Python Requests Library

To fully replace Postman with the Python `requests` library, you can perform all common HTTP operations such as sending different types of requests, handling headers, working with JSON data, uploading files, using proxies, and asserting responses. Below, I'll walk you through examples for each of these functionalities step-by-step, ensuring you have a complete and practical guide.

Step 1: Install the Requests Library

Before you begin, ensure the `requests` library is installed. Run this command in your terminal:

```
pip install requests
```

Now, let's dive into the examples.

Sending HTTP Requests

The `requests` library supports all HTTP methods like GET, POST, PUT, DELETE, etc. Here's how to send a simple GET and POST request:

GET Request

```python
import requests

# Send a GET request
response = requests.get('https://api.example.com/data')

# Print the status code and response body
print("Status Code:", response.status_code)
print("Response Body:", response.text)
```

POST Request

```python
# Send a POST request with no data
response = requests.post('https://api.example.com/submit')

print("Status Code:", response.status_code)
print("Response Body:", response.text)
```

Adding Headers

Headers are often used for authentication, content types, or custom metadata. Pass them as a dictionary to the `headers` parameter.

```
# Define custom headers
headers = {
    'Authorization': 'Bearer my_token',
    'Content-Type': 'application/json',
    'User-Agent': 'MyApp/1.0'
}

# Send a GET request with headers
response = requests.get('https://api.example.com/data', headers=headers)

print("Status Code:", response.status_code)
print("Response Headers:", response.headers)
print("Response Body:", response.text)
```

Sending JSON Data

To send JSON data in a POST request (like selecting JSON in Postman's body tab), use the `json` parameter. This automatically sets the `Content-Type` to `application/json`.

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

# Send a POST request with JSON data
response = requests.post('https://api.example.com/submit', json=data, headers=headers)

print("Status Code:", response.status_code)
print("Response JSON:", response.json())
```

Uploading Files

To upload files (similar to Postman's form-data option), use the `files` parameter. Open files in binary mode (`'rb'`) and optionally include additional form data.

Simple File Upload

```
# Prepare file for upload
files = {
    'file': open('myfile.txt', 'rb')
}

# Send POST request with file
response = requests.post('https://api.example.com/upload', files=files)

print("Status Code:", response.status_code)
print("Response Body:", response.text)

# Close the file manually
files['file'].close()
```

File Upload with Form Data (Recommended Approach) Using a `with` statement ensures the file is closed automatically:

```
# Additional form data
form_data = {
    'description': 'My file upload'
}

# Open and upload file
with open('myfile.txt', 'rb') as f:
    files = {
        'file': f
    }
    response = requests.post('https://api.example.com/upload', data=form_data, files=files)

print("Status Code:", response.status_code)
print("Response Body:", response.text)
```

Using Proxies

To route requests through a proxy (similar to Postman's proxy settings), use the `proxies` parameter with a dictionary.

```
# Define proxy settings
proxies = {
    'http': 'http://myproxy:8080',
    'https': 'https://myproxy:8080'
}

# Send a request through a proxy
response = requests.get('https://api.example.com/data', proxies=proxies)

print("Status Code:", response.status_code)
print("Response Body:", response.text)
```

Handling and Asserting Responses

The `requests` library provides easy access to response details like status codes, JSON data, headers, and cookies. You can use Python's `assert` statements to validate responses, similar to Postman's test scripts.

Parsing JSON Responses

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

# Check status code and parse JSON
if response.status_code == 200:
    data = response.json()  # Converts response to Python dict/list
    print("JSON Data:", data)
else:
    print("Error:", response.status_code)
```

Asserting Response Details

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

# Assert status code
assert response.status_code == 200, f"Expected 200, got {response.status_code}"
```

```python
# Parse JSON and assert content
data = response.json()
assert 'key' in data, "Key not found in response"
assert data['key'] == 'expected_value', "Value does not match"

# Check response headers
assert 'Content-Type' in response.headers, "Content-Type header missing"
assert response.headers['Content-Type'] == 'application/json', "Unexpected Content-Type"

# Check cookies
cookies = response.cookies
assert 'session_id' in cookies, "Session ID cookie missing"

print("All assertions passed!")
```

Handling Errors Wrap requests in a `try-except` block to catch network or HTTP errors:

```python
try:
    response = requests.get('https://api.example.com/data')
    response.raise_for_status()  # Raises an exception for 4xx/5xx errors
    data = response.json()
    print("Data:", data)
except requests.exceptions.RequestException as e:
    print("Request failed:", e)
```

Comprehensive Example

Here's a full example combining headers, file upload, proxies, and response assertions:

```python
import requests

# Define headers
headers = {
    'Authorization': 'Bearer my_token'
}

# Form data and file
```

```
form_data = {
    'description': 'My file upload'
}

# Proxy settings
proxies = {
    'http': 'http://myproxy:8080',
    'https': 'https://myproxy:8080'
}

# Send request with file upload
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()  # Check for HTTP errors

        # Parse and assert response
        data = response.json()
        assert 'upload_id' in data, "Upload ID not found in response"
        print("Upload successful, ID:", data['upload_id'])

except requests.exceptions.RequestException as e:
    print("Request failed:", e)
```

Reusable Functions (Replacing Postman Collections)

To mimic Postman's saved requests or collections, encapsulate logic in functions:

```
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, "Upload ID not found"
            return data['upload_id']
    except requests.exceptions.RequestException as e:
        print("Request failed:", e)
        return None

# Usage
upload_id = upload_file('myfile.txt', 'My file upload', 'my_token', 'http://myproxy:8080')
if upload_id:
    print("Upload successful, ID:", upload_id)
```

Additional Features

- **Sessions**: Use `requests.Session()` to persist cookies or headers across requests.
- **Timeouts**: Add `timeout=5` (seconds) to requests to avoid hanging.
- **Redirects**: Control with `allow_redirects=True/False`.

Summary

With the `requests` library, you can: - Send HTTP requests (`get`, `post`, etc.) - Include headers, JSON data, and file uploads - Use proxies - Parse and assert responses (status, JSON, headers, cookies) - Organize code into reusable functions

This gives you a programmatic, flexible alternative to Postman, suitable for scripting and automation!