Python requests. La librería para hacer peticiones http en Python

es una librería de Python que facilita el trabajo con solicitudes HTTP. Permite enviar y recibir datos de forma sencilla y eficiente, sin necesidad de manejar manualmente los detalles del protocolo HTTP. Con Requests, se puede acceder a servicios web, extraer información de páginas web, enviar formularios, archivos o JSON, y mucho mas.

Algunas de las características y ventajas de Requests son:

- Es compatible con todos los métodos HTTP: GET, POST, PUT, PATCH, DELETE, HEAD, OPTIONS
- Soporta cabeceras personalizadas, parámetros de consulta, cookies, autenticación y proxies
- Maneja automáticamente la codificación y decodificación de los contenidos, así como la compresión y descompresión
- Tiene una API simple y consistente, basada en el objeto Response
- Es segura y confiable, ya que usa el protocolo HTTPS y verifica los certificados SSL
- Es rápida y eficiente, ya que usa conexiones persistentes y un pool de

Instalación

Al tratarse de una librería de terceros, lo primero que debes hacer es instalarla. Lo más fácil para ello, es utilizar el gestor de paquetes pip:

$> pip install requests

Cómo hacer una petición GET en Python con requests

Una de las operaciones más habituales con la librería requests es hacer una petición GET, ya sea para obtener el contenido de una web o para realizar una petición a un API.

Para ello, simplemente tienes que invocar a la función get() indicando la URL a la que hacer la petición.

```
import requests
resp = requests.get('https://www.google.com/')
```

.
La función devuelve un objeto Response, que en este caso se ha asignado a la variable resp, con toda la información de la respuesta.

Cómo hacer una petición POST en Python con requests
Imagina ahora que quieres hacer una petición POST para enviar los datos de un formulario. En este caso, la manera de proceder es muy similar a la anterior, solo que debes llamar a la función post() e indicar en el parámetro data un *diccionario* con los datos del cuerpo de la petición. Al pasar los datos en el parámetro data, requests se encarga de codificarlos correctamente antes de realizar la petición:

```
import requests
auth_data = {'email': 'juanjo@j2logo.com', 'pass': '1234'}
resp = requests.post('https://mipagina.xyz/login/', data=auth_data)
```

Para el caso en que un formulario tenga uno o más campos multivaluados, se pueden especificar los diferentes valores de dos maneras distintas.

En un diccionario, indicando una lista de valores para una clave:

```
import requests

form_data = {'color': ['blanco', 'verde'], 'idioma': 'es'}
resp = requests.post('http://mipagina.xyz/formulario/', data=form_data)
```

O como una lista de tuplas:

```
import requests

form_data = [('color', 'blanco'), ('color', 'verde'), ('idioma', 'es')]
resp = requests.post('http://mipagina.xyz/formulario/', data=form_data)
```

Timeouts
Para cualquier petición, es posible especificar un *timeout* de espera de respuesta. Para ello, debes indicar en el parámetro timeout los segundos que la petición debe esperar, como mucho, antes de recibir el primer byte.

```
import requests
resp = requests.get('http://mipagina.xyz/', timeout=0.01)
```

IMPORTANTE: Si no se especifica un timeout, requests esperará indefinidamente a que se obtenga una respuesta.

Si el servidor no devuelve una respuesta antes del tiempo indicado, se lanzará la excepción requests.exceptions.Timeout.

Excepciones de requests
En caso de que se produzca algún error al realizar la petición, requests lanzará una excepción. La clase base de todas las excepciones es requests.exceptions.RequestException. No obstante, las excepciones más comunes son las siguientes:

- Timeout: Si el servidor no devuelve una respuesta antes del tiempo indicado en el parámetro timeout.

- TooManyRedirects: Si una petición excede el número de redirecciones máximo.
- ConnectionError: Si existe algún problema de red (no hay internet, fallo de DNS, conexión rechazada, …).

Por defecto, requests sigue las redirecciones

Por defecto, al realizar una petición con requests, esta sigue las redirecciones que vaya indicando el servidor antes de devolver la respuesta definitiva (excepto para HEAD, que hay que indicarlo explícitamente).

Si esto ocurre, el objeto con la respuesta guarda en el atributo history una lista con todas las respuestas desde la más antigua a la más reciente.

Por ejemplo, si intentas hacer una petición a http://google.com, obtendrás lo siguiente como respuesta:

```
import requests
r = requests.get('http://google.com')
r.history
[<Response [301]>]
r.status_code
200
r.url
'http://www.google.com/'
```

Como ves, la primera petición te rediríge a http://www.google.com/.

Para modificar este comportamiento, debes establecer el parámetro allow_redirects con valor False.

El objeto Response. Respuesta de la petición

Una vez que hemos repasado los aspectos principales para realizar una petición HTTP, en esta sección nos vamos a centrar en el objeto Response, que se obtiene como resultado de una petición.

Este objeto contiene toda la información referente a la respuesta, como el contenido, el código de respuesta, las cabeceras o las cookies.

Contenido de la respuesta

Cuando la respuesta que devuelve un servidor es de tipo texto, por ejemplo *html* o *xml*, el contenido se encuentra en el atributo text del objeto Response.

requests automáticamente decodifica el contenido devuelto por el servidor, adivinando la codificación a emplear a partir de las cabeceras de la respuesta. Para conocer la codificación empleada puedes acceder al atributo encoding.

```
>>> import requests
>>> r = requests.get('http://www.google.com/')
>>> r.encoding
'ISO-8859-1'
>>> r.text
'<!doctype html><html itemscope="" itemtype="h...'
```

Para aquellos casos en los que la respuesta no es texto, como por ejemplo una imagen o un pdf, entonces se debe acceder al atributo content, ya que este devuelve el contenido como una secuencia de bytes.

Por último, hay un caso especial que permite acceder al socket que devuelve la respuesta del servidor. Es a través del atributo raw. Sin embargo, en lugar de acceder al atributo raw directamente, es preferible llamar a la función iter_content usando el siguiente patrón, especialmente cuando se quiere hacer streaming en crudo de la descarga:

```
for chunk in r.iter_content(chunk_size=128):
# código que maneja la secuencia de bytes descargada
```

Código de estado de la respuesta

Para obtener el código de estado de la respuesta, debes acceder al atributo status_code de la misma.

```
>>> import requests
>>> r = requests.get('http://www.google.com/')
>>> r.status_code
200
```

Cabeceras de la respuesta

Las cabeceras de la respuesta están accesibles a través del atributo headers. Este atributo es un diccionario especial que contiene cada una de las cabeceras devueltas como claves del diccionario.

```
>>> import requests
>>> r = requests.get('http://www.google.com/')
>>> r.headers
{'Expires': '-1', 'Cache-Control': 'private, max-age=0', ...}
```

Cookies de la respuesta

Si quieres consultar las cookies devueltas por el servidor, lo puedes hacer accediendo al atributo cookies de la respuesta. Este atributo es de tipo RequestsCookieJar, que actúa como un diccionario con mejoras, para indicar el dominio y/o el path de una cookie, entre otras cosas:

```
>>> import requests
>>> r = requests.get('http://www.google.com/')
>>> r.cookies
<RequestsCookieJar[Cookie(version=0, name='1P_JAR', value='2020-05-18-12',
port=None, port_specified=False, domain='.google.com', domain_specified=True,
domain_initial_dot=True, path='/', path_specified=True, secure=True,
expires=1592397807, discard=False, comment=None, comment_url=None, rest={},
rfc2109=False), ...]>
>>> r.cookies['1P_JAR']
'2020-05-18-12'
```

Cómo pasar parámetros en la URL en Python con requests

En ocasiones es necesario pasar una serie de parámetros en la URL de la petición. Esto se puede hacer de forma manual añadiendo a la URL una cadena que

comienza por el carácter ? seguido de pares de la forma parm1=valor1&param2=valor2....

Por ejemplo:

```
import requests
r = requests.get('http://unblog.xyz?page=2')
```

No obstante, requests facilita la construcción de una URL con parámetros pasando un *diccionario* en el parámetro params.

```
import requests
parametros = {'clave1': 'valor1', 'clave2': ['val1', 'val2']}
r = requests.get('http://unblog.xyz', params=parametros)
```

Cómo enviar cabeceras con requests

Si necesitas especificar una cabecera en la petición, debes pasar un diccionario de pares clave: valor en el parámetro headers. El valor de cada uno de los elementos del diccionario debe ser un string. La clave se corresponde con el nombre de la cabecera.

Ejemplo:

```
import requests
cabeceras = {'cache-control': 'no-cache', 'accept': 'text/html'}
r = requests.get('http://unblog.xyz', headers=cabeceras)
```

Peticiones y respuestas JSON en Python con requests

Uno de los usos más habituales de la librería requests es hacer peticiones a un API desde una aplicación.

Una de las principales características de consumir un API es que, generalmente, los datos se envían y se obtienen en formato JSON.

A continuación, te mostraré lo fácil que es hacer llamadas a un API utilizando la librería requests.

Python GET requests

Para hacer una petición GET, simplemente hay que llamar a la función get().

Si la respuesta es un JSON, que es lo más común, podemos llamar al método json() de la respuesta para que decodifique los datos y los devuelva como un diccionario con los campos de dicho JSON.

```
import requests
r = requests.get('https://miapi.com/posts/')
posts = r.json()
```

IMPORTANTE: Comprueba el código de la respuesta para verificar si la respuesta es válida o se ha producido un fallo. En muchas ocasiones, un servidor puede devolver un JSON aun cuando se produzca un fallo.

Python POST requests
Para enviar datos en formato JSON a un API empleando los métodos POST, PUT o PATCH, simplemente pasa un diccionario a través del parámetro json. requests ya se encarga de especificar la cabecera Content-Type por ti y de serializar los datos de forma correcta.

```
import requests
payload = {'comentario': 'Está genial', 'estrellas': 5}
r = requests.post('https://miapi.com/comentarios/', json=payload)
```

Cómo enviar un fichero Multipart-Encoded
Para enviar un fichero con requests, tan solo debes cargar su contenido en un diccionario y pasar este a través del parámetro files:

```
import requests
ficheros = {'file1': open('nombre_fichero.pdf', 'rb')}
r = requests.post('https://mipagina.xyz/form/', files=ficheros)
```

También es posible especificar explícitamente el nombre del fichero y el tipo del siguiente modo:

```
import requests
ficheros = {'file1': ('nombre_fichero.pdf', open('nombre_fichero.pdf', 'rb'),
'application/pdf')}
r = requests.post('https://mipagina.xyz/form/', files=ficheros)
```

Cookies con requests
Por último, vamos a ver cómo enviar una *cookie* al servidor generada por nuestra aplicación.

Como te he mencionado anteriormente, requests maneja las cookies a través de un objeto de tipo RequestsCookieJar que es una especie de diccionario con una interfaz para especificar, entre otras cosas, el dominio y/o el path de una cookie.

Por tanto, para enviar una cookie a través de requests, lo podemos hacer de la siguiente manera:

```
import requests
jar = requests.cookies.RequestsCookieJar()
jar.set('nombre_cookie_1', 'valor_1', domain='mipagina.xyz', path='/')
jar.set('nombre_cookie_2', 'valor_2', domain='mipagina.xyz', path='/admin')
r = requests.get('http://mipagina.xyz', cookies=jar)
```

Conclusión
Bueno, ha sido un tutorial intenso pero muy productivo. En él hemos repasado los aspectos principales de cómo usar la librería requests para hacer peticiones *HTTP*, ya sea a una página web o para consumir un API.

REFERENCIAS:

Python requests. La librería para hacer peticiones http en Python (j2logo.com)

Solicitudes HTTP en Python con Requests - ▷ Cursos de Programación de 0 a Experto © Garantizados (unipython.com)