and output:

```
(base) C:\Users\Owner\python_basics>C:/Users/Owner/anaconda3/Scripts/activate
(base) C:\Users\Owner\python_basics>conda activate base
(base) C:\Users\Owner\python_basics>python 2-1.py
1
Alex
(base) C:\Users\Owner\python_basics>
```

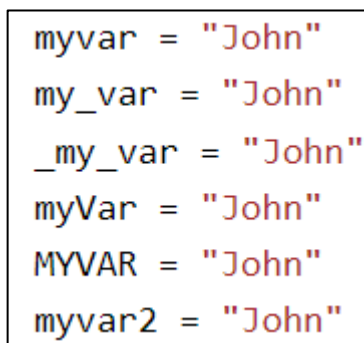
The terminal output shows the number '1' on the first line and the string 'Alex' on the second line, which are the results of the print statements in the code.

圖：綠框為程式碼行號，紅框為程式碼，藍框代表 conda 環境，
紅底線是指令，藍底線是檔名，黃框是程式執行結果(輸出)

變數命名規則、關鍵字與縮排

變數命名規則

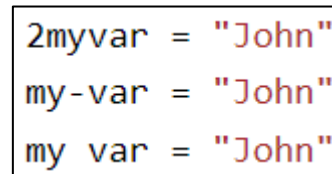
- 由字母 (A-z)、數字 (0-9) 符號和底線 (__) 組成。
- 必須以字母 (A-z) 或底線 (__) 開頭，不能以數字開頭。
- 區分大小寫，name、Name、NAME 是不同的變數。



A list of variable names that are correctly formatted according to Python naming rules:

```
myvar = "John"
my_var = "John"
_my_var = "John"
myVar = "John"
MYVAR = "John"
myvar2 = "John"
```

圖：正確的變數命名方式



A list of variable names that are incorrectly formatted according to Python naming rules:

```
2myvar = "John"
my-var = "John"
my var = "John"
```

圖：錯誤的變數命名方式

關鍵字

所謂關鍵字 (keywords)，是用來保留給程式語言使用的內建語法或是變數名稱，不適合作為一般變數來使用，以下列出常見的 Python 關鍵字：

as	assert	break
class	continue	def
del	elif	else
except	False	finally
for	from	global
if	import	in
is	lambda	None
nonlocal	not	or
pass	raise	return
True	try	while
with	yield	and

縮排

Python 的縮排 (indent) 以冒號「:」來代表程式區塊 (block) 的開始，縮排通常都是使用 4 個空白鍵 (有時候會看到有人用 2 個，建議一行用 4 個)，例如：

sum = 0	宣告 sum 變數的值為 0
score = 60	宣告 score 變數的值為 60
if score >= 60:	從 : 右方開始，到) 右方，都是程式區塊，
print('及格')	程式碼前放了 4 個空白 () 縮排，
	代表都在同一個程式區塊內。
(下一行程式)	如果下一行程式前面還有縮排，
(下一行程式)	代表都在同一個程式區塊範圍內。
sum = sum + score	這是下一行程式，不受上方程式區塊的範圍影響

運算子

運算子 (operators) 就是大家常見的 + (加)，- (減)，* (乘)，/ (除)，% (相除取餘數)，= (賦值)，> (大於)，< (小於)，== (等於) ... 等符號，常見的有：

- 算術運算子 (Arithmetic operators)
- 賦值運算子 (Assignment operators)
- 比較運算子 (Comparison operators)

- 邏輯運算子 (Logical operators)
- 成員運算子 (Membership operators)
- 身分運算子 (Identity operators)
- 位元運算子 (Bitwise Operators)

算術運算子

運算符號	說明	範例
+	加	9 + 4 回傳 13
-	減	9 - 4 回傳 5
*	乘	9 * 4 回傳 36
/	除	9 / 4 回傳 2.25
%	相除取得餘數	9 % 4 回傳 1
**	多少的幾次方	9 ** 4 回傳 6561
//	相除取得整數	9 // 4 回傳 2

賦值運算子

運算符號	說明	範例
=	單純賦值 (值由右往左給)	a = 3 b = 4 c = a + b c 的值為 7
+=	計算加法後賦值	a = 3 b = 4 b += a 等同於 b = b + a b 的值為 7

-=	計算減法後賦值	a = 3 b = 4 b -= a 等同於 b = b - a b 的值為 1
*=	計算乘法後賦值	a = 3 b = 4 b *= a 等同於 b = b * a b 的值為 12
/=	計算除法後賦值	a = 3 b = 4 b /= a 等同於 b = b / a b 的值為 1.3333333333333333
%=	計算取餘數後賦值	a = 3 b = 4 b %= a 等同於 b = b % a b 的值為 1
**=	計算幾次方後賦值	a = 3 b = 4 b **= a 等同於 b = b ** a b 的值為 64
//=	計算取得整數後賦值	a = 3 b = 4 b //= a 等同於 b = b // a b 的值為 1

比較運算子

運算符號	說明	範例
==	等於	5 == 5 回傳 True

!=	不等於	5 != 6 回傳 True
>	大於	5 > 6 回傳 False
<	小於	5 < 6 回傳 True
>=	大於等於	5 >= 6 回傳 False
<=	小於等於	5 <= 6 回傳 True

邏輯運算子

運算符號	說明	範例
and	若是兩邊都是 True，則回傳 True	name = 'Alex' age = 18 if name == 'Alex' and age == 18: print('正確')
or	若是兩邊其中一個是 True，則回傳 True	name = 'Alex' age = 18 if name == 'Alex' or age == 16: print('正確')
not	若變數的值是 True，則回傳 False；若變數的值是 False，則回傳 True	flag = False if not(flag): print('從 False 變成 True')

成員運算子

運算符號	說明	範例
in	在序列 (sequence) 中找到指定的值，則回傳 True，反之則回傳 False。	myList = [9, 5, 2, 7] if 2 in myList: print('2 在 list 當中')
not in	在序列 (sequence)	myList = [9, 5, 2, 7]

	中沒有找到指定的值，則回傳 True，有找到則回傳 False。	<pre>if 3 not in myList: print('3 不在 list 當中')</pre>
--	----------------------------------	--

身分運算子

運算符號	說明	範例
is	當兩個變數是參照同一個物件(同一個記憶體位置)，回傳 True。	<pre>x = 123 y = x # y 參照 x 的記憶體位置 if x is y: print('x 和 y 是同一物件')</pre>
is not	當兩個變數不是參照同一個物件(同一個記憶體位置)，回傳 True。	<pre>x = 123 y = 456 # y 隨機佔用記憶體位置 if x is not y: print('x 和 y 不是同一物件')</pre>

位元運算子

提示：位元運算會將運算元轉成二進制，再進行運算元之間的布林運算

運算符號	說明	範例
&	AND 運算 (相同為 1，就是 1)	<pre>x = 1 y = 3 print(x & y) # 輸出 1</pre>
	OR 運算 (其中為 1，就是 1)	<pre>x = 1 y = 3 print(x y) # 輸出 3</pre>
^	XOR 運算 (相同為 0，相異為 1)	<pre>x = 1 y = 3 print(x ^ y) # 輸出 2</pre>
~	補數運算 (有正負號，運算才有義意)	<pre>x = +3 print(~x)# 輸出 -4</pre>
<<	位元左移	<pre>x = 4 print(x << 1) # 輸出 8</pre>

>>	位元右移	<code>x = 4</code> <code>print(x >> 2) # 輸出 1</code>
----	------	--

運算子優先順序

運算子	說明
()	括號
**	冪、指數
+x -x ~x	一元運算子加號、一元運算子減號、位元運算的 not
* / // %	乘、除、除(只取商數)、相除取餘數
+ -	加、減
<< >>	位元左移、位元右移
&	位元運算 AND
^	位元運算 XOR
	位元運算 OR
== != > >= < <=	比較運算子
is is not	身分運算子
in not in	成員運算子
not	(邏輯運算)True 變成 False，或是 False 變成 True
and	(邏輯運算)且
or	(邏輯運算)或

註解

註解 (comments) 在程式碼當中，通常是用於對撰寫的程式碼進行文字說明，或是希望其它行的程式碼暫時不要執行，先用單行註解或多行註解先將其關閉，待需要的時候才打開。

單行註解	多行註解	多行註解
<code># 單行註解</code>	<pre>''' 多行註解 多行註解 多行註解 '''</pre>	<pre>""" 這也是多行註解 這也是多行註解 這也是多行註解 """</pre>

註：

在 VS code 當中，可以反白幾行程式碼，之後再對它們按下 **Ctrl + /**，可以作到多個單行註解；如果要解除註解，對原先註解的程式碼進行反白，再按下 **Ctrl + /**。

2-2.py

```
# 算術運算子
print(9 + 4)
print(3 / 2)
print(2 ** 3)

# 賦值運算子
a = 9
b = 4
a += b # 等於 a = a + b
print(a)
c = 3
d = 2
d **= c # 等於 d = d ** c
print(d)

# 比較運算子 (用 if 判斷)
e = 4
if e == 4:
    print('e 等於 4')
if e > 2:
    print('e 大於 2')
if e != 5:
    print('e 不等於 5')

# 如果程式區塊只需要執行 1 行程式碼，則可以這麼寫
if e == 4: print('e 等於 4')
```



```
# 邏輯運算子
f = 8
name = 'Bill'
if f == 8 and name == 'Bill':
    print('ok')
if f == 6 or name == 'Bill':
    print('name 等於 Bill')

# 成員運算子
myList = [5, 5, 6, 6, 3, 3, 1, 2]
if 6 in myList:
    print('6 在 list 裡')
if 4 not in myList:
    print('4 不在 list 裡')
```

參考資料

[1] Python Variables

https://www.w3schools.com/python/python_variables.asp

[2] Python 運算符

<https://www.runoob.com/python/python-operators.html>

[3] Python Keywords

https://www.w3schools.com/python/python_ref_keywords.asp

Module 3. 控制結構

循序

循序 (sequence) 指的是通常在程式語言中，程式碼是由上而下、一行一行依序執行。

選擇

選擇 (selection) 則是依條件 (condition) 讓程式分流執行。

3-1.py

```
'''
條件判斷
if ... else; if ... elif ... else; True 和 False
'''

# if
num = 10
'''
可以用以下符號來進行條件判斷
== (等於)
> (大於)
>= (大於等於)
< (小於)
<= (小於等於)
!= (不等於)
'''
if num > 5:
    print("num 大於 5")

# if else
name = 'apple'
if name == 'apple':
    print('名稱是 apple')
else:
    print("名稱不是 apple")

# if elif else
name = 'darren'
if name == "alex":
```

```

    print("名稱: alex")
elif name == "bill":
    print("名稱: bill")
elif name == "carl":
    print("名稱: carl")
else:
    print("Not found")

# 補充: True (真) 和 False (假、偽)
is_available = True
if is_available == True:
    print("真")
else:
    print("假")
# 也可以不用加「 == True」
if is_available:
    print("真")
else:
    print("假")

```

重複

重複 (repetition) 是指當有程式區塊 (block) 需要重複執行，重複的條件為真 (True)，則該程式碼將繼續重複執行，直到條件為假 (False) 時，才會停止重複執行。這種不斷迭代 (iteration)、重複 (repeat) 執行的結構，稱為迴圈 (loop)。

3-2.py

```

'''
while 迴圈
for 迴圈
'''

# while 迴圈
count = 1
while count <= 5:

```

```
print(count, end="")
count = count + 1 # 或是寫成 count += 1

# for 迴圈 01
...
用法
range(n, m-1)

說明
會走訪 n 到 m-1 的數字
...
for i in range(5, 8):
    print(i, end = ",")

# for 迴圈 02
...
用法
range(n, m-1, step)

說明
以每 step 為間距，走訪 n 到 m-1 的數字
...
for i in range(5, 20, 2):
    print(i, end = ",")
```

在執行迴圈的過程中，若是有跳出迴圈、不繼續執行的需要，可以使用 **break** 來中斷迴圈；若是在特定條件下，不執行當前的程式區塊，或是只執行局部的程式區塊，而後立刻跳到下一個項目、下一筆資料，可以使用 **continue**。

若是在迴圈當中撰寫了條件，卻一時不確定要 **break** 或是 **continue**，抑或打算在條件中撰寫其它程式碼，暫時先保留程式區塊可用的範圍，等到想好了再補上去，則可以使用 **pass**。

3-3.py

```
# break
```

```
...
```

說明

當偵測到字母 `t` 時，就會強制結束迴圈

```
...
```

```
for char in 'content':  
    if char == 't':  
        break  
    print(char, end="")
```

```
# continue
```

```
...
```

說明

當偵測到字母 `t` 時，
會跳過本次迴圈剩下的程式碼 `print(char)`，
但不會結束迴圈，仍然會進入下一圈繼續執行

```
...
```

```
for char in 'content':  
    if char == 't':  
        continue  
    print(char, end="")
```

```
# pass
```

```
...
```

說明

當偵測到字母 `t` 時，會忽略該條件，繼續像正常迴圈一樣運程序

備註

有時候寫 `pass`，是為了將某塊或某行列入 `to-do`

```
...
```

```
for char in 'content':  
    if char == 't':  
        pass  
    print(char, end="")
```

Module 4.字串、串列、元組、字典、集合

字串

- 「字串」(string) 如同一串字，同時由兩個以上的字元 (character) 組成。
- 字串需要用「兩個單引號」或「兩個雙引號」包起來，例如 '星期日' 或 "星期日"。
- 包住字串的兩個引號一定要一樣，不能一個單引號、一個雙引號，要同時都是單引號，或是同時都是雙引號。
- 如果包住的字只有一個，則稱之為「字元」(character)。

4-1.py

```
# 字串變數初始化
string01 = "1,2,3,4"
print(string01)

# 替換字串
'''
用法
string.replace(str1, str2)
說明
將 string 中的 str1 替換成 str2
'''
string02 = "Alex"
string02 = string02.replace('ex', 'len')
print(string02)

# 去除字串兩側空格
'''
用法
string.strip()
說明
去除字串 string 左、右兩邊的空格
```

```
'''
string03 = "      __ccc__      "
print(string03)
print(string03.strip())

# 字串變成小寫或大寫
'''

用法
string.lower()
說明
將字串 string 裡的字母全部改成小寫
'''

print("CAR".lower())

'''

用法
string.upper()
說明
將字串 string 裡的字母全部改成大寫
'''

print("good".upper())
```

補充：字串格式化

輸出結果的時候，若是需要嵌入必要的文字，建立字串樣版 (string template)，或是嘗試將文字進行排列與調整，可以將字串格式化。

使用 %

語法	說明
%s	以 str() 函式輸出文字
%f	以 浮點數 輸出文字
%d	以 十進制 輸出文字
%%	輸出百分比 (類似跳脫字元)

4-1-1.py

```
'''
格式化字串 - 使用 %
'''

# 多組文字
msg = '%s, %s!' % ('Hello', 'World')
print(msg)

# 整數
msg = 'I am %d years old.' % 5
print(msg)

# 文字與整數
msg = '%s is %d years old.' % ('Alex', 18)
print(msg)

# 指定寬度（維持 10 個字元長度，預設向右對齊）
msg = ' [%10s]' % 'Hello'
print(msg)

# 靠左對齊（維持 10 個字元長度，向左對齊）
msg = ' [%-10s]' % 'Hello'
print(msg)

# 指定浮點數位數
msg = ' [%8.3f]' % 12.3456
print(msg)

# 指定文字長度上限（只有文字，才在格式化字串中加入「.」來限定字串長度）
msg = ' [%.3s]' % 'Hello'
print(msg)

# 空白補 0
msg = ' [%06.2f]' % 3.1415926
print(msg)
```


使用 .format()

4-1-2.py

```
'''
格式化字串 - 使用 string.format()
'''

# 嵌入文字
msg = '{} {}'.format('Hello', 'World')
print(msg)

# 改變參數順序
msg = '{1} {0}'.format('Hello', 'World')
print(msg)

# 指定寬度
msg = '{:10}'.format('Hello')
print(msg)

# 靠右對齊
msg = '{:>10}'.format('Hello')
print(msg)

# 靠左對齊
msg = '{:<10}'.format('Hello')
print(msg)

# 置中對齊
msg = '{:^10}'.format('Hello')
print(msg)

# 指定浮點數位數
msg = '{:8.3f}'.format(12.3456)
print(msg)

# 指定文字長度上限（只有文字，才在格式化字串中加入「.」來限定字串長度）
msg = '{:.3}'.format('Hello')
```

```
print(msg)

# 空白補 0
msg = '{:06.2f}'.format(3.1415926)
print(msg)
```

使用 f-string (Python 版本 > 3.6)

```
4-1-3.py

# f-string (又稱作 formatted string literals, version >= 3.6)
name = "Darren"
age = 18
result = f"name: {name}, age: {age}"
print(result)

# 靠左對齊、靠右對齊
result = f"name: [{name:<10}], age: [{age:>10}]"
print(result)

# 靠左對齊、靠右對齊，字數不足，補「」(沒寫補什麼，預設空格)
result = f"name: [{name:  <10}], age: [{age:  >10}]"
print(result)

# 靠左對齊、靠右對齊，
result = f"name: [{name:^10}], age: [{age:^10}]"
print(result)

# 指定文字長度上限 (只有文字，才在格式化字串中加入「.」來限定字串長度)
result = f"name: [{name:^10.2}]"
print(result)
```

補充：Input

若是希望能在程式執行期間，讓使用者可以自行輸入資料（類似提示對話視窗 prompt），之後整合到程式碼之後輸出結果，可以使用 `input()` 函式。

4-1-4.py

```
# 取得使用者輸入的資料
data = input('輸入文字後，按下 Enter: ')
print(data)

# 將使用者輸入強制轉型成 int
num = int(input('請輸入數字: '))
print(type(num))
print(num)
```

串列

- 「串列」(list) 或稱為「列表」，跟其它程式語言當中的「陣列」(array) 很類似。
- 使用「中括號」[] 將資料儲存起來。
- 可以想像成一個放置藥丸的盒子，從星期天開始。
- 例如【'星期天', '星期一', '星期二', '星期三', '星期四', '星期五', '星期六'】。
- 這個藥盒子內，有各自存取的代號(位置)，例如 0 代表「星期天」，1 代表「星期一」，2 代表「星期二」，依此類推。
- 這個存取的代號 0、1、2...等等，稱之為「索引」(index)。
- 可以儲存各種資料型態

4-2.py

```
# 初始化一個 list
ids = [1, 2, 3, 4, 5, 6]
print(ids)

# 新增元素在 list 尾端
ids.append(7)
print(ids)

# 修改索引位置的元素
ids[4] = 99
print(ids)
```

刪除索引位置的元素

```
ids.pop(4)
```

```
print(ids)
```

新增元素 9 在指定索引 1，其餘元素往後移

```
ids.insert(1, 9)
```

```
print(ids)
```

移除指定的值

```
ids.remove(9)
```

```
print(ids)
```

補充：字串分割成 list

...

用法

```
string.split()
```

說明

默認以空格、換行字元分割字串 `string`，回傳 `list`

...

```
string04 = "1,2,3,4"
```

```
list04 = string04.split(',')
```

```
print(list04)
```

補充：將 list 元素合併成字串

...

用法

```
string.join(seq)
```

說明

以 `string` 為分隔符號，將 `seq` 中的元素串起來成為一個新的字串

...

```
list05 = ['古道', '西風', '瘦馬']
```

```
string05 = '-'.join(list05)
```

```
print(string05)
```

```
string05 = ''.join(list05)
print(string05)
```

序列物件分割 (Slicing)

- 序列物件包含 String、List 及 Tuple 等，我們可透過其順序取得元素
- 中括號為常見物件的索引值取值 (Indexing) 或分割之語法
 - 索引值取值指的是取出一個值
 - 分割指的是取出一部分值

索引值：	0	1	2	3	4	5	6	7	8	9	10	11
	'H'	'e'	'l'	'l'	'o'	','	'W'	'o'	'r'	'l'	'd'	'!'
索引值：	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

圖：把 sequence 當成 list 來存取

4-3.py

```
# 初始化一個字串
myStr = 'Hello,World!'

...
透過冒號 (:) 進行分割
...

# 從頭取到 5 (不包含第 5 個元素，實際為 0 ~ 4)
print( myStr[0:5] ) # 索引從 0 開始，到 5 - 1 的索引位置

# 從頭取到 5 (不包含第 5 個元素，實際為 0 ~ 4)
print( myStr[:5] ) # 冒號前面留空，效果跟 [0:5] 一樣

# 從 7 取到尾
print( myStr[7:] ) # 冒號後面留空

# 從 7 取到 9 (不包含第 9 個元素，實際為 7 ~ 8)
print( myStr[7:9] )
```

```

...
使用負號
...
# 從倒數第 10 取到倒數第 7（不包含倒數第 7 的元素）
print( myStr[-10:-7] )

# 從倒數第 5 取到尾
print( myStr[-5:] )

# 從頭取到倒數第 7（不包含倒數第 7 的元素）
print( myStr[:-7] )


# 取得檔案全名
string06 = "/home/darren/ebook.txt"
list06 = string06.split("/")
print(list06[-1])


# 搜尋字串
string07 = 'believe'
result07 = string07.find('lie')
...
用法
string.find(str)
說明
回傳 string 第一次在字串 string 中出現的 index，若找不到則回傳 -1
...
print(result07)

```

補充：List Comprehension

類似串列生成（list generation）的概念，僅使用一行程式碼，就可以產生一個 list。

一般 list 用法	List Comprehension
<pre>list01 = [] for x in range(10): list01.append(x) print(list01)</pre>	<pre># 一行直接建立 list list02 = [x for x in range(10)] print(list02) # 還可以設定條件 list03 = [x for x in range(10) if x % 2 == 0] print(list03)</pre>

元組

- 元組 (tuple) 。
- 有序號、索引概念。
- 允許重覆的值。
- 跟 list 很類似，只是不可更改元組的資料。
- 可以使用 for 迴圈讀出一筆筆的資料。

4-4.py
<pre># Tuple 初始化: 第一種 myTuple01 = ("人", "帥", "得體") print(myTuple01) # Tuple 初始化: 第二種 myTuple02 = "哆", "啦", "A", "夢" print(myTuple02) # 透過指定索引輸出值 print(myTuple01[1]) print(myTuple02[1]) # 複數變數修改值 a = 10 b = 20 print("交換前: a = {}, b = {}".format(a, b))</pre>

```
a, b = b, a
print("交換後: a = {}, b = {}".format(a, b))

# list 可以修改指定索引的值
myList = ["人", "帥", "任性"]
myList[2] = "真好"
print(myList)

# tuple 不可以修改指定索引的值
myTuple = ("人", "帥", "任性")
myTuple[2] = "真好"
print(myTuple)

# 用 for 迴圈逐一輸出資料
for value in myTuple01:
    print(value)

# 用 len 計算 tuple 資料個數
print( len(myTuple02) )

# 因為無法修改 tuple 的資料，所以也無法指定索引進行刪除
del myTuple01[1]
print(myTuple01)

# 只能從記憶體刪除整個 tuple 變數
del myTuple01
print(myTuple01)
```

字典

- 「字典」(dict)，跟其它程式語言的「物件」(Object) 很類似。
- 透過大括號 { } 來存放資料。
- 結構以 **key:value** 來儲存資料，key 可以想像成一個變數。
- 一個字典可以放置多組 key:value，每組之間用逗號 (,) 分隔。

- 例如 {'name': 'Darren', 'age': 18, 'height': 172, 'weight': 100}
- 其中 'name' 就是字典其中的 key，而 'Darren' 就是此 key 對應的 value。
- 與 List 不同的是，Dict 通常是透過 key 來存取 value，沒有特定的 index。

4-5.py

```
# 初始化 dict
dict01 = {"蘋果": 100, "橘子": 20, "水梨": 50}
print(dict01)

...

也可以透過排版，寫成：

dict01 = {
    "蘋果": 100,
    "橘子": 20,
    "水梨": 50
}
...

# 印出蘋果的價格
print(dict01["蘋果"])

# 修改橘子的價格
dict01["橘子"] = 30
print(dict01["橘子"])

# 刪除 水梨
del dict01["水梨"]
print(dict01)

...

有時候會遇到比較複雜的結構，需要一層一層去走訪，取得資料
...
```

```
dict02 = {
    'name': 'Darren',
    'age': 18,
    'info': {
        'nickname': 'boatman',
        'favorite_role': 'Doraemon',
        'phone_number': [
            '0911111111',
            '0922222222',
            '0933333333'
        ]
    }
}

# 取得 nickname
print( dict02['info']['nickname'] )

# 取得所有 phone_number
for number in dict02['info']['phone_number']:
    print(number)
```

集合

- 無序號、索引概念
- 無索引(隨機出現)
- 沒有重覆(一樣的只會出現一次)

4-6.py

```
# Set 初始化: 第一種
mySet01 = {"網路", "爬蟲", "真好玩"}
print(mySet01)

# Set 初始化: 第二種 (裡面放 tuple)
mySet02 = set( ("網路", "爬蟲", "真好玩") )
print(mySet02)

# 一筆筆讀資料 : for 迴圈
```

```

for data in mySet02:
    print(data)

# 總共有幾筆資料: len()
print( len(mySet02) )

# 因為沒有序號、索引的概念，所以無法透過指定索引輸出
print( mySet01[0] )

# 添加一筆資料: add()
mySet01.add("嗎?")
mySet02.add("對不對?")

print(mySet01)
print(mySet02)

# 此時新增重複的，資料不會增加
mySet01.add("真好玩")
print(mySet01)

# 添加多筆資料: update()
mySet01.update(["甲", "乙", "丙"])
mySet02.update(["子", "丑", "寅"])
print(mySet01)
print(mySet02)

# 查詢陣列中有沒有我要的資料: in (只回傳 true 或 false)
print( "甲" in mySet01 )
print( "甲" in mySet02 )

if "乙" in mySet01:
    print("有資料")
else:
    print("找不到資料")

# 刪除元素: discard() or remove()
mySet01.discard("丙")

```

```
print(mySet01)
mySet02.remove("丑")
print(mySet02)

# 清空: clear()、del
'''clear() 會清空 set 變數，但變數依然存在，所以會印出空 set'''
mySet01.clear()
print(mySet01)
'''del 會將 set 變數從記憶體中刪除，所以刪除完後，變數會變成未宣告的狀態'''
del mySet02
print(mySet02)
```

參考資料

[1] Python 基礎教學

<https://www.runoob.com/python/python-tutorial.html>

[2] String Formatting in Python

<https://www.scaler.com/topics/python/string-formatting-in-python/>

[3] Python 字串格式化教學與範例

<https://officeguide.cc/python-string-formatters-tutorial/>

[4] 如何使用 Python 進行字串格式化

<https://blog.techbridge.cc/2019/05/03/how-to-use-python-string-format-method/>

[5] Python String Formatting

https://www.w3schools.com/python/python_string_formatting.asp

[6] List Comprehension: Python 的 For Loop 怎樣使用？

<https://pythonviz.com/basic/list-comprehension/>

Module 5. 函式

函式 (function) 將經常使用的程式碼打包起來，變成一種模組，每執行一次函式，相當於把模組的程式碼全部執行過，透過 `def` 關鍵字來定義。

基本用法

函式的定義方式

```
def 函式名稱():
```

 陳述句(statement, 代表程式要執行的動作)

執行函式，直接使用函式名稱，後面加上()

函式名稱()

變數傳遞

回傳 值(結果)	傳遞變數
<p># 傳遞在函式當中執行的結果</p> <pre>def 函式名稱(): return 您的執行結果</pre>	<p># 傳遞在函式當中執行的結果</p> <pre>def 函式名稱(參數[, 參數 2,...]): return 整合變數後的結果</pre>
<p># 函式執行完畢後，會將結果回傳給變數</p> <p>變數 = 函式名稱()</p>	<p># 函式執行完畢後，會將結果回傳給變數</p> <p>變數 = 函式名稱(變數[, 變數 2,...])</p>

5-1.py

```
'''不回傳 值'''  
# 定義函式  
def show_name():  
    print("Doraemon")  
  
# 執行函式  
show_name()  
  
'''回傳 值'''  
# 定義函式  
def get_name():  
    name = "Doraemon"  
    return name  
# 也可以寫成 return "Doraemon"
```

```

# 執行函式
result = get_name()

# 輸出回傳結果
print(result)

'''傳遞變數'''

# 定義函式
def get_greeting(name):
    return "Hello, " + name

# 執行函式
result = get_greeting("Darren")

# 輸出回傳結果
print(result)

```

區域變數與全域變數

- 當程式區塊 (block) 內外都有相同的變數名稱：
 - 程式區塊內部的變數，叫作區域變數 (local variable)。
 - 程式區塊外部的變數，會稱為外部變數。
 - 如果變數在程式碼的最外層，沒有被任何程式區塊 (block) 包起來，該變數叫作全域變數 (global variable)。
- 想在函式內部使用、修改外部變數的值，必須在變數前加上 `global` 關鍵字。
- 不能在加上 `global` 的時候進行變數初始化。

區域變數無法修改全域變數	區域變數可以修改全域變數
<pre> # 全域變數 name = 'Bill' # 定義函式 </pre>	<pre> # 全域變數 name = 'Bill' # 定義函式 </pre>

<pre>def 函式名稱(): # 區域變數 name = 'Doraemon' # 執行函式 函式名稱() # 輸出 name print(name) # 輸出 Bill</pre>	<pre>def 函式名稱(): # 使用 global 關鍵字， # 讓函式可以取存全域變數的值 global name # 區域變數 name = 'Doraemon' # 執行函式 函式名稱() # 輸出 name print(name) # 輸出 Doraemon</pre>
---	---

<pre>5-2.py '''區域變數無法修改全域變數''' # 全域變數 name = "Bill" # 定義函式 def set_name(): # 區域變數 name = "Doraemon" # 執行函式 set_name() # 輸出結果 print(name) '''區域變數可以修改全域變數''' # 全域變數 name = "Bill" # 定義函式</pre>

```
def set_name():
    ...

    想在函式內部使用、修改外部變數的值，
    必須在變數前加上 global 關鍵字，
    但是不能在加上 global 的時候進行變數初始化
    ...

    global name
    name = "Doraemon"

# 執行函式
set_name()

# 輸出結果
print(name)
```

匿名函式 Lambda

所謂匿名函式 (anonymous function)，許多程式語言都有支援，在程式語言 Python 當中叫作 Lambda，只需要撰寫一行的程式碼，就有一般函式的效果，相對其它函式定義方法來得簡潔。

將變數作為函式名稱 = **lambda** 參數 1[, 參數 2,...] : 執行回傳結果

一般函式	Lambda
<pre>def add(x, y): return x * y print(add(3, 5))</pre>	<pre>add = lambda x, y : x * y print(add(3, 5))</pre>
<pre>def abs(x): if x >= 0: return x else: return -x print(abs(5)) print(abs(-5))</pre>	<pre>abs = lambda x : x if x >= 0 else -x print(abs(5)) print(abs(-5))</pre>
<pre>def product(x, y):</pre>	<pre>print((lambda x,y : x * y)(4, 5))</pre>

<pre> return x * y print(product(4, 5)) </pre>	
---	--

Lambda 通常會與 **一次性影響所有序列資料** 的函式整合使用，例如 map 或 sorted 等。

map	sorted
<pre> # 與 map 函式一起使用 myList = list(map(lambda x : x ** 2, [1, 2, 3, 4, 5])) print(myList) </pre>	<pre> # 與 sorted 函式一起使用 myList = [{"name": "Alex", "age": 18}, {"name": "Bill", "age": 25}, {"name": "Carl", "age": 21}, {"name": "Darren", "age": 10},] list_sorted = sorted(myList, key=lambda d: d['age'], reverse=False) print(list_sorted) </pre>

5-3.py
<pre> # 帶入參數，並將計算結果直接回傳 add = lambda x, y : x * y print(add(3, 5)) # 回傳之前的條件判斷（需要練習一下語法，否則一開始不好理解） abs = lambda x : x if x >= 0 else -x print(abs(5)) print(abs(-5)) # 不使用變數當作函式名稱，直接執行 print((lambda x,y : x * y)(4, 5)) # 與 map 函式一起使用 list_map = list(map(lambda x : x ** 2, [1, 2, 3, 4, 5])) </pre>

```
print(list_map)

# 與 sorted 函式一起使用
myList = [
    {"name": "Alex", "age": 18},
    {"name": "Bill", "age": 25},
    {"name": "Carl", "age": 21},
    {"name": "Darren", "age": 10},
]
list_sorted = sorted(myList, key=lambda d: d['age'], reverse=False)
print(list_sorted)
```

雖然匿名函式有它的便利性，若是撰寫的程式邏輯相當複雜，還是比較適合使用一般函式，比較容易維護。

參考資料

- [1] [Python 教學]5 個必知的 Python Function 觀念整理
<https://www.learncodewithmike.com/2019/12/python-function.html>
- [2] Wiki - 匿名函式
<https://zh.wikipedia.org/zh-tw/匿名函数>
- [3] Python- lambda 函式
<https://medium.com/seaniap/python-lambda-函式-7e86a56f1996>
- [4] Python3 map() 函数
<https://www.runoob.com/python3/python3-func-map.html>
- [5] Python3 sorted() 函数
<https://www.runoob.com/python3/python3-func-sorted.html>

Module 6.物件導向程式設計

基本認識物件導向

物件導向程式設計 (Object-Oriented Programming) 是幾乎把所有在 Python 中的東西都看成物件 (object)，擁有它自己的屬性 (properties) 和方法 (methods)。

例如車子有自己的烤漆顏色、4 門或 5 門、車身的寬高、安全氣囊數量等

等，這些是屬性；車子可以按下喇叭、煞車、轉彎、加速等，都是它的方法。例如人類有自己的身高、體重、性別、髮色、膚色、牙齒顆數、是否有戴眼鏡等，這些是屬性；人類可以跑、跳、說話、睡覺等，這些是方法。

任何東西都擁有自己的屬性和方法，將這樣的概念用在程式設計上，就叫做物件導向程式設計。

類別

類別 (class) 就像是一個樣式的名稱，裡面已經事先定義好基本的、通用的屬性和方法，例如人類通常會有兩隻手、兩條腿，同時也會爬行、走路、跑、跳等。

`class` 類別名稱:

(縮排後，撰寫程式碼)

方法

方法 (methods) 跟函式 (functions) 的感覺很像，都是執行一連串在程式區塊中定義好的程式碼 (code)，在每一個類別 (class) 中，都會有一些 Python 內建且需要在一開始定義的函式。

在建立類別的時候，常常會看到 `__init__(self)`，指的是建立物件 (或稱為實體) 的時候，預設第一時間會執行的方法，有時會稱為建構子 (constructor)。例如人類剛出生的時候，通常一開始就會放聲大哭，很可能是在 `__init__(self)` 裡面，定義了放聲大哭的方法。

`class` 類別名稱:

```
def __init__(self):
    print('Python')
def 方法 1():
    return 'Hello World!'
def 方法 2():
    return 'Good job!'
```

```
物件(實體) = 類別名稱() # 建立物件(又稱實體化)時，直接輸出 Python
print(物件.方法 1()) # 輸出 Hello World!
print(物件.方法 2()) # 輸出 Good job!
```

6-1.py

```
# 建立一個家長類別
class Parent():
    def __init__(self):
        pass
    def post(self):
        return '發話中'

# 實體化 (instantiation)
obj_parent = Parent()
print(obj_parent.post())
```

屬性

屬性 (properties) 可以想像成類別內部的變數，`self` 是讓物件或是類別本身可以存取在類別當中定義好的屬性和方法。

```
class 類別名稱:
    def __init__(self, 參數 1, 參數 2):
        self.內部變數 1 = 參數 1
        self.內部變數 2 = 參數 2
```

```
物件(實體) = 類別名稱('Hello', 'World')
print(物件.內部變數 1) # 輸出 Hello
print(物件.內部變數 2) # 輸出 World
```

6-2.py

```
# 建立一個家長類別
class Parent():
    def __init__(self, height, weight):
        self.height = height
        self.weight = weight
    def post(self):
        return '發話中'

# 實體化 (instantiation)
obj_parent = Parent(165, 80)
```

```
print(obj_parent.height)
print(obj_parent.weight)
print(obj_parent.post())
```

繼承

繼承 (Inheritance) 是建立一個新的類別 (class) 時，把前一個定義好的類別內容拿過來使用，包括前一個類別的屬性和方法，同時也可以在新類別當中定義新的屬性和方法。

例如：

```
class Parent():
    def __init__(self):
        self.height = 172
        self.weight = 95
    def post(self):
        return '發話中'

class Child(Parent):
    def __init__(self): # 這裡的 __init__ 屬於 Child 類別的
        super().__init__() # 這裡將 Parent 所有屬性、方法繼承下來
    def reply(self):
        return '回話中'
    def content(self):
        return '我知道了'
```

6-3.py

```
# 建立家長類別
class Parent():
    def __init__(self, height, weight):
        self.height = height
        self.weight = weight
    def post(self):
        return '發話中'

# 建立子類別
class Child(Parent):
```

```
...
這裡的 __init__ 屬於 Child 類別的，會將 Parent 的 __init__ 覆蓋掉
...

def __init__(self, height, weight):
    # 這裡將 Parent 所有屬性、方法繼承下來
    super().__init__(height, weight)
    self.height = height
    self.weight = weight
def reply(self):
    return '回話中'
def content(self):
    return '我知道了'

# 實體化 Parent 物件
obj_parent = Parent(165, 80)
print(obj_parent.height)
print(obj_parent.weight)
print(obj_parent.post())

# 實體化 Child 物件
obj_child = Child(172, 95)
print(obj_child.height)
print(obj_child.weight)
print(obj_child.post())
print(obj_child.reply())
print(obj_child.content())
```

參考資料

[1] Python Inheritance

https://www.w3schools.com/python/python_inheritance.asp

[2] Python 面向對象（物件導向）

<https://www.runoob.com/python/python-object.html>

Module 7. 例外處理

認識例外處理

例外 (Exception) 就是程式執行過程中，發生錯誤，影響程式的正常執行，所立刻執行的事件 (event)。

通常程式執行都會被中斷，偶爾也會顯示一些造成問題的原因；也可以自訂例外發生時的程式碼，不見得一定要使用預設的警示結果。以下是個人常見的事件類別：

事件類別	說明
BaseException	基本例外事件類別
Exception	最常用的例外事件類別
IOError	輸入/輸出發生錯誤的事件類別
IndexError	序列中沒有這個索引(index)的事件類別
RuntimeError	執行過程中發生錯誤的事件類別
SyntaxError	Python 語法錯誤的事件類別
IndentationError	縮排錯誤的事件類別
Warning	顯示警告訊息的事件類別 (通常不會中斷程式)
DeprecationWarning	功能、指令、語法等不再被支援的事件類別
RuntimeWarning	可能造成程式執行錯誤的事件類別
SyntaxWarning	程式碼語法可能有問題的事件類別

try-except 陳述

例外的處理，可以使用「try: ... except: ...」語法，try: 是預設的程式區塊，當 try: 程式區塊的程式碼在執行期間發生錯誤，會直接中斷，跳到 except: 程式區塊來繼續執行。

try:

預設正常執行程式的程式區塊

except 例外事件類別名稱 1:

拋出例外時，符合例外事件類別名稱 1，則在這個程式區塊繼續執行

except 例外事件類別名稱 2:

拋出例外時，符合例外事件類別名稱 2，則在這個程式區塊繼續執行

else:

沒有例外發生的程式區塊

finally:

無論是否正常執行，最後都會執行這裡的程式區塊

7-1.py

```
# 可以連續定義數個例外處理機制，
# 最符合的例外事件類別會先拋出
try:
    print(x)
except NameError:
    print("x 尚未宣告")
except:
    print("某個東西出錯了")

# 多層例外處理：內部的例外處理會優先補捉，
# 捕捉不到，才會由在外部的例外處理來補捉
try:
    try:
        print(y)
    except NameError as err:
        print("NameError:")
        print(str(err))
except Exception as err:
    print("Exception:")
    print(str(err))

# 如果沒有錯誤，就會執行 else 程式區塊
# finally 程式區塊是無論有無錯誤，都會執行
try:
    z = 5
    print(z)
except NameError:
    print("變數尚未宣告")
else:
    print('沒有例外發生')
```



```
finally:  
    print('程式結束')
```

例外處理 raise 陳述

如果有自訂例外的需求，可以使用 `raise` 語法來拋出例外訊息。

```
7-2.py  
  
# 自訂例外處理  
x = -1  
if x < 0:  
    raise Exception("數字小於 0")  
  
# 自訂型別錯誤的例外處理  
x = "hello"  
if type(x) is not int:  
    raise TypeError("只接受整數型別")
```

參考資料

[1] Python 异常处理（例外處理）

<https://www.runoob.com/python/python-exceptions.html>

[2] Python Try Except

https://www.w3schools.com/python/python_try_except.asp

Module 8. 檔案處理

open() 函式

可以使用內建的 `open()` 函式來進行檔案處理，其中包含了 4 種檔案模式：

- `"r"`: Read（讀取檔案）
- `"a"`: Append（附加資料到檔案內容的最尾端或最後一行）
- `"w"`: Write（寫入檔案）
- `"x"`: Create（建立檔案）
- `"t"`: Text（文字模式）
- `"b"`: Binary（二進制模式，例如用於 `images` 的讀取）

讀取檔案

8-1.py

```
# 讀取檔案內容 - 舊的寫法
f = open(file="./2-1.py", mode="r", encoding="utf-8")
print( f.read() )
f.close() # 關閉檔案，否則執行期間，在關閉前，會變成唯讀狀態

# 印出 50 個 等號
print("=" * 50)

# 讀取檔案內容 - 現今常用的寫法（執行完畢會自動關閉檔案）
with open(file="./2-1.py", mode="r", encoding="utf-8") as f:
    print( f.read() )

# 印出 50 個 等號
print("=" * 50)

# 一行一行讀出來
with open(file="./2-1.py", mode="r", encoding="utf-8") as f:
    for line in f:
        print(line) # 每一行後面會用 \n 結尾，所以輸出會有多次斷行的效果
```

註：

使用 `open()` 的時候，如果參數位置沒有改變，可以不用特別寫「`file=...`」，`mode=...`」

寫入檔案

8-2.py

```
# 文字內容
text = '''教我撩妹
你長得很像我的一個朋友
像誰
...'''
```

```
# 寫入檔案
with open("./8-2.txt", "w", encoding="utf-8") as f:
    f.write(text)

# 寫入最後一行 (附加文字在檔案內容最後一行)
with open("./8-2.txt", "a", encoding="utf-8") as f:
    f.write("你長得很像我下一任女朋友")
```

刪除檔案

```
8-3.py

# 刪除檔案
import os # 匯入套件

# 檔案路徑
file_path = './8-2.txt'

# 如果指定路徑的檔案存在，則進行刪除
if os.path.exists(file_path):
    os.remove(file_path)
else:
    print('Not found')
```

參考資料

[1] Python File(文件) 方法

<https://www.runoob.com/python/file-methods.html>

[2] Python File Open

https://www.w3schools.com/python/python_file_handling.asp

Module 9. 模組

定義模組

將常用的程式碼 (通常是函式) 整合在一個 .py 檔案中，變成一個函式庫 (library)，或是重複使用的工具，在其它程式需要用到的時候，可以不用

重新撰寫，只要透過 `import` 語法將其引入、匯入，即可在當前的程式檔案裡使用。

myModule.py

```
# 計算幾次方
def pow(x, y):
    return x**y

# 簡單斷句
def segmentSentence(text):
    list_sentences = text.split(' ')
    return list_sentences
```

使用模組

匯入模組的程式檔，通常是主程式；myModule.py 就是自訂模組，被主程式匯入。

9-1.py

```
import myModule

# 計算幾次方
num = myModule.pow(2, 3)
print(num)

# 簡單斷句
txt = "I will always love you"
list_result = myModule.segmentSentence(txt)
print(list_result)
```

9-2.py

```
# 直接從模組中匯入函式
from myModule import pow, segmentSentence

# 計算幾次方
num = pow(2, 3)
print(num)
```

```
# 簡單斷句
txt = "I will always love you"
list_result = segmentSentence(txt)
print(list_result)
```

__name__ 內建變數的用法

__name__ 是內建變數，用來確認匯入模組的名稱（例如 myModule，會自動去掉 .py），如果執行的主程式檔案裡面有輸出 __name__，它的值就會是 __main__。

myModule.py	9-2.py
<pre># 計算幾次方 def pow(x, y): return x**y # 簡單斷句 def segmentSentence(text): list_sentences = text.split(' ') return list_sentences # 想先在模組裡測試，再給其它程式檔案匯入 print(pow(7, 2))</pre>	<pre># 直接從模組中匯入函式 from myModule import pow, segmentSentence # 計算幾次方 num = pow(2, 3) print(num) # 簡單斷句 txt = "I will always love you" list_result = segmentSentence(txt) print(list_result)</pre>

如果想要測試自訂模組的執行結果是否如預期，可以執行 myModule.py：

```
(base) C:\Users\Owner\python_basics>python myModule.py
49
(base) C:\Users\Owner\python_basics>
```

圖：myModule.py 執行結果

此時執行 9-2.py 的話，會發生什麼事？

```
(base) C:\Users\Owner\python_basics>python 9-2.py
49
8
['I', 'will', 'always', 'love', 'you']
(base) C:\Users\Owner\python_basics>
```

圖：突然輸出 49 這個模組內部測試結果

那是因為匯入模組的時候，Python 會將模組內部的程式碼（myModule.py）由上而下執行過一遍，連同模組的執行輸出也顯示在主要程式（9-2.py）裡。若是希望模組測試的結果，不要在主程式執行階段輸出，此時要將 myModule.py 改成：

myModule.py	9-2.py
<pre># 計算幾次方 def pow(x, y): return x**y # 簡單斷句 def segmentSentence(text): list_sentences = text.split(' ') return list_sentences ''' 將模組當作主程式來執行， 此時 myModule.py 的 __name__ 就會等於 __main__， __main__ 就是主程式的意思 ''' if __name__ == "__main__": print(pow(7, 2))</pre>	<pre># 直接從模組中匯入函式 from myModule import pow, segmentSentence # 計算幾次方 num = pow(2, 3) print(num) # 簡單斷句 txt = "I will always love you" list_result = segmentSentence(txt) print(list_result)</pre>

雖然主程式（9-2.py）執行時，模組的輸出不會顯示出來，實務上，通常都會在撰寫程式碼的檔案加上「if __name__ == "__main__":」，來確保每個程式進行模組化（modularization）時，都能避免掉不必要的麻煩。請參考以下的結果：

myModule.py	9-2.py
<pre> # 計算幾次方 def pow(x, y): return x**y # 簡單斷句 def segmentSentence(text): list_sentences = text.split(' ') return list_sentences if __name__ == "__main__": print(pow(7, 2)) </pre>	<pre> # 直接從模組中匯入函式 from myModule import pow, segmentSentence if __name__ == "__main__": # 計算幾次方 num = pow(2, 3) print(num) # 簡單斷句 txt = "I will always love you" list_result = segmentSentence(txt) print(list_result) </pre>

參考資料

[1] Python - if __name__ == '__main__' 涵義

<http://blog.castman.net/%E6%95%99%E5%AD%B8/2018/01/27/python-name-main.html>

[2] __main__ --- 最高层级代码环境

https://docs.python.org/zh-tw/3/library/_main_.html

[3] Python 中的 if __name__ == __main__

<https://www.freecodecamp.org/chinese/news/if-name-main-python-example/>