## 1. Python requests module

The Python Requests module is a powerful, elegant, and user-friendly HTTP library that has become the de facto standard for making HTTP requests in Python. Requests is one of the most downloaded Python packages today, pulling in around 30 million downloads per week, and is currently depended upon by over 1,000,000 repositories on GitHub. It is an easy-to-use library with a lot of features ranging from passing parameters in URLs to sending custom headers and SSL Verification. Requests allows you to send HTTP/1.1 requests extremely easily, with no need to manually add query strings to URLs or form-encode PUT and POST data.

## 2. JSON and XML

JSON (JavaScript Object Notation) and XML (eXtensible Markup Language) are two of the most widely used data interchange formats in modern software development. Both formats facilitate the storage and transmission of structured data, yet they differ significantly in syntax, complexity, use cases, and overall design philosophy.

JSON

JSON is a a lightweight, text-based, open standard data interchange format that is easy for both humans to read and write and for machines to parse and generate. JSON is derived from JavaScript, but it is language-independent and can be parsed by many programming languages.

JSON advantages:

1. JSON's syntax is lightweight and straightforward, using a simple key-value pair notation similar to JavaScript objects. This makes it easy for humans to read and write, and the files look cleaner and more organised without empty tags and verbose data. The simplicity of its structure and minimal syntax makes JSON easier to understand and use, reducing the learning curve for developers

2. One of the most significant advantages of using JSON is that the file size is smaller compared to XML, thus transferring data is faster. JSON is more concise and compact, making it faster to parse and generate. This reduces the size and bandwidth of data transfers, as well as improving the performance and efficiency of data processing. JSON requires less coding and has a smaller size, making it faster to process and transmit data.

3. Despite being written in JavaScript, JSON is language-independent, meaning you can use it with any programming language. However, it has native support in JavaScript, making it ideal for web development. JSON integrates seamlessly with JavaScript, and you can parse JSON by a standard JavaScript function, eliminating the need for complex parsers. With JavaScript-based frameworks like Node.js becoming more popular, getting data in JSON format makes it easy to load data into an object tree.

4. JSON can be parsed using standard functions available in most programming languages, making it truly interoperable and easy to work with across different platforms. JSON is perfect for data interchange because every programming language is capable of parsing JSON. JSON's ease of parsing is a significant advantage - it can be parsed using standard JavaScript functions, which is more accessible than XML parsing that requires specialized parsers.

JSON disadvantages:

1. JSON supports a limited set of data types (strings, numbers, booleans, null, objects, and arrays) and lacks support for more complex types like binary data or dates. It primarily handles text data and lacks native support for complex data types. When dealing with binary data, JSON requires extra encoding and decoding, adding complexity to the implementation.

2. JSON does not support comments, namespaces, or attributes, making it difficult to add metadata or annotations to the data. There is no standard schema definition language for

JSON (though JSON Schema exists as an extension), making it hard to validate data or ensure interoperability. JSON is less expressive and flexible than XML, meaning there are restrictions on the complexity and variety of data structures and schemas that can be represented. This lack of built-in validation mechanisms means JSON lacks the robust validation that is inherent in XML.

3. Standard JSON does not allow comments within the data, which can hinder documentation and make it harder for developers to annotate and explain the data structure within the file itself. This limitation makes JSON files less self-documenting compared to XML, where comments can be freely added to explain the structure and purpose of elements.

4. JSON is less secure than XML in certain contexts, leaving the data vulnerable to injection attacks or cross-site scripting, particularly when JSONP (JSON with Padding) is used, as it can result in CSRF (Cross-Site Request Forgery) attacks. While JSON is usually safe, it may be more at risk in specific implementations. Additionally, JSON doesn't have the same level of schema validation and security features that XML provides through DTD and XML Schema.

XML

XML stands for eXtensible Markup Language. It is a markup language designed to store and transport data in a way that is readable by both humans and machines. Developed by the World Wide Web Consortium (W3C), XML has been in use since 1998 and is widely employed for data exchange and storage across various platforms and systems.

XML was compiled by a working group of eleven members at Sun Microsystems. Its first draft was published between August and November 1996, and the first version of XML was released in February 1998. XML was developed to prevent web fragmentation and facilitate data interchange between applications, serving as a solution to standardise data representation.

XML advantages:

1. An XML document is normally self-describing, usually having a link to its schema in the header. Schemas are also written in XML and defined in the XML specification by W3C. Because the schema describes what can or cannot be in a document, it provides two key advantages: authors know what fields need to be included, and documents can be validated against the schema. XML benefits from a well-defined schema language (XML Schema) that allows you to create and validate complex data structures, ensuring data consistency and reducing the risk of errors.

2. XML handles comments, metadata, attributes, and namespaces exceptionally well. This feature makes it easier for developers to keep track of what is happening and to share documents with other team members. XML enables you to define unique identifiers for your data elements through namespaces, making it more appropriate for complex and large-scale data structures. The tag-based format clearly defines data elements, making the structure explicit and self-documenting.

3. XML enables various data types such as images, charts, and other complex structures, unlike JSON which only supports strings, objects, numbers, and boolean arrays. XML provides support for XSLT (Extensible Stylesheet Language Transformations), a powerful language for transforming XML documents into other formats like HTML, PDF, or plain text. This capability makes XML more powerful for complex document structures and transformations

4. XML is extensively used in today's online world for banking services, online retail stores, integrating industrial systems, among other things. It's widely supported by virtually every programming language having a parser for it, making it the default choice for data interchange in many enterprise environments. XML has been around for more than three decades and is an integral part of many enterprise applications, providing mature tooling and extensive ecosystem support.

XML Disadvantages:

1. XML is more verbose and substitutes certain characters for entity references (for example, instead of the < character, XML uses the entity reference <). XML also uses end tags, which makes it longer than JSON. The tag structure certainly adds complexity and results in larger file sizes compared to JSON, leading to increased bandwidth usage and slower data transfer. This verbosity makes XML files bigger and harder to read compared to the cleaner JSON format.

2. You must parse XML with an XML parser, which often slows and complicates the process compared to JSON's simple parsing. XML is often characterised by its complexity and the old-fashioned standard due to the tag structure. This complexity can make XML more difficult for some developers to work with, especially those who are new to the format or prefer simpler alternatives.

3. Because of syntax and file size differences, you can parse JSON faster than XML. XML parsing typically requires more computational resources and time compared to JSON. The larger file sizes and more complex structure of XML can lead to slower processing times and reduced performance, particularly in applications where speed is critical.

4. When using XML, DTD (Document Type Definition) validation and external entity expansion are enabled by default, making structures disposed to some attacks such as XML External Entity (XXE) attacks. Disabling these features makes XML structures safer, but requires additional configuration. XML's complexity can introduce security vulnerabilities if not properly configured and validated.

**RESTful API**

RESTful APIs (Representational State Transfer Application Programming Interfaces) have become the de facto standard for building web services and enabling communication between applications over the internet.A RESTful API is an application programming interface that conforms to the constraints and architectural principles of REST (Representational State Transfer). REST is not a protocol but an architectural style for designing networked applications that interact over HTTP. RESTful APIs provide a standardised way for you to access, use, and modify resources on a server while creating efficient web applications that can scale across distributed systems. RESTful APIs enable communication between front-end applications (web or mobile) and back-end servers. An online store might use a RESTful API to connect its inventory system with its website and mobile app, ensuring all platforms have the same information about products.

RESTful API advantages:

1. RESTful APIs are easy to understand and learn due to their simplicity and use of standard HTTP methods. Because REST leverages standard HTTP methods, many developers find them easy to understand and use. The uniform interface provided by REST services reduces the learning curve for developers integrating these APIs. REST APIs are based on the same standards used for the web - same language, same code, and architecture - making them accessible and manageable for team members with limited experience in web development.

2. One of the most significant advantages of using a RESTful API is its scalability. REST APIs are highly scalable and can handle thousands, millions, and billions of data requests simultaneously without affecting performance. The reason is the server and client separation, which allows the protocol to work on various development projects independently. REST APIs employ a layered architecture where the application operates separately from the server, allowing software to request data through intermediaries like load balancers and proxies, enhancing scalability and performance. The stateless nature of REST also contributes to scalability, as servers don't need to maintain session information between requests.

3. RESTful APIs are highly flexible and can support various applications. Unlike protocols that are limited in data format choices, RESTful APIs support all data formats - you can send an HTTP request and receive JSON, XML, HTML, or plain text in response. REST is not constrained to XML but instead can return any format depending on what the client requests. The APIs are platform-agnostic and can handle various data formats, making them suitable for

conducting fast and lightweight operations. This flexibility makes them adaptable to various working syntax and platforms and brings opportunities to work on different projects. REST APIs can communicate using any data format and can be adapted to almost any application on the web regardless of its format, language, or architecture.

4. REST is widely used and supported by many frameworks, tools, and programming languages, making it easier to find resources and community support. RESTful APIs are the most commonly used APIs in the world of web services and are extensively supported by current programming languages and frameworks. This wide adoption means developers can find extensive documentation, libraries, and community support when working with RESTful APIs. The maturity of the REST ecosystem provides robust tooling for development, testing, and monitoring.

RESTful API disadvantages:

1. One of the disadvantages of RESTful APIs is that you can lose the ability to maintain state in REST, such as within sessions. While statelessness provides benefits for scalability, it can complicate scenarios where maintaining user context across multiple requests is necessary. The lack of built-in session management means applications must implement their own mechanisms for maintaining user state, which can add complexity to the development process.

2. Clients may receive more data than needed (over-fetching) or may need to make multiple requests to get all the required data (under-fetching). The structure of the REST response is fixed, which may not always align with the needs of the client. This limitation can lead to inefficient data transfer and increased bandwidth usage, particularly problematic for mobile applications with limited data plans.

3. Managing different versions of an API can become complex as the application evolves. When you need to make breaking changes to your API, you often need to maintain multiple versions simultaneously to support existing clients. This versioning requirement can lead to increased maintenance burden and code duplication.

4. Although REST APIs are generally considered easy to use, the design of a REST API can be more complex than other APIs, especially if you are not familiar with web architecture. RESTful APIs can be complex to develop, requiring much technical knowledge and understanding of REST principles. For developers new to REST or those working with complex data structures, implementing a proper RESTful API can present challenges.