

Gyöngyösi Máté

The organization of Python development





gy-mate commented on Oct 5

Owner ...

Description

If the `requires-python` Trove classifier in `pyproject.toml` is ignored and package is ran with Python < 3.12, the following error occurs at the end of the process:

A screenshot of a terminal window titled "Letöltesek -- zsh -- 150x24". The command run was `moodle-to-vikwikiquiz -v -n /Users/Downloads/kviz.html`. The output shows a long stack trace from the `moodle_to_vikwikiquiz` module, followed by a `SyntaxError` at the end:

```
Traceback (most recent call last):
  File "/Library/Frameworks/Python.framework/Versions/3.11/bin/moodle-to-vikwikiquiz", line 5, in <module>
    from moodle_to_vikwikiquiz.main import main
  File "/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-packages/moodle_to_vikwikiquiz/main.py", line 16, in <module>
    from .quiz import Quiz # type: ignore
  File "/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-packages/moodle_to_vikwikiquiz/quiz.py", line 27, in <module>
    from .quiz_helpers import * # type: ignore
  File "/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-packages/moodle_to_vikwikiquiz/quiz_helpers.py", line 12, in <module>
    from .question import Question # type: ignore
  File "/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-packages/moodle_to_vikwikiquiz/question.py", line 48
    f"\n|Válasz={",".join([str(answer) for answer in ordered_correct_answers])}"
                                         ^
SyntaxError: f-string: expecting '' %
```

Reported by [@TancsicsGergely](#) on Discord

Version

2.0.2

Preferred solution

Build backends (`pipx` and `shiv` uses `setuptools`) should fail when building `moodle-to-vikwikiquiz` with a Python version that doesn't meet the `requires-python` Trove classifier.

Related: [pypa/pipx#387](#)

To implement this, submitting a PEP [was hinted at](#) by a CPython developer. The steps for this are:

- Check if an issue was opened previously in the [CPython GitHub repo](#)

Assignees



Labels

[blocked](#) [enhancement](#) [publishing](#)



Projects

[moodle-to-vikwikiquiz](#)

Status: to do

+2 more



Milestone

No milestone



Development

Create a branch for this issue or link a pull request.

Customize

[Unsubscribe](#)

You're receiving notifications because you're watching this repository.

1 participant



[Lock conversation](#)

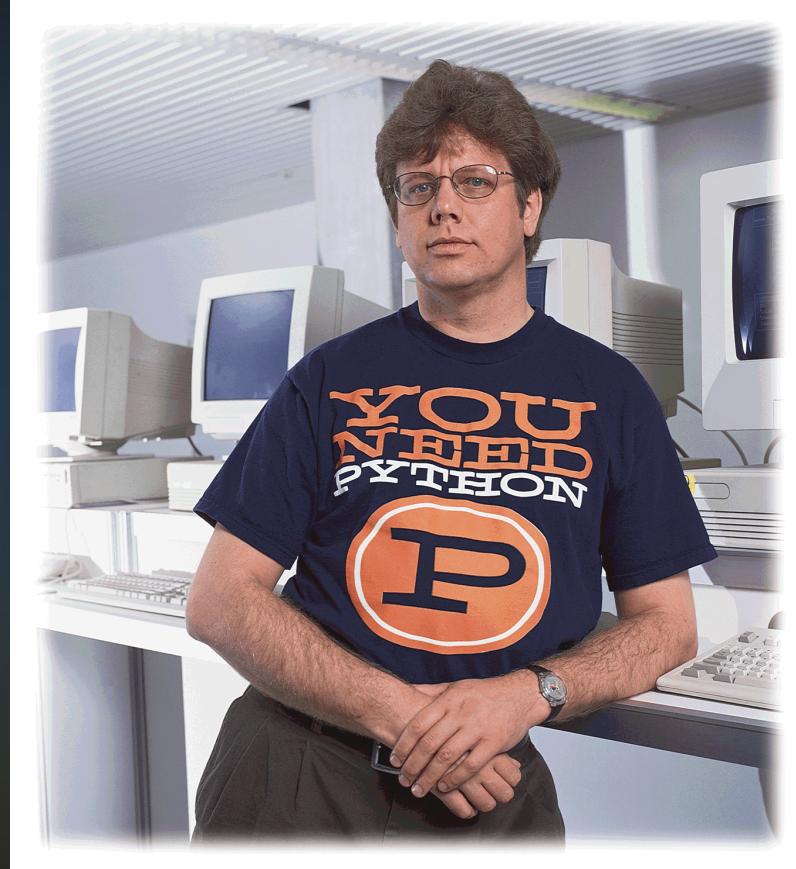
[Pin issue](#) ⓘ

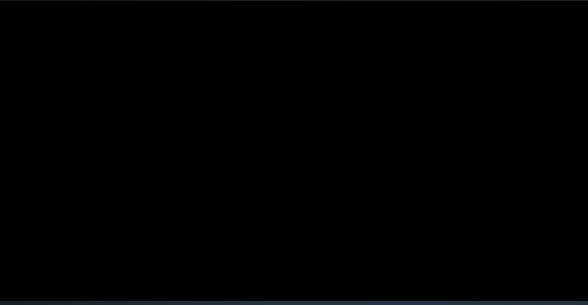
[Transfer issue](#)

[Delete issue](#)

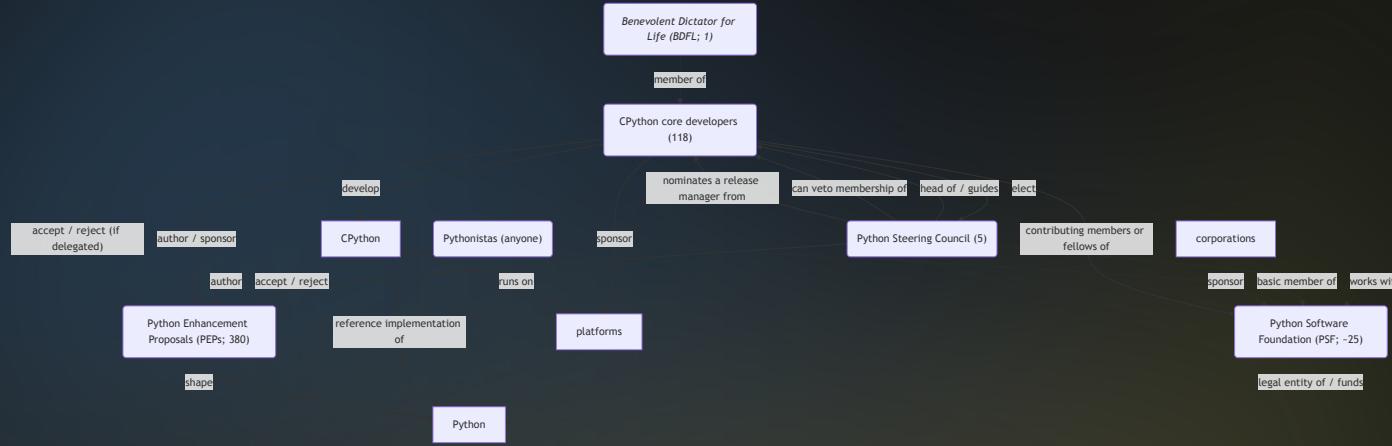


Python history





Python governance



What is Python?

The Python Language Reference

This reference manual describes the syntax and “core semantics” of the language. It is terse, but attempts to be exact and complete. The semantics of non-essential built-in object types and of the built-in functions and modules are described in [The Python Standard Library](#). For an informal introduction to the language, see [The Python Tutorial](#). For C or C++ programmers, two additional manuals exist: [Extending and Embedding the Python Interpreter](#) describes the high-level picture of how to write a Python extension module, and the [Python/C API Reference Manual](#) describes the interfaces available to C/C++ programmers in detail.

- 1. Introduction
 - 1.1. Alternate Implementations
 - 1.2. Notation
- 2. Lexical analysis
 - 2.1. Line structure
 - 2.2. Other tokens
 - 2.3. Identifiers and keywords
 - 2.4. Literals
 - 2.5. Operators
 - 2.6. Delimiters
- 3. Data model
 - 3.1. Objects, values and types
 - 3.2. The standard type hierarchy
 - 3.3. Special method names
 - 3.4. Coroutines

The Python Standard Library

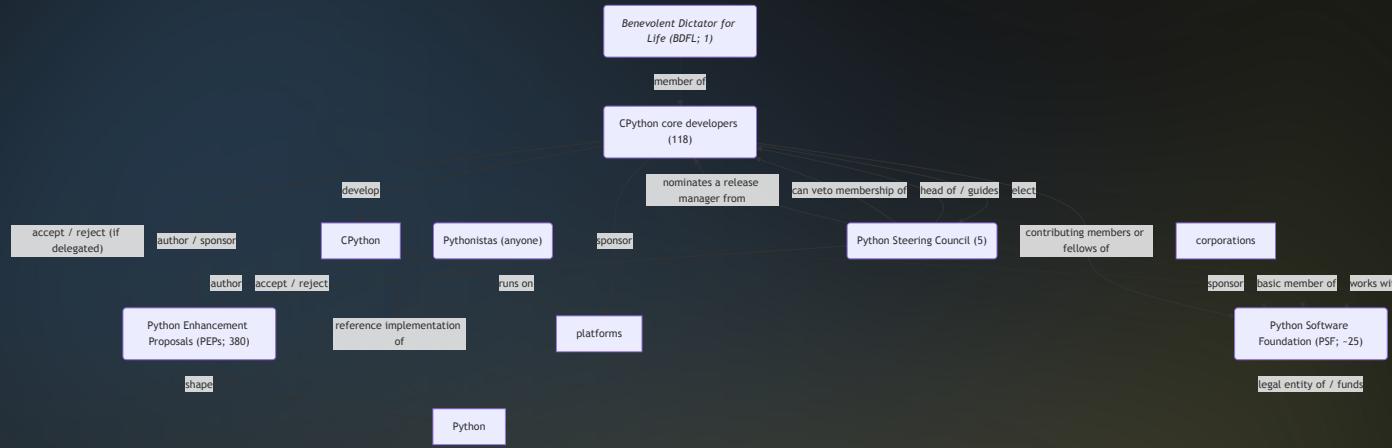
While [The Python Language Reference](#) describes the exact syntax and semantics of the Python language, this library reference manual describes the standard library that is distributed with Python. It also describes some of the optional components that are commonly included in Python distributions.

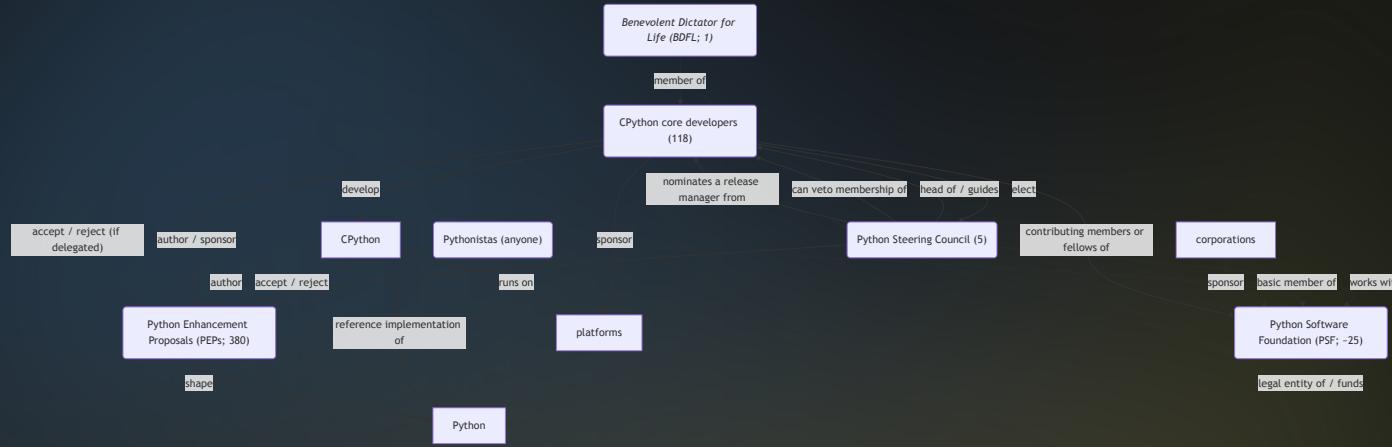
Python's standard library is very extensive, offering a wide range of facilities as indicated by the long table of contents listed below. The library contains built-in modules (written in C) that provide access to system functionality such as file I/O that would otherwise be inaccessible to Python programmers, as well as modules written in Python that provide standardized solutions for many problems that occur in everyday programming. Some of these modules are explicitly designed to encourage and enhance the portability of Python programs by abstracting away platform-specifics into platform-neutral APIs.

The Python installers for the Windows platform usually include the entire standard library and often also include many additional components. For Unix-like operating systems Python is normally provided as a collection of packages, so it may be necessary to use the packaging tools provided with the operating system to obtain some or all of the optional components.

In addition to the standard library, there is an active collection of hundreds of thousands of components (from individual programs and modules to packages and entire application development frameworks), available from the [Python Package Index](#).

- [Introduction](#)
 - [Notes on availability](#)
- [Built-in Functions](#)
 - [abs\(\)](#)
 - [aiter\(\)](#)





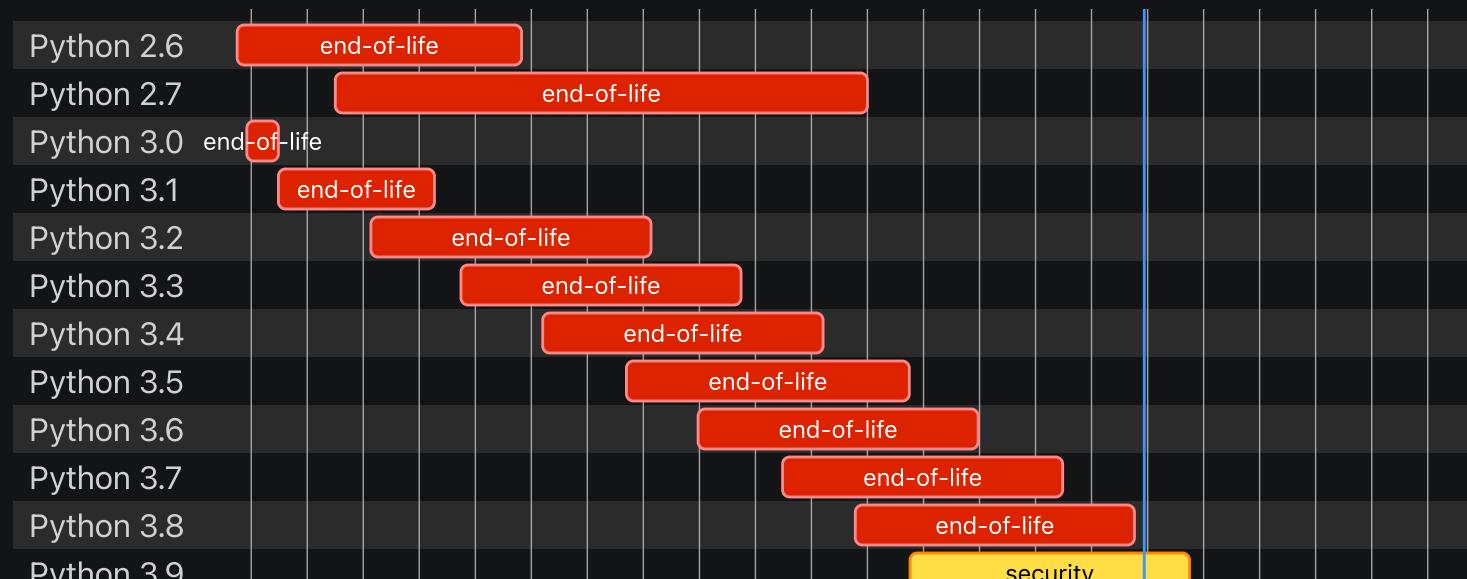
Release cycle

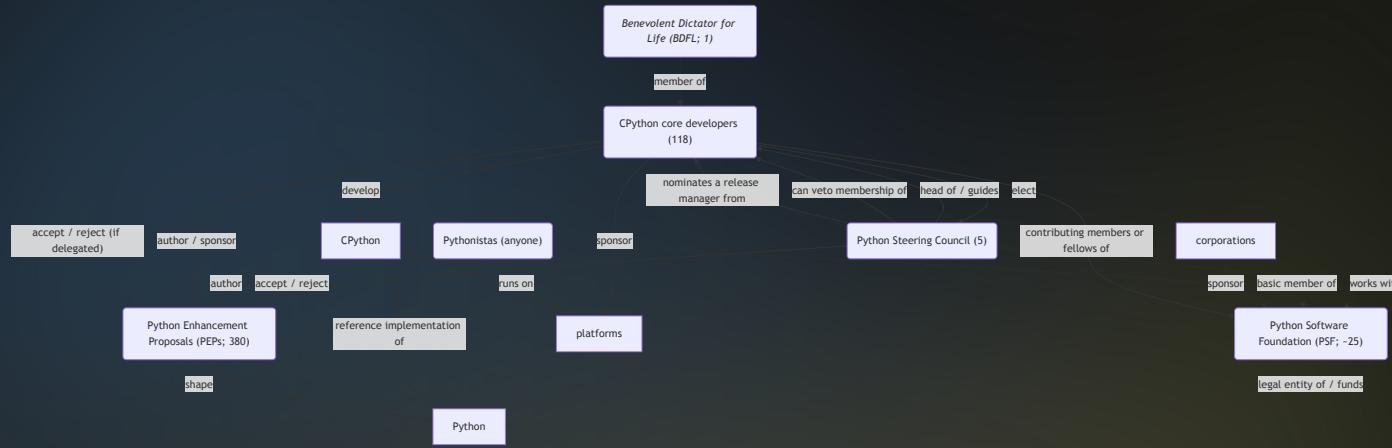


Status of Python versions

The `main` branch is currently the future Python 3.14, and is the only branch that accepts new features. The latest release for each Python version can be found on the [download page](#).

Python release cycle



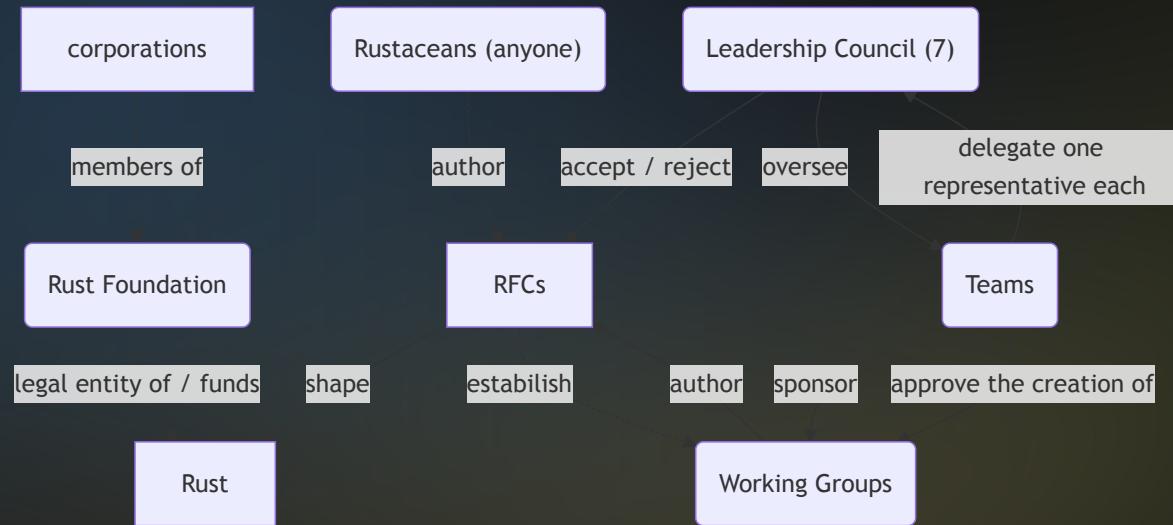


Join the PSF today!

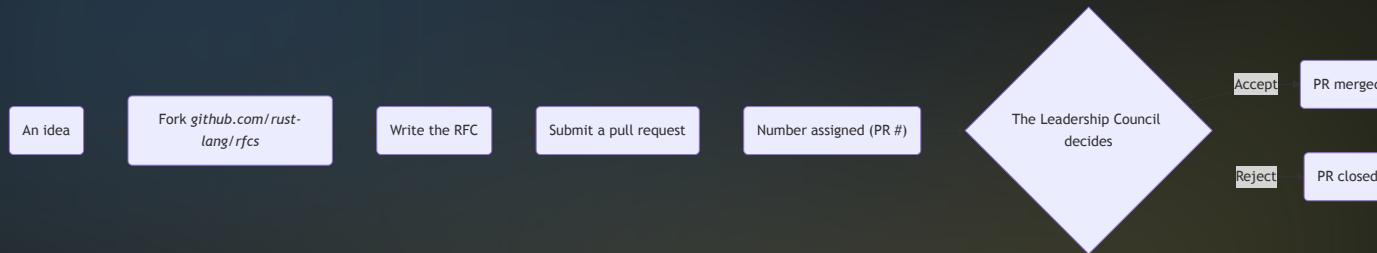
python.org/users/membership

Rust governance

Organogram

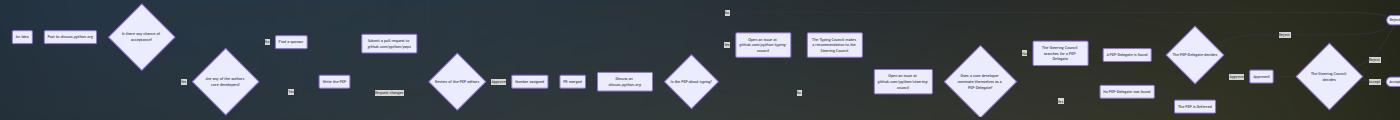


RFC process



vs. Python

PEP process





Contents

- Abstract
- Rationale
- Voting Scores
- References
- Copyright

[Page Source \(GitHub\)](#)

PEP 10 – Voting Guidelines

Author: Barry Warsaw <barry at python.org>

Status: ~~Active~~

Type: ~~Process~~

Created: 07-Mar-2002

Post-History: 07-Mar-2002

► [Table of Contents](#)

Abstract

This PEP outlines the python-dev voting guidelines. These guidelines serve to provide feedback or gauge the “wind direction” on a particular proposal, idea, or feature. They don’t have a binding force.

Looks are important!

Contents

- Introduction
- Topics
- API
- Numerical Index
- Index by Category
 - Process and Meta-PEPs
 - Other Informational PEPs
 - Provisional PEPs

(provisionally accepted;
interface may still change)
 - Accepted PEPs (accepted;
may not be implemented yet)
 - Open PEPs (under
consideration)
 - Finished PEPs (done, with a
stable interface)
 - Historical Meta-PEPs and
Informational PEPs
 - Deferred PEPs (postponed
pending further research or

PEP 0 – Index of Python Enhancement Proposals (PEPs)

Author: The PEP Editors

Status: Active

Type: Informational

Created: 13-Jul-2000

► Table of Contents

Introduction

This PEP contains the index of all Python Enhancement Proposals, known as PEPs. PEP numbers are assigned by the PEP editors, and once assigned are never changed. The version control history of the PEP texts represent their historical record.

```
1  from datetime import datetime
2  import json
3  import requests
4
5  peps_json = json.loads(requests.get("https://peps.python.org/api/peps.json").text)
6  peps = [value for key, value in peps_json.items()]
7  peps = sorted(peps, key=lambda pep: datetime.strptime(pep["created"], "%d-%b-%Y"))
8
9  retrospect = 5
10 print(f"The last {retrospect} PEPs are:")
11 for pep in peps[-retrospect:]:
12     print(f"- PEP {pep['number']} created on {pep['created']}: {pep['title']}
```

The last 5 PEPs are:

- PEP 763 created on 24-Oct-2024: Limiting deletions on PyPI
- PEP 765 created on 15-Nov-2024: Disallow return/break/continue that exit a finally block
- PEP 766 created on 18-Nov-2024: Explicit Priority Choices Among Multiple Indexes
- PEP 767 created on 18-Nov-2024: Annotating Read-Only Attributes
- PEP 768 created on 25-Nov-2024: Safe external debugger interface for CPython

Contents

- Introduction
- Topics
- API
- Numerical Index
- Index by Category
 - Process and Meta-PEPs
 - Other Informational PEPs
 - Provisional PEPs

(provisionally accepted;
interface may still change)
 - Accepted PEPs (accepted;
may not be implemented yet)
 - Open PEPs (under
consideration)
 - Finished PEPs (done, with a
stable interface)
 - Historical Meta-PEPs and
Informational PEPs
 - Deferred PEPs (postponed
pending further research or

PEP 0 – Index of Python Enhancement Proposals (PEPs)

Author: The PEP Editors

Status: Active

Type: Informational

Created: 13-Jul-2000

► Table of Contents

Introduction

This PEP contains the index of all Python Enhancement Proposals, known as PEPs. PEP numbers are assigned by the PEP editors, and once assigned are never changed. The version control history of the PEP texts represent their historical record.



Rust RFCs - RFC Book - Active RFC List

The “RFC” (request for comments) process is intended to provide a consistent and controlled path for changes to Rust (such as new features) so that all stakeholders can be confident about the direction of the project.

Many changes, including bug fixes and documentation improvements can be implemented and reviewed via the normal GitHub pull request workflow.

Some changes though are “substantial”, and we ask that these be put through a bit of a design process and produce a consensus among the Rust community and the [sub-teams](#).

Table of Contents

- [Opening](#)
- [Table of Contents](#)
- [When you need to follow this process](#)
- [Sub-team specific guidelines](#)
- [Before creating an RFC](#)

Summary

https://github.com/gy-mate/talks/tree/main/2024-12-10_BME_FLOSS_assignment

References

- [1] C. F. Bolz-Tereick, 'The First 15 Years of PyPy — a Personal Retrospective'. Accessed: Dec. 08, 2024. [Online]. Available: <https://pypy.org/posts/2018/09/the-first-15-years-of-pypy-3412615975376972020.html>
- [2] bstriie and B. Anderson, '0002-rfc-process'. Mar. 11, 2014. Accessed: Dec. 10, 2024. [Online]. Available: <https://rust-lang.github.io/rfcs/0002-rfc-process.html>
- [3] Cython developers, 'About Cython'. Accessed: Dec. 08, 2024. [Online]. Available: <https://cython.org/>
- [4] Data Is Beautiful, Most Popular Programming Languages 1965 - 2022, (Dec. 12, 2022). Accessed: Dec. 10, 2024. [Online Video]. Available: <https://www.youtube.com/watch?v=qQXXI5QFUfw>
- [5] S. J. Turner et al., '3392-leadership-council'. Aug. 01, 2022. Accessed: Dec. 10, 2024. [Online]. Available: <https://rust-lang.github.io/rfcs/3392-leadership-council.html>
- [6] Ł. Langa, 'PEP 602 – Annual Release Cycle for Python'. Jun. 04, 2019. Accessed: Dec. 10, 2024. [Online]. Available: <https://peps.python.org/pep-0602/>
- [7] .NET Foundation, 'Python.NET documentation'. 2022. Accessed: Dec. 08, 2024. [Online]. Available: <https://pythonnet.github.io/pythonnet/>
- [8] .NET Foundation, '.NET Foundation'. 2024. Accessed: Dec. 08, 2024. [Online]. Available: <https://dotnetfoundation.org/>
- [9] Python Software Foundation, 'Experts index'. 2011. Accessed: Dec. 10, 2024. [Online]. Available: <https://devguide.python.org/core-developers/experts/#>
- [10] Python Software Foundation, 'About the Python Software Foundation'. 2024. Accessed: Dec. 09, 2024. [Online]. Available: <https://www.python.org/psf/about/>
- [11] Python Software Foundation, 'The Python Language Reference'. 2024. Accessed: Dec. 08, 2024. [Online]. Available: <https://docs.python.org/3/reference/index.html>
- [12] Python Software Foundation, 'The Python Standard Library'. 2024. Accessed: Dec. 08, 2024. [Online]. Available: <https://docs.python.org/3/library/index.html>
- [13] Rust Team, 'Governance'. Accessed: Dec. 10, 2024. [Online]. Available: <https://www.rust-lang.org/governance>
- [14] The PEP Editors, 'PEP 0 – Index of Python Enhancement Proposals (PEPs)'. Jul. 13, 2000. Accessed: Dec. 10, 2024. [Online]. Available: <https://peps.python.org>

Made with



vsodium



yt-dlp



ffmpeg



Mermaid



Zotero

```
sys.exit()
```