Python, Requests, & The Standard Library

2013

3 min read • 633 words

Themes: Technology

Introduction

This presentation examined a critical question facing the Python community: whether the wildly popular Requests library should be included in Python's standard library. The discussion explored both the implications and reasoning behind this consequential decision, weighing the benefits of official inclusion against the risks of losing the agility that made Requests successful in the first place.

Requests Overview

Requests had established itself as the gold standard for HTTP interactions in Python by prioritizing security and thoughtful design above all else. The library featured optimized interfaces that made best practices in SSL, connection pooling, encoding, and headers accessible to everyday developers, dramatically simplifying interaction with web services compared to the standard library's complex urllib2.

The numbers spoke to its impact: Requests had become the most downloaded Python package, with approximately 42 million downloads from PyPI. This unprecedented adoption led to regular suggestions from the community for its inclusion in the standard library—a proposal that seemed both obvious and fraught with complications.

Arguments for Inclusion

The case for including Requests in the standard library rested on several compelling arguments. From a social responsibility perspective, inclusion seemed like simply "the right thing to do"—given Requests' critical role in the Python ecosystem, why shouldn't every Python installation include this essential functionality?

Sustainability concerns also favored inclusion. Standard library status could facilitate funding for core contributors, ensuring the project's long-term viability without relying solely on volunteer effort. The ecosystem benefits would extend beyond Requests itself—Chardet, a key dependency, emerged as a particularly strong candidate for standard library inclusion due to its universal utility in character encoding detection.

> The argument for standard library inclusion gained additional weight from Python's adoption of ensurepip, which demonstrated the language's willingness to embrace widely-used third-party tools as core infrastructure.

Arguments Against Inclusion

However, equally compelling arguments emerged against standard library inclusion. The independence argument struck at the heart of what made Requests valuable: its superiority over the standard library stemmed from its ability to innovate freely and respond quickly to security incidents or specification changes. Standard library inclusion could paradoxically diminish the very agility that created its excellence.

> This argument proved prescient. The debate over Requests' standard library inclusion highlighted fundamental questions about Python's development philosophy and the role of third-party packages in a language ecosystem.

The flexibility concerns extended beyond philosophical considerations to practical development realities. Standard library inclusion would fundamentally limit the project's ability to release updates and improvements promptly, potentially slowing the very innovation cycles that kept Python's HTTP capabilities at the forefront of web development.

Broader Questions

The Requests debate illuminated deeper questions about Python's evolving ecosystem. What should the goals of the standard library be in an era where tools like `ensurepip` made package installation increasingly seamless? The traditional argument for standard library inclusion—ensuring universal availability—seemed less compelling when package management had become so straightforward.

This led to fundamental questions about critical infrastructure. While Requests undoubtedly qualified as critical infrastructure for the Python community, standard library inclusion might paradoxically make it less adaptable to the rapid changes that characterize modern web development. The tension between stability and innovation had no easy resolution.

Conclusion

After weighing all considerations, the presentation ultimately argued against including Requests in the standard library. The decision emphasized that the project's greatest value to the Python community lay in its ability to remain agile and independent—qualities that would be compromised by standard library inclusion.

This stance reflected a mature understanding of software ecosystems: sometimes the most responsible choice is to resist the apparent honor of official inclusion in favor of maintaining the flexibility that created value in the first place. Requests would continue serving the Python community most effectively by remaining outside the standard library, free to evolve at the pace of web technology rather than the necessarily more conservative pace of language development.