

About Requests Module

Thursday, October 31, 2024 2:32 PM

- The Requests module is a powerful and user-friendly HTTP library for Python, designed to make it easier to send HTTP requests.
- It abstracts away the complexities of making requests and handling responses, allowing developers to focus on building applications.



Requests
http for humans

Star 52,143

Requests is an elegant and simple HTTP library for Python, built for human beings.

Requests: HTTP for Humans™

Release v2.32.3. ([Installation](#))

downloads/month 564M license Apache-2.0 wheel yes python 3.8 | 3.9 | 3.10 | 3.11 | 3.12

Requests is an elegant and simple HTTP library for Python, built for human beings

Behold, the power of Requests:

```
>>> r = requests.get('https://api.github.com/user', auth=('user', 'pass'))
>>> r.status_code
200
>>> r.headers['content-type']
'application/json; charset=utf8'
>>> r.encoding
'utf-8'
```

Key Features:

- **Simplicity:** Known for its clean and straightforward syntax, making it accessible for beginners and experienced developers alike.
- **Flexibility:** Supports a wide range of HTTP methods and allows for customization, such as adding headers, query parameters, and more.
- **User-Friendly:** Compared to the built-in `urllib` library, Requests provides a more intuitive API which is particularly beneficial for rapid development and prototyping.
- **Automatic Content Decoding:** Requests automatically decodes the content based on the response headers, so you can work with the data directly without worrying about the encoding.
- **Session Management:** You can persist certain parameters across multiple requests using session objects, which can be useful for maintaining state.
- **Built-in Error Handling:** Requests come with built-in mechanisms to handle common HTTP errors and exceptions.
- **Community & Support:** The Requests library has a large and active community. Can easily find tutorials, documentation, and support for any issues you encounter.

Beloved Features

Requests is ready for today's web.

- Keep-Alive & Connection Pooling
- International Domains and URLs
- Sessions with Cookie Persistence
- Browser-style SSL Verification
- Automatic Content Decoding
- Basic/Digest Authentication
- Elegant Key/Value Cookies
- Automatic Decompression
- Unicode Response Bodies
- HTTP(S) Proxy Support
- Multipart File Uploads
- Streaming Downloads
- Connection Timeouts
- Chunked Requests
- `.netrc` Support

Requests officially supports Python 3.8+, and runs great on PyPy.

Applications of Requests:

1. Web Scraping:

- Requests is often the first step in web scraping.
- It allows developers to send HTTP requests to retrieve the HTML content of web pages, which can then be parsed and analyzed using libraries like BeautifulSoup or lxml.

2. API Interaction:

- Commonly used to interact with RESTful APIs.
- It can send various types of HTTP requests (GET, POST, PUT, DELETE) to perform operations like retrieving data, creating new records, or updating existing ones.

3. File Uploads:

- Requests supports file uploads, which is essential for applications where users need to submit files to a server.

4. Session Management:

- Requests provides the ability to maintain sessions across multiple requests, which is useful for applications that require authentication.
- A web application that requires users to log in can use a session to maintain the user's login state while making subsequent requests.

5. Data Submission:

- Requests can be used to submit data to web servers, especially in web forms where users enter information.

- Example: An application could use Requests to send user feedback or comments to a server.

Thursday, October 31, 2024 5:47 PM

```
import requests ✓  
  
response = requests.get("<uri>")
```

- They are parameters which are included in the URI as part of the request message.
- Mainly used to filter the data received from the server.
- In a URI, these are placed after the `?` and separated by `&`
- **Application:** Filter phones by a price range in an e-commerce website

{ www.flipkart.com? product=phones & min_price=10000
& max_price=30000 }

POST requests

Thursday, October 31, 2024


6:28 PM

Basic Syntax:

```
import requests

data = {
    "param1": "val1",
    "param2": "val2"
}

response = requests.post(url, data=data)
```

A diagram with red arrows showing the flow of data from the dictionary 'data' to the 'data' parameter in the 'requests.post' function call. A blue bracket underlines the entire function call 'requests.post(url, data=data)'.

- Since POST requests are used to send data to the server to create a resource, the Requests library provides the `data` parameter for this
- We can also pass JSON data directly using the `json` parameter
- These parameters accept dictionary-like objects as arguments

Headers

Thursday, October 31, 2024 7:33 PM

- Headers enable communicate of additional information such as authorization credentials, content types, and user-agent data, etc. between the clients and servers.
- Understanding and managing headers is essential for authenticating requests, specifying response formats, handling cookies, and personalizing user experiences.

Common types of Headers:

- **Authorization:** Used for passing authentication tokens.
- **Content-Type:** Specifies the format of the request data, such as JSON or XML.
- **User-Agent:** Provides information about the client making the request, such as the browser or app.
- **Accept:** Informs the server about the types of content the client can process, like JSON or HTML.
- **Custom Headers:** Headers that may be unique to an API or web application.

Practical tips for using Headers:

- Always check API documentation to see required headers for a particular request.
- Use headers for efficient server interaction because servers expect headers for security, data formatting, and client identification.
- Perform appropriate error handling because missing or incorrect headers can lead to HTTP errors like 401 Unauthorized or 415 Unsupported Media Type.