1. Module for Python Requests

A well-known third-party HTTP library that makes submitting HTTP requests in Python easier is the Python requests module. It offers a sophisticated and straightforward API for communicating with web services and other APIs.

What it is: Developers may submit HTTP/1.1 queries without having to manually add query strings to URLs or form-encode POST data thanks to the requests module, an easy-to-use framework. It automatically manages headers, cookies, authentication, and other HTTP functions.

How it is used to make HTTP requests:

import requests

- Making a GET request

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

- Making a POST request

```
response = requests.post('https://api.example.com/data', json={'key': 'value'})
```

- Making a PUT request

```
response = requests.put('https://api.example.com/data/1', json={'key': 'updated_value'})
```

- Making a DELETE request

```
response = requests.delete('https://api.example.com/data/1')
```

- Accessing response data

```
print(response.status_code)          -HTTP status code
print(response.json())               -JSON response
```

print(response.text) -Text response

print(response.headers) -Response headers

Sending data to online servers, retrieving data from APIs, web scraping, authenticating web services, uploading and downloading files, and managing sessions and cookies are examples of common use cases.

2. Data Formats: JSON and XML

JavaScript Object Notation, or JSON

JSON is a lightweight, text-based data interchange format that uses human-readable text to store and transmit data objects consisting of attribute-value pairs and arrays.

Common uses include Data sharing via APIs, configuration files, web apps, NoSQL databases (MongoDB), and real-time data streaming

Benefits

1. Lightweight and quick: Faster parsing and transmission result from smaller file sizes.

2. Simple syntax that is easy to read and comprehend for humans

3. Native JavaScript support: JavaScript parses data directly without the need for other libraries

4. Language independent: With built-in libraries, it is compatible with almost all programming languages.

5. Simple data structure: Makes use of well-known structures like arrays and objects

6. Improved performance: Parsing is quicker than with XML

Drawbacks

1. Restricted data types: Only arrays, objects, strings, numbers, booleans, and null are supported.

2. No schema validation: Unlike XML Schema, it lacks built-in validation.

3. No comments - Cannot include comments in JSON files for documentation

4. No support for namespaces: Namespaces cannot be defined to prevent naming conflicts.

5. Security issues: If improperly validated, it may be susceptible to injection attacks.

6. No support for metadata: Extra details about the data cannot be included.

XML stands for Extensible Markup Language.

XML is a markup language that establishes guidelines for encoding texts in a machine-readable and human-readable format. It uses tags to organise data in a hierarchical manner.

Its applications include SOAP web services; document storage and transfer; configuration files (Maven, Spring); data sharing in corporate systems; RSS feeds; and office documents (OpenOffice, Microsoft Office).

Benefits

1. Self-descriptive: Tags give the data context and meaning; 2. Schema validation: XML Schema (XSD) offers robust type verification and data validation.

3. Support for namespaces: Enables combining various XML vocabularies without creating naming conflicts

4. Extensible: It is simple to add new components without disrupting the current structure.

5. Metadata is supported; characteristics and comments can be added for more details.

6. Industry standard: extensively used in legacy and corporate systems

7. Complex data structures: Ideal for depicting intricate hierarchical data

8. Support for Unicode: Outstanding assistance with internationalisation

Drawbacks:

1. Bulky and verbose: requires opening and closing tags, which causes files to be larger.

2. Slower parsing: Processing is slowed down by more complicated structures.

3. Complexity: Compared to JSON, it is more challenging to understand and write

4. Greater bandwidth usage: More network bandwidth is needed for larger file sizes.

5. Less understandable by humans: Nested tags may make it more difficult to quickly comprehend

6. Overhead: Needs additional processing power and memory

3. The RESTful API

Networked applications can be designed using the REST (Representational State Transfer) architectural style. A web service that adheres to REST principles and uses HTTP requests to carry out CRUD (Create, Read, Update, Delete) actions on resources is known as a RESTful API.

How it operates RESTful APIs function by:

1. Making use of common HTTP techniques

Data can be retrieved using GET, created using POST, updated using PUT, and deleted using DELETE.

2. Communication without a state: Every request includes all required data; client context is not stored by the server.

3. Resource-based URLs: These URLs, such /api/users/123, use URIs to identify resources.

4. Common forms for data: usually exchanges data using JSON or XML.

5. Status codes for HTTP: Standard codes (200 OK, 404 Not Found, and 500 Internal Server Error) are returned.

Example:

GET /api/notes - Get all notes

GET /api/notes/1 - Get note with ID 1

POST /api/notes - Create a new note

PUT /api/notes/1 - Update note with ID 1

DELETE /api/notes/1 - Delete note with ID 1

Mobile application backends, web application APIs, microservices architecture, third-party integrations, cloud services, and IoT device connection are some of its uses.

Benefits

1. Scalability: The stateless design makes load balancing and horizontal scalability simple.

2. Flexibility: Depending on the demands of the client, it can return various data forms (JSON, XML, HTML).

3. Platform independence: Capable of operating on several platforms and programming languages

4. Simple to comprehend—Uses common HTTP techniques and adheres to predictable patterns

5. Support for caching: HTTP caching techniques lower server load and increase performance.

6. Concern separation: The client and server can change on their own

7. Lightweight: Compared to SOAP-based services, there is less overhead

8. Widespread adoption of the industry standard with copious documentation and tools

Drawbacks:

1. Over-fetching or under-fetching: This can provide either too much or too little information for requirements.

2. Statelessness overhead: Every request needs to provide all relevant data, which could result in a larger payload.

3. There is no built-in security; OAuth, JWT, and API keys must be implemented individually.

4. Limited standardization: Inconsistencies may result from various developers using REST in different ways.

5. Multiple round trips: To retrieve relevant data, it could be necessary to make several API requests.

6. Versioning issues: As an API changes, maintaining its versions might become difficult.

7. There is no built-in error management; instead, error handling techniques must be explicitly applied. 8. Inadequate optimization may result in increased bandwidth consumption.