Python-Anfragen-Bibliothek

Um Postman vollständig durch die Python-Bibliothek `requests` zu ersetzen, können Sie alle gängigen HTTP-Operationen wie das Senden verschiedener Arten von Anfragen, das Bearbeiten von Headern, das Arbeiten mit JSON-Daten, das Hochladen von Dateien, das Verwenden von Proxys und das Überprüfen von Antworten durchführen. Im Folgenden werde ich Sie Schritt für Schritt durch Beispiele für jede dieser Funktionen führen, um sicherzustellen, dass Sie eine vollständige und praktische Anleitung haben.

Schritt 1: Installieren der Requests-Bibliothek

Stellen Sie sicher, dass die `requests`-Bibliothek installiert ist, bevor Sie beginnen. Führen Sie diesen Befehl in Ihrem Terminal aus:

```
pip install requests
```

Nun lassen Sie uns in die Beispiele eintauchen.

HTTP-Anfragen senden

Die `requests`-Bibliothek unterstützt alle HTTP-Methoden wie GET, POST, PUT, DELETE usw. Hier ist, wie Sie eine einfache GET- und POST-Anfrage senden:

GET-Anfrage

```python
import requests

# Senden einer GET-Anfrage
response = requests.get('https://api.example.com/data')

# Drucken des Status-Codes und des Antworttextes
print("Status Code:", response.status_code)
print("Response Body:", response.text)
```

POST-Anfrage

```python
# Senden einer POST-Anfrage ohne Daten
response = requests.post('https://api.example.com/submit')

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

Headers hinzufügen

Header werden oft für Authentifizierung, Inhaltsarten oder benutzerdefinierte Metadaten verwendet. Geben Sie sie als Dictionary an den `headers`-Parameter weiter.

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

# Senden einer GET-Anfrage mit Headern
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)
```

JSON-Daten senden

Um JSON-Daten in einer POST-Anfrage zu senden (ähnlich wie die Auswahl von JSON in Postmans Body-Tab), verwenden Sie den `json`-Parameter. Dies setzt automatisch den `Content-Type` auf `application/json`.

```
# JSON-Daten definieren
data = {
    'key1': 'value1',
    'key2': 'value2'
}

# Senden einer POST-Anfrage mit JSON-Daten
response = requests.post('https://api.example.com/submit', json=data, headers=headers)

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

Dateien hochladen

Um Dateien hochzuladen (ähnlich wie die Form-Daten-Option in Postman), verwenden Sie den `files`-Parameter. Öffnen Sie Dateien im Binärmodus (`'rb'`) und fügen Sie optional zusätzliche Formulardaten hinzu.

Einfaches Dateiupload

```
# Datei für den Upload vorbereiten
files = {
    'file': open('myfile.txt', 'rb')
}

# Senden einer POST-Anfrage mit Datei
response = requests.post('https://api.example.com/upload', files=files)

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

# Datei manuell schließen
files['file'].close()
```

Dateiupload mit Formulardaten (Empfohlener Ansatz) Verwenden Sie eine `with`-Anweisung, um sicherzustellen, dass die Datei automatisch geschlossen wird:

```
# Zusätzliche Formulardaten
form_data = {
    'description': 'Mein Dateiupload'
}

# Datei öffnen und hochladen
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)
```

Verwenden von Proxys

Um Anfragen über einen Proxy zu leiten (ähnlich wie die Proxy-Einstellungen in Postman), verwenden Sie den `proxies`-Parameter mit einem Dictionary.

```
# Proxy-Einstellungen definieren
proxies = {
    'http': 'http://myproxy:8080',
    'https': 'https://myproxy:8080'
}

# Senden einer Anfrage über einen Proxy
response = requests.get('https://api.example.com/data', proxies=proxies)

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

Antworten verarbeiten und überprüfen

Die `requests`-Bibliothek bietet einen einfachen Zugriff auf Antwortdetails wie Status-Codes, JSON-Daten, Header und Cookies. Sie können Python's `assert`-Anweisungen verwenden, um Antworten zu validieren, ähnlich wie Postmans Test-Skripte.

JSON-Antworten parsen

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

# Überprüfen des Status-Codes und Parsen von JSON
if response.status_code == 200:
    data = response.json()  # Konvertiert die Antwort in ein Python-Dictionary/Liste
    print("JSON Data:", data)
else:
    print("Error:", response.status_code)
```

Antwortdetails überprüfen

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

# Überprüfen des Status-Codes
```

```
assert response.status_code == 200, f"Expected 200, got {response.status_code}"

# JSON parsen und Inhalt überprüfen
data = response.json()
assert 'key' in data, "Key not found in response"
assert data['key'] == 'expected_value', "Value does not match"

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

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

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

Fehler behandeln Umfassen Sie Anfragen in einem `try-except`-Block, um Netzwerk- oder HTTP-Fehler abzufangen:

```
try:
    response = requests.get('https://api.example.com/data')
    response.raise_for_status()  # Wirft eine Ausnahme für 4xx/5xx-Fehler
    data = response.json()
    print("Data:", data)
except requests.exceptions.RequestException as e:
    print("Request failed:", e)
```

Umfassendes Beispiel

Hier ist ein vollständiges Beispiel, das Header, Dateiupload, Proxys und Antwortüberprüfungen kombiniert:

```
import requests

# Header definieren
headers = {
    'Authorization': 'Bearer my_token'
}
```

```
# Formulardaten und Datei
form_data = {
    'description': 'Mein Dateiupload'
}

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

# Anfrage mit Dateiupload senden
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()  # Überprüfen auf HTTP-Fehler

        # Antwort parsen und überprüfen
        data = response.json()
        assert 'upload_id' in data, "Upload ID not found in response"
        print("Upload erfolgreich, ID:", data['upload_id'])

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

Wiederverwendbare Funktionen (Ersetzen von Postman-Sammlungen)

Um Postmans gespeicherte Anfragen oder Sammlungen nachzuahmen, kapseln Sie die Logik in Funktionen:

```
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

# Verwendung
upload_id = upload_file('myfile.txt', 'Mein Dateiupload', 'my_token', 'http://myproxy:8080')
if upload_id:
    print("Upload erfolgreich, ID:", upload_id)
```

Zusätzliche Funktionen

- **Sitzungen**: Verwenden Sie `requests.Session()`, um Cookies oder Header über Anfragen hinweg zu speichern.
- **Timeouts**: Fügen Sie `timeout=5` (Sekunden) zu Anfragen hinzu, um das Hängenbleiben zu vermeiden.
- **Weiterleitungen**: Steuern Sie mit `allow_redirects=True/False`.

Zusammenfassung

Mit der `requests`-Bibliothek können Sie: - HTTP-Anfragen (`get`, `post`, etc.) senden - Header, JSON-Daten und Dateiuploads hinzufügen - Proxys verwenden - Antworten parsen und überprüfen (Status, JSON, Header,

Cookies) - Code in wiederverwendbare Funktionen organisieren

Dies bietet Ihnen eine programmatische, flexible Alternative zu Postman, die sich für Skripte und Automatisierung eignet!