Is Python your TYPE of programming language?

How to use static typing in Python with type hints, MyPy and Pydantic

Jack Bennett • PyOhio • 2024-07-27

What is a data type?

- A category for data in your Python program.
- Knowing the data type answers these questions:
    - "*What* ***is*** *this data?*" **(noun)**
    - "*What {can, can't} we* ***do*** *{with it, to it}?*" **(verb)**

Data types determine *what the data is* and *how we use it*

- ✅ multiply two numeric types together
- ✅ add two strings together
- ✅ add a time interval to a date/time stamp
- ✅ transform the coordinate system of a 3D point cloud
- ❌ multiply a string by a float
- ❌ divide a string by a dictionary

Python has a versatile set of built-in data types

- **Scalar types**
  - integer (int), floating-point number (float), Boolean (bool), date and time (date, time, datetime)
- **Sequence types**
  - list, tuple, range
  - Text sequence: str
  - Binary sequence: bytes
- **Mapping types**
  - dict
- Other built-in types exist, and in Python, classes define many more types.

Types of typing

- **Static typing:** data types of variables are **set at compile time**
- **Dynamic typing:** data types of variables are **set at run time**
- **Strong typing:** you **may not** use a value of one type as if it were a value of another type.
- **Weak typing:** you **may** use a value of one type as if it were a value of another type.
- Python is a **strongly-typed** and **dynamically-typed** language. (As a dynamic, *interpreted* language, Python does not have a "compile time.")

A (very incomplete) history of static typing in Python

- 2006: **PEP 3107** introduces standardized **function annotations**.
- 2012: Jukka Lehtosalo starts the **mypy** project based on his PhD work.
- 2014: **PEP 484** introduces **type hints** ... heavily influenced by **mypy**.
- 2015: **Python 3.5** includes PEP 484 with the **typing** module.
- 2017: Samuel Colvin starts the **Pydantic** project.

Show 👁️, don't just tell 🗣️

Type hints in the Python language

- Type hints have been part of the language since Python 3.5, when the **typing** module arrived.
- No accidental run time effects. (Pydantic *deliberately* uses type hints at run time.)
- Type hints provide a foundation for all kinds of useful capabilities.
    - **mypy** uses the built-in type hint syntax in Python to provide static type checking.
    - **Pydantic** uses the built-in type hint syntax in Python to provide type validation and data serialization capabilities at run time.
- https://docs.python.org/3/library/typing.html

What is the syntax for type hints?

- Specify the type after a variable name or argument using the colon operator
- Specify the return type of a function using the arrow operator
- Specify the type(s) contained in a sequence in square brackets
- typing module provides definitions for a variety of complex types - sequences, generics, unions, optional, etc

mypy summary

- **mypy** is a static type checker for Python that uses the type hints built in to the language (PEP 484 ➡️ **typing** module)
- **mypy** provides the missing "compiler" step to validate your Python code before running (testing, deploying) it.
- Type hints don't interfere with normal program operation. Your code may be valid Python even if **mypy** reports type inconsistencies.
- "Gradual typing" - **mypy** doesn't make you use static typing everywhere all at once.
- Powerful features: type inference, generics, callable types, tuple types, union types, structural subtyping and more.
- https://mypy.readthedocs.io/en/stable/

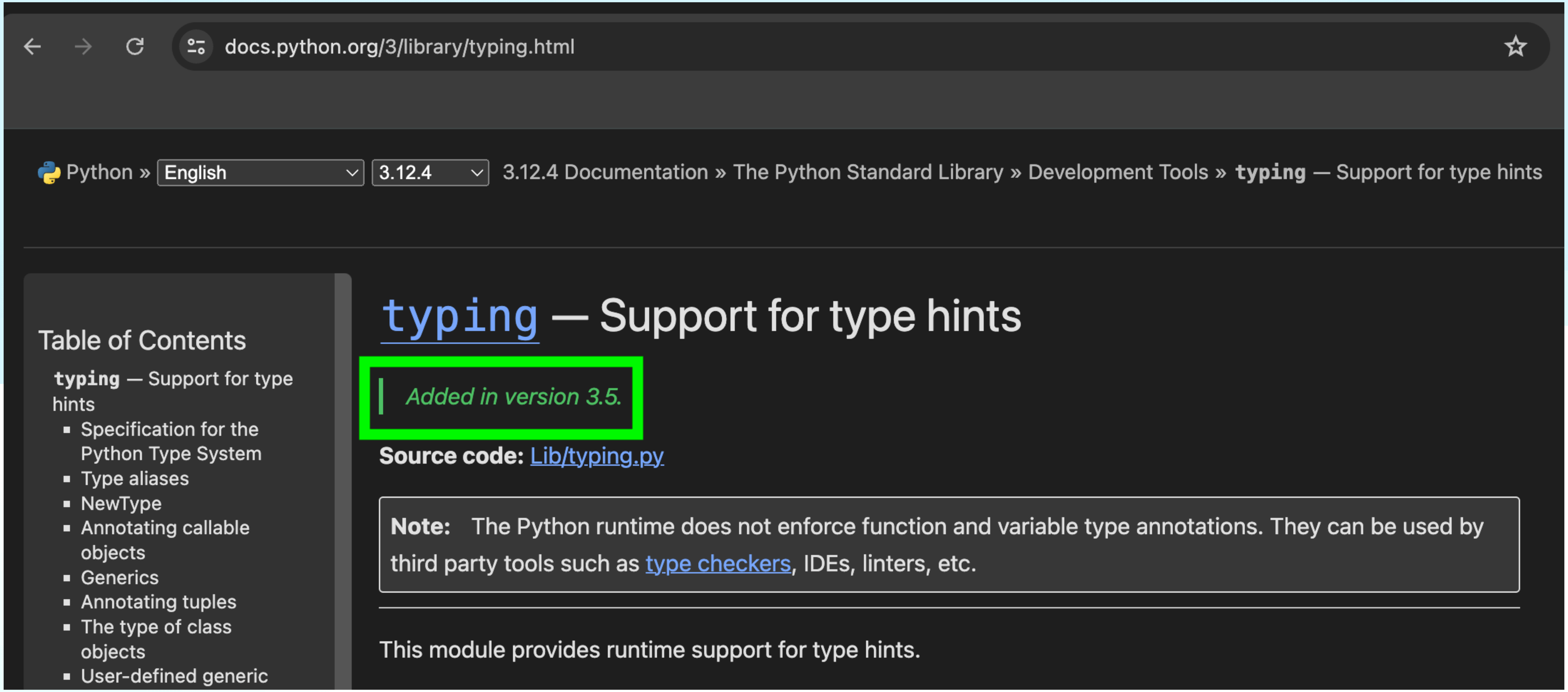
Pydantic summary

- Pydantic is a powerful 3rd party library for ingesting and validating incoming data, possibly from untrusted sources (external API, incoming file, etc).
- Pydantic uses type hints *at run time* to control data validation and serialization.
- The Pydantic designers wrote the core validation logic in Rust 🦀 to make it fast.
- "Strict mode"
  - strict=True - never convert the type of incoming data
  - strict=False - Pydantic casts data to the correct type (if it can)
- Customization: custom **validators** can use arbitrary Python code to inspect and transform incoming data according to business rules; custom **serializers** can use arbitrary Python code to emit outgoing data according to desired business rules and schemas.
- https://docs.pydantic.dev/latest/

⭐ These tools work together. ⭐

Type hints alone are just comments. mypy and Pydantic don't do anything without type hints.

Are these tools mainstream / supported / best practice?



Python » English 3.12.4 3.12.4 Documentation » The Python Standard Library » Development Tools » **typing** — Support for type hints

Table of Contents

typing — Support for type hints

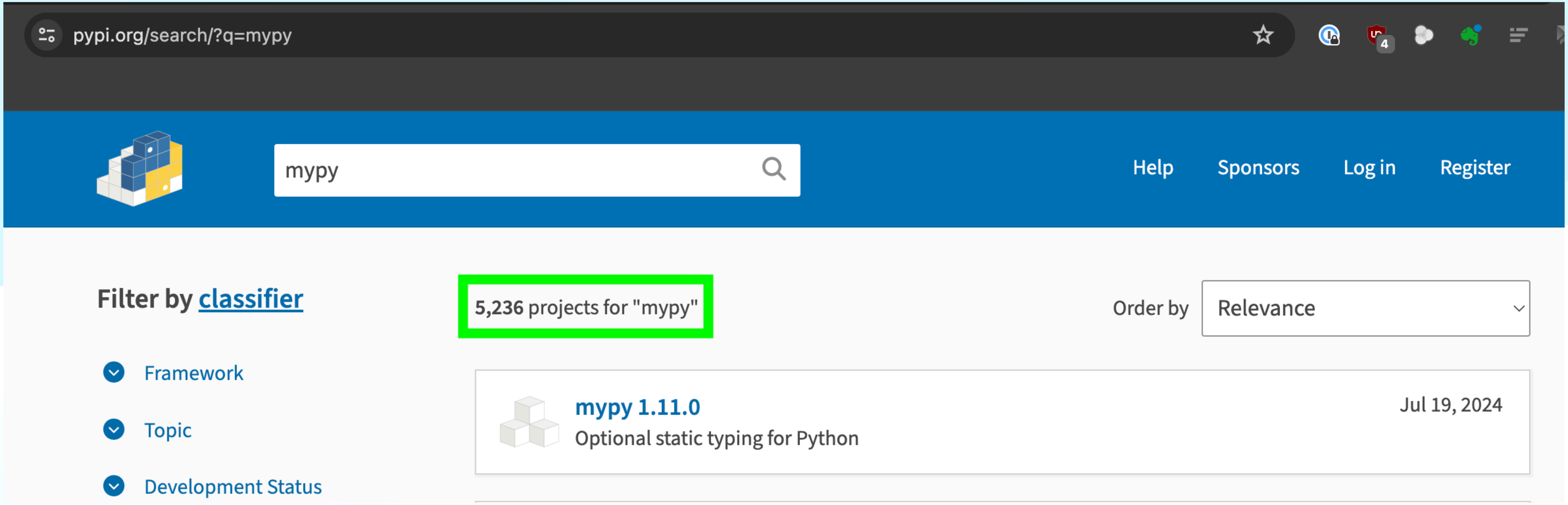
Added in version 3.5.

Source code: Lib/typing.py

Note: The Python runtime does not enforce function and variable type annotations. They can be used by third party tools such as type checkers, IDEs, linters, etc.

This module provides runtime support for type hints.

Type hints (**typing** module) have been in the Python language definition since 2015.


pypi.org/search/?q=mypy

mypy

Help Sponsors Log in Register

Filter by classifier

Framework

Topic

Development Status

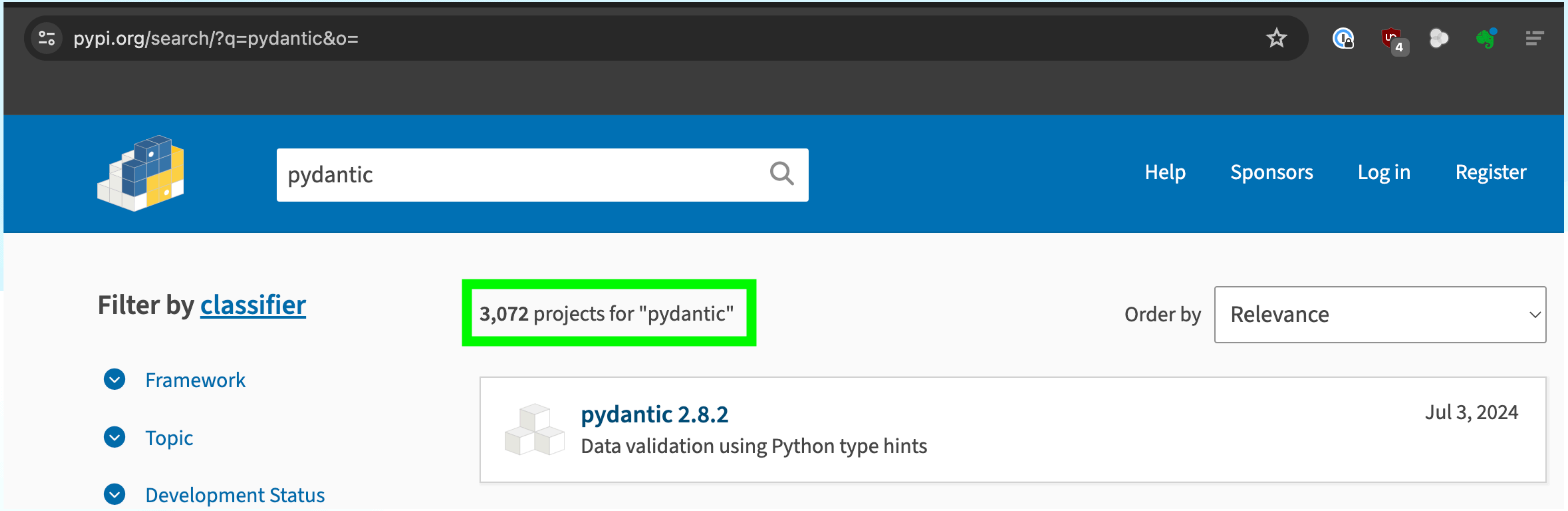
5,236 projects for "mypy"

Order by Relevance

mypy 1.11.0 Jul 19, 2024

Optional static typing for Python

mypy: Started 2012, 11,948 commits, 2.8k forks, 18k stars, 692 contributors (GitHub)



pypi.org/search/?q=pydantic&o=

pydantic

Help Sponsors Log in Register

Filter by classifier

Framework

Topic

Development Status

3,072 projects for "pydantic"

Order by Relevance

pydantic 2.8.2
Data validation using Python type hints
Jul 3, 2024

Pydantic: Started 2017, 3,041 commits, 1.8k forks, 19.9k stars, 566 contributors (GitHub)

Strong yes 👍

Migrating your codebase: quick start tips

- Start adding **type hints** incrementally
  - To all new or refactored code
  - To the most critical areas of your codebase
- Add **mypy** to your IDE, pre-commit hooks, CI/CD process.
- Use **Pydantic** as needed for validating data at system boundaries - i.e. places where data enters and exits your system.

Summary

- Specifying types in your Python code using type hints and the **typing** module is straightforward and makes your code clearer to the reader.
- **mypy** static type checking should be part of any significant Python code base, and integrated with your IDE, pre-commit hooks, and CI/CD process to improve code reliability.
- The **Pydantic** library provides a powerful set of tools for validating incoming data against simple schema definitions, and emitting JSON and other data formats.
- ➡️ **Type hints, mypy, and Pydantic are mainstream, reliable, and best practice.** ⬅️
- ➡️ **Using these language features and tools together will make your programs easier to understand, debug, and maintain.** ⬅️

How to get started?

Read the (excellent) documentation

🔄

Play and experiment (REPL, Jupyter, etc)

🎉 Have fun adding type checking to your Python programs! 🎉

Thank you for your attention! 🙏