Bibliothèque Requests Python

Pour remplacer complètement Postman par la bibliothèque Python `requests`, vous pouvez effectuer toutes les opérations HTTP courantes telles que l'envoi de différents types de requêtes, la gestion des en-têtes, le travail avec des données JSON, le téléchargement de fichiers, l'utilisation de proxys et l'affirmation des réponses. Ci-dessous, je vais vous guider étape par étape à travers des exemples pour chacune de ces fonctionnalités, en m'assurant que vous avez un guide complet et pratique.

Étape 1 : Installer la bibliothèque Requests

Avant de commencer, assurez-vous que la bibliothèque `requests` est installée. Exécutez cette commande dans votre terminal :

```
pip install requests
```

Maintenant, plongeons dans les exemples.

Envoi de requêtes HTTP

La bibliothèque `requests` prend en charge toutes les méthodes HTTP comme GET, POST, PUT, DELETE, etc. Voici comment envoyer une simple requête GET et POST :

Requête GET

```python
import requests

# Envoyer une requête GET
response = requests.get('https://api.example.com/data')

# Afficher le code de statut et le corps de la réponse
print("Code de statut :", response.status_code)
print("Corps de la réponse :", response.text)
```

Requête POST

```python
# Envoyer une requête POST sans données
response = requests.post('https://api.example.com/submit')

print("Code de statut :", response.status_code)
print("Corps de la réponse :", response.text)
```

Ajout d'en-têtes

Les en-têtes sont souvent utilisés pour l'authentification, les types de contenu ou les métadonnées personnalisées. Passez-les sous forme de dictionnaire au paramètre `headers`.

```
# Définir des en-têtes personnalisés
headers = {
    'Authorization': 'Bearer my_token',
    'Content-Type': 'application/json',
    'User-Agent': 'MyApp/1.0'
}

# Envoyer une requête GET avec des en-têtes
response = requests.get('https://api.example.com/data', headers=headers)

print("Code de statut :", response.status_code)
print("En-têtes de la réponse :", response.headers)
print("Corps de la réponse :", response.text)
```

Envoi de données JSON

Pour envoyer des données JSON dans une requête POST (comme sélectionner JSON dans l'onglet corps de Postman), utilisez le paramètre `json`. Cela définit automatiquement le `Content-Type` à `application/json`.

```
# Définir des données JSON
data = {
    'key1': 'value1',
    'key2': 'value2'
}

# Envoyer une requête POST avec des données JSON
response = requests.post('https://api.example.com/submit', json=data, headers=headers)

print("Code de statut :", response.status_code)
print("JSON de la réponse :", response.json())
```

Téléchargement de fichiers

Pour télécharger des fichiers (similaire à l'option form-data de Postman), utilisez le paramètre `files`. Ouvrez les fichiers en mode binaire (`'rb'`) et incluez éventuellement des données de formulaire supplémentaires.

Téléchargement de fichier simple

```
# Préparer le fichier pour le téléchargement
files = {
    'file': open('myfile.txt', 'rb')
}

# Envoyer une requête POST avec un fichier
response = requests.post('https://api.example.com/upload', files=files)

print("Code de statut :", response.status_code)
print("Corps de la réponse :", response.text)

# Fermer le fichier manuellement
files['file'].close()
```

Téléchargement de fichier avec données de formulaire (Approche recommandée) L'utilisation d'une instruction `with` garantit que le fichier est fermé automatiquement :

```
# Données de formulaire supplémentaires
form_data = {
    'description': 'Mon téléchargement de fichier'
}

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

print("Code de statut :", response.status_code)
print("Corps de la réponse :", response.text)
```

Utilisation de proxys

Pour acheminer les requêtes via un proxy (similaire aux paramètres proxy de Postman), utilisez le paramètre `proxies` avec un dictionnaire.

```
# Définir les paramètres proxy
proxies = {
    'http': 'http://myproxy:8080',
    'https': 'https://myproxy:8080'
}

# Envoyer une requête via un proxy
response = requests.get('https://api.example.com/data', proxies=proxies)

print("Code de statut :", response.status_code)
print("Corps de la réponse :", response.text)
```

Gestion et affirmation des réponses

La bibliothèque `requests` fournit un accès facile aux détails de la réponse comme les codes de statut, les données JSON, les en-têtes et les cookies. Vous pouvez utiliser les instructions `assert` de Python pour valider les réponses, de manière similaire aux scripts de test de Postman.

Analyse des réponses JSON

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

# Vérifier le code de statut et analyser le JSON
if response.status_code == 200:
    data = response.json()  # Convertit la réponse en dictionnaire/list Python
    print("Données JSON :", data)
else:
    print("Erreur :", response.status_code)
```

Affirmation des détails de la réponse

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

# Affirmer le code de statut
```

```
assert response.status_code == 200, f"200 attendu, {response.status_code} obtenu"

# Analyser le JSON et affirmer le contenu
data = response.json()
assert 'key' in data, "Clé non trouvée dans la réponse"
assert data['key'] == 'expected_value', "La valeur ne correspond pas"

# Vérifier les en-têtes de réponse
assert 'Content-Type' in response.headers, "En-tête Content-Type manquant"
assert response.headers['Content-Type'] == 'application/json', "Content-Type inattendu"

# Vérifier les cookies
cookies = response.cookies
assert 'session_id' in cookies, "Cookie session_id manquant"

print("Toutes les affirmations réussies !")
```

Gestion des erreurs Enveloppez les requêtes dans un bloc `try-except` pour attraper les erreurs réseau ou HTTP :

```
try:
    response = requests.get('https://api.example.com/data')
    response.raise_for_status()  # Lève une exception pour les erreurs 4xx/5xx
    data = response.json()
    print("Données :", data)
except requests.exceptions.RequestException as e:
    print("Requête échouée :", e)
```

Exemple complet

Voici un exemple complet combinant des en-têtes, un téléchargement de fichier, des proxys et des affirmations de réponse :

```
import requests

# Définir les en-têtes
headers = {
    'Authorization': 'Bearer my_token'
```

```
}

# Données de formulaire et fichier
form_data = {
    'description': 'Mon téléchargement de fichier'
}

# Paramètres proxy
proxies = {
    'http': 'http://myproxy:8080',
    'https': 'https://myproxy:8080'
}

# Envoyer une requête avec un téléchargement de fichier
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()  # Vérifier les erreurs HTTP

        # Analyser et affirmer la réponse
        data = response.json()
        assert 'upload_id' in data, "ID de téléchargement non trouvé dans la réponse"
        print("Téléchargement réussi, ID :", data['upload_id'])

except requests.exceptions.RequestException as e:
    print("Requête échouée :", e)
```

Fonctions réutilisables (Remplacement des collections Postman)

Pour imiter les requêtes enregistrées ou les collections de Postman, encapsulez la logique dans des fonctions :

```python
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, "ID de téléchargement non trouvé"
            return data['upload_id']
    except requests.exceptions.RequestException as e:
        print("Requête échouée :", e)
        return None

# Utilisation
upload_id = upload_file('myfile.txt', 'Mon téléchargement de fichier', 'my_token', 'http://myproxy:8080')
if upload_id:
    print("Téléchargement réussi, ID :", upload_id)
```

Fonctionnalités supplémentaires

- **Sessions** : Utilisez `requests.Session()` pour persister les cookies ou les en-têtes à travers les requêtes.
- **Délais d'attente** : Ajoutez `timeout=5` (secondes) aux requêtes pour éviter les blocages.
- **Redirections** : Contrôlez avec `allow_redirects=True/False`.

Résumé

Avec la bibliothèque `requests`, vous pouvez : - Envoyer des requêtes HTTP (`get`, `post`, etc.) - Inclure des en-têtes, des données JSON et des téléchargements de fichiers - Utiliser des proxys - Analyser et affirmer les réponses (statut, JSON, en-têtes, cookies) - Organiser le code en fonctions réutilisables

Cela vous donne une alternative programmatique et flexible à Postman, adaptée à la scriptisation et à l’automatisation !