Python Pros and Cons

Jason Olson
Staff Software Engineer

@jolson88 www.jolson88.com

Pros and Cons

Pro: Comprehensive Standard Library

Pro: Community-driven

Pro: 3rd Party Libraries

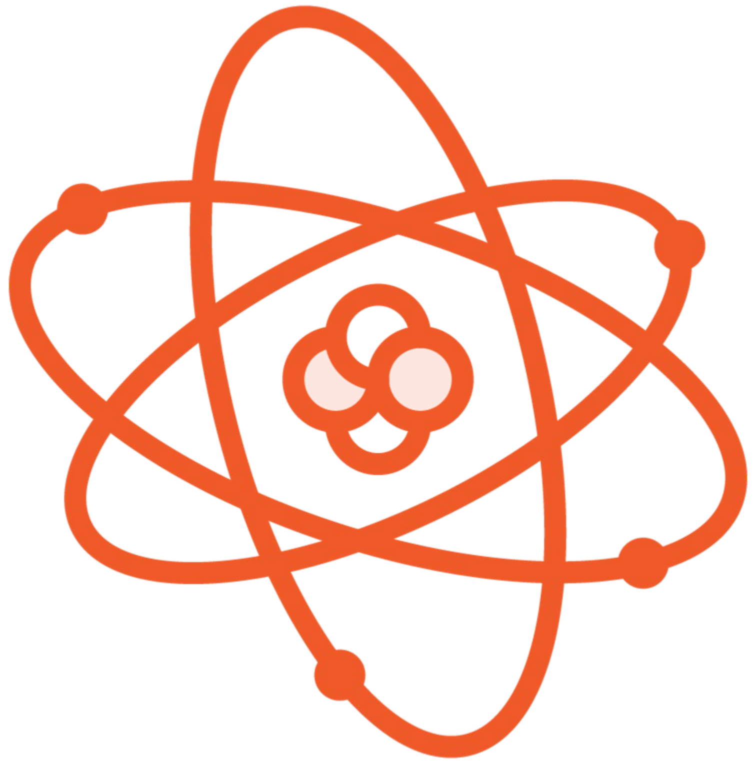
Pro: 3rd Party Tools

Python Cons

Pro: Comprehensive Standard Library

What things can we do right out of the box?

Standard Library Philosophies



Minimal Standard Library

Pay only for what you use

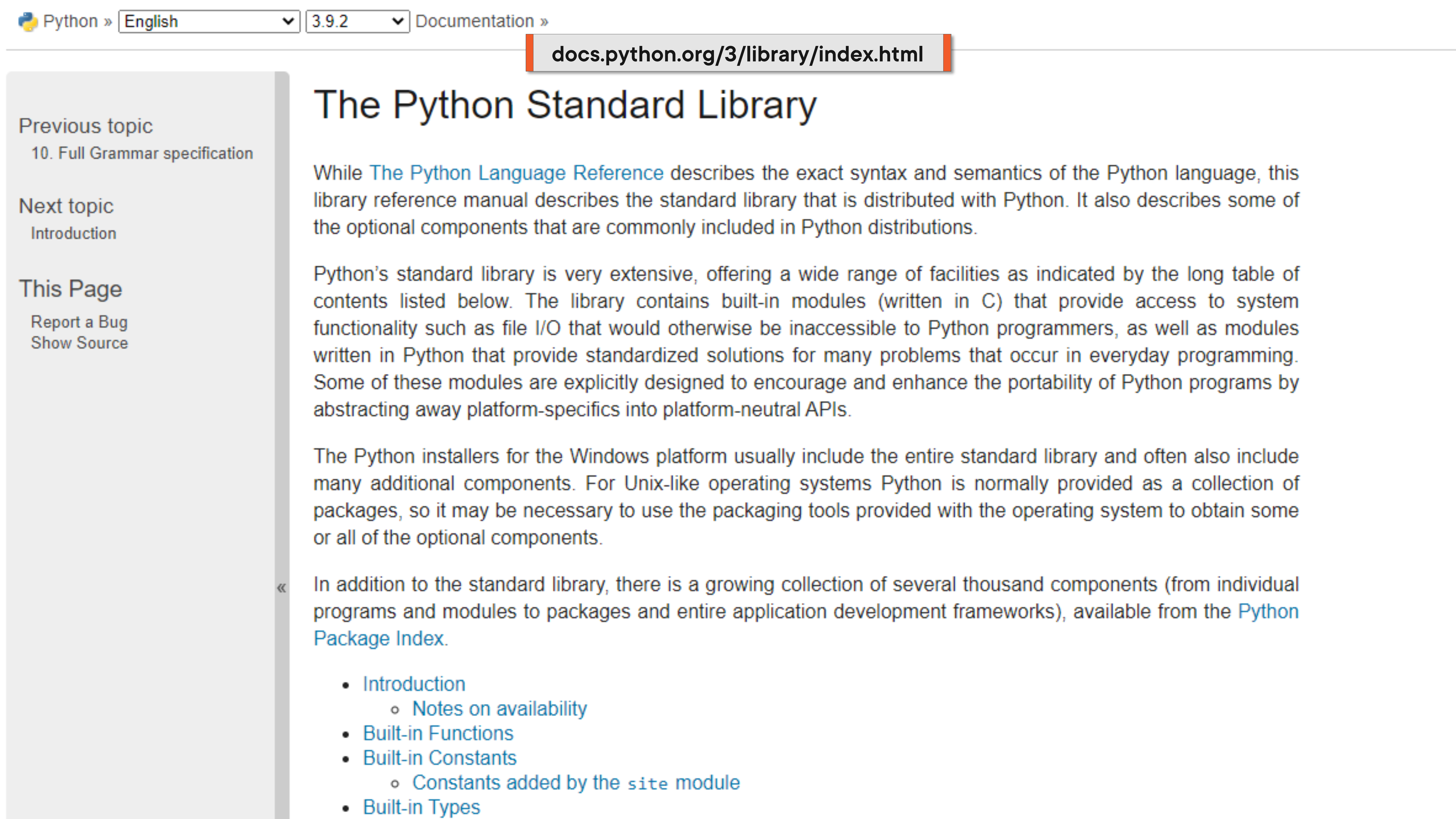
Comprehensive Standard Library

You can do it all

A Comprehensive Standard Library

- Collections
- File I/O
- Dates and times
- Compression
- User interfaces
- Much more...

docs.python.org/3/library/index.html



The Python Standard Library

Previous topic

10. Full Grammar specification

Next topic

Introduction

This Page

Report a Bug

Show Source

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.

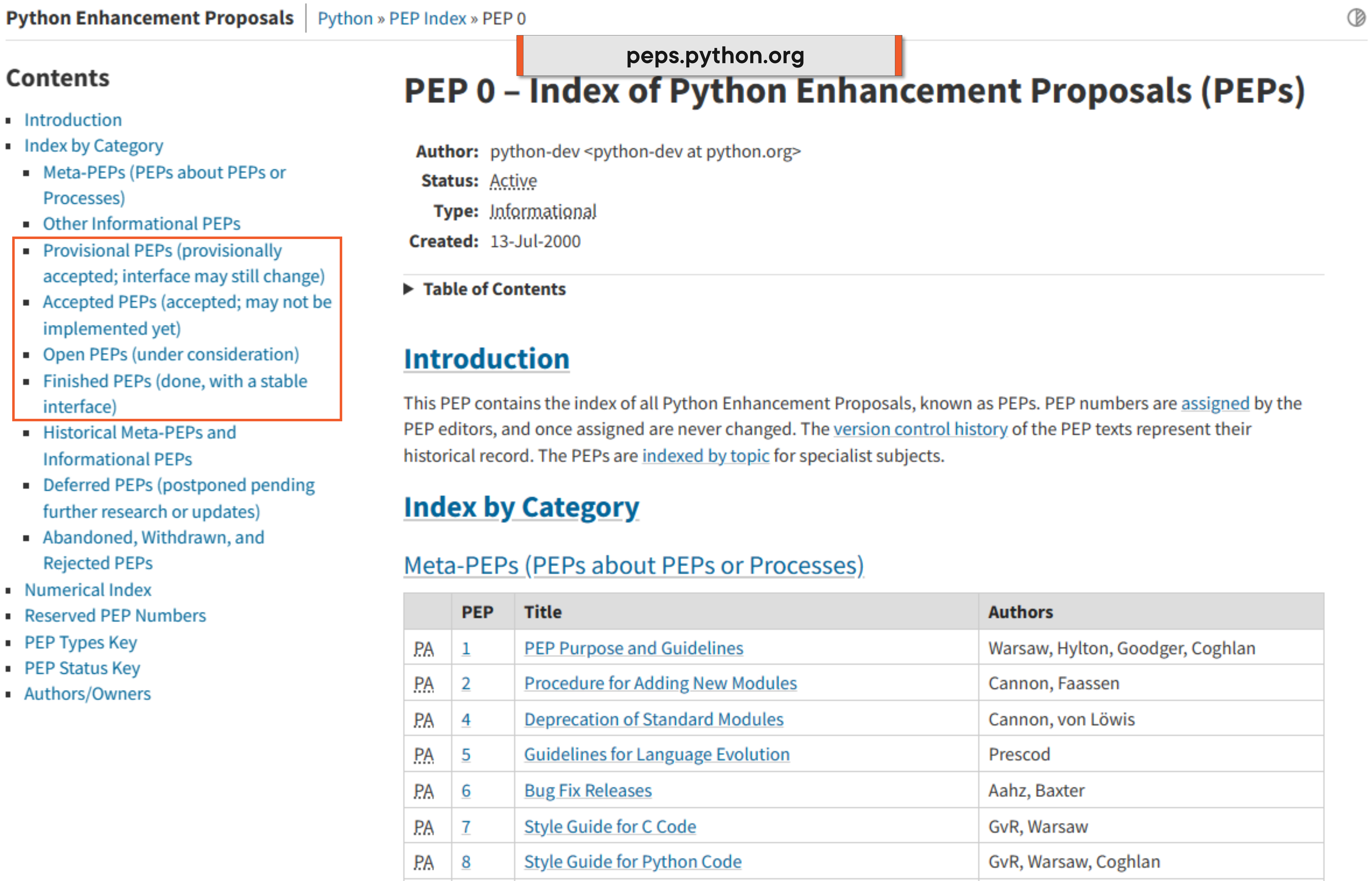
Python's standard library is very extensive, offering a wide range of facilities as indicated by the long table of contents listed below. The library contains built-in modules (written in C) that provide access to system functionality such as file I/O that would otherwise be inaccessible to Python programmers, 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 a growing collection of several thousand components (from individual programs and modules to packages and entire application development frameworks), available from the Python Package Index.

Pro: Community-driven

peps.python.org



Python Enhancement Proposals | Python » PEP Index » PEP 0

Contents

- Introduction
- Index by Category
  - Meta-PEPs (PEPs about PEPs or Processes)
  - 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 updates)
  - Abandoned, Withdrawn, and Rejected PEPs
- Numerical Index
- Reserved PEP Numbers
- PEP Types Key
- PEP Status Key
- Authors/Owners

PEP 0 – Index of Python Enhancement Proposals (PEPs)

Author: python-dev <python-dev at python.org>
Status: Active
Type: Informational
Created: 13-Jul-2000

▶ **Table of Contents**

Introduction

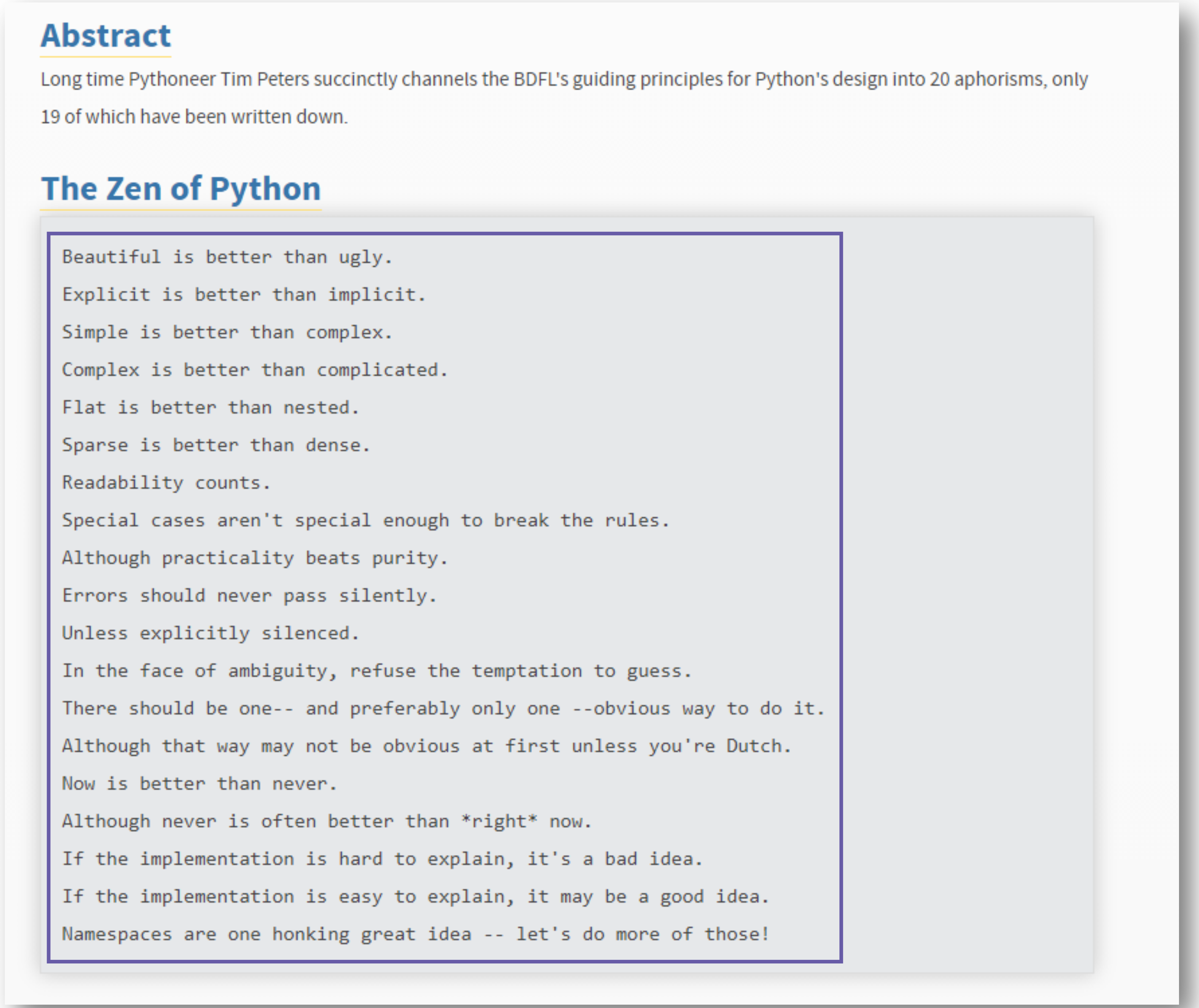
This PEP contains the index of all Python Enhancement Proposals, known as PEPs. PEP numbers are assigned by the PEP editors, and once assigned are never changed. The version control history of the PEP texts represent their historical record. The PEPs are indexed by topic for specialist subjects.

Index by Category

Meta-PEPs (PEPs about PEPs or Processes)

	PEP	Title	Authors
PA	1	PEP Purpose and Guidelines	Warsaw, Hylton, Goodger, Coghlan
PA	2	Procedure for Adding New Modules	Cannon, Faassen
PA	4	Deprecation of Standard Modules	Cannon, von Löwis
PA	5	Guidelines for Language Evolution	Prescod
PA	6	Bug Fix Releases	Aahz, Baxter
PA	7	Style Guide for C Code	GvR, Warsaw
PA	8	Style Guide for Python Code	GvR, Warsaw, Coghlan

peps.python.org/pep-0020/



Abstract

Long time Pythoneer Tim Peters succinctly channels the BDFL's guiding principles for Python's design into 20 aphorisms, only 19 of which have been written down.

The Zen of Python

```
Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```

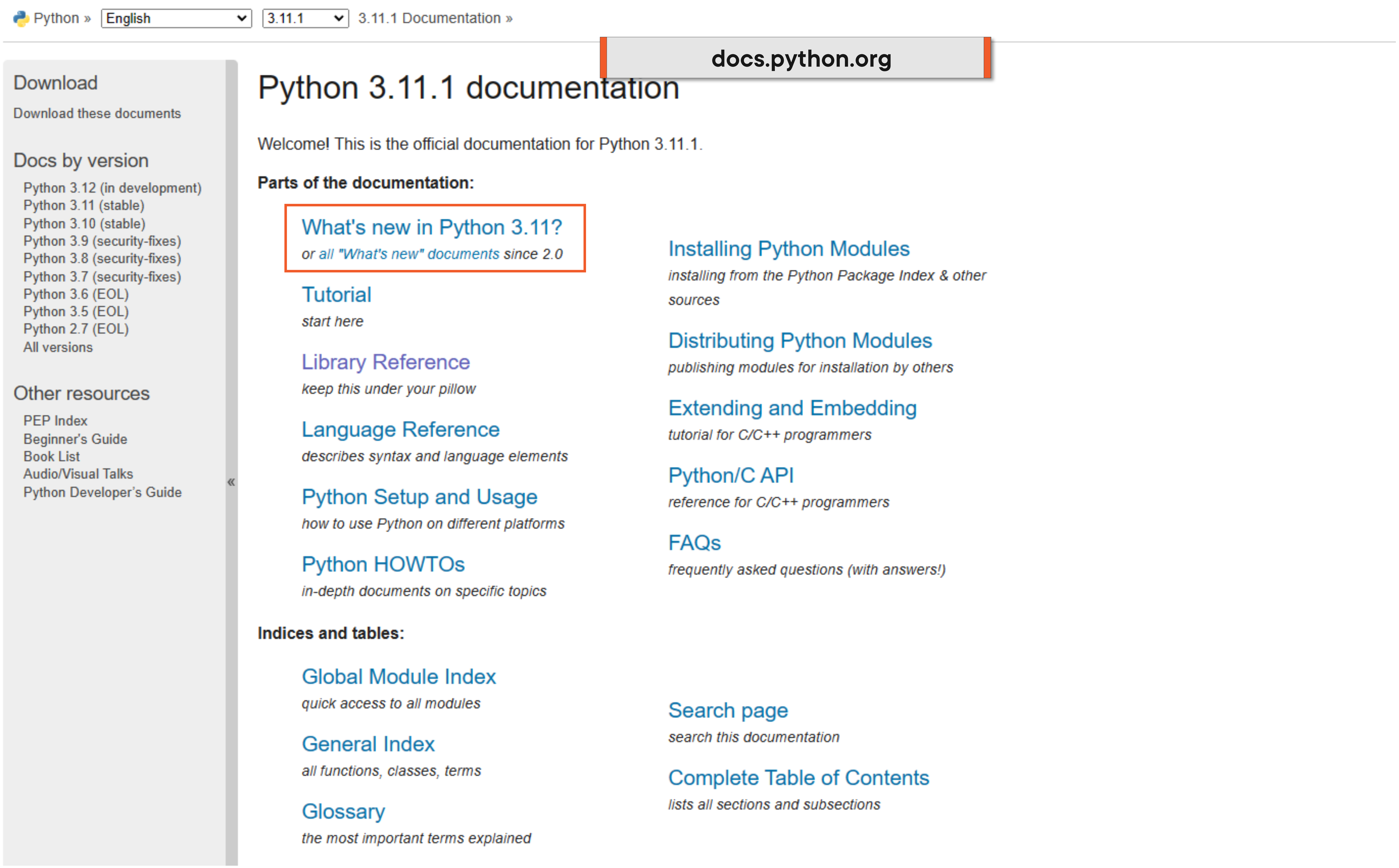
Beautiful is better than ugly

Simple is better than complex

Flat is better than nested

Sparse is better than dense

Readability counts



docs.python.org

Python » English 3.11.1 3.11.1 Documentation »

Download

Download these documents

Docs by version

- Python 3.12 (in development)
- Python 3.11 (stable)
- Python 3.10 (stable)
- Python 3.9 (security-fixes)
- Python 3.8 (security-fixes)
- Python 3.7 (security-fixes)
- Python 3.6 (EOL)
- Python 3.5 (EOL)
- Python 2.7 (EOL)
- All versions

Other resources

- PEP Index
- Beginner's Guide
- Book List
- Audio/Visual Talks
- Python Developer's Guide

Python 3.11.1 documentation

Welcome! This is the official documentation for Python 3.11.1.

Parts of the documentation:

What's new in Python 3.11?
or all "What's new" documents since 2.0

Tutorial
start here

Library Reference
keep this under your pillow

Language Reference
describes syntax and language elements

Python Setup and Usage
how to use Python on different platforms

Python HOWTOs
in-depth documents on specific topics

Installing Python Modules
installing from the Python Package Index & other sources

Distributing Python Modules
publishing modules for installation by others

Extending and Embedding
tutorial for C/C++ programmers

Python/C API
reference for C/C++ programmers

FAQs
frequently asked questions (with answers!)

Indices and tables:

Global Module Index
quick access to all modules

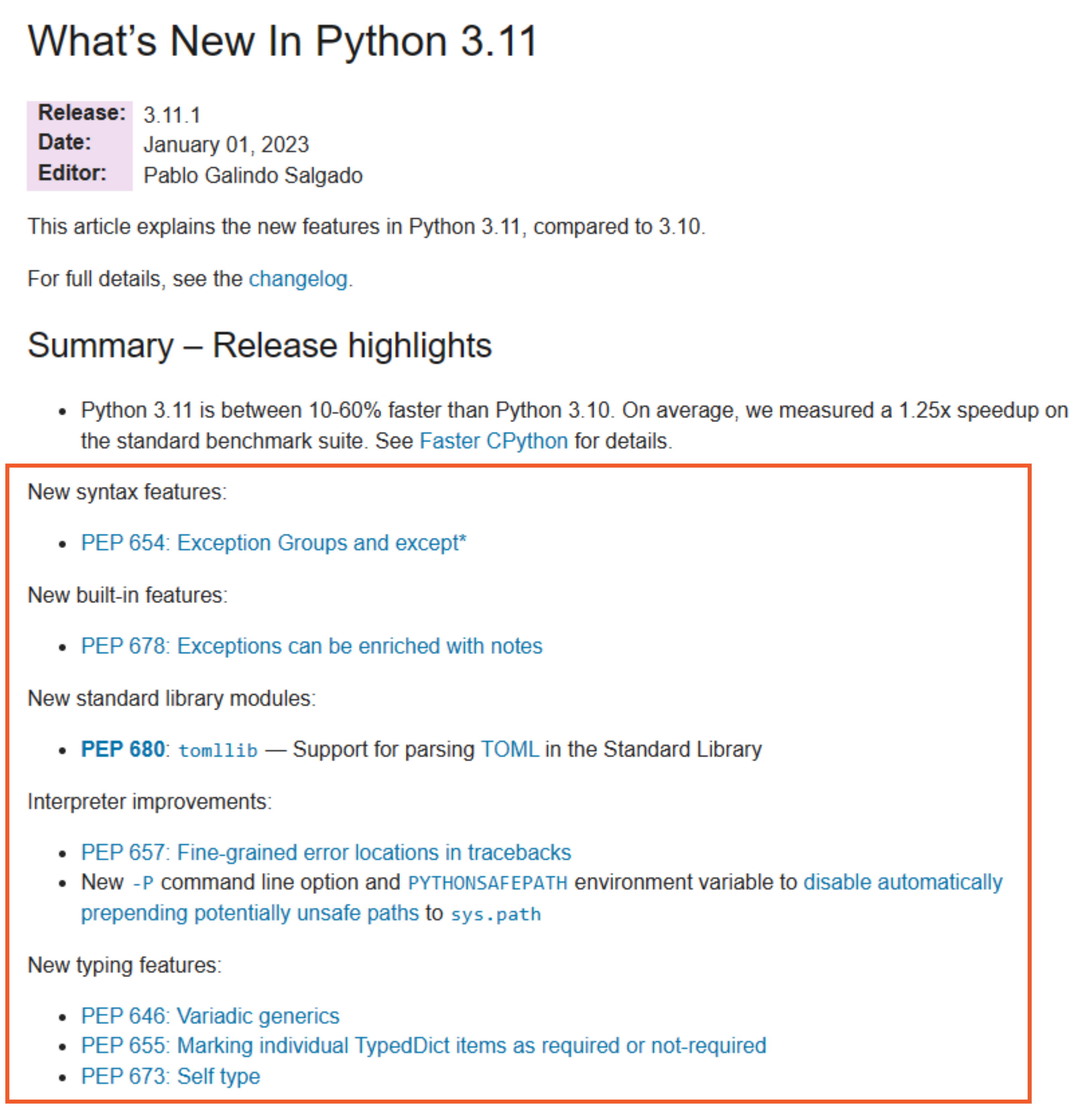
General Index
all functions, classes, terms

Glossary
the most important terms explained

Search page
search this documentation

Complete Table of Contents
lists all sections and subsections

Python Release Notes



What's New In Python 3.11

Release:	3.11.1
Date:	January 01, 2023
Editor:	Pablo Galindo Salgado

This article explains the new features in Python 3.11, compared to 3.10.

For full details, see the changelog.

Summary – Release highlights

- Python 3.11 is between 10-60% faster than Python 3.10. On average, we measured a 1.25x speedup on the standard benchmark suite. See Faster CPython for details.

New syntax features:

- PEP 654: Exception Groups and except*

New built-in features:

- PEP 678: Exceptions can be enriched with notes

New standard library modules:

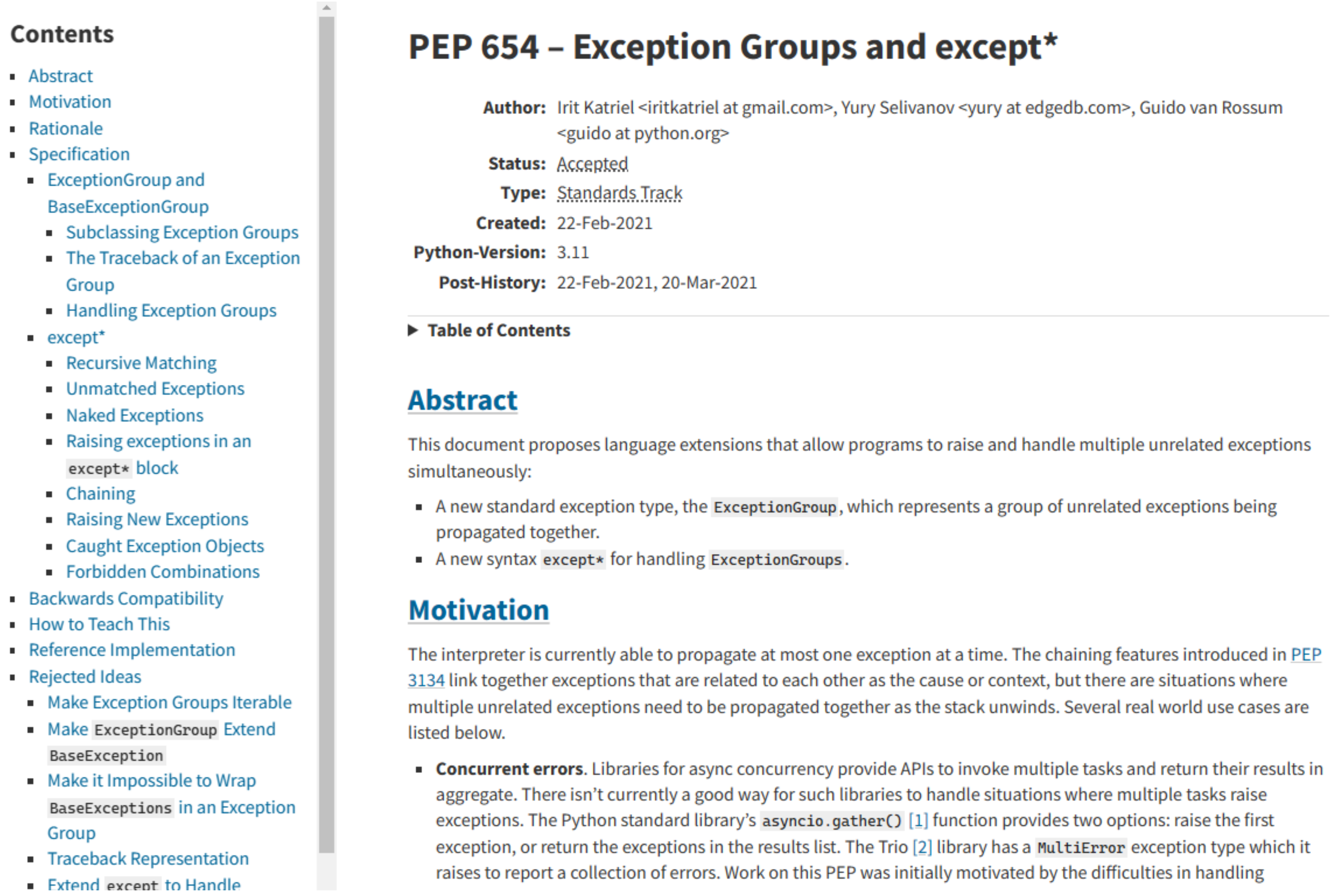
- **PEP 680**: `tomllib` — Support for parsing TOML in the Standard Library

Interpreter improvements:

- PEP 657: Fine-grained error locations in tracebacks
- New `-P` command line option and `PYTHONSAFEPATH` environment variable to disable automatically prepending potentially unsafe paths to `sys.path`

New typing features:

- PEP 646: Variadic generics
- PEP 655: Marking individual TypedDict items as required or not-required
- PEP 673: Self type



PEP 654 – Exception Groups and except*

Author: Irit Katriel <iritkatriel at gmail.com>, Yury Selivanov <yury at edgedb.com>, Guido van Rossum <guido at python.org>

Status: Accepted

Type: Standards Track

Created: 22-Feb-2021

Python-Version: 3.11

Post-History: 22-Feb-2021, 20-Mar-2021

▶ **Table of Contents**

Abstract

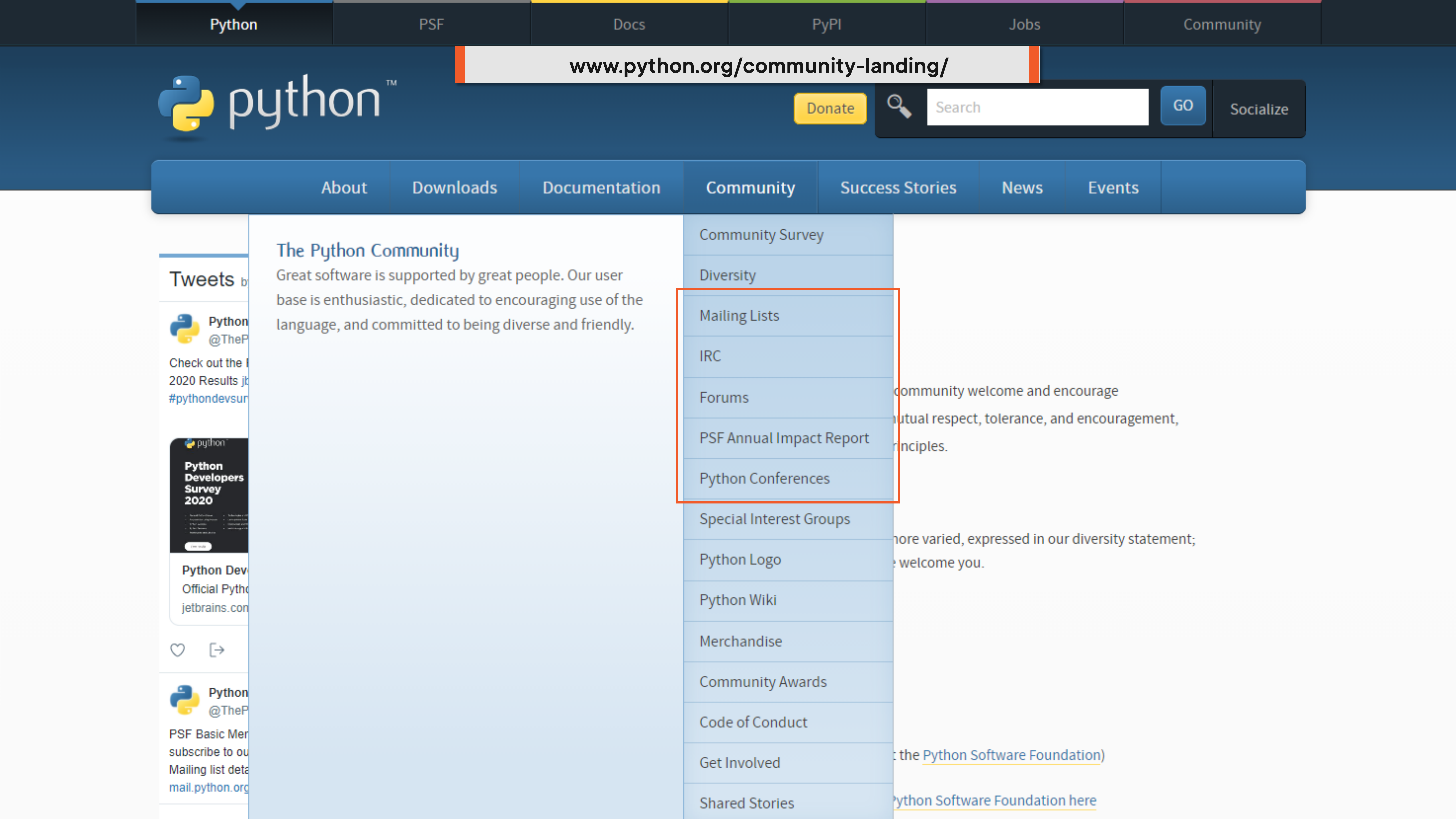
This document proposes language extensions that allow programs to raise and handle multiple unrelated exceptions simultaneously:

- A new standard exception type, the `ExceptionGroup`, which represents a group of unrelated exceptions being propagated together.
- A new syntax `except*` for handling `ExceptionGroups`.

Motivation

The interpreter is currently able to propagate at most one exception at a time. The chaining features introduced in PEP 3134 link together exceptions that are related to each other as the cause or context, but there are situations where multiple unrelated exceptions need to be propagated together as the stack unwinds. Several real world use cases are listed below.

- **Concurrent errors**. Libraries for async concurrency provide APIs to invoke multiple tasks and return their results in aggregate. There isn't currently a good way for such libraries to handle situations where multiple tasks raise exceptions. The Python standard library's `asyncio.gather()` [1] function provides two options: raise the first exception, or return the exceptions in the results list. The Trio [2] library has a `MultiError` exception type which it raises to report a collection of errors. Work on this PEP was initially motivated by the difficulties in handling www.python.org/community-landing/



The Python Community

Great software is supported by great people. Our user base is enthusiastic, dedicated to encouraging use of the language, and committed to being diverse and friendly.

Python
@TheP
Check out the
2020 Results j
#pythondevsur

Python Developers Survey 2020

Python Dev
Official Pytho
jetbrains.com

Python
@TheP
PSF Basic Mer
subscribe to ou
Mailing list deta
mail.python.org

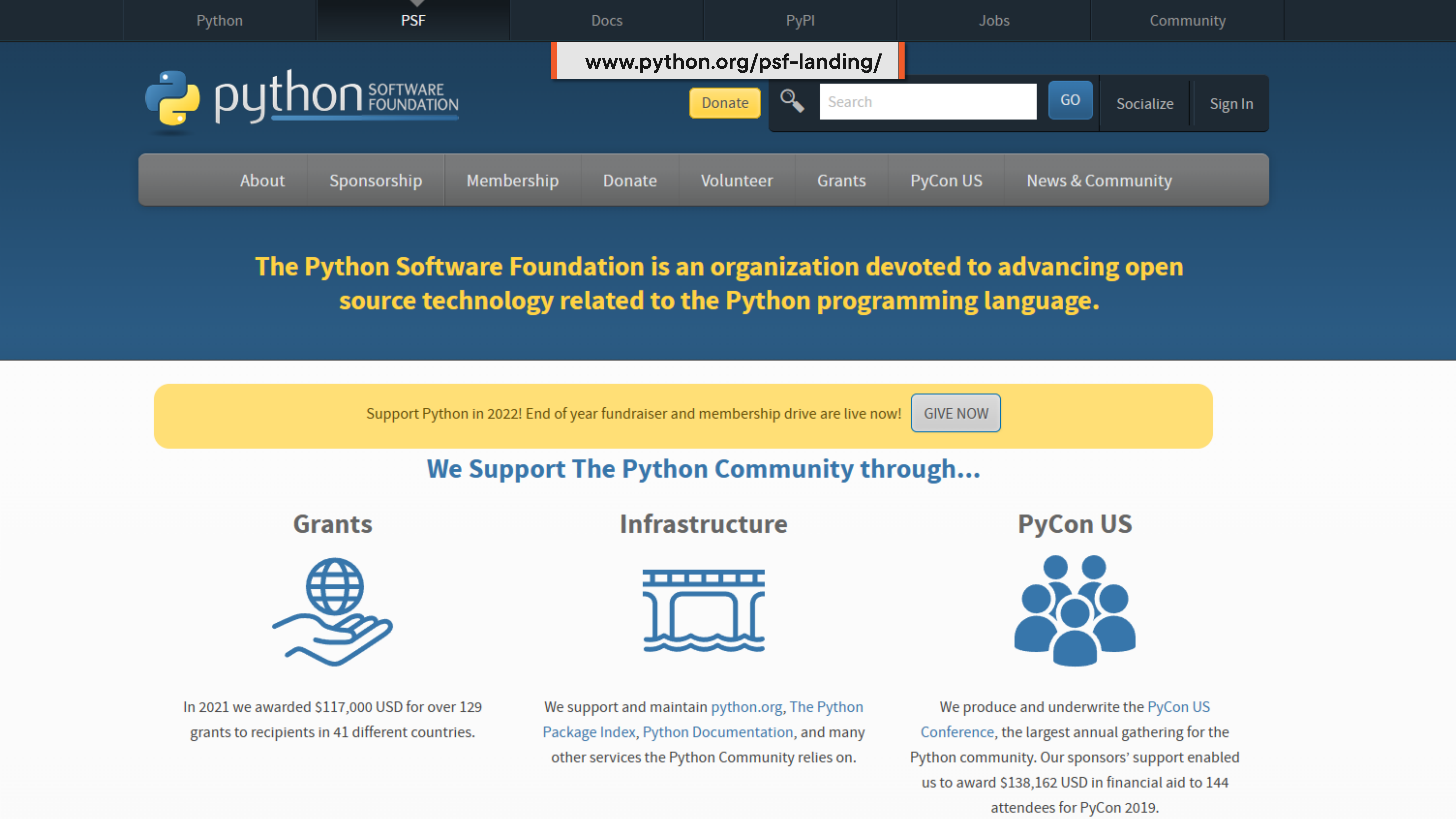
ommunity welcome and encourage
utual respect, tolerance, and encouragement,
nciples.

ore varied, expressed in our diversity statement;
welcome you.

the Python Software Foundation)

ython Software Foundation here

www.python.org/psf-landing/



The Python Software Foundation is an organization devoted to advancing open source technology related to the Python programming language.

Support Python in 2022! End of year fundraiser and membership drive are live now!

We Support The Python Community through...

Grants

In 2021 we awarded $117,000 USD for over 129 grants to recipients in 41 different countries.

Infrastructure

We support and maintain python.org, The Python Package Index, Python Documentation, and many other services the Python Community relies on.

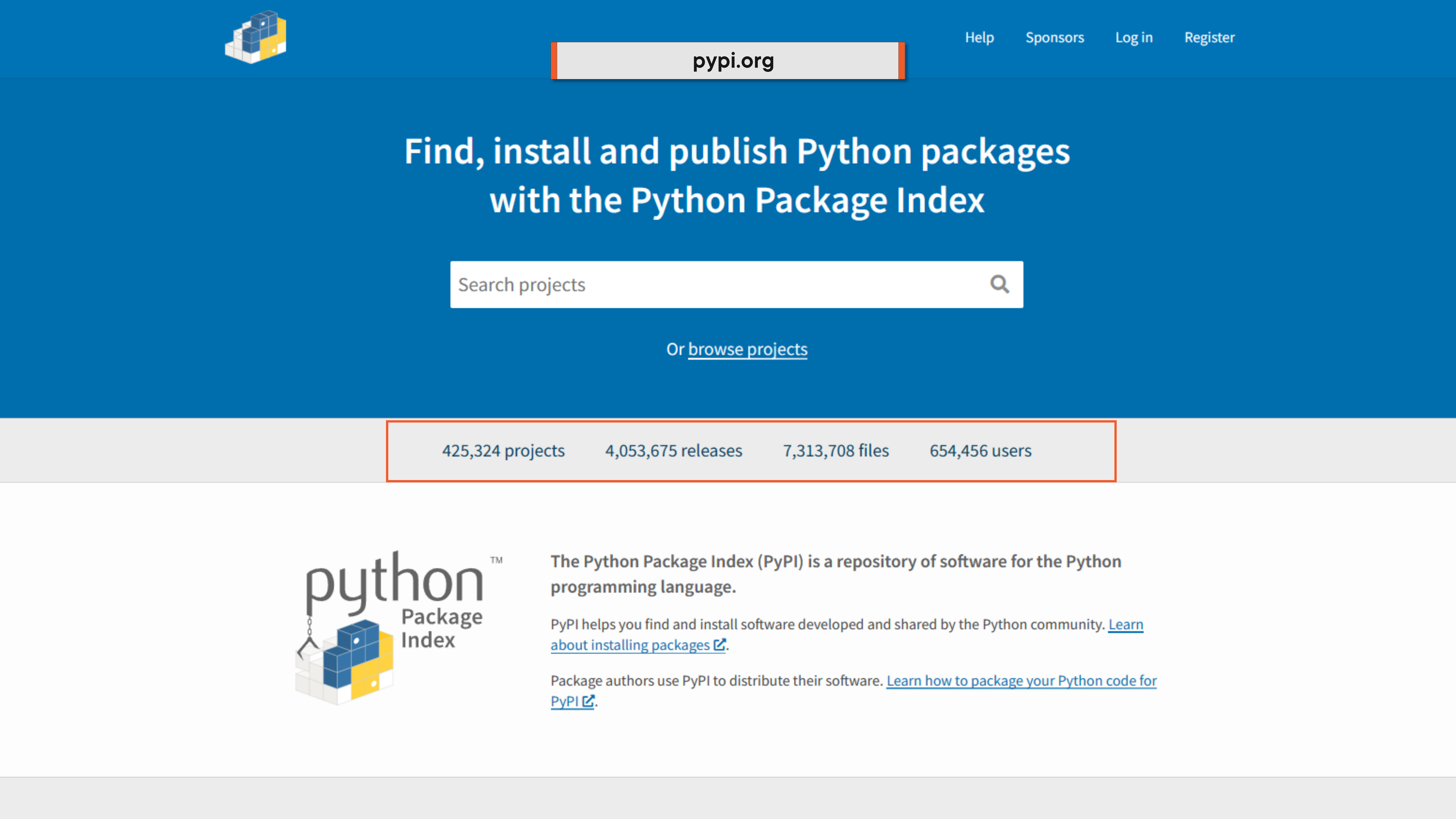
PyCon US

We produce and underwrite the PyCon US Conference, the largest annual gathering for the Python community. Our sponsors' support enabled us to award $138,162 USD in financial aid to 144 attendees for PyCon 2019.

Pro: 3rd Party Libraries

What 3rd party libraries exist?

pypi.org



Help Sponsors Log in Register

Find, install and publish Python packages with the Python Package Index

Search projects

Or browse projects

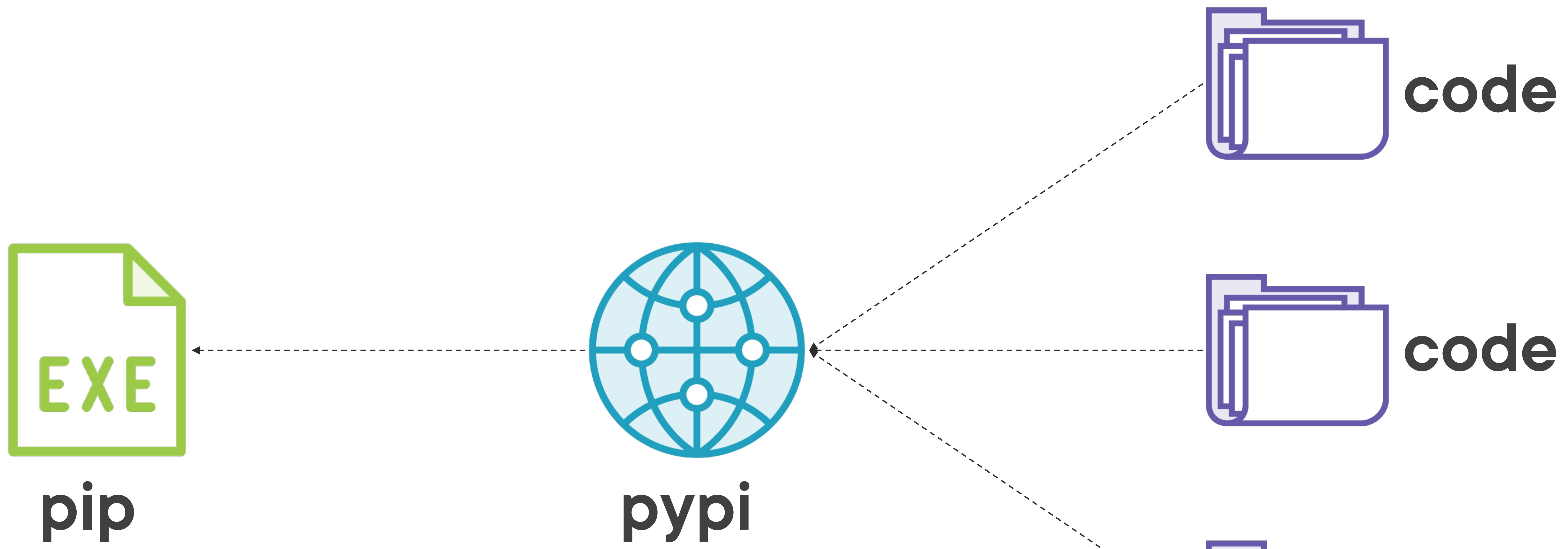
425,324 projects 4,053,675 releases 7,313,708 files 654,456 users

The Python Package Index (PyPI) is a repository of software for the Python programming language.

PyPI helps you find and install software developed and shared by the Python community. Learn about installing packages.

Package authors use PyPI to distribute their software. Learn how to package your Python code for PyPI.

Working with the Code of Others



```
$> pip install SomePackage
```

pip

Install

Uninstall

Dependencies

Package Groups

Versions

Popular 3rd Party Libraries

Data Science

NumPy, Pandas, Matplotlib, Tensorflow, Keras, Scikit Learn

Web Development

Flask, Requests, Django

Images / Computer Vision

Pillow, Pygal, OpenCV, Mahotas

Applications

wxPython (GUI), PyGTK (GUI), Fire (CLI), Kivy (Mobile/Multi-touch), Pygame

If you need it, it probably exists!

Pro: 3rd Party Tools

Python IDEs and Editors

Pydev

Pycharm

Visual Studio Code

Spyder

Sublime

Vim

GNU/Emacs

Python Code Tools

Style Guide Enforcement

flake8

Code Analysis

PyLint

Code Formatter

black

Performance Analyzers

Python Cons

Python Drawbacks

Interpreted

Slow

Not Native

High memory usage, lack of native security sandbox

Dynamic

Runtime errors

Addressing Python's Drawbacks

Interpreted

Not Native

Dynamic

Compilation / CPython

Optional Typing

Do the pros outweigh the cons?

The Big Picture Summary

Why Python?

Simple to learn

Simple to use

Great community

Widely used

High demand

What Is Python?

Dynamically-typed

Syntax

Garbage-collected

python™

General-purpose

Interpreted

Multi-paradigm

Python Pros and Cons

Pro: Comprehensive Standard Library

Pro: Community-driven

Pro: 3rd Party Libraries

Pro: 3rd Party Tools

Python Drawbacks (Interpreted, Not Native, and Dynamic)

Thank you so much!