FastAPI 入門：打造你的第一個 Web API

超新手最小路徑 - Mac/Linux (1/2)

環境設置：

```
# 1) 新建資料夾並進入
mkdir ~/Desktop/my-api && cd ~/Desktop/my-api

# 2) 建立並啟用虛擬環境
python3 -m venv venv
source venv/bin/activate

# 3) 安裝 FastAPI
pip install "fastapi[standard]"
```

超新手最小路徑 - Mac/Linux (2/3)

建立 `main.py` ：

```
from fastapi import FastAPI

app = FastAPI()

@app.get("/")
def home():
    return {"message": "Hello FastAPI"}
```

超新手最小路徑 - Mac/Linux (3/3)

啟動：

```
fastapi dev main.py
```

開啟瀏覽器：

- http://127.0.0.1:8000

超新手最小路徑 - Windows (1/3)

環境設置：

```
# 1) 新建資料夾並進入
mkdir ~/Desktop/my-api; cd ~/Desktop/my-api

# 2) 設定權限
Set-ExecutionPolicy Bypass -Scope Process -Force

# 3) 建立並啟用虛擬環境
python -m venv venv
venv\Scripts\activate

# 4) 安裝 FastAPI
pip install "fastapi[standard]"
```

超新手最小路徑 - Windows (2/3)

建立 `main.py` ：

```
from fastapi import FastAPI

app = FastAPI()

@app.get("/")
def home():
    return {"message": "Hello FastAPI"}
```

超新手最小路徑 - Windows (3/3)

啟動：

```
fastapi dev main.py
```

開啟瀏覽器：

- http://127.0.0.1:8000

常見卡關與疑難排解

啟動問題：

- 連不到 8000 → 看終端機是否正在跑，或被占用改成 `--port 8001`
- 虛擬環境未啟用 → 確認終端機提示符前有 `(venv)`

資料格式問題：

- 422 錯誤多半是 JSON 格式問題（最後一行不要加逗號）
- 確保 JSON 中所有字串使用雙引號 `"`

什麼是 FastAPI?

FastAPI 是一個超好用的 Python 工具箱

專門用來打造「網路應用程式介面」(API)，讓不同程式可以互相通訊。

FastAPI 的超能力：

- 🚀 **跑很快**：比很多工具還要快
- 📝 **自動寫說明**：幫你寫 API 文件
- 🔍 **智能檢查**：自動檢查資料對不對
- 🛠️ **開發爽爽**：少寫錯程式碼

安裝 FastAPI - Windows

```
# 設定權限
Set-ExecutionPolicy Bypass -Scope Process -Force

# 建立虛擬環境
python -m venv venv
venv\Scripts\activate

# 安裝 FastAPI
pip install "fastapi[standard]"
```

安裝 FastAPI - macOS

```
# 建立虛擬環境
python3 -m venv venv
source venv/bin/activate

# 安裝 FastAPI
pip install "fastapi[standard]"
```

FastAPI 包含的工具

- `fastapi` - 主要框架
- `uvicorn` - 網路伺服器
- `pydantic` - 資料驗證

虛擬環境的重要性

💡 小提示：為什麼需要虛擬環境？

專案環境隔離很重要！

沒有虛擬環境，所有專案共用同一個工具箱，
結果誰都不能爽快開發！

環境髒掉的問題超級常見！

VSCode 開發環境設定

1. 安裝 Python 擴充套件：

- 在 VSCode 中安裝 "Python" 擴充套件
- 安裝 "Pylance" 擴充套件 (更好的 Python 支援)

2. 選擇 Python 直譯器：

- 按 `Ctrl+Shift+P` (Mac: `Cmd+Shift+P`)
- 輸入 "Python: Select Interpreter"
- 選擇你的虛擬環境 Python

3. 在 VSCode 中開啟終端機：

第一步：建立 API - Windows

```
# 建立 main.py
New-Item main.py -Value @'
from fastapi import FastAPI

app = FastAPI()

@app.get("/")
def read_root():
    return {"Hello": "World"}
'@

# 啟動
fastapi dev main.py
```

第一步：建立 API - macOS

```
# 建立 main.py
cat > main.py << 'EOF'
from fastapi import FastAPI

app = FastAPI()

@app.get("/")
def read_root():
    return {"Hello": "World"}
EOF

# 啟動
fastapi dev main.py
```

裝飾器 (Decorator)

`@app.get("/")` 是什麼？

告訴 FastAPI：「這個函數處理首頁的 GET 請求」

- `@app.get()` = GET 請求
- `@app.post()` = POST 請求
- `@app.put()` = PUT 請求

啟動與測試

開啟瀏覽器：

- http://127.0.0.1:8000
- http://127.0.0.1:8000/docs (自動文檔)

測試：

```
curl http://127.0.0.1:8000
# 回應: {"Hello": "World"}
```

Port 通訊埠

為什麼是 8000？

- `80` - HTTP 網站
- `443` - HTTPS 安全連線
- `8000` - 開發用

開發伺服器 (Dev Server)

特色：

- 修改程式碼自動重啟
- 顯示詳細錯誤訊息
- 方便開發測試

路徑參數 (Path Parameters)

網址中的變數：

```
/users/123/posts/456
       ↑          ↑
    user_id   post_id
```

路徑參數：程式碼

```python
@app.get("/items/{item_id}")
def read_item(item_id: int):
    return {"item_id": item_id}
```

測試：

```
GET /items/42 → {"item_id": 42}
```

類型提示 (Type Hints)

`: int` 的作用：

- 自動轉換字串為數字
- 自動驗證資料類型
- 錯誤時自動回傳 422

範例：

- `/items/42` → 成功 ✅
- `/items/abc` → 422 錯誤 ❌

路徑參數：字串類型

```python
@app.get("/users/{user_id}")
def read_user(user_id: str):
    return {"user_id": user_id}
```

測試：

```
GET /users/johndoe
→ {"user_id": "johndoe"}
```

預設路徑參數都是字串

查詢參數 (Query Parameters)

問號後面的參數：

```
/search?q=iphone&num=10
         ↑          ↑
       query     數量
```

用於：搜尋、過濾、排序

查詢參數：程式碼

```python
@app.get("/items/")
def read_items(skip: int = 0, limit: int = 10):
    return {"skip": skip, "limit": limit}
```

測試：

```
GET /items/?skip=20&limit=50
→ {"skip": 20, "limit": 50}

GET /items/
→ {"skip": 0, "limit": 10}
```

預設值

有 `=` 的參數可省略：

```
def read_items(skip: int = 0):  # 可省略
def search(q: str):             # 必填
```

- 有預設值 → 選填
- 無預設值 → 必填

查詢參數：必要參數

```python
@app.get("/search/")
def search_items(q: str, limit: int = 10):
    return {"q": q, "limit": limit}
```

測試：

```
GET /search/?q=apple
→ {"q": "apple", "limit": 10}
```

沒有預設值的參數為必要

請求主體 (Request Body)

傳送較大的資料：

- 路徑/查詢參數 → 簡單資料
- 請求主體 → 複雜結構化資料

用於：註冊、新增、更新資料

Pydantic BaseModel

```python
from pydantic import BaseModel

class Item(BaseModel):
    name: str
    price: float
    is_offer: bool = None
```

自動驗證資料格式

POST 方法

```python
@app.post("/items/")
def create_item(item: Item):
    return item
```

測試：

```
POST /items/
{
  "name": "iPhone",
  "price": 999.99
}
```

BaseModel 的好處

- ✅ 自動檢查資料格式
- ✅ 自動轉換類型
- ✅ 清楚的錯誤訊息
- ✅ 自動生成 API 文檔

JSON 格式

```
{
  "key": "value",
  "number": 123,
  "boolean": true
}
```

規則：

- ✅ 字串用雙引號 `"`
- ❌ 最後不加逗號 `,`

JSON 常見錯誤

```
{"name": "iPhone",}   ❌ 結尾逗號
{"name": "iPhone"}    ✅ 正確
```

422 錯誤：

資料格式不符合模型定義

查詢參數與字串驗證

```python
from fastapi import FastAPI

app = FastAPI()

@app.get("/items/")
def read_items(q: str | None = None):
    results = {"items": [{"item_id": "Foo"}, {"item_id": "Bar"}]}
    if q:
        results.update({"q": q})
    return results
```

使用 `str | None` 表示可選字串

路徑參數與數值驗證

```python
from fastapi import FastAPI

app = FastAPI()

@app.get("/items/{item_id}")
def read_item(item_id: int):
    return {"item_id": item_id}
```

路徑參數會自動轉換並驗證類型

測試：

```
GET /items/42 → 成功
GET /items/three → 錯誤（不是數字）
```

查詢參數模型

```python
from fastapi import FastAPI
from pydantic import BaseModel

class FilterParams(BaseModel):
    limit: int = 100
    offset: int = 0
    q: str | None = None

app = FastAPI()

@app.get("/items/")
def read_items(filter_query: FilterParams):
    return filter_query.dict()
```

將查詢參數組織成 Pydantic 模型

請求主體：多個參數

```python
from fastapi import FastAPI, Path
from pydantic import BaseModel

class Item(BaseModel):
    name: str
    price: float
    is_offer: bool = None

app = FastAPI()

@app.put("/items/{item_id}")
def update_item(
    item_id: int,
    item: Item,
    q: str | None = None
):
    result = {"item_id": item_id, **item.dict()}
    if q:
        result.update({"q": q})
    return result
```

請求主體：欄位設定

```python
from fastapi import FastAPI
from pydantic import BaseModel, Field

class Item(BaseModel):
    name: str = Field(examples=["iPhone"])
    price: float = Field(description="價格必須大於零")
    is_offer: bool = Field(default=None, description="是否特價")

app = FastAPI()

@app.put("/items/{item_id}")
def update_item(item_id: int, item: Item):
    return {"item_id": item_id, **item.dict()}
```

使用 Field() 設定欄位屬性

巢狀模型 (Nested Models)

```python
from fastapi import FastAPI
from pydantic import BaseModel

class Image(BaseModel):
    url: str
    name: str

class Item(BaseModel):
    name: str
    price: float
    image: Image | None = None

app = FastAPI()

@app.post("/items/")
def create_item(item: Item):
    return item
```

巢狀模型：請求範例

```
{
  "name": "iPhone",
  "price": 999.99,
  "image": {
    "url": "http://example.com/iphone.jpg",
    "name": "iPhone 15"
  }
}
```

巢狀結構會自動驗證

宣告請求範例資料

```python
from fastapi import FastAPI
from pydantic import BaseModel, Field

class Item(BaseModel):
    name: str = Field(examples=["iPhone"])
    price: float
    is_offer: bool = Field(default=None, examples=[True])

app = FastAPI()

@app.post("/items/")
def create_item(item: Item):
    return item
```

examples 會顯示在 API 文檔中

額外資料類型

```python
from datetime import datetime
from fastapi import FastAPI

app = FastAPI()

@app.put("/items/{item_id}")
def update_item(
    item_id: int,
    start_datetime: datetime,
    end_datetime: datetime,
    repeat_at: str | None = None
):
    return {
        "item_id": item_id,
        "start_datetime": start_datetime,
        "end_datetime": end_datetime,
        "repeat_at": repeat_at
    }
```

Cookie 參數

```python
from fastapi import FastAPI, Cookie

app = FastAPI()

@app.get("/items/")
def read_items(ads_id: str | None = Cookie(default=None)):
    return {"ads_id": ads_id}
```

使用 Cookie() 取得 Cookie 值

Header 參數

```python
from fastapi import FastAPI, Header

app = FastAPI()

@app.get("/items/")
def read_items(user_agent: str | None = Header(default=None)):
    return {"User-Agent": user_agent}
```

使用 Header() 取得 HTTP Header

Cookie 參數模型

```python
from fastapi import FastAPI, Cookie
from pydantic import BaseModel

class Cookies(BaseModel):
    session_id: str
    ads_id: str | None = None

app = FastAPI()

@app.get("/items/")
def read_items(cookies: Cookies):
    return cookies
```

將所有 Cookies 組織成模型

Header 參數模型

```python
from fastapi import FastAPI, Header
from pydantic import BaseModel

class CommonHeaders(BaseModel):
    host: str
    save_data: bool
    if_modified_since: str | None = None
    traceparent: str | None = None
    x_tag: list[str] = []

app = FastAPI()

@app.get("/items/")
def read_items(headers: CommonHeaders):
    return headers
```

回應模型：回傳類型

```python
from fastapi import FastAPI
from pydantic import BaseModel

class Item(BaseModel):
    name: str
    price: float
    is_offer: bool = None

app = FastAPI()

@app.post("/items/")
def create_item(item: Item) -> Item:
    return item
```

使用 -> Item 宣告回應類型

額外模型

```python
from fastapi import FastAPI
from pydantic import BaseModel, EmailStr

class UserIn(BaseModel):
    username: str
    password: str
    email: EmailStr
    full_name: str | None = None

class UserOut(BaseModel):
    username: str
    email: EmailStr
    full_name: str | None = None

app = FastAPI()

@app.post("/user/", response_model=UserOut)
def create_user(user: UserIn):
    return user
```

回應狀態碼

```python
from fastapi import FastAPI

app = FastAPI()

@app.post("/items/", status_code=201)
def create_item(name: str):
    return {"name": name}
```

常見 HTTP 狀態碼 (像收據上面的編號)：

- `200` - 成功搞定！(最常見的成功訊息)
- `201` - 新東西建立成功！
- `400` - 你送的資料怪怪的

表單資料 (Form Data)

```python
from fastapi import FastAPI, Form

app = FastAPI()

@app.post("/login/")
def login(username: str = Form(), password: str = Form()):
    return {"username": username}
```

使用 Form() 處理表單資料

請求檔案

```python
from fastapi import FastAPI, File, UploadFile

app = FastAPI()

@app.post("/files/")
def create_file(file: bytes = File()):
    return {"file_size": len(file)}

@app.post("/uploadfile/")
def create_upload_file(file: UploadFile):
    return {"filename": file.filename}
```

使用 File() 和 UploadFile 處理檔案上傳

請求表單與檔案

```python
from fastapi import FastAPI, File, Form, UploadFile

app = FastAPI()

@app.post("/files/")
def create_file(
    file: bytes = File(),
    fileb: UploadFile = File(),
    token: str = Form()
):
    return {
        "file_size": len(file),
        "token": token,
        "fileb_content_type": fileb.content_type,
    }
```

錯誤處理

```python
from fastapi import FastAPI, HTTPException

app = FastAPI()

@app.get("/items/{item_id}")
def read_item(item_id: int):
    if item_id == 3:
        raise HTTPException(status_code=418, detail="Nope! I don't like 3.")
    return {"item_id": item_id}
```

使用 HTTPException 拋出 HTTP 錯誤

路徑操作設定

```python
from fastapi import FastAPI

app = FastAPI()

@app.get(
    "/items/",
    summary="取得物品列表",
    description="取得所有物品的詳細列表",
    tags=["items"]
)
def read_items():
    return [{"name": "Item 1"}, {"name": "Item 2"}]
```

設定 API 文檔的描述和標籤

JSON 相容編碼器

```python
from fastapi import FastAPI
from fastapi.encoders import jsonable_encoder

app = FastAPI()

@app.get("/items/")
def read_items():
    data = [{"name": "Item 1", "price": 10.5}]
    # 將資料轉換為 JSON 相容格式
    json_compatible_data = jsonable_encoder(data)
    return json_compatible_data
```

處理非 JSON 相容的 Python 物件

請求主體：更新

```python
from fastapi import FastAPI
from pydantic import BaseModel

class Item(BaseModel):
    name: str | None = None
    price: float | None = None
    is_offer: bool | None = None

app = FastAPI()

@app.patch("/items/{item_id}")
def update_item(item_id: int, item: Item):
    return {"item_id": item_id, **item.dict(exclude_unset=True)}
```

使用 PATCH 方法進行部分更新

依賴注入簡介

```python
from fastapi import FastAPI, Depends

app = FastAPI()

def get_current_user(token: str):
    # 模擬驗證邏輯
    return {"username": "john", "token": token}

@app.get("/users/me")
def read_current_user(current_user: dict = Depends(get_current_user)):
    return current_user
```

使用 Depends() 注入依賴

安全性：第一步

```python
from fastapi import FastAPI, Depends, HTTPException, status
from fastapi.security import HTTPBasic, HTTPBasicCredentials

security = HTTPBasic()

app = FastAPI()

@app.get("/users/me")
def read_current_user(credentials: HTTPBasicCredentials = Depends(security)):
    return {"username": credentials.username}
```

基本 HTTP 認證

常見誤區與除錯

🚨 **五大常見錯誤：**

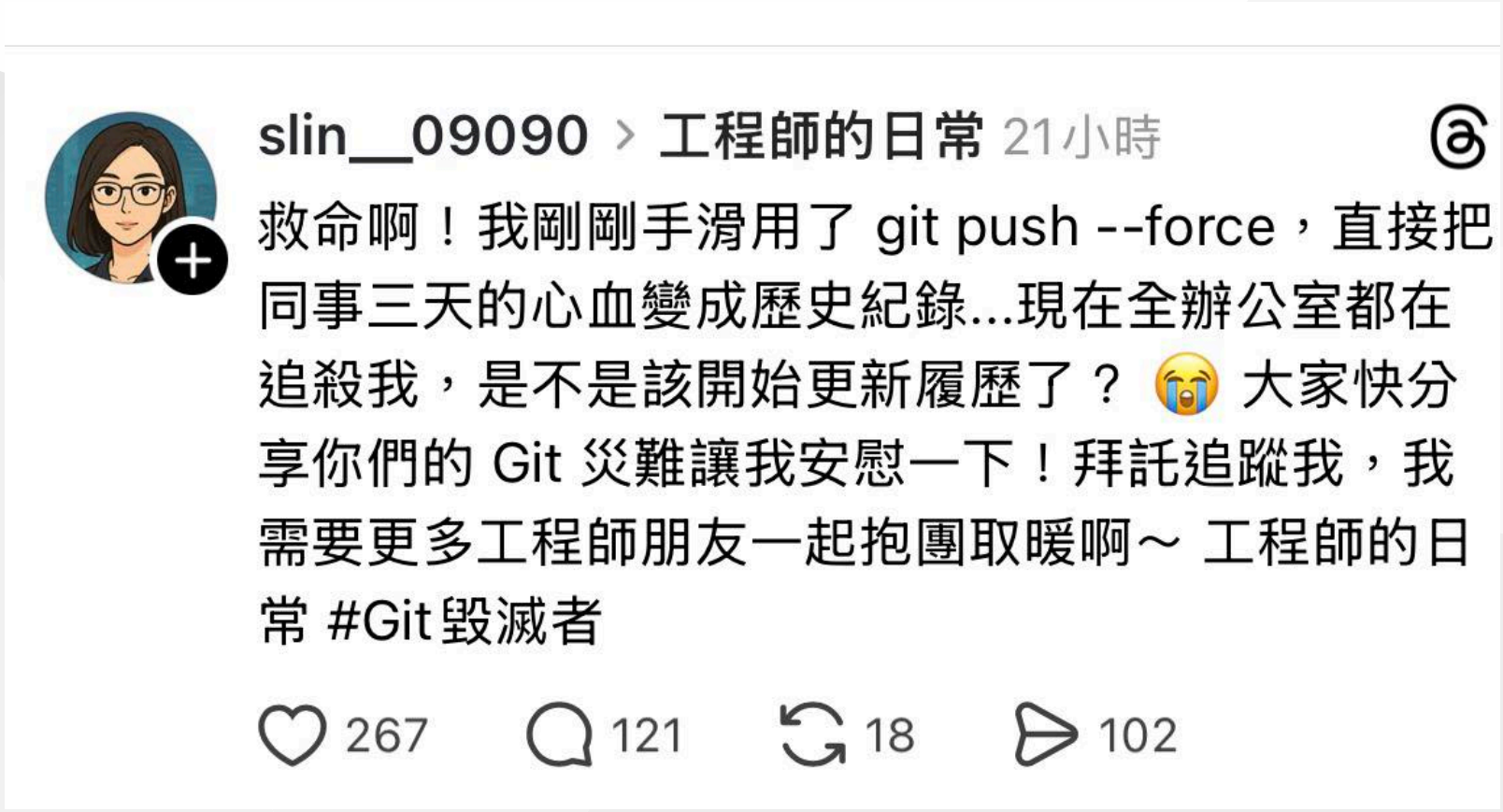
1. **JSON 格式錯誤** - 最後一行加逗號 → 422 錯誤
2. **參數類型混淆** - 路徑 vs 查詢 vs 請求主體
3. **忘記 return** - 函數必須回傳資料
4. **路徑順序錯誤** - `/users/me` 要在 `/users/{user_id}` 之前
5. **型別提示錯誤** - 使用 `str | None` 語法

🔧 **快速除錯：**

- 查看 `/docs` 自動文檔

⚠️ 命令列的危險地雷區

這些指令可能會摧毀你的電腦！



🚨 絕對不要執行的指令！

檔案系統毀滅者：

- `rm -rf /` 或 `rm -rf ~` - 刪除所有檔案
- `mkfs.* /dev/sdX` - 格式化硬碟
- `dd if=/dev/zero of=/dev/sdX` - 用垃圾資料覆蓋硬碟

❓ 為什麼這麼危險？

就像把房子所有門窗都打破，
風雨進來就什麼都沒了！
系統檔案被刪，電腦就開不了機。

安全使用命令列的守則

🔒 三思而後行：

- 執行前先想：「這個指令會刪除什麼？」
- 重要操作前先備份
- 在虛擬機或測試環境先試

🔍 先小範圍測試：

```
# 不要直接 rm -rf /
# 先試 ls 看看會列出什麼
ls /

# 不要直接 rm -rf ~/Documents
```

命令列學習心態

💡 **學習心態：**
命令列很強大，但也像雙刃劍。
學會尊重它，就不會傷到自己！

安全守則總結：

- 永遠先在測試環境試
- 複製貼上前先閱讀
- 重要操作先備份

FastAPI 快速參考表

新手必備指令：

```python
from fastapi import FastAPI
from pydantic import BaseModel

# 建立應用程式
app = FastAPI(title="My API", version="1.0.0")

# 基本路由
@app.get("/")                    # 首頁
def home():
    return {"message": "Hello World"}

@app.get("/items/{item_id}")     # 路徑參數
def get_item(item_id: int):
    return {"item_id": item_id}

@app.get("/items/")              # 查詢參數
def list_items(limit: int = 10):
    return {"limit": limit}

@app.post("/items/")             # 建立資料
def create_item(item: Item):     # Item 是 Pydantic 模型
    return item
```

請求主體參數

請求主體參數：

```
# 請求主體 - 傳送資料用
def func(item: Item):
```

🎯 VSCode Git 操作：

1. 初始化 Git 倉庫：

- 在 VSCode 中按 `Ctrl+Shift+G` 開啟原始碼控制面板
- 按 "Initialize Repository" 按鈕
- 或在終端機執行 `git init`

2. 提交變更：

- 在原始碼控制面板中看到變更檔案
- 輸入 commit 訊息
- 按 ✓ 按鈕提交

學習資源與下一步

學習路徑建議

🎯 第一階段：打好基礎 (1-2 週)

1. ✅ 學會基本路由 (GET, POST)
2. ✅ 了解路徑參數和查詢參數
3. ✅ 會用 Pydantic 定義資料格式
4. ✅ 熟悉 JSON 格式和請求測試

第二階段：進階功能 (2-4 週)

1. 🔄 學習資料庫整合 (SQLite/PostgreSQL)
2. 🔒 認識使用者認證和安全性
3. 📁 處理檔案上傳和下載
4. ⚡ 探索非同步處理

第三階段：實戰應用 (1-2 月)

1. 🏗️ 建置完整的 CRUD API
2. 🧪 撰寫測試程式碼
3. 🚀 部署到雲端服務
4. 👥 學習團隊協作

學習心態

💡 學習心態：

- **不要怕錯**：每個人都是從錯誤中學習
- **多實作**：看再多不如自己動手做一次
- **找問題**：卡關時 Google + ChatGPT 是好朋友
- **看程式碼**：GitHub 上有很多開源專案可以學

有用的學習資源

🔗 有用的學習資源：

- FastAPI 官方文檔 (英文較完整)
- Python 官方教學網站
- YouTube 程式教學影片
- 台灣 Python 社群論壇

最終建議

🚀 最終建議：

多做 side project！做一個屬於自己的小專案，
比如：個人部落格 API、Todo 應用、簡單的電商後端。

理論學一百遍，不如實作一個小專案！