


TRƯỜNG: ĐH Công thương TP HCM KHOA: CÔNG NGHỆ THÔNG TIN BỘ MÔN: KHDL&TTNT MH: LẬP TRÌNH PYTHON	BUỔI 9. Request	
---	------------------------	---

A. MỤC TIÊU

- Làm việc với Thư viện Request

B. DỤNG CỤ - THIẾT BỊ THÍ NGHIỆM CHO MỘT SV

STT	Chủng loại – Quy cách vật tư	Số lượng	Đơn vị	Ghi chú
1	Computer	1	1	

C. NỘI DUNG THỰC HÀNH

1. Giới thiệu về thư viện Request

1.1. Tại sao cần sử dụng thư viện Request?

Thư viện Requests là một trong những công cụ mạnh mẽ và linh hoạt nhất trong Python để thực hiện các yêu cầu HTTP một cách dễ dàng và hiệu quả. Dưới đây là một số lý do tại sao thư viện Requests trở thành lựa chọn phổ biến cho việc xử lý HTTP requests trong Python:

- **Sử dụng đơn giản:** Thư viện Requests cung cấp một cách thức sử dụng rất dễ hiểu và dễ sử dụng. Với chỉ vài dòng mã, bạn có thể tạo và gửi các yêu cầu HTTP một cách nhanh chóng.
- **Tính linh hoạt:** Requests hỗ trợ tất cả các phương thức HTTP chính, bao gồm GET, POST, PUT, DELETE và nhiều phương thức khác. Bạn có thể thực hiện mọi loại yêu cầu mà bạn cần một cách dễ dàng.
- **Xử lý đa dạng dữ liệu:** Requests cho phép bạn truyền và nhận dữ liệu theo nhiều định dạng khác nhau như JSON, form data, hoặc cả dữ liệu nhị phân.
- **Hỗ trợ Cookie và Session:** Thư viện Requests giúp quản lý cookies và session một cách thuận tiện, cho phép bạn duy trì trạng thái của phiên làm việc với server.
- **Xử lý ngoại lệ dễ dàng:** Requests tự động xử lý nhiều tình huống ngoại lệ như redirects, timeout, và các lỗi HTTP một cách linh hoạt, giúp bạn tập trung vào logic của ứng dụng mà không phải lo lắng về các vấn đề kỹ thuật.

- Hỗ trợ đa luồng: Requests có khả năng thực hiện các yêu cầu HTTP đồng thời thông qua việc sử dụng luồng (threads) hoặc phiên bản bất đồng bộ (asynchronous) với `async/await`.

Như vậy, nhờ vào những tính năng và ưu điểm nổi bật của mình, thư viện Requests là một công cụ mạnh mẽ và không thể thiếu đối với việc xử lý HTTP requests trong Python.

1.2. Cài đặt thư viện Request

- Cài đặt bằng pip (chạy lệnh trong Anaconda Prompt)

```
pip install requests
```

- Kiểm tra phiên bản đã cài đặt

```
pip show requests
```

2. Cơ bản về HTTP Requests

2.1. HTTP là gì?

HTTP (Hypertext Transfer Protocol) là một giao thức dùng để truyền tải và nhận dữ liệu trên Internet.

Giao thức này được sử dụng để truyền tải các trang web, hình ảnh, video, âm nhạc, và nhiều loại dữ liệu khác qua mạng.

2.2. Phương thức HTTP

HTTP định nghĩa một tập hợp các phương thức (method) để xác định hành động cụ thể mà client (trình duyệt web, ứng dụng, vv.) muốn thực hiện trên server.

Các phương thức phổ biến bao gồm:

- GET: Truy vấn dữ liệu từ server.
- POST: Gửi dữ liệu mới lên server.
- PUT: Cập nhật hoặc thêm mới một tài nguyên trên server.
- DELETE: Xóa một tài nguyên trên server.

2.3. Cấu trúc của một HTTP Request

Một HTTP Request bao gồm ba phần chính: Request line, Headers, và Body.

** Request line:*

Request line bao gồm ba phần: phương thức (method), URL, và phiên bản của HTTP.

Ví dụ: GET /page HTTP/1.1.

** Headers:*

Headers chứa các thông tin bổ sung về yêu cầu như Content-Type, User-Agent, Accept, vv.

Mỗi header được phân tách bằng dấu xuống dòng (\r\n) và có dạng Key: Value.

Ví dụ:

Host: example.com

User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/96.0.4664.110 Safari/537.36

Accept:

text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9

** Body:*

Body chứa dữ liệu gửi đi từ client đến server (chỉ có khi cần).

Đối với phương thức GET, thông thường không có body, nhưng với các phương thức như POST, PUT, DELETE, body thường chứa dữ liệu được gửi đi.

Dữ liệu trong body thường được mã hóa theo định dạng được chỉ định trong header Content-Type.

Ví dụ một HTTP Request hoàn chỉnh:

GET /page HTTP/1.1

Host: example.com

User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/96.0.4664.110 Safari/537.36

Accept:

text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9

2.4. Cách Request và Response hoạt động

*** Quá trình gửi Request:**

- Khi một client muốn tương tác với một server thông qua HTTP, nó gửi đi một HTTP Request.
- Request bao gồm thông tin như phương thức (GET, POST, PUT, DELETE), URL, headers, và có thể chứa body nếu cần.

*** Quá trình xử lý Request trên Server:**

- Server nhận được Request từ client và xử lý theo nội dung của nó.
- Server kiểm tra phương thức, URL, và các headers trong Request để hiểu yêu cầu của client là gì và phản hồi tương ứng.

*** Quá trình tạo Response:**

- Sau khi xử lý yêu cầu từ client, server tạo ra một HTTP Response.
- Response bao gồm status code, headers, và có thể chứa body nếu cần.

*** Phản hồi từ Server đến Client:**

- Server gửi HTTP Response tới client, chứa thông tin về kết quả của yêu cầu từ client.
- Response bao gồm status code để chỉ ra thành công hay thất bại của yêu cầu, headers với thông tin bổ sung, và body chứa dữ liệu (nếu có).

*** Quá trình xử lý Response trên Client:**

- Client nhận được Response từ server và xử lý theo nội dung của nó.
- Dựa vào status code, client có thể biết được kết quả của yêu cầu (thành công, lỗi, vv.) và thực hiện các hành động tương ứng.

*** Kết thúc phiên tương tác:**

- Sau khi hoàn tất quá trình gửi Request và nhận Response, phiên tương tác giữa client và server có thể kết thúc hoặc tiếp tục với các yêu cầu và phản hồi khác.
- Quá trình này diễn ra liên tục và song song trên Internet, cho phép client và server tương tác với nhau thông qua giao thức HTTP.

2.5. Mã trạng thái (Status Code) trong HTTP

Mã trạng thái là một phần quan trọng của phản hồi HTTP, được sử dụng để chỉ ra kết quả của yêu cầu từ client đến server. Dưới đây là một số mã trạng thái phổ biến và ý nghĩa của chúng:

* 1xx: Informational (Thông tin)

- 100 Continue: Server đã nhận yêu cầu và client có thể tiếp tục gửi dữ liệu.

* 2xx: Success (Thành công)

- 200 OK: Yêu cầu thành công, phản hồi chứa dữ liệu được yêu cầu.
- 201 Created: Yêu cầu đã được tạo thành công, thường được sử dụng cho các phương thức POST để chỉ ra rằng tài nguyên đã được tạo mới.

* 3xx: Redirection (Chuyển hướng)

- 301 Moved Permanently: Tài nguyên đã được chuyển sang một địa chỉ mới vĩnh viễn.
- 302 Found: Tài nguyên đã được chuyển hướng tạm thời đến một địa chỉ mới.

* 4xx: Client Error (Lỗi từ client)

- 400 Bad Request: Yêu cầu không hợp lệ từ phía client.
- 404 Not Found: Tài nguyên được yêu cầu không tồn tại trên server.

* 5xx: Server Error (Lỗi từ server)

- 500 Internal Server Error: Lỗi xảy ra trên server khi xử lý yêu cầu từ client.
- 503 Service Unavailable: Server không thể xử lý yêu cầu do quá tải hoặc bảo trì.

Mã trạng thái là một phần quan trọng trong việc xác định kết quả của yêu cầu HTTP và thường được sử dụng để điều chỉnh luồng điều khiển trong ứng dụng web.

3. Sử dụng Request để gửi HTTP Requests

3.1. Gửi HTTP GET Request

Để gửi một HTTP GET Request bằng thư viện Requests trong Python, bạn có thể sử dụng hàm `requests.get()` và truyền URL của tài nguyên muốn truy cập. Dưới đây là một ví dụ minh họa:

```

import requests

# URL của tài nguyên bạn muốn truy cập
url = 'https://api.github.com'

# Gửi HTTP GET Request
response = requests.get(url)

# Kiểm tra mã trạng thái của Response
if response.status_code == 200:
    # Nếu mã trạng thái là 200 OK, in ra nội dung của
    Response
    print("Dữ liệu nhận được:")
    print(response.text)
else:
    # Nếu mã trạng thái không phải là 200, in ra mã trạng
    thái
    print("Lỗi! Mã trạng thái:", response.status_code)

```

3.2. Gửi HTTP POST Request

Để gửi một HTTP POST Request bằng thư viện Requests trong Python, bạn có thể sử dụng hàm `requests.post()` và truyền URL của endpoint mà bạn muốn gửi dữ liệu đến, cùng với dữ liệu bạn muốn gửi. Dưới đây là một ví dụ minh họa:

```

import requests

# URL của endpoint bạn muốn gửi dữ liệu đến
url = 'https://nghiait.wiremockapi.cloud/post'

# Dữ liệu bạn muốn gửi (ở định dạng dictionary)
data = {'key1': 'value1', 'key2': 'value2'}

# Gửi HTTP POST Request với dữ liệu
response = requests.post(url, data=data)

```

```
# Kiểm tra mã trạng thái của Response
if response.status_code == 200:
    # Nếu mã trạng thái là 200 OK, in ra nội dung của
    Response
    print("Dữ liệu nhận được:")
    print(response.text)
else:
    # Nếu mã trạng thái không phải là 200, in ra mã trạng
    thái
    print("Lỗi! Mã trạng thái:", response.status_code)
```

Lưu ý rằng ở đây ta gửi POST request đến url. URL này là một dịch vụ test POST request. Nếu ta truy cập trực tiếp trang web sẽ nhận được thông báo “Phương thức không được hỗ trợ”. Lý do: Truy cập trang web bình thường mặc định là GET request.

3.3. Truyền dữ liệu với Request

Để truyền dữ liệu với một HTTP Request trong thư viện Requests, có một số cách tiếp cận khác nhau tùy thuộc vào loại dữ liệu bạn muốn gửi và cách server đích xử lý dữ liệu đó. Dưới đây là các cách phổ biến để truyền dữ liệu:

a. Truyền dữ liệu qua URL parameters (query parameters):

Đây là cách đơn giản và phổ biến nhất để truyền dữ liệu trong HTTP GET Request. Bạn có thể thêm các tham số vào URL bằng cách sử dụng dấu ? sau URL, và sau đó là các cặp key-value cách nhau bởi dấu &. Ví dụ:

```
import requests

url = 'https://api.example.com/data'
params = {'key1': 'value1', 'key2': 'value2'}

response = requests.get(url, params=params)
```

b. Truyền dữ liệu qua body của Request

Đối với HTTP POST Request hoặc các phương thức khác mà yêu cầu gửi dữ liệu qua body, bạn có thể truyền dữ liệu bằng cách sử dụng tham số data hoặc json. Tham số data được sử dụng cho các dữ liệu không phải là JSON, trong khi json được sử dụng cho JSON data. Ví dụ:

```
import requests

url = 'https://api.example.com/post_data'
data = {'key1': 'value1', 'key2': 'value2'}

response = requests.post(url, data=data)

url = 'https://api.example.com/post_json'
json_data = {'key1': 'value1', 'key2': 'value2'}

response = requests.post(url, json=json_data)
```

c. Truyền dữ liệu qua Header

Một số yêu cầu đòi hỏi dữ liệu được truyền qua header của Request thay vì qua URL hoặc body. Bạn có thể thêm các thông tin vào header bằng cách sử dụng tham số headers. Ví dụ:

```
import requests

url = 'https://api.example.com/data'
headers = {'Authorization': 'Bearer my_access_token'}

response = requests.get(url, headers=headers)
```

3.4. Xử lý Response từ Server

Xử lý Response từ server là một phần quan trọng trong việc tương tác với các dịch vụ web và API. Dưới đây là một số phương pháp phổ biến để xử lý Response từ server sau khi bạn đã gửi một HTTP Request bằng thư viện Requests:

** Truy cập dữ liệu trong Response*

Bạn có thể truy cập dữ liệu trong Response thông qua thuộc tính text hoặc content.

response.text trả về nội dung của Response dưới dạng string, trong khi response.content trả về dữ liệu dưới dạng byte.

```
import requests

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

# Truy cập dữ liệu trong Response
data_text = response.text
data_content = response.content
```

** Xử lý dữ liệu JSON*

Nếu dữ liệu trả về từ server là JSON, bạn có thể sử dụng phương thức json() để chuyển đổi nó thành đối tượng Python (ví dụ: dictionary).

Điều này giúp bạn dễ dàng truy cập và xử lý dữ liệu trong Python.

```
import requests

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

# Chuyển đổi dữ liệu JSON thành dictionary
data_json = response.json()
```

** Kiểm tra mã trạng thái*

Mã trạng thái trong Response cho biết kết quả của yêu cầu. Bạn có thể kiểm tra mã trạng thái bằng thuộc tính status_code.

Nếu mã trạng thái là 200, yêu cầu đã thành công. Nếu không, bạn cần xử lý tình huống tương ứng.

* *Xử lý Headers*

Headers trong Response cung cấp thông tin về Response như loại nội dung, độ dài, vv.

Bạn có thể truy cập headers bằng thuộc tính headers.

```
import requests

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

# Truy cập headers của Response
headers = response.headers
```

4. Thao tác với Header và Cookies

4.1. *Thêm Header vào Request*

Để thêm header vào một HTTP Request bằng thư viện Requests trong Python, bạn có thể sử dụng tham số headers của hàm tương ứng như requests.get() hoặc requests.post(). Dưới đây là cách thêm header vào một HTTP Request:

```
import requests

# URL của endpoint bạn muốn gửi Request đến
url = 'https://api.example.com/data'

# Định nghĩa các headers bạn muốn thêm vào Request
headers = {
    'User-Agent': 'My User Agent',
    'Authorization': 'Bearer my_access_token',
    'Content-Type': 'application/json'
}

# Gửi HTTP GET Request với các headers đã xác định
response = requests.get(url, headers=headers)

# Xử lý Response như thường
```

4.2. Thao tác với Cookies

Để thao tác với cookies trong thư viện Requests của Python, bạn có thể sử dụng thuộc tính cookies của đối tượng Response hoặc truyền một đối tượng cookies vào các phương thức get(), post(), vv. Dưới đây là cách thao tác với cookies:

** Nhận cookies từ Response:*

```
import requests

url = 'https://example.com'

response = requests.get(url)

# Truy cập cookies từ Response
cookies = response.cookies
```

** Gửi cookies trong Request*

```
import requests

url = 'https://example.com'

# Định nghĩa cookies bạn muốn gửi
cookies = {'cookie_name': 'cookie_value'}

response = requests.get(url, cookies=cookies)
```

** Duy trì cookies giữa các Request*

Nếu bạn muốn duy trì cookies giữa các request (giống như một phiên làm việc trên web), bạn có thể sử dụng một đối tượng Session. Đối tượng Session sẽ tự động duy trì cookies và thông tin phiên làm việc giữa các request:

```
import requests
```

```
url = 'https://example.com'

# Tạo một đối tượng Session
session = requests.Session()

# Gửi Request bằng cách sử dụng Session
response = session.get(url)

# Request tiếp theo sẽ tự động gửi cookies đã nhận được từ
Response trước đó
response = session.get(url)
```

D. BÀI TẬP

Bài 1. Viết chương trình python thực hiện các tính năng sau

- Hiển thị danh sách sản phẩm (ID, Name) lấy từ API:
https://thongthai.work/simple_api/<MSSV>

- Chức năng truy vấn 1 sản phẩm: Cho phép người dùng nhập vào 1 ID sản phẩm, thực hiện truy vấn và hiển thị thông tin chi tiết 1 sản phẩm ra màn hình (ID, Name, Description, Price).
API để lấy thông tin 1 sản phẩm: https://thongthai.work/simple_api/<MSSV>?id=<id>

- Chức năng thêm sản phẩm: Cho phép người dùng nhập vào Name, Description và Price.
Thêm sản phẩm mới vào CSDL thông qua API.

Chú ý: Để thêm sản phẩm mới, thực hiện request POST đến API:
https://thongthai.work/simple_api/<MSSV>

Trong đó: json của request chứa thông tin dữ liệu cần thêm dưới dạng

```
{
    "name": "New Product",
    "description": "Description of the new product.",
    "price": 99
}
```

Dữ liệu trả về bao gồm thông tin thêm vào.

- Chức năng xóa: Cho phép người dùng nhập vào ID cần xóa. Thực hiện xóa sản phẩm với ID tương ứng.

Để xóa, thực hiện request DELETE đến API:

https://thongthai.work/simple_api/<MSSV>?id=<id>

- Chức năng sửa: Cho phép người dùng nhập ID cần sửa. Nếu ID hợp lệ, cho phép nhập thông tin mới như Name, Description, Price. Cập nhật lại thông tin sản phẩm tương ứng.

Để sửa thông tin sản phẩm, gửi request PUT đến API:

https://thongthai.work/simple_api/<MSSV>?id=<id>

Trong đó, dữ liệu kèm theo là thông tin sản phẩm sau khi cập nhật.

- Hiện thị menu để người dùng chọn chức năng

1. Hiện thị danh sách sản phẩm

2. Truy vấn 1 sản phẩm

3. Thêm sản phẩm mới

4. Xóa 1 sản phẩm

5. Sửa thông tin sản phẩm

0. Thoát chương trình

Vui lòng chọn chức năng:

Bài 2: Tạo và test API

- Vào trang <https://www.freemysqlhosting.net/> để tạo một CSDL mySQL miễn phí.

- Sau khi hoàn thành đăng ký, truy cập vào phpMyAdmin và tạo 1 CSDL và 1 table students (MSSV, Name, Gender, Birthday, City).

- Sử dụng câu lệnh SQL INSERT INTO để thêm khoảng 10 dòng vào table students.

- Truy cập trang <https://slashapi.com/> để tạo một tài khoản API miễn phí.

- Kết nối SlashAPI với CSDL mySql.

- Viết chương trình python thực hiện các chức năng tương tự như Bài 1. Tuy nhiên, ở đây thực hiện API của chính mình.

Bài 3: Dịch văn bản

Mô tả: Viết chương trình cho phép người dùng nhập một câu bằng tiếng Anh, sử dụng API dịch thuật để dịch sang tiếng Việt và hiển thị kết quả.

Gợi ý: Có thể dùng API như LibreTranslate hoặc Google Translate unofficial API (nếu tự tìm được).

Bài 4: Tìm thời tiết theo thành phố

Mô tả: Cho phép người dùng nhập tên thành phố hoặc hiển thị danh sách một số thành phố để lựa chọn, sử dụng API thời tiết để truy vấn và hiển thị nhiệt độ, mô tả thời tiết.

Gợi ý: Dùng OpenWeatherMap API

Bài 5: Tìm thông tin sách theo tên tác giả

Mô tả: Viết chương trình nhập tên tác giả từ bàn phím, truy vấn API và hiển thị danh sách sách kèm năm xuất bản.

Gợi ý: Dùng API từ Open Library.

Bài 6: Tra cứu quốc gia theo mã quốc gia

Mô tả: Nhập vào mã quốc gia (ví dụ: vn, us), dùng API để hiển thị tên nước, dân số, thủ đô, và khu vực.

Gợi ý: Dùng REST Countries API.

Bài 7: Hiển thị hình ảnh ngẫu nhiên từ API

Mô tả: Sử dụng API để lấy một hình ảnh ngẫu nhiên (ảnh chó, mèo, ảnh nghệ thuật...) và hiển thị đường dẫn.

Gợi ý: Cat API

Bài 8: Tìm phim theo từ khóa

Mô tả: Viết chương trình cho phép người dùng nhập từ khóa (tên phim), sử dụng API để hiển thị danh sách phim phù hợp với tiêu đề, năm phát hành, và mô tả ngắn.

Gợi ý: Dùng OMDb API

Bài 9: Tìm quote ngẫu nhiên

Mô tả: Gọi API để hiển thị một câu trích dẫn ngẫu nhiên (quote), bao gồm nội dung và tác giả.

Gợi ý: Dùng API như Quotable.

Bài 10: Truy vấn thông tin tiền mã hóa

Mô tả: Viết chương trình nhập vào tên đồng tiền (vd: bitcoin, ethereum), trả về giá hiện tại (USD) và phần trăm thay đổi trong 24h.

Gợi ý: Dùng API từ CoinGecko.