

Python 程式設計：從零開始

快速開始

超新手先這樣做 - Mac/Linux (1/2)

建立環境：

```
# 1) 建一個練習資料夾  
mkdir ~/Desktop/my-first-python  
cd ~/Desktop/my-first-python
```

```
# 2) 用 VSCode 打開  
code .
```

建立 **hello.py** 檔案，輸入：

超新手先這樣做 - Mac/Linux (2/2)

執行程式：

```
python3 hello.py
```

看到：

```
Hello, Python!
```

成功！

超新手先這樣做 - Windows (1/2)

建立環境：

```
# 1) 建一個練習資料夾  
mkdir ~/Desktop/my-first-python  
cd ~/Desktop/my-first-python  
  
# 2) 用 VSCode 打開  
code .
```

建立 `hello.py` 檔案，輸入：

```
print("Hello, Python!")
```

超新手先這樣做 - Windows (2/2)

執行程式：

```
python hello.py
```

看到：

```
Hello, Python!
```

成功！

基礎概念

什麼是 Python ？

Python 是一個友善易學的程式語言

特色：

-  簡單好懂：程式碼像說話
-  應用廣泛：網頁、AI、數據分析
-  社群龐大：資源超多
-  工具豐富：有很多現成套件

為什麼學 Python ？

就像學智慧型手機：

- 簡單操作就能做很多事
- 全世界都在用
- 直接上手，不用太複雜

Python 版本

Python 2 vs 3 :

```
# Python 2 (已停止支援)  
print "Hello"
```

```
# Python 3 (目前版本)  
print("Hello")
```

建議：使用 Python 3.x

為什麼有兩個版本？

就像手機系統更新：

- 功能一樣，但新版更安全、更流暢
- Python 3 是「更好用的新版」

安裝 Python

Windows

推薦：從官網下載安裝

1. 訪問官網：開啟瀏覽器到 <https://python.org/downloads/>
2. 下載最新版本：點擊 "Download Python 3.x.x"
3. 執行安裝程式：
 - 務必勾選 **"Add Python to PATH"**
 - 選擇 "Customize installation"
 - 確保 pip 被選中
4. 驗證安裝：

macOS

推薦：使用 Homebrew

1. 安裝 Homebrew (如果還沒裝)：

```
/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"
```

2. 安裝 Python：

```
brew install python
```

3. 驗證安裝：

```
python3 --version
```

Linux

使用 apt 套件管理器

```
# 更新套件列表
```

```
sudo apt update
```

```
# 安裝 Python 3 和 pip
```

```
sudo apt install python3 python3-pip
```

```
# 驗證安裝
```

```
python3 --version
```

```
pip3 --version
```

進階：使用 pyenv 管理多版本

```
# 安裝 pyenv
```

你的第一個 Python 程式

Mac/Linux

```
# 建立 hello.py 檔案  
echo 'print("Hello, Python!")' > hello.py  
  
# 執行程式  
python3 hello.py
```

Windows

建立 *hello.py* 檔案

```
New-Item hello.py -Value 'print("Hello, Python!")'
```

執行程式

```
python hello.py
```

print 是什麼？

`print` 就像 Python 的「嘴巴」：

- `print("hello")` → 說 hello
- `print(123)` → 說 123
- `print(variable)` → 說出變數的內容

變數與資料類型

```
# 文字 (字串)  
name = "小明"
```

```
# 數字  
age = 25  
height = 170.5
```

```
# 布林值  
is_student = True
```

```
# 印出來看看  
print(name, age, is_student)
```


變數是什麼？

變數就像「手機聯絡人」：

- 名字（變數名稱）
- 存電話、email（資料）
- 可以隨時改資訊

條件判斷

if-else 就像十字路口的選擇：

```
age = int(input("你幾歲？"))

if age >= 18:
    print("你可以投票！")
    print("也可以開車！")
elif age >= 13:
    print("你是青少年")
else:
    print("你是小孩")
    print("要聽爸爸媽媽的話！")
```

💡 小提示：縮排很重要！

迴圈

for 迴圈：重複做事

```
# 印出 1 到 5
for i in range(1, 6):
    print(f"這是第 {i} 次")

# 走訪清單
fruits = ["蘋果", "香蕉", "橘子"]
for fruit in fruits:
    print(f"我喜歡吃 {fruit}")
```

while 迴圈：條件重複

```
# 猜數字遊戲
```

函數

```
# 定義函數
def greet(name):
    return f"哈囉，{name}！"

def calculate_area(width, height):
    return width * height

# 使用函數
message = greet("小明")
area = calculate_area(10, 5)
print(message, area)
```

為什麼要用函數？

想像做菜：

- 沒有函數：每次都要重新切菜
- 有函數：用預切的菜包

函數就是「可以重複用的工具」

模組與套件

使用內建模組

```
import random
```

```
import datetime
```

產生隨機數字

```
number = random.randint(1, 100)
```

取得現在時間

```
now = datetime.datetime.now()
```

取別名使用

```
import numpy as np
```

模組 vs 套件

- 模組：單一的 `.py` 檔案（一個 app）
- 套件：包含多個模組的資料夾（整個 app 商店）

虛擬環境 (venv)

每個專案有獨立的套件環境

Windows :

```
Set-ExecutionPolicy Bypass -Scope Process -Force  
python -m venv myproject_env  
myproject_env\Scripts\activate
```

Mac/Linux :

```
python3 -m venv myproject_env  
source myproject_env/bin/activate
```


為什麼需要虛擬環境？

就像每個專案的「私人房間」：

- A 專案：自己的 venv，有自己的套件
- B 專案：自己的 venv，有自己的套件
- 互不干擾，和平共處

套件管理：pip

pip 就像 App Store，但專門裝 Python 套件！

```
# 安裝套件
pip install requests      # 安裝網路請求套件
pip install fastapi       # 安裝 web 框架
pip install pandas        # 安裝資料分析套件
```

```
# 安裝特定版本
pip install requests==2.28.0
```

```
# 更新套件
pip install --upgrade requests
```

```
# 移除套件
pip uninstall requests
```

```
# 看已安裝的套件
pip list
```

依賴管理檔案

requirements.txt：舊版的管理方式

```
# requirements.txt
requests==2.28.0
fastapi==0.100.0
uvicorn==0.23.0
pandas>=1.5.0
```

安裝所有依賴：

```
pip install -r requirements.txt
```

問題：

pyproject.toml：現代的管理方式

pyproject.toml：新世代的專案設定檔

```
[build-system]
requires = ["hatchling"]
build-backend = "hatchling.build"
```

```
[project]
name = "my-awesome-project"
version = "0.1.0"
description = "一個超棒的專案"
readme = "README.md"
requires-python = ">=3.8"
dependencies = [
    "fastapi>=0.100.0",
    "uvicorn>=0.23.0",
    "requests>=2.28.0"
]
```

```
[project.optional-dependencies]
dev = [
    "pytest>=7.0.0",
    "black>=23.0.0"
```

比較：requirements.txt vs pyproject.toml

功能	requirements.txt	pyproject.toml
套件版本	✓	✓
專案資訊	✗	✓
Python 版本要求	✗	✓
選用依賴	✗	✓
工具設定	✗	✓
現代化	舊版	新版

Python 開發工具大集合

程式碼編輯器：

- **VSCode**：免費，超強大，Python 支援一流
- **PyCharm**：專業 IDE，功能完整但收費
- **Sublime Text**：輕量快速

版本管理：

- **pyenv**：管理多個 Python 版本
- **conda**：科學計算環境管理

程式碼品質：

開發環境設定步驟

新手完整設定指南：

1. 安裝 Python

```
# 確認安裝  
python --version # 應該是 3.8 或以上
```

2. 安裝程式碼編輯器

- 下載 VSCode
- 安裝 Python 擴充套件

3. 建立專案資料夾

常見錯誤與解決方案

✗ 錯誤：ModuleNotFoundError

原因：忘記安裝套件或啟動虛擬環境

解決：

1. 檢查是否在虛擬環境中（提示字元有 `(venv)`）
2. `pip install` 缺少的套件
3. 或 `pip install -r requirements.txt`

✗ 錯誤：SyntaxError

原因：語法錯誤

解決：

1. 檢查括號是否成對
2. 檢查縮排是否正確

Python 學習資源

官方資源：

- Python 官方文檔 (英文最佳)
- Python 教學網站

線上學習平台：

- Codecademy Python 課程
- freeCodeCamp
- Coursera Python 課程

社群與論壇：

Python Cheat Sheet

基本語法：

```
# 變數
name = "小明"
age = 25

# 條件
if age >= 18:
    print("成年")
else:
    print("未成年")

# 迴圈
for i in range(5):
    print(i)

# 函數
def greet(name):
    return f"哈囉 {name}"
```

總結：Python 的學習心態

學習 Python 的心法

💡 學習 Python 的心法：

1. 不要怕錯 - 錯誤是最好的老師
2. 從小開始 - 先學基本語法，再學進階
3. 多實作 - 看再多不如動手做一次
4. 找問題解 - 卡關時是進步的機會

學習心法續

5. 看別人程式 - GitHub 是寶庫

6. 加入社群 - 大家一起學比較快樂

 你的 Python 之路：

Python 學習路徑

新手	→	基本語法	→	專案實作	→	深入研究	→	專業開發
↓		↓		↓		↓		↓
初學		語法熟悉		獨立開發		框架精通		架構設計

記住：每一個專家都曾是新手！