# Python 基礎學習


**學期:**114-1 | **課程:** 程式設計與統計軟體 | **教師:** 吳漢銘

**教學網站:** http://www.hmwu.idv.tw/

**2025/12/11**

## 主題

0. **從 R 語言進到學習 Python 語言**
1. **RStudio 環境建置與學習資源**
2. **變數、輸入輸出與運算**
3. **核心資料型態與陣列**
4. **程式流程控制與函式設計**
5. **Pandas 資料讀取與檢視**
6. **資料視覺化與統計圖**

** 此文件與 Gemini 3.0 一同協作*

# 0. 從 R 語言進到學習 Python 語言

對於熟悉 R 語言的使用者（通常是統計背景或資料分析師）來說，學習 Python 並不是「從零開始」，而是「概念的遷移」。R 和 Python 在資料分析上有 80% 的邏輯是通用的，但那 20% 的差異（語法慣例、底層邏輯）往往是造成挫折的來源。以下是給 R 語言使用者的 Python 學習建議與關鍵注意事項。

---

## 一、 核心觀念的轉換 (Mindset Shift)

**1. 廣用語言 vs. 統計語言**

- **R** 是為統計學家設計的，很多統計檢定（t-test, ANOVA）是內建函式。
- **Python** 是通用程式語言（General Purpose），「資料分析」只是它的其中一個功能。因此，很多 R 裡直接呼叫的統計功能，在 Python 中通常需要 import 特定的套件（如 scipy 或 statsmodels）才能使用。

**2. 物件導向 (OOP) 的使用習慣**

- **R (函數式思維):** 習慣 函數(資料)。例如：mean(x)、filter(df, condition)。
- **Python (物件導向思維):** 習慣 物件.方法()。例如：x.mean()、df.dropna()。
    - *建議：* 剛開始會很不習慣一直打點 .，但要適應這是 Python「物件呼叫自己功能」的邏輯。

**3. 變數是「標籤」而非「盒子」 (參照 vs. 複製)**

- **R:** 通常是 **Copy-on-modify**。當你寫 b <- a，然後修改 b，a 通常不會變。
- **Python:** 變數是物件的 **Reference (參照)**。
    - *陷阱：* 對於 List 或 Dictionary，若寫 b = a，修改 b **會連帶修改到 a**！
    - *解法：* 若要獨立複製，需使用 .copy()。

## 二、 絕對要避開的「語法陷阱」 (Syntax Pitfalls)

**1. 索引 (Indexing) 的地獄**

這是 R 使用者最痛苦的改變：

- **R:** 從 **1** 開始 (vec[1])。
- **Python:** 從 **0** 開始 (list[0])。

**2. 切片 (Slicing) 的範圍**

- **R:** 1:3 代表包含 1, 2, 3。
- **Python:** 0:3 代表 index 0, 1, 2 (**含頭不含尾**，不包含 3)。
    - *口訣：* Python 的切片長度 = end - start (例如 3-0=3 個元素)。

**3. 縮排 (Indentation) 就是語法**

- **R:** 使用 { } 來包覆程式區塊，縮排只是為了好看。
- **Python: 強制縮排**。沒有 { }，若縮排不對齊，程式會直接報錯 (IndentationError) 或執行邏輯錯誤。

**4. 賦值符號**

- **R:** 習慣用 <- (雖然 = 也可以用，但 R 社群偏好箭頭)。
- **Python:** 只能用 =。Python 不支援 <-。

## 三、 資料處理慣用法的對照 (Mapping)

將你腦中的 R 函數對應到 Pandas 的寫法，可以加速學習：

| 動作 | R (Tidyverse) | Python (Pandas) | 注意事項 |
|---|---|---|---|
| 選欄位 | `select(df, col)` | `df[['col']]` 或 `df.col` | Pandas 選單欄會變 Series |
| 篩選 | `filter(df, cond)` | `df[df['col'] > 0]` | 條件需用 () 包起來 |
| 排序 | `arrange(df, col)` | `df.sort_values('col')` | |
| 新增欄 | `mutate(df, new=...)` | `df['new'] = ...` | Pandas 直接賦值即可 |
| 分組 | `group_by(df, grp)` | `df.groupby('grp')` | |
| 管線 | `%>%` 或 ` | `>` | . (Method Chaining) |
| 缺失值 | `NA` | `NaN (Not a Number)` | 需用 np.nan 或 pd.NA |

## 四、 工具與環境建議

1. **善用 RStudio (Posit) 的雙棲功能：**既然你熟悉 RStudio，不需要馬上強迫自己跳到 VS Code。RStudio 對 Python 的支援（透過 reticulate）已經非常成熟，你可以在 R Markdown (Quarto) 中同時寫 R 和 Python chunk，這對過渡期非常有幫助。
2. **擁抱 Pandas 和 NumPy：**R 的基礎結構是 Vector 和 DataFrame；Python 的基礎 List 運算效能很差。做資料分析時，請務必將資料轉為 NumPy Array 或 Pandas DataFrame，不要用 Python 原生的 List 做數學運算（既慢又麻煩）。
3. **視覺化選擇：**如果你是 ggplot2 的鐵粉，你會覺得 Python 的 Matplotlib 很難用。*建議：*改學 **Seaborn** (基於 Matplotlib 但語法簡潔) 或 **Plotnine** (這是 ggplot2 的 Python 移植版，語法幾乎一樣)。

## 五、 學習建議

如果是先學過 R 再學 Python，記得：「學 Python 不是要拋棄 R，而是讓我們變成**雙語使用者 (Bilingual)**。R 在統計推論、學術繪圖上依然無敵；而 Python 在自動化、文字探勘、機器學習上更強大。未來的職場趨勢是兩種都會用，並知道在什麼時候該用什麼工具。」

# 1. RStudio 環境建置與學習資源

## 1: 安裝 Python 與設定 RStudio

在 RStudio 中撰寫 Python 程式主要依賴 **reticulate** 這個 R 套件。它能讓 R 與 Python 在同一個環境中溝通，甚至共用變數。

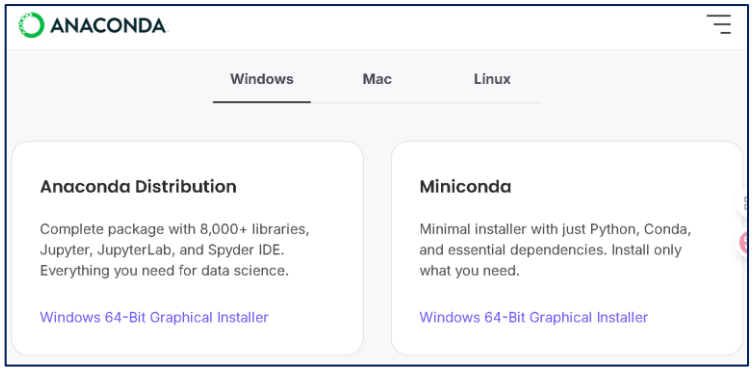
### ❖ 第一階段：環境準備與軟體套件安裝

在開始之前，你的電腦必須先有「RStudio」以及「Python 直譯器」。

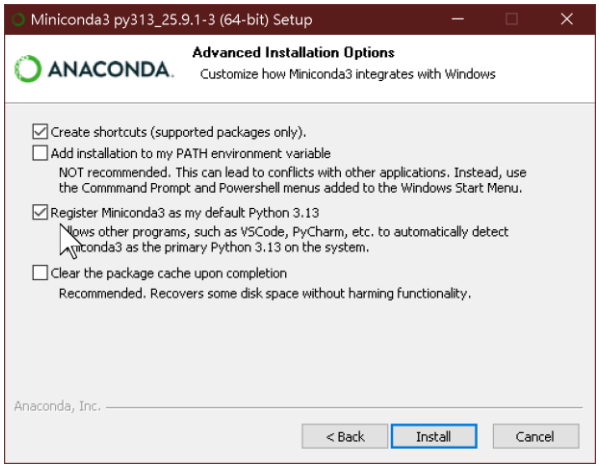
1. 安裝 Python (若電腦中尚未安裝)

RStudio 只是一個編輯器，它需要呼叫電腦底層的 Python 引擎。

- 可從 Python 官網下載(Python 3.14.2 - Dec. 5, 2025)：https://www.python.org/
- 建議方式： 下載並安裝 **Anaconda** (或 Miniconda)。這是資料科學界最標準的懶人包，內建了 Python 以及常用的 pandas, numpy 等套件。
- 下載點： Anaconda 官網 (https://www.anaconda.com/download)
- 檔案: Miniconda3-latest-Windows-x86_64.exe




- 注意：安裝過程中，建議勾選 "Register Anaconda3 as my default Python" (若為 Windows)。



- 安裝完後，務必將 RStudio 整個關掉重開。

2. 安裝 R 套件 reticulate

打開 RStudio，在 Console (控制台) 輸入以下指令來安裝並載入套件：

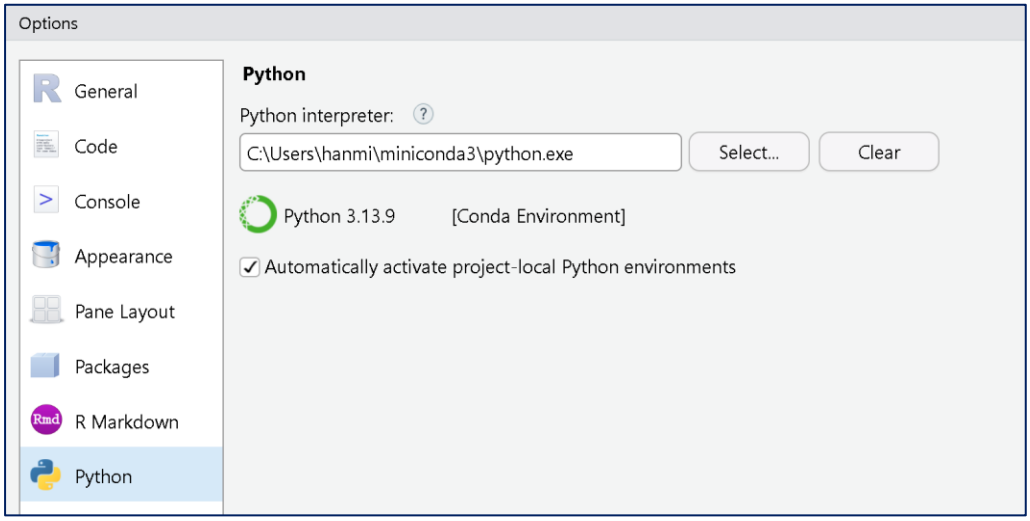
[R Code]

```
install.packages("reticulate")
library(reticulate)
```

❖ **第二階段：設定 RStudio 的 Python 引擎**

告訴 RStudio 你的 Python 安裝在哪裡。這是最重要的一步。

1. 在 RStudio 上方選單點選 **Tools** > **Global Options**。
2. 在左側列表選擇 **Python**。
3. 在 **Python interpreter** 欄位旁，點選 **Select...**。
4. RStudio 會自動掃描你電腦裡的 Python 環境。
   - 如果你有裝 Anaconda，通常會看到類似 .../anaconda3/python.exe 或 .../conda/envs/... 的路徑。
   - 選擇你想使用的那個環境，然後點選 **OK** -> **Apply**。
5. **重新啟動 RStudio** 讓設定生效。




**註 1:**

RStudio 預設會去幾個標準位置找 Anaconda，如果找不到，我們必須「手動告訴它」位置在哪裡。

[R Code]

```
> library(reticulate)
> conda_list()
Error: Unable to find conda binary. Is Anaconda installed?
```

1. 找出你的 conda 在哪裡

- **Windows:** 通常在 C:/Users/你的使用者名稱/anaconda3/Scripts/conda.exe 或 C:/ProgramData/anaconda3/Scripts/conda.exe。
- **Mac:** 通常在 /opt/anaconda3/bin/conda 或 /usr/local/bin/conda。

2. 在 RStudio 指定路徑: 請在 RStudio 的 Console 執行以下指令 (請將路徑換成你電腦實際的路徑，注意 Windows 要用 | 或 /)：

[R Code]

```
> library(reticulate)
# 設定 conda 路徑 (以下是 Windows 範例，請替換 "User" 為你的使用者名稱)
# 注意：路徑分隔線要用 / (斜線)，不要用 \ (反斜線)
> options(reticulate.conda_binary = "C:/Users/User/anaconda3/Scripts/conda.exe")
> #options(reticulate.conda_binary = "C:/Users/hanmi/miniconda3/Scripts/conda.exe")
# 再試一次
> conda_list()
  name                                  python
1 base C:\\Users\\hanmi\\miniconda3/python.exe
```

3. 再進行<**第二階段：設定 RStudio 的 Python 引擎**>步驟。

**註 2:**

直接讓 RStudio 自己裝一個 Miniconda。reticulate 提供了一個指令，可以直接幫你在 R 的環境下安裝一個輕量版的 Python (Miniconda)。

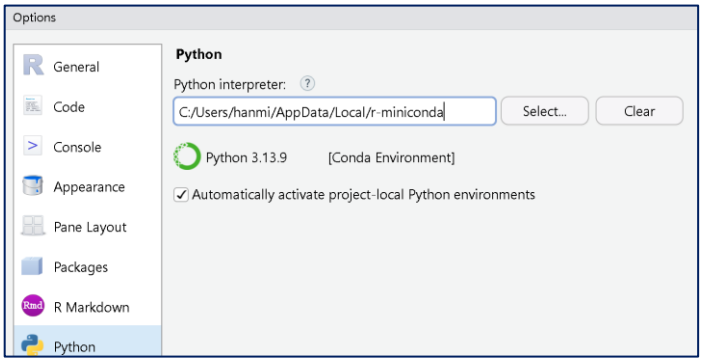
在 RStudio Console 執行：

[R Code]

```
> library(reticulate)
> install_miniconda()
* Installing Miniconda -- please wait a moment ...
...
Welcome to Miniforge3 25.11.0-1
...
...
* Miniconda has been successfully installed at "C:/Users/hanmi/AppData/Local/r-miniconda".
[1] "C:/Users/hanmi/AppData/Local/r-miniconda"
# 或者讓 R 自動下載並安裝一個標準版的 Python (這可能需要幾分鐘)
> # install_python()    [★極力建議以此方式安裝 Python★]
> py_config()
```

1. 這會跑一陣子，它會下載並安裝 Python 環境。
2. 安裝完成後，重啟 RStudio。
3. 執行 py_config() 確認是否成功抓到 Python。

4. 再進行<**第二階段：設定 RStudio 的 Python 引擎**>步驟。




**註 3:**

執行 py_config()，查看 R 找到的 Python 資訊。(若無相關套件，會順路安裝)

[R Code]

```
> py_config()
Downloading uv...Done!
Downloading cpython-3.12.12-windows-x86_64-none (download) (20.8MiB)
 Downloaded cpython-3.12.12-windows-x86_64-none (download)
Downloading numpy (12.2MiB)
 Downloaded numpy
Installed 1 package in 1.05s
python:         C:/Users/hanmi/AppData/Local/R/cache/R/reticulate/uv/cache/archive-v0/zEUMGbmaSAGTZTEJ8xBtF/Scripts/python.exe
libpython:      C:/Users/hanmi/AppData/Local/R/cache/R/reticulate/uv/python/cpython-3.12.12-windows-x86_64-none/python312.dll
pythonhome:     C:/Users/hanmi/AppData/Local/R/cache/R/reticulate/uv/cache/archive-v0/zEUMGbmaSAGTZTEJ8xBtF
virtualenv:     C:/Users/hanmi/AppData/Local/R/cache/R/reticulate/uv/cache/archive-v0/zEUMGbmaSAGTZTEJ8xBtF/Scripts/activate_this.py
version:        3.12.12 (main, Dec  9 2025, 19:02:55) [MSC v.1944 64 bit (AMD64)]
Architecture:   64bit
numpy:          C:/Users/hanmi/AppData/Local/R/cache/R/reticulate/uv/cache/archive-v0/zEUMGbmaSAGTZTEJ8xBtF/Lib/site-packages/numpy
numpy_version:  2.3.5

NOTE: Python version was forced by py_require()
```


之後再進行<**第二階段：設定 RStudio 的 Python 引擎**>步驟。

- **第三階段：開始在 RStudio 寫 Python**

在 RStudio 中寫 Python 主要有三種方式，(先 Create “New Project” (與制造 R 專案同一步驟)，再進行寫程式)，最推薦第 1 種：

**【方法 1】：使用 R Markdown (或 Quarto) 筆記本 (最強大)**

這種方式可以讓你**同時**跑 R 和 Python 程式碼，變數還可以互通。

1. 建立新檔案：File > New File > R Markdown。

2. 插入一個 Python 程式碼區塊 (Chunk)：
    - 點選編輯器右上角的 +C 按鈕，選擇 Python。
    - 或者手動輸入：

[Markdown]

```
```{python}
import pandas as pd
data = [10, 20, 30]
print(data)
```
```

3. 執行該區塊 (按綠色播放鍵)，結果會直接顯示在下方。
    - 在 Python 中讀取 R 的變數： 使用 r.變數名
    - 在 R 中讀取 Python 的變數： 使用 py$變數名

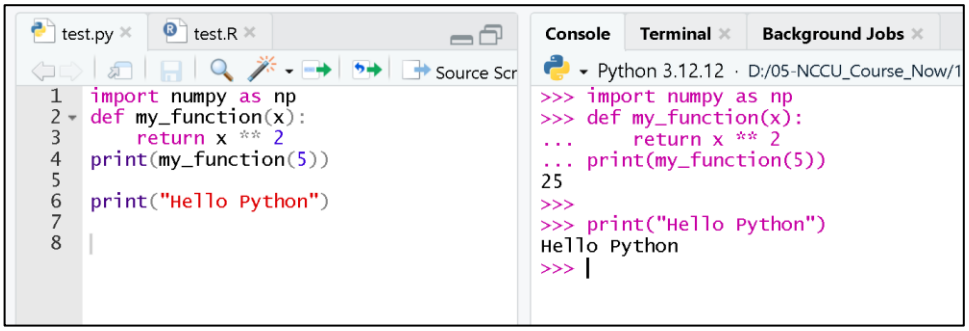
**【方法 2】：使用 Python Script (.py 檔)**

如果你只是想寫純 Python 腳本，不需要 R 的功能。

1. 建立新檔案：File > New File > Python Script。
2. 就像在 VS Code 裡一樣寫程式：

```
import numpy as np
def my_function(x):
    return x ** 2
print(my_function(5))
```

3. 存檔 test.py，按 Ctrl + Enter (Windows) 或 Cmd + Enter (Mac) 執行程式碼，結果會顯示在 Console。(如同寫 R 程式)




**【方法 3】：使用 Console (互動模式)**

如果你只是想快速測試一兩行 Python 指令：

1. 在 R 的 Console 輸入 repl_python()。
2. 你會發現提示符號從 > 變成 >>> (且變成黃色/綠色)，這代表你進入了 Python 模式。
3. 輸入 exit 即可回到 R 模式。

```
repl_python()
print("Hello Python")
exit
```

# 2: Python 程式設計 IDE 軟體與線上平台

- **IDE 軟體**

  撰寫 Python 程式的工具非常多，選擇哪一個通常取決於您的用途（是做資料分析？還是寫網站系統？）以及個人習慣。以下列出目前全球最主流的 5 款 Python IDE 與編輯器，並分析其優缺點：

  **1. Visual Studio Code (VS Code):** 目前全球市佔率第一的程式碼編輯器，由微軟開發。它本身是一個輕量級的編輯器，透過安裝「擴充套件 (Extensions)」變成強大的 IDE。

  優點：

  - 完全免費且開源。
  - 擴充性極強：想要 Python 功能？裝 Python 套件。想要繁體中文介面？裝中文套件。
  - 跨語言支援：今天寫 Python，明天寫 C++ 或 HTML，不用換軟體。
  - 輕量快速：啟動速度比完整的 IDE 快。

  缺點：

  - 初學者需設定：剛下載下來是一個「空殼」，需要自己搜尋並安裝 Python 相關套件才能開始跑程式，對完全新手有一點點門檻。
  - 記憶體佔用：若安裝太多套件，電腦可能會變慢。
  - 適用對象： 所有類型的開發者、資工系學生、網頁工程師。

  **2. PyCharm:** 由 JetBrains 公司專為 Python 打造的「重裝級」IDE。它就像是一輛配備齊全的跑車，所有功能原廠都幫你裝好了。

  優點：

  - 開箱即用：安裝好就能寫，不需要像 VS Code 那樣自己裝套件。
  - 強大的代碼分析：它的自動補全 (Autocomplete) 和除錯功能 (Debugging) 非常聰明，能幫你抓出很多潛在錯誤。
  - 專案管理強：適合處理幾百個檔案的大型專案。

  缺點：

  - 啟動慢、吃資源：開啟軟體需要較長時間，舊電腦跑起來會卡。
  - 收費機制：雖然有免費的「社群版 (Community)」，但進階功能的「專業版 (Professional)」需要付費。
  - 適用對象： 專業軟體工程師、開發大型系統者。

  **3. Jupyter Notebook / JupyterLab:** 資料科學領域的標準配備。它不是傳統寫「軟體」的介面，而是一種「互動式筆記本」。

優點：

- 互動性強：程式碼是一塊一塊執行的，執行完馬上能在下方看到結果（表格、圖表）。
- 圖文並茂：可以在程式碼中間穿插文字說明 (Markdown)、數學公式，非常適合教學和撰寫報告。
- 資料探索方便：不需要重跑整個程式，只需修改其中一小段並執行。

缺點：

- 不適合寫大型軟體：若程式碼有幾千行，管理起來會很混亂。
- 版控困難：很難使用 Git 進行版本控制。
- 適用對象： 資料分析師、科學家、教學演示。

**4. Spyder:** 這款軟體通常隨 Anaconda 一起安裝。它的介面設計模仿 MATLAB，非常適合習慣 RStudio 或 MATLAB 的使用者。

優點：

- 變數瀏覽器 (Variable Explorer)：這是 R 使用者最愛的功能，可以直觀地看到目前記憶體中有哪些變數、DataFrame 長什麼樣子。
- 科學計算導向：專為數據分析設計，整合了繪圖視窗。

缺點：

- 介面較老舊：看起來比較像十年前的軟體。
- 功能單一：除了做資料分析，不適合拿來寫網頁或一般腳本。
- 適用對象： 從 R 或 MATLAB 轉過來的研究人員、工程師。

**5. Google Colab (Colaboratory):** 這是 Google 推出的「雲端版 Jupyter Notebook」。

優點：

- 完全免安裝：只要有瀏覽器、有 Google 帳號就能寫程式。
- 免費算力：Google 免費提供 GPU (顯示卡) 和 TPU 資源，跑深度學習（AI 模型）。
- 易於分享：就像 Google 文件一樣，連結丟給學生就能看。

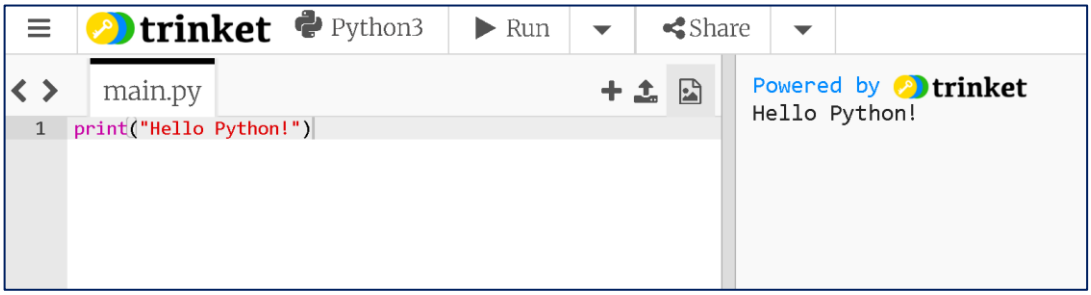
缺點：

- 需要網路：沒網路就不能用。
- 執行時間限制：閒置太久或跑太久會斷線，檔案存在雲端重開機後會消失（需連結 Google Drive）。
- 適用對象： 學生（電腦配備不好時）、AI 初學者、臨時測試程式碼。

**總結建議**

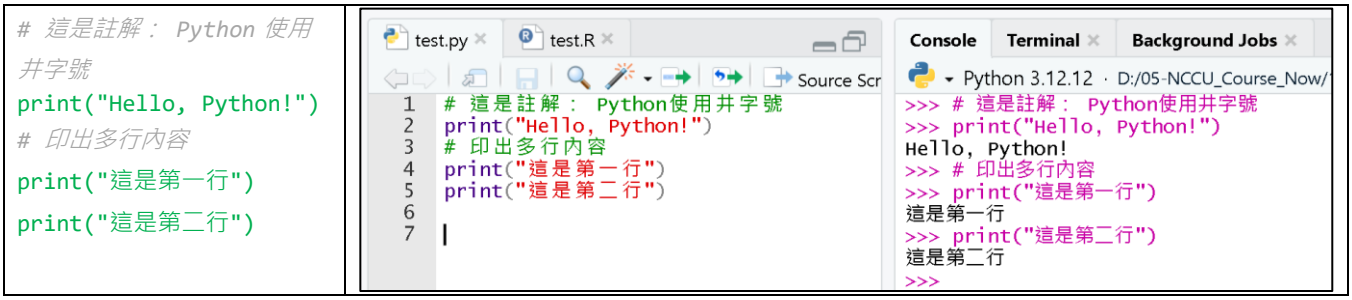
| 如果你是... | 推薦選擇 | 原因 |
|---|---|---|
| 資料分析課程的學生 | Jupyter Notebook (或 Anaconda 裡的 Spyder) | 可以一步步執行，馬上看圖表，學習挫折感低。 |
| 資工系/要寫軟體的學生 | VS Code | 目前業界標準，學會用它對未來找工作最有幫助。 |
| 電腦跑不動/不想安裝環境 | Google Colab | 打開網頁就能上課，完全沒負擔。 |
| 習慣 RStudio | RStudio (透過 reticulate) 或 Spyder | 介面邏輯最接近您原本的習慣，無痛轉移。 |

- **線上寫 Python 程式**
  - https://trinket.io/embed/python3
  - https://www.programiz.com/python-programming/online-compiler/
  - https://pythononline.net/
  - https://www.online-python.com/

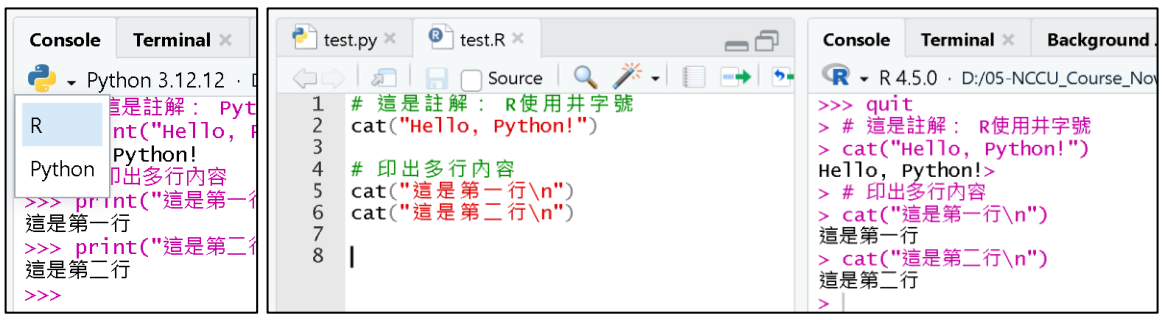



## 3: 第一支 Python 程式：Hello World

- 體驗 print() 函數與字串。




- 以 R 程式示範:



## 4: 變數賦值與命名

- Python 使用 = 賦值 (R 使用 <-)。

```
# 正確的命名 (Snake Case)
student_name = "Alice"
math_score = 90

# 同時賦值
x = y = 10
a, b = 1, 2

print(student_name)
print(a, b)
```

## 5: 基本輸入與輸出

- input() 讀入皆為字串，需搭配 f-string 輸出。

```
# 1. 取得使用者輸入
> name = input("請輸入您的名字: ")
請輸入您的名字:  Han-Ming Wu

# 2. 格式化輸出 (f-string)
# 注意字串前的 'f' 以及大括號 {}
> print(f"歡迎 {name} 來到 Python 的世界！")
歡迎 Han-Ming Wu 來到 Python 的世界！
> print("歡迎 {name} 來到 Python 的世界！")
歡迎 {name} 來到 Python 的世界！

# 3. 簡單運算輸出
> age = 20
> print(f"{name} 明年將會是 {age + 1} 歲")
Han-Ming Wu 明年將會是 21 歲
```

## 6: 安裝 Python 套件

在 RStudio 中安裝 Python 套件主要有兩種方式：一種是**用 R 的指令安裝 (推薦)**，另一種是**用終端機 (Terminal) 指令安裝**。

對於 RStudio 的使用者，建議優先教學 **方法一**，因為這最符合 R 的操作習慣。

**【方法 1】: 使用 R 指令 py_install() (最推薦)**

這是 reticulate 套件提供的功能，它的邏輯跟 R 的 install.packages() 非常像，RStudio 會自動幫你把套件裝進目前設定好的 Python 環境中。優點是你不需要煩惱現在是在哪個 Conda 環境或虛擬環境，reticulate 會自動偵測並安裝到對的地方。

**步驟：**

- 開啟 RStudio 的 **Console** (左下角 R 的視窗)。

```
# R
library(reticulate)

# 安裝單一套件 (例如安裝 pandas)
py_install("pandas")

# 一次安裝多個套件 (用 c() 包起來)
py_install(c("numpy", "matplotlib", "scikit-learn"))
```

- 用 R 指令檢查任何 Python 套件是否已經安裝成功 (最快、最推薦)

```
library(reticulate)

#這會回傳 TRUE (有安裝) 或 FALSE (沒安裝)
py_module_available("pandas")

# 這會列出目前 Python 環境下所有的套件
py_list_packages()

# 如果發現沒安裝，請直接在 R Console 執行安裝指令
py_install("pandas")
```

**【方法 2】: 使用 RStudio 的 Terminal (標準 Python 做法)**

如果你希望熟悉未來在 VS Code 或純 Python 環境的操作方式，可以試這個方法。

**步驟：**

1. 點選 RStudio 左下角視窗的 **Terminal** 分頁 (就在 Console 旁邊)。
    - 如果找不到 **Terminal** 分頁，請選按 Shift+Alt+T (RStudio: Tools -> Terminal -> Move Focus to Terminal).)
    - 或 RStudio: View -> Move Focus to Terminal
    - **RStudio:** Tools -> Global Options。在左側選擇 Pane Layout (版面配置)。檢查 Console 那個區塊 ( 通常是左下角 ) 的標籤列表中，Terminal 有沒有被勾選？
    - Using the RStudio Terminal in the RStudio IDE: https://support.posit.co/hc/en-us/articles/115010737148-Using-the-RStudio-Terminal-in-the-RStudio-IDE
2. (選用) 如果你有使用特定的 conda 環境，記得先啟動它 (例如 conda activate r-reticulate)。
3. 輸入 Python 的標準安裝指令 pip：

```
# Bash
# 這是 Linux/Mac/Windows 通用的指令
pip install pandas numpy matplotlib
# (注意：在 Terminal 裡不需要加引號，也不需要逗號分隔，用空白鍵隔開即可)
```

### ⚠ 重要觀念[1]：標準套件管理工具 (Package Installer for Python, pip)

1. 在 Python 程式設計中，**pip** 是 Python 的標準套件管理工具 (Package Installer for Python)。

| 概念 | R 語言 | Python |
|---|---|---|
| 動作 | 安裝套件 | 安裝套件 |
| 指令 | install.packages("pandas") | pip install pandas |
| 套件倉庫 | **CRAN** (Comprehensive R Archive Network) | **PyPI** (Python Package Index) |
| 執行位置 | R Console | **Terminal (終端機)** |

2. pip 的具體功能: 它的全名通常被戲稱為 **"Pip Installs Packages"** (遞迴縮寫)。它的主要功能是從 **PyPI** (Python 的雲端倉庫) 下載別人寫好的程式碼庫到你的電腦裡。當你想用 pandas 做資料分析，或用 matplotlib 畫圖時，這些功能不是 Python 內建的，你必須先呼叫 pip 把這幾個「工具包」下載下來安裝好。

3. 常見指令 (在 Terminal 輸入): 請注意，這些指令**不是**寫在 Python 程式碼(.py)裡面，而是寫在**終端機 (Terminal / Command Prompt)** 裡面的：

```
# Bash
# 安裝套件：
pip install pandas
# 安裝特定版本：
pip install pandas==2.0.0
# 列出已安裝的套件：
pip list
# 移除套件：
pip uninstall pandas
```

4. 在 RStudio 中的特例: 如果您使用 RStudio 透過 reticulate 套件來管理 Python，您可以使用 R 的函數 py_install()，它其實就是在背後自動幫您執行 pip install 的動作。

### ⚠ 重要觀念[2]：虛擬環境 (Virtual Environments)

這是 R 使用者最容易卡關的地方。

- **R 的習慣：** 套件裝了就是裝了，全電腦通用。
- **Python 的習慣：** 套件是裝在「特定的環境」裡。如果你在 **環境 A** 裝了 pandas，切換到 **環境 B** 是讀不到的！

常見問題排除：

如果執行 py_install("pandas") 顯示成功，但在 Python script 裡 import pandas 卻說找不到？」有可能安裝的環境跟 RStudio 現在用的環境不同。請執行 py_config()，確認目前 RStudio 到底連到哪個 Python，並確保套件是裝在那個路徑下。(必要時，R Session 重啟)

**⚠ 重要觀念[3]：Windows 環境變數問題**

在 terminal 中，有錯誤訊息: 這是一個非常經典的 Windows 環境變數問題。這段錯誤訊息 'pip' 不是內部或外部命令... 的意思是：**「Windows 的命令提示字元 (cmd) 不知道 pip 這個程式藏在哪個資料夾裡，所以它找不到。」**這通常是因為安裝 Python 或 Anaconda 時，沒有勾選「加入環境變數 (Add to PATH)」。

```
# Bash
D: \PythonBasic> pip
'pip' 不是內部或外部命令、可執行的程式或批次檔。
```

請依照以下三種解法，由簡入深嘗試：

**【解法一】: 加上 python -m (最快、成功率最高)**

既然 Windows 找不到 pip，那我們就叫 python 去幫我們執行它。請在原本出現錯誤的 Terminal 中，輸入以下指令（**在原本的 pip 指令前加上 python -m**）:

```
# 原本的指令：
# Bash
pip install pandas
# 修正後的指令：
# Bash
python -m pip install pandas
```

**【解法二】: 如果您是安裝 Anaconda (最正確的做法)**

Anaconda 的設計邏輯是希望您使用它專屬的終端機，而不是 Windows 預設的黑底白字 cmd。

- 按 Windows 的 **「開始」** 按鈕。
- 搜尋 **"Anaconda Prompt"** (會有一個黑色圖示)。開啟它。
- 在這個專屬視窗中，直接輸入 pip install pandas 通常就會成功了。

**【解法三】: 回到 RStudio 的舒適圈 (避開問題)**

如果您是在 RStudio 裡面遇到這個挫折，其實可以完全不用管 Terminal。直接回到 **R Console** (左下角那個有 > 符號的視窗)，使用 R 的指令來安裝，RStudio 會自動幫您找到路徑：

```
# R
# 在 R Console 執行
library(reticulate)
py_install("pandas")
```

## 7: 求助 Help

在 RStudio 中查詢 Python 語法的體驗與 R 略有不同。R 的 ?函數名 會開啟漂亮的 Help 視窗（HTML 格式），但 Python 的說明文件通常是純文字格式（Docstring），主要會直接顯示在 **Console（控制台）** 中。以下是在 RStudio 查詢 Python 語法的 **三種主要方式**：

**【方法 1】: 使用 reticulate 套件的 py_help() (在 R 模式下)**

如果你正在 R 的環境中操作 Python 物件（例如已經 import("pandas")），可以使用 reticulate 提供的專用函數 py_help，說明文件會顯示在 **Console** 視窗中（純文字）。

- 指令： py_help(物件)

```
# R
library(reticulate)
pd <- import("pandas")

# 查詢 read_csv 的用法
py_help(pd$read_csv)
```

**【方法 2】: 使用 Python 原生 help() (在 Python 模式下)**

這是最標準的 Python 查詢方式，無論是在 .py 腳本、R Markdown 的 Python 區塊，或是 Python REPL 中都通用。

- 指令： help(物件)
- 範例 (在 Python 區塊或腳本中)：

```
# Python
import pandas as pd

# 查詢 read_csv
help(pd.read_csv)
```

- **小技巧**： 如果你是在 RStudio 的 Console 進入 Python 模式 (repl_python())，也可以使用 ? (IPython 風格)：

```
# Python
>>> import pandas as pd
>>> pd.read_csv?  # 這樣也可以查
```

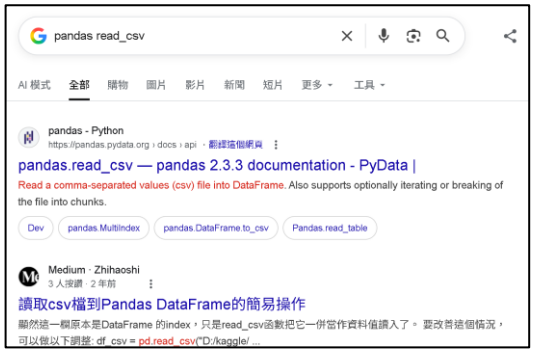
**【方法 3】: 直接查看屬性 __doc__ (快速查看)**

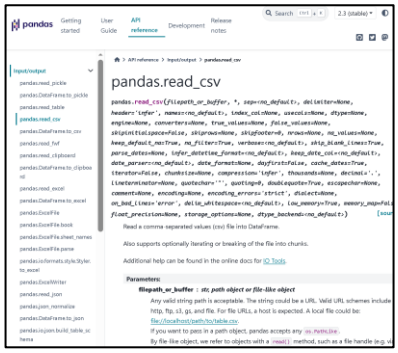
Python 的說明文件其實就是一段儲存在物件裡的字串，稱為 "Docstring"。你可以直接印出來看。

- **指令**： print(物件.__doc__)

```
# Python
import math
print(math.sqrt.__doc__)
```

**【方法 4】:** 由於 Python 在 RStudio 內的 Help 顯示不如 R 那麼美觀易讀，建議直接 Google 「套件名 + 函數名」(例如："pandas read_csv")，直接看官方線上文件通常是體驗最好的方式。

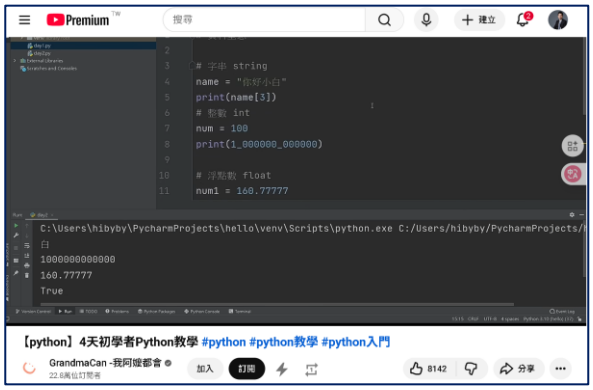


## 8: 學習資源

以下列出 Python 繁體中文學習資源清單，涵蓋了從初學者到進階開發者的各種需求：

- **線上教學影片** (YouTube 頻道，可搜尋 Python 教學)
  - **GrandmaCan - 我阿嬤都會**：(https://www.youtube.com/@GrandmaCan) 非常適合完全零基礎的新手，影片風格輕鬆有趣，通常會用簡單的專案（如計算機、猜數字）來帶入語法。
    【python】4 天初學者 Python 教學 #python #python 教學 #python 入門 - YouTube
    https://www.youtube.com/watch?v=Ob_LKCLxg2o



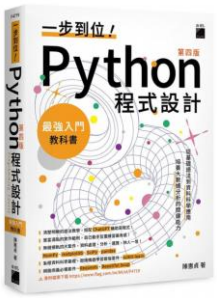

  - **彭彭的課程 (Peng Peng)**：(https://www.youtube.com/@cwpeng-course) 台灣非常資深的程式教學頻道，邏輯清晰，涵蓋 Python 基礎語法、網路爬蟲到網頁開發 (Flask)，內容紮實。
  - **Code Me Maybe 碼上學**：(https://www.youtube.com/@CodeMeMaybe) 適合初學者的教學頻道，有許多關於環境建置與基礎語法的教學。

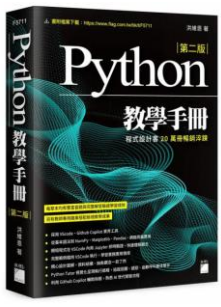

- **線上課程平台 (付費/免費)**
  - **Hahow 好學校**：台灣最大的線上課程平台。推薦關鍵字：搜尋「Python 入門特訓」、「Python 爬蟲」、「FinLab」(專注於金融理財應用)。
  - **Udemy**：全球最大的課程平台，但有很多繁體中文講師。推薦關鍵字：搜尋「Python 3 零基礎」、「Python 資料分析」。購買前請留意課程評價與更新日期。

- **社群與討論區**
  - **Facebook 社團**：

    - **Python Taiwan**：台灣最大的 Python 社團，討論風氣活躍，從新手問題到職缺分享都有。
    - **Python 技術交流區**：另一個熱門的技術討論版塊。
  - **PTT (批踢踢實業坊)**：
    - **Soft_Job 版**：主要討論軟體工程師職涯、面試心得，也會有 Python 相關職缺。
    - **Python 版**：專門討論 Python 技術的看板（流量較少，建議也可去 **CodeJob** 或 **DataScience** 版）。
  - **Dcard**：
    - **程式設計版 (Coding)**：大學生為主的討論區，很多新手入門、轉職或作業相關的討論。
- **實體與線上研討會 (Conferences & Meetups)**
  - **PyCon Taiwan (Python 年會)**：台灣年度最盛大的 Python 研討會，通常在每年 9 月左右舉辦，有技術演講、工作坊和社群攤位。
  - **地方性聚會 (Meetups)**：各地定期舉辦的小型聚會，通常會有 1-2 個短講: **Taipei.py** (台北)、**PyHUG** (新竹)、**Taichung.py** (台中)、**Tainan.py** (台南)、**Kaohsiung.py** (高雄)、**Hualien.py** (花蓮)。

- **推薦書籍 (繁體中文) (從博客來查詢)**
  - 《**精通 Python**》(Introducing Python)：歐萊禮 (O'Reilly) 出版，內容廣泛且紮實，適合想打好基礎的人。
  - 《**Python 自動化的樂趣**》(Automate the Boring Stuff with Python)：非常實用，教你用 Python 處理 Excel、Email、檔案整理等日常行政工作。
  - 《**流暢的 Python**》(Fluent Python)：被譽為 Python 聖經，深入講解 Python 的底層機制與最佳實踐，適合想從「會寫」進階到「寫得好」的人。
  - [GOTOP] 《Python 功力提昇的樂趣》、[五南] 《量表編製與統計分析使用 Python 語言》、[Springer] 《Python 統計分析: 生命科學應用》、[深智] 《Python 最強入門王者歸來》、[旗標] 《一步到位 Python 程式設計》、 [旗標] 《Python 教學手冊》





# 2. 數值運算、型態轉換與內建函式

## 1: 數學基本運算

- 重點： 整除 // 與次方 **。

```
x = 10
y = 3

print(x + y)   # 加法: 13
print(x / y)   # 除法: 3.3333... (永遠是 float)
print(x // y)  # 整除: 3 (R 沒有這個)
print(x % y)   # 餘數: 1
print(x ** 2)  # 次方: 100 (R 是 x^2)
```

- 重點： `abs, max, min, round` 的實際用法。

```
data = -3.14159
numbers = [10, 50, 20]

print(abs(data))        # 絕對值: 3.14159
print(round(data, 2))   # 四捨五入到小數點後 2 位: -3.14
print(max(numbers))     # 最大值: 50
print(min(numbers))     # 最小值: 10
print(len(numbers))     # 長度: 3
```

- 重點：資料型態轉換 (Casting)， input() 進來的資料通常需要轉型才能運算。

```
# 字串轉整數/浮點數
s_num = "100"
s_float = "3.5"

val_int = int(s_num)      # 變成整數 100
val_float = float(s_float) # 變成浮點數 3.5

print(val_int + 50)       # 輸出 150

# 轉為字串
age = 20
print("我今年 " + str(age) + " 歲") # 字串串接需先轉型
```

## 2: 統計基本運算

在 Python 中，我們通常有三種方式來計算基礎的統計量。

- **使用內建的 statistics 模組 (最簡單，免安裝)**

如果您只是想對簡單的 List 做計算，不想安裝任何套件，這是首選。

```
import statistics

# 準備資料
data = [1, 2, 3, 4, 5, 6, 7, 8, 9, 100] # 故意加一個 100 看看對平均數的影響
x = [1, 2, 3, 4, 5]
y = [2, 4, 6, 8, 10]

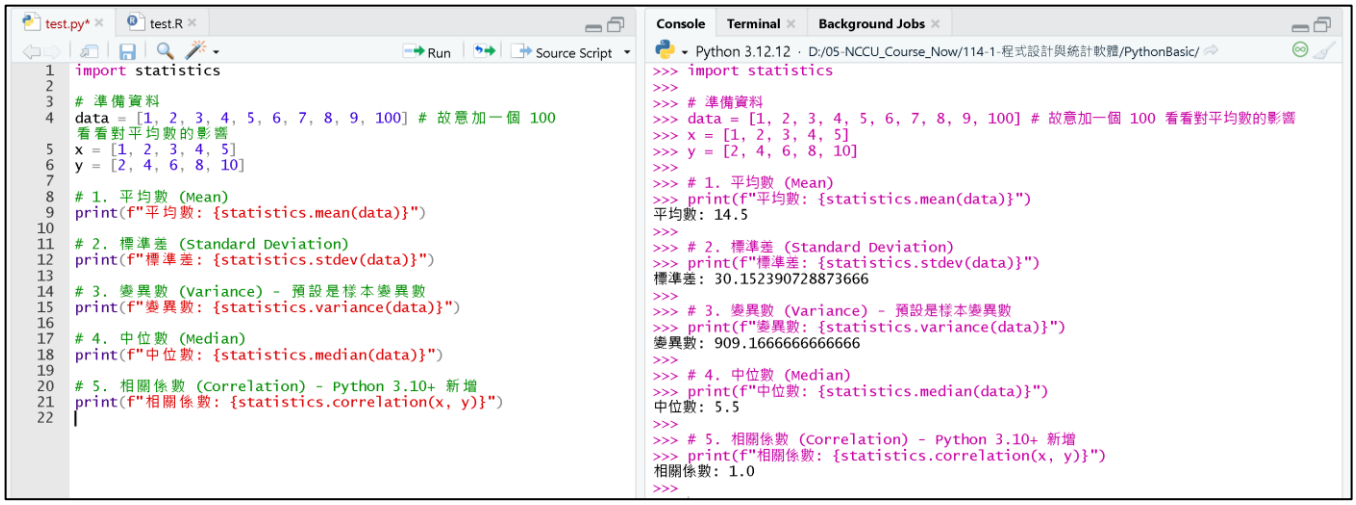
# 1. 平均數 (Mean)
print(f"平均數: {statistics.mean(data)}")

# 2. 標準差 (Standard Deviation)
print(f"標準差: {statistics.stdev(data)}")

# 3. 變異數 (Variance) - 預設是樣本變異數
print(f"變異數: {statistics.variance(data)}")

# 4. 中位數 (Median)
print(f"中位數: {statistics.median(data)}")

# 5. 相關係數 (Correlation) - Python 3.10+ 新增
print(f"相關係數: {statistics.correlation(x, y)}")
```




- **使用 NumPy 套件 (科學運算標準，類似 R 的 Vector)**

這是做數值分析最常用的方式，速度快，適合大量數據。

**注意：** NumPy 的標準差預設是 **母體標準差 (除以 n)**，這點跟 R (除以 n-1) 不同，需要加上 ddof=1 參數才會一樣。

```
import numpy as np

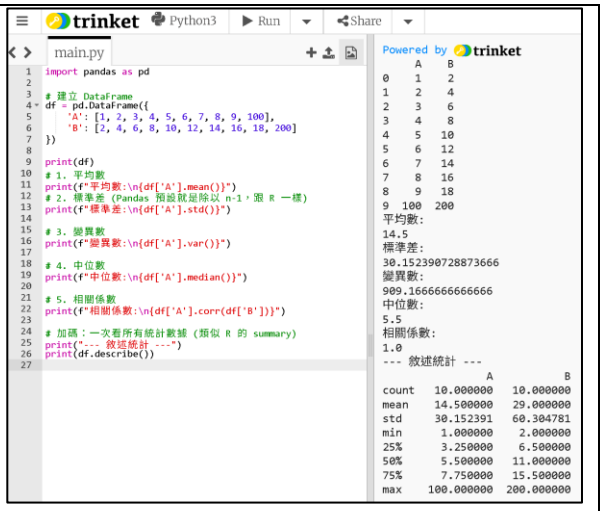
# 建立 Array (向量)
data = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 100])
x = np.array([1, 2, 3, 4, 5])
y = np.array([2, 4, 6, 8, 10])
```

```python
# 1. 平均數
print(f"平均數: {np.mean(data)}")

# 2. 標準差 (ddof=1 代表樣本標準差，除以 n-1)
print(f"標準差: {np.std(data, ddof=1)}")

# 3. 變異數 (同上，ddof=1)
print(f"變異數: {np.var(data, ddof=1)}")

# 4. 中位數
print(f"中位數: {np.median(data)}")

# 5. 相關係數 (回傳矩陣)
# np.corrcoef 回傳的是相關係數矩陣 (Correlation Matrix)
# [[1. 1.]
#  [1. 1.]]
# 取 [0, 1] 才是兩個變數的相關係數
r = np.corrcoef(x, y)[0, 1]
print(f"相關係數: {r}")
```

- **使用 Pandas 套件 (類似 R 的 DataFrame，最推薦)**

如果您處理的是表格資料 (Excel, CSV)，這是最方便的方法，語法邏輯也最接近 R。

```python
import pandas as pd

# 建立 DataFrame
df = pd.DataFrame({
    'A': [1, 2, 3, 4, 5, 6, 7, 8, 9, 100],
    'B': [2, 4, 6, 8, 10, 12, 14, 16, 18, 200]
})

print(df)
# 1. 平均數
print(f"平均數:\n{df['A'].mean()}")
# 2. 標準差 (Pandas 預設就是除以 n-1，跟 R 一樣)
print(f"標準差:\n{df['A'].std()}")

# 3. 變異數
print(f"變異數:\n{df['A'].var()}")

# 4. 中位數
print(f"中位數:\n{df['A'].median()}")

# 5. 相關係數
print(f"相關係數:\n{df['A'].corr(df['B'])}")

# 加碼：一次看所有統計數據 (類似 R 的 summary)
print("--- 敘述統計 ---")
print(df.describe())
```



- **總結比較**

| 指標 | statistics (內建) | numpy (數值計算) | pandas (表格分析) | R 語言對照 |
|---|---|---|---|---|
| 平均數 | `mean()` | `np.mean()` | `.mean()` | `mean()` |
| 標準差 | `stdev()` | `np.std(ddof=1)` | `.std()` | `sd()` |
| 變異數 | `variance()` | `np.var(ddof=1)` | `.var()` | `var()` |
| 中位數 | `median()` | `np.median()` | `.median()` | `median()` |
| 相關係數 | `correlation()*` | `np.corrcoef()` | `.corr()` | `cor()` |

**註：statistics.correlation 需 Python 3.10 以上版本。*

# 3. 核心資料結構：List 與 Dictionary

這兩者是 Python 中最核心、最常用的兩種**資料容器 (Container)**。如果把變數比喻成「物品」，那麼 List 和 Dictionary 就是用來裝這些物品的「箱子」或「櫃子」。

---

## 1: List (列表)

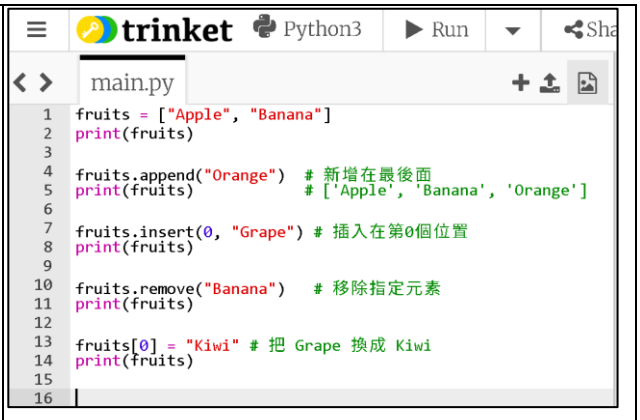
1. **概念：** 就像一個**有編號的置物櫃**，或者是你的「購物清單」。
   - 特性： 有順序性 (Ordered)、內容可變 (Mutable)、可重複。
   - 符號： 使用 中括號 [ ]。
   - R 語言對照： 類似 R 的 list (因為它可以裝不同型態的東西)，或是 R 的 vector (如果裡面都裝數字)。
   - 核心運作邏輯：它是靠 「位置 (Index)」 來找資料的。**注意：** Python 的位置是從 **0** 開始編號！(R 是從 1 開始)
   - 重點： List 索引與切片 (Indexing & Slicing)，索引從 0 開始。

```
# 建立 List
scores = [80, 90, 75, 60, 100]

# 索引 (Index)
print(scores[0])    # 第 1 個元素: 80
print(scores[-1])   # 最後 1 個元素: 100

# 切片 (Slice) [開始 : 結束(不含)]
print(scores[1:3])  # 取 index 1 到 2: [90, 75]
print(scores[:3])   # 取前 3 個: [80, 90, 75]
```

## 2: List 常用指令

- 重點： 新增、刪除與排序。

```
fruits = ["Apple", "Banana"]
print(fruits)

fruits.append("Orange")  # 新增在最後面
print(fruits)            # ['Apple', 'Banana',
'Orange']

fruits.insert(0, "Grape") # 插入在第0個位置
print(fruits)

fruits.remove("Banana")   # 移除指定元素
print(fruits)

fruits[0] = "Kiwi" # 把 Grape 換成 Kiwi
print(fruits)
```



```python
nums = [3, 1, 5]
nums.sort()              # 排序 (會改變原本的 list)
print(nums)              # [1, 3, 5]
```

```
Powered by trinket
['Apple', 'Banana']
['Apple', 'Banana', 'Orange']
['Grape', 'Apple', 'Banana', 'Orange']
['Grape', 'Apple', 'Orange']
['Kiwi', 'Apple', 'Orange']
```

## 3: Dictionary (字典)

- **概念：** 就像一本真的**字典**，或者是「通訊錄」。你需要透過「名字」來找到「電話」。
  - **特性：** 鍵值對 (Key-Value Pair)、鍵 (Key) 不可重複、搜尋速度極快。
  - **符號：** 使用 **大括號 { }**。
  - **R 語言對照：** 類似 R 的 **Named List** (例如 list(name="John", age=20))。
- 核心運作邏輯：它是靠「標籤 (Key)」來找資料的，而不是靠位置。

```python
# 建立一個 Dictionary
# 結構是 "Key": Value
student = {
    "Name": "John",
    "Age": 20,
    "Score": 90
}
# 也可以直接寫
# student = {"Name": "John", "Age": 20, "Score": 90}

# 1. 讀取 (靠標籤 Key)
print(student["Name"]) # 印出 John
# print(student[0])    # 錯誤！字典不能用 0, 1, 2 來查

# 2. 修改
student["Score"] = 95  # 把 90 改成 95

# 3. 新增
student["City"] = "Taipei" # 直接給一個新的 Key 就會新增

print(student)
# 結果: {'Name': 'John', 'Age': 20, 'Score': 95, 'City': 'Taipei'}

# 檢查 Key 是否存在
print("Age" in student)  # True
```

## 3: List (列表)與 Dictionary (字典)比較

| 特性 | List (列表) | Dictionary (字典) |
|---|---|---|
| 符號 | `[1, 2, 3]` | `{"A": 1, "B": 2}` |

| **主要用途** | 儲存一連串的資料 (序列) | 儲存資料的詳細屬性 (對照表) |
|---|---|---|
| **如何取資料** | **靠位置 (Index)** | **靠標籤 (Key)** |
| **順序性** | 有順序 (第 1 個, 第 2 個...) | Python 3.7+ 之後有紀錄順序，但邏輯上主要看 Key |
| **生活類比** | 排隊的人龍、火車車廂 | 通訊錄、身份證資料 |
| **R 語言對照** | `list()` 或 `c()` | `list(key=value)` |

最容易混淆的是取值的方式：

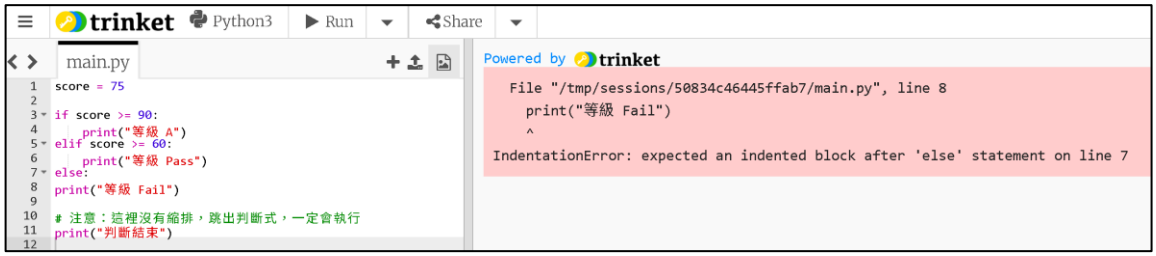
- 看到 [] 裡面是 0, 1, 2 這種數字通常是 List。
- 看到 [] 裡面是 "字串" (如 ["Name"])通常是 Dictionary。

# 4. 流程控制與函式設計

## 1: 邏輯判斷： if-elif-else

- 縮排 (Indentation) 決定程式區塊。

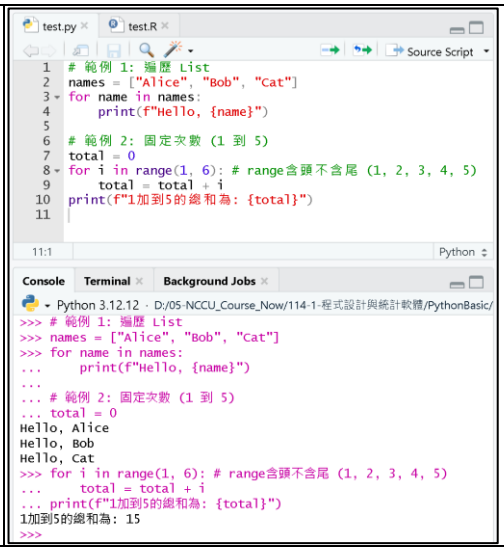
```
score = 75

if score >= 90:
    print("等級 A")
elif score >= 60:
    print("等級 Pass")
else:
    print("等級 Fail")

 # 注意：這裡沒有縮排，跳出判斷式，一定會執行
 print("判斷結束")
```


## 2: 迴圈控制：For Loop

- 重點： 搭配 range() 使用。

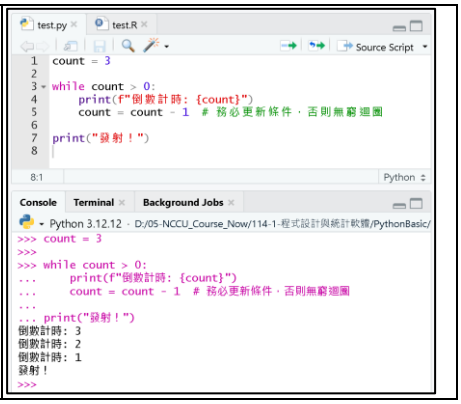
```
# 範例 1: 遍歷 List
names = ["Alice", "Bob", "Cat"]
for name in names:
    print(f"Hello, {name}")

# 範例 2: 固定次數 (1 到 5)
total = 0
for i in range(1, 6): # range 含頭不含尾 (1, 2, 3, 4, 5)
    total = total + i
print(f"1 加到 5 的總和為: {total}")
```




## 3: 迴圈控制：While Loop

- 重點： 條件成立時持續執行。

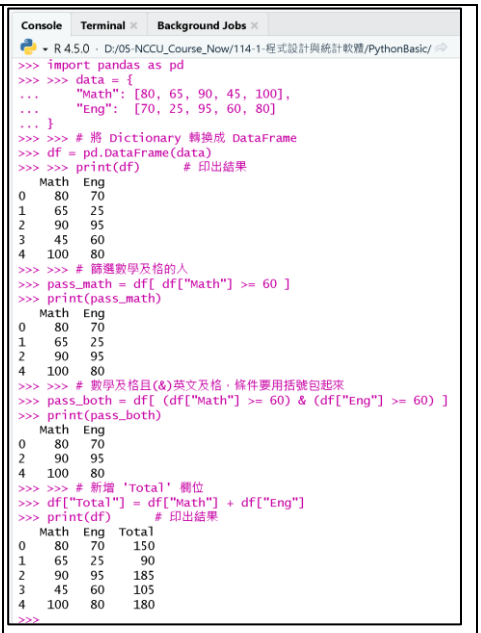
```python
count = 3

while count > 0:
    print(f"倒數計時: {count}")
    count = count - 1  # 務必更新條件，否則無窮迴圈

print("發射！")
```




## 4: 條件篩選

- 重點： Boolean Indexing (條件篩選)。

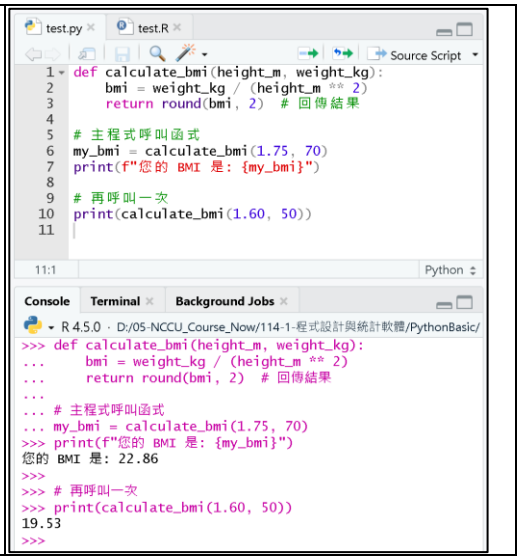
```python
import pandas as pd

data = {
    "Math": [80, 65, 90, 45, 100],
    "Eng":  [70, 25, 95, 60, 80]
}

# 將 Dictionary 轉換成 DataFrame
df = pd.DataFrame(data)

print(df)      # 印出結果

# 篩選數學及格的人
pass_math = df[ df["Math"] >= 60 ]
print(pass_math)

# 數學及格且(&)英文及格，條件要用括號包起來
pass_both = df[ (df["Math"] >= 60) & (df["Eng"] >= 60) ]
print(pass_both)

# 新增 'Total' 欄位
df["Total"] = df["Math"] + df["Eng"]
print(df)      # 印出結果
```




## 4: 函式設計與編寫

- 重點： def 定義，return 回傳值。

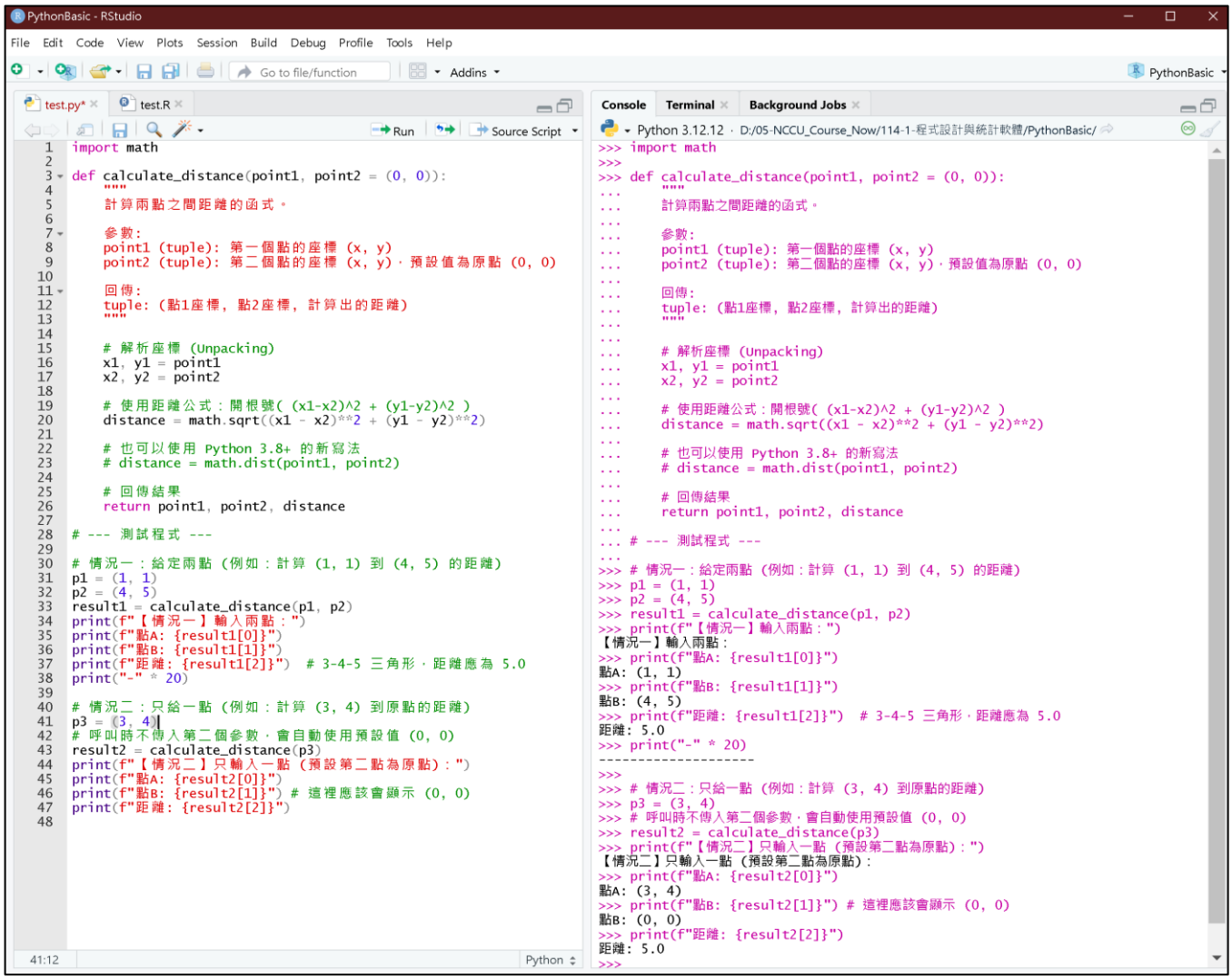
```python
# 定義函式：計算 BMI
def calculate_bmi(height_m, weight_kg):
    bmi = weight_kg / (height_m ** 2)
    return round(bmi, 2)  # 回傳結果

# 主程式呼叫函式
my_bmi = calculate_bmi(1.75, 70)
print(f"您的 BMI 是: {my_bmi}")

# 再呼叫一次
print(calculate_bmi(1.60, 50))
```




- 範例：用 Python 寫一個程式: 座標上兩點，計算其距離，若第二點沒給座標，就以(0，0)為預設值，回傳輸入的兩點座標及其距離。
- 三個重要觀念：
    - 函式的預設參數 (Default Arguments)：處理「若沒給座標，就以 (0,0) 為預設值」的需求。預設參數的寫法： 在定義函式時寫 def calculate_distance(point1, point2=(0, 0)):，這個 =(0, 0) 就是預設參數。
    - 元組 (Tuple)：用 (x, y) 這種資料型態來代表一個座標點。程式碼中的 x1, y1 = point1 是一個很 Pythonic 的寫法，可以一次把 (3, 4) 分別賦值給 x1 和 y1，這在其他語言比較少見。
    - 數學模組 (Math Module)：使用平方根函數。sqrt (開根號) 不是內建指令，必須先 import math 才能使用。如果有 Python 3.8 以上的環境，也可以順便介紹 math.dist(p1, p2) 這個指令，它直接幫你算好距離，不用自己寫公式。

```python
import math

def calculate_distance(point1, point2 = (0, 0)):
    """
    計算兩點之間距離的函式。

    參數:
    point1 (tuple): 第一個點的座標 (x, y)
    point2 (tuple): 第二個點的座標 (x, y)，預設值為原點 (0, 0)

    回傳:
    tuple: (點 1 座標, 點 2 座標, 計算出的距離)
    """
```

```
    # 解析座標 (Unpacking)
    x1, y1 = point1
    x2, y2 = point2

    # 使用距離公式：開根號( (x1-x2)^2 + (y1-y2)^2 )
    distance = math.sqrt((x1 - x2)**2 + (y1 - y2)**2)

    # 也可以使用 Python 3.8+ 的新寫法
    # distance = math.dist(point1, point2)

    # 回傳結果
    return point1, point2, distance

# --- 測試程式 ---

# 情況一：給定兩點 (例如：計算 (1, 1) 到 (4, 5) 的距離)
p1 = (1, 1)
p2 = (4, 5)
result1 = calculate_distance(p1, p2)
print(f"【情況一】輸入兩點：")
print(f"點 A: {result1[0]}")
print(f"點 B: {result1[1]}")
print(f"距離: {result1[2]}")  # 3-4-5 三角形，距離應為 5.0
print("-" * 20)

# 情況二：只給一點 (例如：計算 (3, 4) 到原點的距離)
p3 = (3, 4)
# 呼叫時不傳入第二個參數，會自動使用預設值 (0, 0)
result2 = calculate_distance(p3)
print(f"【情況二】只輸入一點 (預設第二點為原點)：")
print(f"點 A: {result2[0]}")
print(f"點 B: {result2[1]}") # 這裡應該會顯示 (0, 0)
print(f"距離: {result2[2]}")
```


PythonBasic - RStudio
File Edit Code View Plots Session Build Debug Profile Tools Help
Go to file/function
Addins
PythonBasic
test.py*
test.R
Run
Source Script
import math
def calculate_distance(point1, point2 = (0, 0)):
"""
計算兩點之間距離的函式。
參數:
point1 (tuple): 第一個點的座標 (x, y)
point2 (tuple): 第二個點的座標 (x, y)，預設值為原點 (0, 0)
回傳:
tuple: (點1座標, 點2座標, 計算出的距離)
"""
# 解析座標 (Unpacking)
x1, y1 = point1
x2, y2 = point2
# 使用距離公式：開根號( (x1-x2)^2 + (y1-y2)^2 )
distance = math.sqrt((x1 - x2)**2 + (y1 - y2)**2)
# 也可以使用 Python 3.8+ 的新寫法
# distance = math.dist(point1, point2)
# 回傳結果
return point1, point2, distance
# --- 測試程式 ---
# 情況一：給定兩點 (例如：計算 (1, 1) 到 (4, 5) 的距離)
p1 = (1, 1)
p2 = (4, 5)
result1 = calculate_distance(p1, p2)
print(f"【情況一】輸入兩點：")
print(f"點A: {result1[0]}")
print(f"點B: {result1[1]}")
print(f"距離: {result1[2]}")  # 3-4-5 三角形，距離應為 5.0
print("-" * 20)
# 情況二：只給一點 (例如：計算 (3, 4) 到原點的距離)
p3 = (3, 4)
# 呼叫時不傳入第二個參數，會自動使用預設值 (0, 0)
result2 = calculate_distance(p3)
print(f"【情況二】只輸入一點 (預設第二點為原點)：")
print(f"點A: {result2[0]}")
print(f"點B: {result2[1]}") # 這裡應該會顯示 (0, 0)
print(f"距離: {result2[2]}")
41:12
Python
Console
Terminal
Background Jobs
Python 3.12.12 · D:/05-NCCU_Course_Now/114-1-程式設計與統計軟體/PythonBasic/
>>> import math
>>>
>>> def calculate_distance(point1, point2 = (0, 0)):
... """
... 計算兩點之間距離的函式。
... 參數:
... point1 (tuple): 第一個點的座標 (x, y)
... point2 (tuple): 第二個點的座標 (x, y)，預設值為原點 (0, 0)
... 回傳:
... tuple: (點1座標, 點2座標, 計算出的距離)
... """
... # 解析座標 (Unpacking)
... x1, y1 = point1
... x2, y2 = point2
... # 使用距離公式：開根號( (x1-x2)^2 + (y1-y2)^2 )
... distance = math.sqrt((x1 - x2)**2 + (y1 - y2)**2)
... # 也可以使用 Python 3.8+ 的新寫法
... # distance = math.dist(point1, point2)
... # 回傳結果
... return point1, point2, distance
... # --- 測試程式 ---
>>> # 情況一：給定兩點 (例如：計算 (1, 1) 到 (4, 5) 的距離)
>>> p1 = (1, 1)
>>> p2 = (4, 5)
>>> result1 = calculate_distance(p1, p2)
>>> print(f"【情況一】輸入兩點：")
【情況一】輸入兩點：
>>> print(f"點A: {result1[0]}")
點A: (1, 1)
>>> print(f"點B: {result1[1]}")
點B: (4, 5)
>>> print(f"距離: {result1[2]}")  # 3-4-5 三角形，距離應為 5.0
距離: 5.0
>>> print("-" * 20)
--------------------
>>>
>>> # 情況二：只給一點 (例如：計算 (3, 4) 到原點的距離)
>>> p3 = (3, 4)
>>> # 呼叫時不傳入第二個參數，會自動使用預設值 (0, 0)
>>> result2 = calculate_distance(p3)
>>> print(f"【情況二】只輸入一點 (預設第二點為原點)：")
【情況二】只輸入一點 (預設第二點為原點)：
>>> print(f"點A: {result2[0]}")
點A: (3, 4)
>>> print(f"點B: {result2[1]}") # 這裡應該會顯示 (0, 0)
點B: (0, 0)
>>> print(f"距離: {result2[2]}")
距離: 5.0
>>>

# 5. Pandas 資料讀取與核心操作

Pandas 是 Python 程式語言中，專門用來進行資料分析與處理的最核心套件。如果不使用太艱澀的術語，你可以這樣理解它：Pandas 就是「Python 版的 Excel」，而且是可程式化、能處理更大量數據的超級 Excel。它是建立在 NumPy（數值運算套件）之上的工具，讓 Python 能夠像 R 語言一樣，輕鬆處理「表格」型態的資料。

---

## 1: Pandas 套件介紹

❖ **為什麼 Pandas 這麼重要？**

在沒有 Pandas 之前，Python 處理資料非常麻煩（要自己寫迴圈去解析文字檔）。Pandas 出現後解決了以下問題：

- 讀取方便：一行指令就能讀取 CSV、Excel、SQL 資料庫、JSON 等格式。
- 清理資料：處理缺失值（空值）、去除重複、格式轉換非常直覺。
- 資料操作：可以像 Excel 一樣做樞紐分析表（Pivot Table），或像 SQL 一樣進行資料合併（Merge/Join）。
- 時間序列：它對「時間日期」的處理非常強大（這是因為 Pandas 最初是為了金融交易分析而開發的）。

❖ **Pandas 的兩個核心資料結構:** 學習 Pandas 其實就是學這兩個東西：

- Series (單維度)
  - 概念：你可以把它想像成 Excel 裡面的「某一欄」，或者 R 語言中的 Vector (向量)。
  - 它帶有索引（Index），所以你可以知道這一排數據分別代表什麼。
- DataFrame (雙維度)
  - 概念：把它想像成一張完整的 Excel 工作表，或者 R 語言中的 data.frame。
  - 它是由很多個 Series 組成的，有「列索引 (Index)」和「欄位名稱 (Columns)」。

❖ **給 R 語言使用者的快速對照**

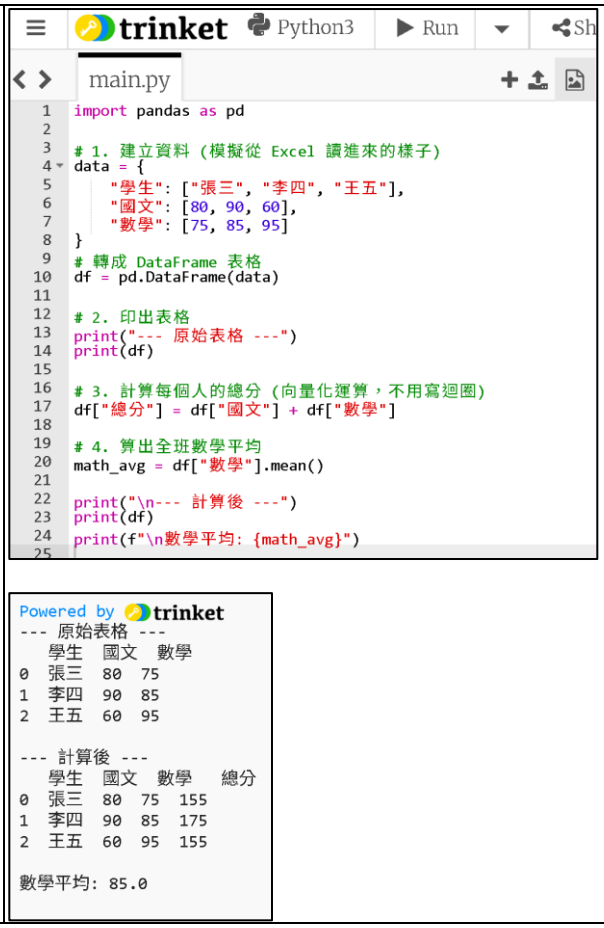
| R 語言概念 | Pandas 對應概念 | 備註 |
|---|---|---|
| `data.frame` | `pd.DataFrame` | 幾乎一模一樣，都是列與欄的表格。 |
| `vector / list` | `pd.Series` | 單一欄位的資料。 |
| `NA` | `NaN (Not a Number)` | 代表缺失值。 |
| `summary()` | `df.describe()` | 快速看平均值、標準差等統計量。 |

| | | |
|---|---|---|
| head() | df.head() | 看前幾筆資料。 |
| subset() / filter() | Boolean Indexing | 例如 df[df['age'] > 20] |

## 2: 建立範例資料與 DataFrame

- 重點：一個典型的 Pandas 流程：建立手動建立 DataFrame 資料 -> 查看資料 -> 簡單計算。

```python
import pandas as pd

# 1. 建立資料 (模擬從 Excel 讀進來的樣子)
data = {
    "學生": ["張三", "李四", "王五"],
    "國文": [80, 90, 60],
    "數學": [75, 85, 95]
}
# 轉成 DataFrame 表格
df = pd.DataFrame(data)

# 2. 印出表格
print("--- 原始表格 ---")
print(df)

# 3. 計算每個人的總分 (向量化運算，不用寫迴圈)
df["總分"] = df["國文"] + df["數學"]

# 4. 算出全班數學平均
math_avg = df["數學"].mean()

print("\n--- 計算後 ---")
print(df)
print(f"\n 數學平均: {math_avg}")
```




## 3: 讀取外部資料 (CSV)

- 重點： read_csv 與中文編碼處理。

```python
import pandas as pd

# 假設檔案在同目錄下。若是中文檔名或內容，建議加 encoding
try:
    df_csv = pd.read_csv("scores.csv", encoding="utf-8")
    print("CSV 讀取成功")
except:
    print("找不到檔案，請確認路徑")
```

## 4: 讀取外部資料 (Excel)

- 重點： read_excel。

```
# 讀取 Excel (需先 pip install openpyxl)
# 或於 R 環境中安裝
# library(reticulate)
# py_install("pandas")

import pandas as pd
df_xlsx = pd.read_excel("data.xlsx", sheet_name="Sheet1")
```

## 5: 資料檢視指令

- 重點： head, info, describe 。

```
print(df_xlsx.head(2))   # 看前 2 筆

print(df_xlsx.info())    # 看資料型態、是否有缺失值

print(df_xlsx.describe()) # 看數值統計 (平均、標準差等)
```

# 7. 資料視覺化與統計繪圖

## 1: 基礎繪圖設定

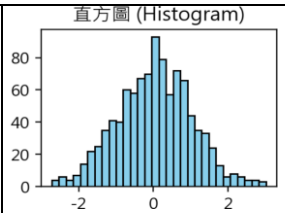
- R 繪圖 習慣用 Data Frame，Python 繪圖則常直接傳入 List 或 Numpy Array。
- 每個繪圖指令都有豐富的參數(顏色與樣式)，如 color (顏色), alpha (透明度), bins (直方圖的柱數) 。
- 中文字型亂碼問題，是 Python 繪圖最常見的坑。必須設定 plt.rcParams['font.sans-serif']，否則中文字會變成方框 □□。(這段程式碼必備)

```
import matplotlib.pyplot as plt
import numpy as np

# Windows 使用 'Microsoft JhengHei' (微軟正黑體)
# Mac 使用 'Heiti TC' (黑體-繁) 或 'Arial Unicode MS'
import platform
if platform.system() == "Windows":
    plt.rcParams['font.sans-serif'] = ['Microsoft JhengHei']
else:
    plt.rcParams['font.sans-serif'] = ['Arial Unicode MS']

plt.rcParams['axes.unicode_minus'] = False   # 解決負號顯示為方塊的問題
```

## 2: 直方圖 (Histogram)

- 適用：看連續資料的分佈 (如：全校身高分佈)

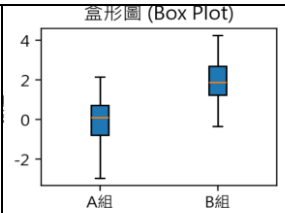
```
plt.figure() # (養成好習慣) 宣告建立第一張畫布
data_hist = np.random.randn(1000) # 產生 1000 筆常態分佈數據
plt.hist(data_hist, bins = 30, color = 'skyblue', edgecolor = 'black')
plt.title('直方圖 (Histogram)')
plt.xlabel('數值')
plt.ylabel('頻率')
plt.show()
```




## 3: 盒形圖 (Box Plot)

- 適用：看資料的四分位數、中位數及離群值

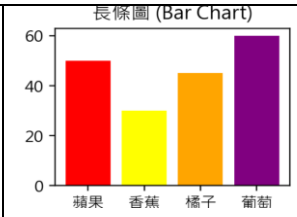
```
# 兩組資料
# 結束上一張，開新的一張畫布，不然圖形會和上一張重疊
plt.figure()
data_box = [np.random.normal(0, 1, 100), np.random.normal(2, 1, 100)]
plt.boxplot(data_box, patch_artist = True, labels=['A 組', 'B 組'])
plt.title('盒形圖 (Box Plot)')
plt.ylabel('數值')
plt.show()
```



## 4: 長條圖 (Bar Chart)

- 適用：比較不同類別的數值大小 (如：各部門業績)

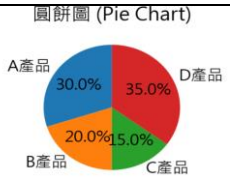
```
categories = ['蘋果', '香蕉', '橘子', '葡萄']
values = [50, 30, 45, 60]
plt.bar(categories, values, color = ['red', 'yellow', 'orange',
'purple'])
plt.title('長條圖 (Bar Chart)')
plt.ylabel('銷售量')
plt.show()
# 避免畫圖重疊的第二種方法:
# 強制關閉當前畫布，下次畫圖時會被迫開新的
plt.close()
```




## 5: 圓餅圖 (Pie Chart)

- 適用：看各類別佔總體的比例

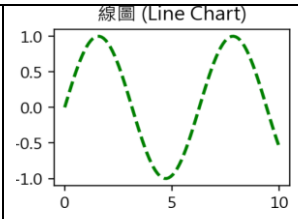
```
plt.figure()
sizes = [30, 20, 15, 35]
labels = ['A 產品', 'B 產品', 'C 產品', 'D 產品']
plt.pie(sizes, labels = labels, autopct = '%1.1f%%', startangle = 90)
plt.title('圓餅圖 (Pie Chart)')
plt.show()
plt.close()
```




## 6: 線圖 (Line Chart)

- 適用：看趨勢變化 (如：股價、氣溫)

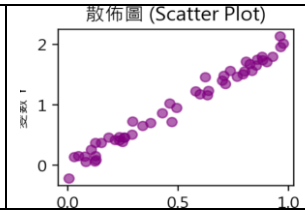
```
plt.figure()
x = np.linspace(0, 10, 100) # 0 到 10 之間產生 100 個點
y = np.sin(x) # sin 波形
plt.plot(x, y, color = 'green', linewidth = 2, linestyle = '--')
plt.title('線圖 (Line Chart)')
plt.xlabel('時間')
plt.ylabel('數值')
plt.show()
```




## 7: 散佈圖 (Scatter Plot)

- 適用：看兩個變數之間的相關性 (如：身高 vs 體重)

```
plt.figure()
x_scatter = np.random.rand(50)
y_scatter = x_scatter * 2 + np.random.normal(0, 0.1, 50) # 正相關資料
plt.scatter(x_scatter, y_scatter, color = 'purple', alpha = 0.6) # alpha
為透明度
```

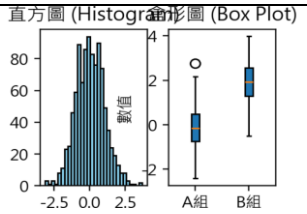


```
plt.title('散佈圖 (Scatter Plot)')
plt.xlabel('變數 X')
plt.ylabel('變數 Y')
plt.show()
```

## 8: 子圖概念 (Subplot)

- plt.subplot(列, 行, 編號)：這是將多張圖畫在同一視窗的技巧。

```
# 設定畫布大小(寬, 高)
plt.figure(figsize = (15, 10))

# 直方圖 (Histogram)
plt.subplot(1, 2, 1) # 切成 1 列2 行，這是第 1 張
data_hist = np.random.randn(1000)
plt.hist(data_hist, bins = 30, color = 'skyblue', edgecolor = 'black')
plt.title('直方圖 (Histogram)')
plt.xlabel('數值')
plt.ylabel('頻率')

# 盒形圖 (Box Plot)
plt.subplot(1, 2, 2) # 第 2 張
data_box = [np.random.normal(0, 1, 100), np.random.normal(2, 1, 100)]
plt.boxplot(data_box, patch_artist = True, labels = ['A 組', 'B 組'])
plt.title('盒形圖 (Box Plot)')
plt.ylabel('數值')

# 自動調整子圖間距，避免重疊
plt.tight_layout()

# 顯示圖形
plt.show()
```




## 8: 錯誤解決: 套件引入有錯誤

- 錯誤訊息

**[Python Code]**

```
>>> import matplotlib.pyplot as plt
Traceback (most recent call last):
  File "<string>", line 1, in <module>
  File "C:\Users\hanmi\AppData\Local\R\win-library\4.5\reticulate\python\rpytools\loader.py", line 122, in
_find_and_load_hook
    return _run_hook(name, _hook)
           ^^^^^^^^^^^^^^^^^^^^^^
           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
ModuleNotFoundError: No module named 'matplotlib'
```

**[R Code]**

```
> reticulate::py_install("matplotlib")
```

```
Warning message:
In reticulate::py_install("matplotlib") :
  An ephemeral virtual environment managed by 'reticulate' is currently in use.
To add more packages to your current session, call `py_require()` instead
of `py_install()`. Running:
  `py_require(c("matplotlib"))`
```

**[R Code]**

```
> reticulate::py_require("matplotlib")
> reticulate::repl_python()
Python 3.12.12 (C:/Users/hanmi/AppData/Local/R/cache/R/reticulate/uv/cache/archive-
v0/mrbzvGiJfkTwWd3_gaab_/Scripts/python.exe)
Reticulate 1.44.1 REPL -- A Python interpreter in R.
Enter 'exit' or 'quit' to exit the REPL and return to R.
```

**[Python Code]**

```
>>> import matplotlib.pyplot as plt
Traceback (most recent call last):
  File "<string>", line 1, in <module>
  File "C:\Users\hanmi\AppData\Local\R\win-library\4.5\reticulate\python\rpytools\loader.py", line 122, in
_find_and_load_hook
    return _run_hook(name, _hook)
…           ^^^^^^^^^^^^^^^^^^^^^
ModuleNotFoundError: No module named 'matplotlib'
```

- 解決方式

**[R code]**

```r
library(reticulate)

# 先讓 R 自動下載並安裝一個標準版的 Python (這可能需要幾分鐘)
install_python()

# 1. 建立一個固定的虛擬環境 (名稱叫 my_course_env)
# 這樣就不會是用完即丟的暫時環境了
virtualenv_create("my_course_env")

# 2. 強制指定使用這個環境
use_virtualenv("my_course_env", required = TRUE)

# 3. 在這個特定環境中安裝您要的套件
# 我們一次把它裝好：pandas (資料處理), matplotlib (畫圖)
py_install(c("pandas", "matplotlib"), envname = "my_course_env")
```

**[Python Code]**

```python
import pandas as pd
import matplotlib.pyplot as plt

print("pandas version:", pd.__version__)
print("matplotlib is ready!")
print("安裝成功！")
```