Python 要求庫

要完全用 Python 的 `requests` 庫取代 Postman，你可以執行所有常見的 HTTP 操作，例如發送不同類型的請求、處理標頭、處理 JSON 資料、上傳文件、使用代理以及斷言回應。以下，我將逐步引導你完成每個功能的範例，確保你有一個完整且實用的指南。

第 1 步：安裝 Requests 庫

在開始之前，請確保 `requests` 庫已安裝。在你的終端中運行以下命令：

```
pip install requests
```

現在，讓我們深入了解範例。

發送 HTTP 請求

`requests` 庫支持所有 HTTP 方法，如 GET、POST、PUT、DELETE 等。以下是如何發送簡單的 GET 和 POST 請求：

GET 請求

```python
import requests

# 發送 GET 請求
response = requests.get('https://api.example.com/data')

# 打印狀態碼和回應主體
print(" 狀態碼:", response.status_code)
print(" 回應主體:", response.text)
```

POST 請求

```python
# 發送 POST 請求，沒有數據
response = requests.post('https://api.example.com/submit')

print(" 狀態碼:", response.status_code)
print(" 回應主體:", response.text)
```

添加標頭

標頭通常用於身份驗證、內容類型或自定義元數據。將它們作為字典傳遞給 `headers` 參數。

```
# 定義自定義標頭
headers = {
    'Authorization': 'Bearer my_token',
    'Content-Type': 'application/json',
    'User-Agent': 'MyApp/1.0'
}

# 發送帶有標頭的 GET 請求
response = requests.get('https://api.example.com/data', headers=headers)

print(" 狀態碼:", response.status_code)
print(" 回應標頭:", response.headers)
print(" 回應主體:", response.text)
```

發送 JSON 資料

要在 POST 請求中發送 JSON 資料（如在 Postman 的主體選項卡中選擇 JSON），使用 `json` 參數。這將自動將 `Content-Type` 設置為 `application/json`。

```
# 定義 JSON 資料
data = {
    'key1': 'value1',
    'key2': 'value2'
}

# 發送帶有 JSON 資料的 POST 請求
response = requests.post('https://api.example.com/submit', json=data, headers=headers)

print(" 狀態碼:", response.status_code)
print(" 回應 JSON:", response.json())
```

上傳文件

要上傳文件（類似於 Postman 的 form-data 選項），使用 `files` 參數。以二進制模式打開文件（`'rb'`）並可選地包含其他表單數據。

簡單文件上傳

```
# 準備文件上傳
files = {
    'file': open('myfile.txt', 'rb')
}

# 發送帶有文件的 POST 請求
response = requests.post('https://api.example.com/upload', files=files)

print(" 狀態碼:", response.status_code)
print(" 回應主體:", response.text)

# 手動關閉文件
files['file'].close()
```

帶有表單數據的文件上傳（推薦方法） 使用 `with` 語句可以確保文件自動關閉：

```
# 額外的表單數據
form_data = {
    'description': 'My file upload'
}

# 打開並上傳文件
with open('myfile.txt', 'rb') as f:
    files = {
        'file': f
    }
    response = requests.post('https://api.example.com/upload', data=form_data, files=files)

print(" 狀態碼:", response.status_code)
print(" 回應主體:", response.text)
```

使用代理

要通過代理路由請求（類似於 Postman 的代理設置），使用 `proxies` 參數和字典。

```
# 定義代理設置
proxies = {
    'http': 'http://myproxy:8080',
    'https': 'https://myproxy:8080'
}

# 通過代理發送請求
response = requests.get('https://api.example.com/data', proxies=proxies)

print(" 狀態碼:", response.status_code)
print(" 回應主體:", response.text)
```

處理和斷言回應

`requests` 庫提供了對回應詳細信息（如狀態碼、JSON 資料、標頭和 cookies）的易於訪問。你可以使用 Python 的 `assert` 語句來驗證回應，類似於 Postman 的測試腳本。

解析 JSON 回應

```
response = requests.get('https://api.example.com/data')

# 檢查狀態碼並解析 JSON
if response.status_code == 200:
    data = response.json()  # 將回應轉換為 Python dict/list
    print("JSON 資料:", data)
else:
    print(" 錯誤:", response.status_code)
```

斷言回應詳細信息

```
response = requests.get('https://api.example.com/data')

# 斷言狀態碼
assert response.status_code == 200, f" 預期 200，但得到了 {response.status_code}"
```

```
# 解析 JSON 並斷言內容
data = response.json()
assert 'key' in data, " 回應中未找到鍵"
assert data['key'] == 'expected_value', " 值不匹配"

# 檢查回應標頭
assert 'Content-Type' in response.headers, "Content-Type 標頭缺失"
assert response.headers['Content-Type'] == 'application/json', " 意外的 Content-Type"

# 檢查 cookies
cookies = response.cookies
assert 'session_id' in cookies, "Session ID cookie 缺失"

print(" 所有斷言通過!")
```

處理錯誤 將請求包裹在 try-except 塊中以捕捉網絡或 HTTP 錯誤：

```
try:
    response = requests.get('https://api.example.com/data')
    response.raise_for_status()  # 為 4xx/5xx 錯誤引發異常
    data = response.json()
    print(" 資料:", data)
except requests.exceptions.RequestException as e:
    print(" 請求失敗:", e)
```

綜合範例

以下是一個結合標頭、文件上傳、代理和回應斷言的完整範例：

```
import requests

# 定義標頭
headers = {
    'Authorization': 'Bearer my_token'
}

# 表單數據和文件
form_data = {
```

```
    'description': 'My file upload'
}

# 代理設置
proxies = {
    'http': 'http://myproxy:8080',
    'https': 'https://myproxy:8080'
}

# 發送帶有文件上傳的請求
try:
    with open('myfile.txt', 'rb') as f:
        files = {'file': f}
        response = requests.post(
            'https://api.example.com/upload',
            headers=headers,
            data=form_data,
            files=files,
            proxies=proxies
        )
        response.raise_for_status()  # 檢查 HTTP 錯誤

        # 解析並斷言回應
        data = response.json()
        assert 'upload_id' in data, " 回應中未找到 Upload ID"
        print(" 上傳成功，ID:", data['upload_id'])

except requests.exceptions.RequestException as e:
    print(" 請求失敗:", e)
```

可重用函數（取代 Postman 集合）

要模仿 Postman 的保存請求或集合，將邏輯封裝在函數中：

```
def upload_file(file_path, description, token, proxy=None):
    headers = {'Authorization': f'Bearer {token}'}
    data = {'description': description}
    proxies = {'http': proxy, 'https': proxy} if proxy else None
```

```
    try:
        with open(file_path, 'rb') as f:
            files = {'file': f}
            response = requests.post(
                'https://api.example.com/upload',
                headers=headers,
                data=data,
                files=files,
                proxies=proxies
            )
            response.raise_for_status()
            data = response.json()
            assert 'upload_id' in data, " 未找到 Upload ID"
            return data['upload_id']
    except requests.exceptions.RequestException as e:
        print(" 請求失敗:", e)
        return None

# 使用
upload_id = upload_file('myfile.txt', 'My file upload', 'my_token', 'http://myproxy:8080')
if upload_id:
    print(" 上傳成功，ID:", upload_id)
```

額外功能

- **會話**：使用 `requests.Session()` 來在請求之間保持 cookies 或標頭。
- **超時**：添加 `timeout=5`（秒）到請求以避免掛起。
- **重定向**：使用 `allow_redirects=True/False` 控制。

總結

使用 `requests` 庫，你可以：- 發送 HTTP 請求（`get`、`post` 等）- 包含標頭、JSON 資料和文件上傳 - 使用代理 - 解析和斷言回應（狀態、JSON、標頭、cookies）- 將代碼組織成可重用的函數

這給你提供了一個程式化、靈活的替代方案，適合腳本和自動化！