PyCon Typing Summit 2023

Welcome!

State of Typing

—

Jelle Zijlstra
Quora

About me

- Contributor to typed Python since 2016
- Maintainer of typeshed, typing-extensions, CPython, the PEPs repo, mypy
- Author of pyanalyze (Quora's type checker)
- Sponsored six typing PEPs and authored two

History of typing

- A review of typing PEPs through the years
- Marking PEPs with one of:
  - T: type system improvements
  - U: usability improvements

Python 3.5: When it all started

- PEP 484: Type Hints (T)
  - A great start, and still covers most of the types we use most frequently
  - Tried hard to avoid changes to the core language

```
def greeting(name: str) -> str:
    return 'Hello ' + name
```

Python 3.6: First new syntax

- PEP 526: Variable Annotations (U)
  - Difficult to imagine going without this now
  - Unlocked support for dataclasses (which arrived in 3.7)
  - Also introduced ClassVar

```
class BasicStarship:
    captain: str = 'Picard'
    damage: int
    stats: ClassVar[Dict[str, int]] = {}
```

Python 3.7: Impactful changes

- PEP 561: Distributing typed packages (U)
  - A necessary enhancement
  - But has some rough edges
- PEP 563: `from __future__ import annotations` (U)
  - Aimed to solve the awkwardness of forward references and improve performance
  - The future that has stubbornly refused to arrive

Python 3.8: Basic type system features

- PEP 544: `Protocol` (T)
- PEP 586: `Literal` (T)
- PEP 589: `TypedDict` (T)
- PEP 591: `Final` / `@final` (T)
  - All of these have turned out to be essential features that are core to the type system

```python
from typing import Protocol

class SupportsClose(Protocol):
    def close(self) -> None:
        ...
```

```python
from typing import Literal

def accepts_only_four(x: Literal[4]) -> None:
    pass
```

Interlude: Is TypedDict just a compatibility hack?

- Or an important part of the core type system?
- My view: It is useful and important even in modern Python

Python 3.9: Integrating with the runtime

- PEP 585: `list[int]` [U]
  - A nice usability improvement
- PEP 593: `Annotated` [T]
  - Allow stashing non-typing information in annotations
  - Used widely, but few standardized uses have emerged

```
def find(haystack: dict[str, list[int]]) -> int:
    ...
```

Python 3.10: Various directions

- PEP 604: | for `Union`/`Optional` [U]
  - Another great usability improvement
- PEP 612: `ParamSpec` [T]
  - A complex feature that allows expressing some previously inexpressible types
- PEP 613: `TypeAlias` [U]
  - Now superseded by PEP 695's type ... = syntax

```python
# Instead of
# def f(list: List[Union[int, str]], param: Optional[int]) -> Union[float, str]
def f(list: List[int | str], param: int | None) -> float | str:
    pass
```

Python 3.11: A type system cornucopia

- PEP 646: `TypeVarTuple` (and new syntax) [T]
  - The most ambitious type system change since 3.8
- PEP 647: `TypeGuard` [T]
- PEP 655: `Required` and `NotRequired` [U]
- PEP 673: `Self` [U]
- PEP 675: `LiteralString` [T]
- PEP 681: `dataclass_transform` [T]
- (rejected) PEP 677: Callable syntax [U]

Python 3.12: The cornucopia continues

- PEP 688: Buffer types [T]
- PEP 692: `Unpack[]` for kwargs [U]
- PEP 698: `@override` [T]
- (might not make it) PEP 695: `TypeVar` syntax [U]
  - Perhaps the biggest core language change motivated by typing
- (might not make it) PEP 649: The new future [U]
- (pending) PEP 696: `TypeVar` defaults [T]
- (pending) PEP 702: `@deprecated` [T]
- (pending) PEP 705: `TypedMapping` [T]

T vs. U

- Marked all PEPs with:
  - T: type system extensions
  - U: usability improvements
- Both are important!
- T-type PEPs help type checkers understand common Python idioms
- U-type PEPs make it easier to write typed Python
  - And these often have the highest impact
  - But they’re also the most controversial
- Let’s think of more ways to improve usability

Successes and unique features

- Innovations unlocked as a result of typing
  - dataclasses (thanks to PEP 526), Pydantic, much better IDE support (e.g. Pylance)
- typing_extensions
  - Allow new type system features to be used without waiting for runtime upgrades
- Many type checkers
  - With complementary strengths

The future

- Improving usability
  - PEP 695 makes generics easy
  - PEP 649 should improve the story for runtime use of annotations
  - Better TypedDict syntax?
  - More ideas needed!
- Type system extensions
  - Some kinds of Python are still hard to express with types
  - Higher-kinded types, type-level math, intersection types, a Map[] operator?
    - Though some of these may not be worth it
- Improving documentation
  - A spec that isn't just PEPs? (Kevin Millikin's talk)
  - General user documentation for typing

Questions?