Biblioteca de solicitudes de Python

Para reemplazar completamente Postman con la biblioteca `requests` de Python, puedes realizar todas las operaciones HTTP comunes, como enviar diferentes tipos de solicitudes, manejar encabezados, trabajar con datos JSON, subir archivos, usar proxies y afirmar respuestas. A continuación, te guiaré paso a paso a través de ejemplos para cada una de estas funcionalidades, asegurándome de que tengas una guía completa y práctica.

Paso 1: Instalar la Biblioteca Requests

Antes de comenzar, asegúrate de que la biblioteca `requests` esté instalada. Ejecuta este comando en tu terminal:

```
pip install requests
```

Ahora, sumerjámonos en los ejemplos.

Enviar Solicitudes HTTP

La biblioteca `requests` admite todos los métodos HTTP como GET, POST, PUT, DELETE, etc. Aquí te muestro cómo enviar una solicitud GET y POST sencilla:

Solicitud GET

```
import requests

# Enviar una solicitud GET
response = requests.get('https://api.example.com/data')

# Imprimir el código de estado y el cuerpo de la respuesta
print("Código de Estado:", response.status_code)
print("Cuerpo de la Respuesta:", response.text)
```

Solicitud POST

```
# Enviar una solicitud POST sin datos
response = requests.post('https://api.example.com/submit')

print("Código de Estado:", response.status_code)
print("Cuerpo de la Respuesta:", response.text)
```

Agregar Encabezados

Los encabezados se utilizan a menudo para la autenticación, tipos de contenido o metadatos personalizados. Pásalos como un diccionario al parámetro `headers`.

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

# Enviar una solicitud GET con encabezados
response = requests.get('https://api.example.com/data', headers=headers)

print("Código de Estado:", response.status_code)
print("Encabezados de la Respuesta:", response.headers)
print("Cuerpo de la Respuesta:", response.text)
```

Enviar Datos JSON

Para enviar datos JSON en una solicitud POST (como seleccionar JSON en la pestaña de cuerpo de Postman), usa el parámetro `json`. Esto establece automáticamente el `Content-Type` a `application/json`.

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

# Enviar una solicitud POST con datos JSON
response = requests.post('https://api.example.com/submit', json=data, headers=headers)

print("Código de Estado:", response.status_code)
print("JSON de la Respuesta:", response.json())
```

Subir Archivos

Para subir archivos (similar a la opción de form-data de Postman), usa el parámetro `files`. Abre los archivos en modo binario (`'rb'`) y opcionalmente incluye datos de formulario adicionales.

Subida de Archivo Sencilla

```
# Preparar archivo para subir
files = {
    'file': open('myfile.txt', 'rb')
}

# Enviar solicitud POST con archivo
response = requests.post('https://api.example.com/upload', files=files)

print("Código de Estado:", response.status_code)
print("Cuerpo de la Respuesta:", response.text)

# Cerrar el archivo manualmente
files['file'].close()
```

Subida de Archivo con Datos de Formulario (Enfoque Recomendado) Usar una declaración `with` asegura que el archivo se cierre automáticamente:

```
# Datos de formulario adicionales
form_data = {
    'description': 'Mi subida de archivo'
}

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

print("Código de Estado:", response.status_code)
print("Cuerpo de la Respuesta:", response.text)
```

Usar Proxies

Para enrutar solicitudes a través de un proxy (similar a la configuración de proxy de Postman), usa el parámetro `proxies` con un diccionario.

```
# Definir configuración de proxy
proxies = {
    'http': 'http://myproxy:8080',
    'https': 'https://myproxy:8080'
}

# Enviar una solicitud a través de un proxy
response = requests.get('https://api.example.com/data', proxies=proxies)

print("Código de Estado:", response.status_code)
print("Cuerpo de la Respuesta:", response.text)
```

Manejar y Afirmar Respuestas

La biblioteca `requests` proporciona acceso fácil a detalles de la respuesta como códigos de estado, datos JSON, encabezados y cookies. Puedes usar declaraciones `assert` de Python para validar respuestas, similar a los scripts de prueba de Postman.

Analizar Respuestas JSON

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

# Verificar código de estado y analizar JSON
if response.status_code == 200:
    data = response.json()  # Convierte la respuesta a diccionario/lista de Python
    print("Datos JSON:", data)
else:
    print("Error:", response.status_code)
```

Afirmar Detalles de la Respuesta

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

# Afirmar código de estado
```

```
assert response.status_code == 200, f"Se esperaba 200, se obtuvo {response.status_code}"

# Analizar JSON y afirmar contenido
data = response.json()
assert 'key' in data, "Clave no encontrada en la respuesta"
assert data['key'] == 'valor_esperado', "El valor no coincide"

# Verificar encabezados de la respuesta
assert 'Content-Type' in response.headers, "Encabezado Content-Type faltante"
assert response.headers['Content-Type'] == 'application/json', "Content-Type inesperado"

# Verificar cookies
cookies = response.cookies
assert 'session_id' in cookies, "Cookie de session_id faltante"

print("Todas las afirmaciones pasaron!")
```

Manejar Errores Envuelve las solicitudes en un bloque `try-except` para capturar errores de red o HTTP:

```
try:
    response = requests.get('https://api.example.com/data')
    response.raise_for_status()  # Levanta una excepción para errores 4xx/5xx
    data = response.json()
    print("Datos:", data)
except requests.exceptions.RequestException as e:
    print("Solicitud fallida:", e)
```

Ejemplo Completo

Aquí tienes un ejemplo completo que combina encabezados, subida de archivos, proxies y afirmaciones de respuesta:

```
import requests

# Definir encabezados
headers = {
    'Authorization': 'Bearer my_token'
}
```

```python
# Datos de formulario y archivo
form_data = {
    'description': 'Mi subida de archivo'
}

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

# Enviar solicitud con subida de archivo
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()  # Verificar errores HTTP

        # Analizar y afirmar respuesta
        data = response.json()
        assert 'upload_id' in data, "ID de subida no encontrado en la respuesta"
        print("Subida exitosa, ID:", data['upload_id'])

except requests.exceptions.RequestException as e:
    print("Solicitud fallida:", e)
```

Funciones Reutilizables (Reemplazando Colecciones de Postman)

Para imitar las solicitudes guardadas o colecciones de Postman, encapsula la lógica en funciones:

```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 subida no encontrado"
            return data['upload_id']
    except requests.exceptions.RequestException as e:
        print("Solicitud fallida:", e)
        return None

# Uso
upload_id = upload_file('myfile.txt', 'Mi subida de archivo', 'my_token', 'http://myproxy:8080')
if upload_id:
    print("Subida exitosa, ID:", upload_id)
```

Características Adicionales

- **Sesiones**: Usa `requests.Session()` para persistir cookies o encabezados a través de solicitudes.
- **Tiempos de Espera**: Agrega `timeout=5` (segundos) a las solicitudes para evitar que se bloqueen.
- **Redirecciones**: Controla con `allow_redirects=True/False`.

Resumen

Con la biblioteca `requests`, puedes: - Enviar solicitudes HTTP (`get`, `post`, etc.) - Incluir encabezados, datos JSON y subidas de archivos - Usar proxies - Analizar y afirmar respuestas (estado, JSON, encabezados, cookies) - Organizar el código en funciones reutilizables

¡Esto te da una alternativa programática y flexible a Postman, adecuada para scripting y automatización!